UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Uma proposta de projeto e implementação de aplicações web com foco na satisfação de requisitos não funcionais críticos

Rafael Almeida de Oliveira

CASCAVEL

2010

RAFAEL ALMEIDA DE OLIVEIRA

UMA PROPOSTA DE PROJETO E IMPLEMENTAÇÃO DE APLIÇACÕES WEB COM FOCO NA SATISFAÇÃO DE REQUISITOS NÃO FUNCIONAIS CRÍTICOS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel.

Orientador: Prof. Dr. Victor Francisco Araya Santander.

CASCAVEL

2010

RAFAEL ALMEIDA DE OLIVEIRA

UMA PROPOSTA DE PROJETO E IMPLEMENTAÇÃO DE APLICAÇÕES WEB COM FOCO NA SATISFAÇÃO DE REQUISITOS NÃO FUNCIONAIS CRÍTICOS

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Victor Francisco Araya Santander
(Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. MSc. Aníbal Mantovani Diniz
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Adair Santa Catarina
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 04 de Novembro de 2010.

DEDICATÓRIA

Dedico este trabalho a Deus e a todas as pessoas que acreditaram e me apoiaram desde o início da minha jornada neste mundo.

AGRADECIMENTOS

Agradeço a Deus por ter permitido minha existência, minha família por ter moldado a pessoa que sou, amigos que comigo lutaram para enfrentar os obstáculos com o objetivo de vencer nesta vida. A todos os professores que transmitiram um pouco de seus conhecimentos desde minha infância e em especial ao professor Victor por me auxiliar na realização deste trabalho, professores pertencentes a banca e ao meus pais por sempre dizerem que a maior herança que poderiam me deixar era a educação, além de me ajudarem a caminhar pela senda do progresso. Agradeço também minha namorada pelo incentivo e tempo dedicado para a realização deste trabalho.

Lista de Figuras

Lista de Abreviaturas e Siglas

AIR	*Adobe Integraded Runtime*
AJAX	*Asynchronous Javascript And XML*
AMF	*Action Message Format*
API	*Application Programming Interface*
AS	*Action Script*
DAO	*Data Access Object*
HDM	*Hypertext Design Model*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
IIS	*Internet Information Services*
MVC	*Model-view-controller*
MXML	*Magic Extensible Markup Language*
OASIS	*Organization for the Advancement of Structured Information Standards*
OO	Orientação a Objeto
OOHDM	*Object-Oriented Hypermedia Design Method*
PHP	*Hypertext Preprocessor*
RIA	*Rich Internet Application*
RF	Requisitos Funcionais
RMM	*Relationship Management Desgin Methodology*
RNF	Requisitos não-funcionais
SDK	*Software Development Kit*
SGBD	Sistema de Gerenciamento de Base de Dados
SOA	*Service-oriented Architecture*
SOAP	*Simple Object Access Protocol*
TI	Tecnologia da Informação
URI	*Uniform Resource Identifiers*
W3C	*World Wide Web Consortium*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*

Sumário

Resumo

Este trabalho apresenta uma proposta de projeto e implementação de aplicações *Web* usando o *framework* Adobe Flex em conjunto com a linguagem de programação PHP para satisfazer os requisitos não-funcionais críticos para esses tipos de aplicações. De forma mais específica propõe-se uma abordagem arquitetural que em conjunto com a tecnologia proposta permite aumentar significativamente o grau de satisfação de RNFs críticos para aplicações *Web*. Para demonstrar como a nossa proposta permite satisfazer estes RNFs críticos, utilizamos o *NFR Framework*, o qual permite correlacionar elementos que operacionalizam um determinado RNF com outros RNFs críticos, possibilitando que engenheiros de *software* possam realizar avaliações e tomar decisões quanto ao uso de determinadas abordagens arquiteturais. Com o catálogo do *NFR Framework* juntamente com a normativa NBR ISO/IEC 9126 elaborou-se um catálogo que auxilia o engenheiro de *software* a compreender as influências positivas e negativas que os componentes que constituem a aplicação podem exercer nos RNFs.

Palavras-chave: FLEX RNF PHP APLICAÇÃO WEB SOA

Capítulo 1

Introdução

Neste capítulo apresenta-se uma visão geral da proposta. Na seção 1.1 realiza-se uma descrição a respeito da evolução das aplicações *Web* até o momento e o surgimento da RIA (*Rich Internet Application*). Em seguida, na seção 1.2, tratam-se as principais motivações para a realização deste trabalho e na seção 1.3, são mencionados os principais elementos envolvidos na nossa proposta. As contribuições esperadas para a comunidade acadêmica e científica são descritas na seção 1.4 e ,por fim, na seção 1.5 detalha-se a estrutura dos capítulos do presente trabalho.

1.1 Contexto

Com o passar dos tempos, vem aumentando a necessidade do uso de computadores para as rotinas diárias nas empresas. Com o auxílio da Internet, as operações que anteriormente exigiam maior tempo na troca de dados entre funcionários de empresas, familiares e colegas, tornaram-se mais ágeis. Com toda essa tecnologia existente, houve a necessidade de transferir os sistemas que eram *Desktop* para a *Web*, o que possibilitou o surgimento das RIAs [1].

O termo RIA (*Rich Internet Application*) foi introduzido pela Macromedia em 2002, e definem aplicações *Web* que possuem características e funcionalidades de aplicações *Desktop*, além de poderem utilizar recursos multimídias, transferindo todo o processamento da interface para o *Browser*.

As preocupações no momento de se desenvolver uma aplicação para a *Web* são: Organizar toda a estrutura de forma que se torne mais leve, ter uma aparência amigável para o usuário, adequação do volume de requisições de dados, aumento da capacidade do servidor e segurança no acesso de dados. Estes requisitos são conhecidos na literatura da área como

requisitos não-funcionais. Entre as propostas mais utilizadas para lidar com a satisfação destes requisitos está a proposta apresentada por Chung [2]. Neste trabalho estudaremos esta proposta no contexto de satisfação de requisitos não-funcionais de aplicações *Web*.

Por outro lado, algumas das metodologias propostas para o desenvolvimento de aplicações *Web* tais como RMM [3], HDM [4] e OOHDM [5] auxiliam no planejamento e desenvolvimento de aplicações *Web*, preocupando-se tanto com a parte de codificação (estruturação do código), quanto com a parte de criação visual. Também propõem a criação de modelos conceituais e de navegação no processo de desenvolvimento destas aplicações [6]. Contudo, estudos iniciais mostram que essas metodologias se preocupam particularmente em como lidar com a navegação e organização da interface com o usuário para aplicações em que se utilizam HTML como linguagem de criação. Acreditamos que é necessário realizar estudos mais aprofundados que mostrem como utilizar estas e outras metodologias considerando novas abordagens arquiteturais como SOA (*Service-Oriented Architecture*), bem como *frameworks* emergentes propostos pela plataforma Flash.

A escolha de orientar o foco do nosso trabalho para SOA e Flex está fundamentada em alguns critérios. Inicialmente consideramos que SOA é uma abordagem arquitetural bastante recente e que tenta solucionar vários problemas existentes no desenvolvimento de aplicações *Web* através da inserção de um novo conceito denominado serviço, bem como pela definição de uma nova estratégia de estruturação de componentes arquiteturais que facilita o acesso remoto, reuso, manutenibilidade, entre outros aspectos. Um serviço em SOA é a maneira em que as operacionalizações existentes em uma aplicação são organizadas. Contém as regras de negócios necessárias para o seu funcionamento, podendo promover a organização da codificação, trazendo benefícios em relação à manutenibilidade, interoperabilidade, entre outros benefícios que serão abordados neste estudo. Por outro lado focamos nosso trabalho no uso da plataforma Flash, em específico o *framework* Adobe Flex devido ao fato de proporcionar uma nova experiência ao usuário, mostrando que aplicações RIAs são formas inovadoras de interação e exposição de informações na *Web*, além da ampla documentação existente na Internet, permitindo maior facilidade no desenvolvimento de sistemas para a *Web*.

Contudo, é importante ressaltar que existem outros *frameworks* no mercado que competem com Adobe Flex como Ajax [7], Microsoft Silverlight [8], Oracle JavaFx, entre outras. O Ajax é uma tecnologia que usa Javascript para poder fazer interações com componentes da

interface, podendo-se realizar trocas de dados com o servidor sem a necessidade de se carregar uma nova página. Já o Silverlight é um *framework* desenvolvido pela Microsoft que concorre com o Adobe Flex. Sua proposta é a de fornecer o mesmo suporte que o Flex disponibiliza para os desenvolvedores de aplicações para a *Web*, além de proporcionar soluções para transmissões multimídias de alta qualidade. Existe também o JavaFx que é outro *framework* para desenvolvimento de aplicações RIA onde juntamente com o Java propicia as mesmas experiências que os demais *frameworks*. O Flex é um *framework* que está há muito tempo no mercado e esta maturidade é um ponto positivo que o favorece na sua escolha, além de apresentar uma ferramenta que agiliza e auxilia na criação das interfaces, bem como a forma em que a interface é criada e seus componentes. Utilizando Flex, a forma de se interagir com o usuário assemelha-se com a interface de uma aplicação *Desktop* permitindo que usuários que possuem experiência com aplicações *Desktop*, possam usá-la sem maiores problemas.

Desta forma, buscar mecanismos tanto arquiteturais quanto tecnológicos que permitam facilitar o desenvolvimento de aplicações *Web* e principalmente permitam satisfazer RNFs críticos associados e estas aplicações, é uma necessidade atual.

1.2 Motivação

O desenvolvimento de aplicações *Web* tem sido um desafio para a comunidade acadêmica e industrial na área de engenharia de *software*. Faltam processos e metodologias que permitam principalmente satisfazer adequadamente os requisitos não-funcionais associados a estas aplicações.

Diversas metodologias têm sido propostas tais como OOHDM, HDM, RMM, entre outras visando apoiar o processo de desenvolvimento deste tipo de aplicações. No entanto estas metodologias não definem mecanismos que permitam relacionar diretamente artefatos desenvolvidos durante o processo de desenvolvimento com a satisfação de requisitos não-funcionais críticos. Isto implica em afirmar que é necessário estabelecer meios que efetivamente conduzam engenheiros *Web* no desenvolvimento orientado à satisfação deste tipo de requisitos. Também neste contexto é importante avaliar se determinadas tecnologias efetivamente contribuem na satisfação destes requisitos. É consensual que requisitos não-

funcionais são na maioria das vezes negligenciados, sendo determinantes para a não aceitação de produtos computacionais para a *Web* [6].

Considerando este contexto, o intuito deste trabalho é propor um modelo arquitetural apropriado para a utilização dos *frameworks* RIAs como o Adobe Flex e linguagens de programação *Web* como o PHP, com base nas metodologias existentes, tendo em vista a satisfação de RNF críticos. Pretende-se assim obter um meio de minimizar o custo de manutenção, atualização, agilidade no acesso, usabilidade, entre outros, para o desenvolvimento de aplicações *Web*.

1.3 Proposta

Neste trabalho estudaremos como o desenvolvimento de aplicações *Web* pode ser melhorado através do uso de abordagens arquiteturais e *frameworks* mais específicos como arquitetura orientada a serviço e Adobe Flex.

Assim, propõe-se um modelo arquitetural orientado a serviços que consideramos apropriados para a utilização dos *frameworks* de desenvolvimento de aplicações RIA. Particularmente neste trabalho propomos a utilização do *framework* Adobe Flex, juntamente com a linguagem de programação *Web* denominada PHP. Esta proposta surge com base no estudo e avaliação de abordagens arquiteturais, metodologias e tecnologias existentes atualmente e a necessidade de projetar e implementar aplicações *Web* que considerem adequadamente as restrições impostas pelos seus RNFs associados. Objetivamos elaborar uma proposta que permita ao engenheiro de *software* projetar uma aplicação com alto grau de qualidade, e que possa satisfazer os requisitos não-funcionais existentes no catálogo elaborado, além de ser um modelo de qualificação de aplicação.

1.4 Contribuições Esperadas

A principal contribuição do trabalho é a formulação de uma proposta que apoie engenheiros de *softwares* no desenvolvimento de aplicações *Web* via definição arquitetural utilizando o *framework* Adobe Flex com foco na satisfação dos requisitos não-funcionais críticos.

Entre outras contribuições esperadas podemos destacar:

- Obter um estudo mais aprofundado do *framework* Adobe Flex e sua relação direta com a Arquitetura Orientada a Serviços tendo como meta principal a satisfação de requisitos não-funcionais críticos para aplicações *Web*;
- Obter uma proposta de projeto arquitetural para aplicações *Web* usando como modelo de qualidade de referência o *NFR Framework* e suas estratégias de operacionalização de RNF;
- Realização de estudos de caso que mostrem a efetividade da proposta apresentada em comparação com outras estratégias de projeto arquitetural de aplicações *Web* bem como com outros *frameworks* diferentes do proposto pela Adobe;

1.5 Estrutura do Trabalho

Este trabalho está dividido em 6 capítulos. O capítulo 2 apresenta uma visão sobre Arquitetura de Aplicações *Web*, a sua evolução e metodologias de desenvolvimento existentes, além de abordar sobre os conceitos associados à RNFs, a norma NBR ISO/IEC 9126 e o catálogo do *NFR Framework*, utilizados como modelos de qualidade de *software*.

No capítulo 3 aborda-se o SOA, seus conceitos e como os seus serviços são disponibilizados para serem acessados pela Internet. Um exemplo prático da sua utilização é apresentado, e por fim uma comparação com as outras arquiteturas *Web* existentes.

O capítulo 4 descreve os conceitos associados às tecnologias RIAs (*Rich Internet Applications*) e uma descrição dos principais *frameworks* existentes no mercado como Microsft Silverligt, Oracle JavaFX, Adobe Flex e HTML 5.

No capítulo 5 apresenta-se a proposta, uma descrição da arquitetura criada para o desenvolvimento de aplicações *Web* e as estratégias tomadas para que se possam satisfazer os RNFs críticos.

Por fim o capítulo 6 tratará do estudo de caso, descrevendo os passos tomados para que a proposta seja bem sucedida.

Capítulo 2

Arquitetura de Aplicações WEB e RNFs

Neste capítulo serão abordados na seção 2.1, os conceitos básicos associados a arquiteturas das aplicações *Web* e na seção 2.2 uma descrição dos RNFs, e mais especificamente sobre a proposta apresentada pelo *NFR Framework*. Na seção 2.3 apresenta-se uma discussão da necessidade de se considerar de forma mais explícita a proposta arquitetural e implementação de aplicações *Web*, e também os RNFs críticos para esses tipos de aplicações. Na seção 2.4 são apresentadas as considerações finais do capítulo.

2.1 Arquitetura de Aplicações *Web*

Com o passar dos tempos, as formas de se construir aplicações foram sendo aprimoradas, como mostra a figura a seguir:

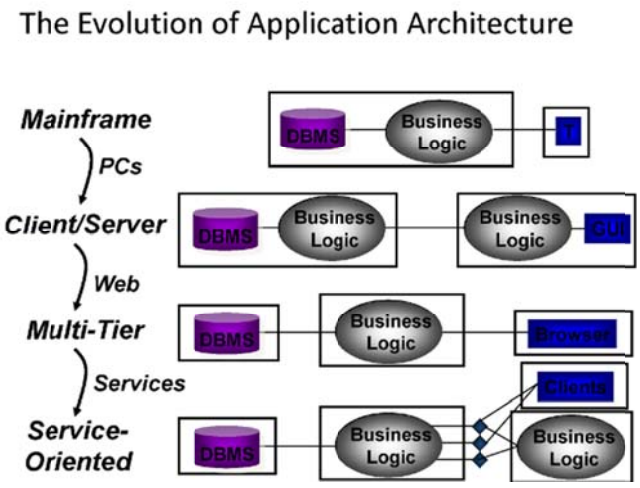


Figura 2.1: Evolução das arquiteturas das aplicações (extraído de [9]).

No início da informatização, todas as aplicações eram produzidas para *mainframes*, os códigos não possuíam uma organização adequada, eram todos aglomerados. Era uma época em que se preocupavam em fazer aplicações que atendessem as necessidades imediatas dos usuários sem a preocupação quanto as melhorias que a aplicação pode exigir futuramente .

Com o crescimento na utilização dos *Desktops*, houve também o crescimento na aquisição de *softwares*. Ao perceberem que os meios utilizados para a produção de sistemas possuíam deficiências, os desenvolvedores da época buscaram outros métodos para organizarem as codificações através de estruturas mais elaboradas. Foi a partir daí que nasceu o termo arquitetura de *software*, que segundo Antonio [10], é o estudo da organização global dos sistemas de *software* bem como do relacionamento entre seus subsistemas e componentes. Ainda nesta época usavam-se aplicações cliente/servidor que possuíam uma interface *Desktop* que se comunicava com um servidor operando em outra máquina.

Neste contexto, cabe ressaltar a separação do modelo de implementação do modelo de domínio de interface humana, o qual é um princípio de design de componentes destinados a manter os detalhes da "interação humana" separada do modelo de implementação de domínio. A justificativa para esta estratégia de particionamento, é que a interação humana pode assumir várias formas, tais como uma interface de *Web Browser* e uma interface gráfica convencional. Sem essa separação, o componente de domínio é escrito várias vezes, aumentando custos e dificultando sua reutilização. Esta técnica é a base do padrão MVC, o qual é um dos padrões mais utilizados para estruturar aplicações *Web*.

O MVC (*Model–View–Controller*) é segundo [11], “um padrão arquitetônico que prevê a separação da interface do modelo de negócio. Esta abordagem divide um sistema que requer uma interface humano-computador que manipula um conjunto subjacente de informações em três componentes. O modelo define as informações de domínio no sistema, independente de como o usuário interage com a informação. A visão define o modo como a informação é apresentada ao ser humano, e do conjunto aceitável de recursos de manipulação. O controlador processa as seqüências de entradas humanas. Um mecanismo é fornecido também para garantir a consistência, propagando mudanças na interface para o modelo”.

Anteriormente ao MVC, os sistemas continham sua estrutura fundida, sem nenhum senso de separação, prejudicando no entendimento do código criado para a aplicação, acarretando em problemas de manutenção, tanto no momento de resolver problemas contidos no sistema,

quanto em agregar novas funcionalidades, além de não permitir uma fácil reutilização do código.

Também cabe ressaltar neste contexto o estilo arquitetural que propõe a divisão do sistema em camadas. Esse estilo surgiu da vontade de criar um número maior de camadas de acordo com a necessidade do projeto, permitindo uma melhor organização do sistema a fim de separar os elementos em grupos com diferentes níveis de abstração, permitindo maior adaptação de código, manutenção e reutilização. Mais recentemente foram apresentadas algumas metodologias de desenvolvimento de aplicações *Web* que propunham arquiteturas que consideravam alguns aspectos diferenciados em relação aos estilos arquiteturais tradicionais. Assim, estas metodologias propõem etapas específicas para determinar os componentes necessários para uma aplicação *Web*, bem como o modo como estes componentes são organizados e se comunicam.

Por exemplo, a metodologia HDM proposta por Garzotto, Mainetti e Paolini, no ano de 1993, foi a primeira metodologia criada para definir as estruturas e interações em aplicações para *Web* que fossem estáticas. O RMM foi criado por Isakowitz, Stohr e Balasubramanian no mesmo ano, mas com o diferencial de permitir projetar aplicações que possuíam conteúdo freqüentemente alterado. Mas havia um problema que afetava as duas metodologias, o de não ter formas de projetar a navegação e interface. Visando sanar esse obstáculo, surge o OOHDM criado por Rossi e Schwabe, no ano de 1994, que permitia o uso de características de orientação a objetos e propunha solucionar os problemas de reusabilidade e manutenção, através da utilização de quatro atividades (Projeto Conceitual, Projeto Navegacional, Projeto de Interface Abstrata e Implementação), sendo documentadas todas as decisões de projetos tomadas durante o processo de desenvolvimento e aperfeiçoamento do sistema *Web*. Na fase de Projeto Conceitual são definidos os objetos, seus relacionamentos e colaborações a fim de representar todo o esquema conceitual da aplicação. A fase Projeto Navegacional tem como objetivo definir a navegação do sistema *Web*, podendo ser definidos por nós, links, estruturas de acesso e esquema de classe navegacional, delimitando desta forma, as transformações navegacionais que podem ocorrer no espaço navegacional. Com a separação existente entre o Projeto Navegacional e Projeto Conceitual que o OOHDM promove, há a possibilidade de utilizar o mesmo esquema conceitual caso necessite ter níveis de usuários diferentes. A atividade seguinte, denominada Projeto de Interface Abstrata, permite desenvolver as telas de

apresentação do sistema. A última fase do OOHDM é denominada implementação, fase em que todas as descrições definidas nas atividades anteriores são codificadas.

Apesar destas metodologias apresentarem propostas que melhoram o processo de construção de aplicações *Web*, a comunidade cientifica percebeu que havia uma grande quantidade de aplicações com funcionalidades redundantes e as metodologias atuais não consideravam estes aspectos. Neste cenário surge o SOA, visando transformar funcionalidades em serviços, para que possam ser reutilizados posteriormente, permitindo que vários aplicativos possam se valer das mesmas funcionalidades. Este padrão arquitetural será explicado com maiores detalhes no capítulo 3, focando a descrição do seu funcionamento, bem como vantagens e desvantagens de sua utilização.

2.2 RNFs

As aplicações, em geral, possuem suas complexidades determinadas através de seus RFs, que são as funcionalidades disponíveis na aplicação, e dos RNFs, que fixam restrições sobre os RFs na forma de como serão implementados.

Chung [2] aponta que RNFs são de grande importância para o desenvolvimento de aplicações, pois são cruciais e definem se uma aplicação será ou não bem sucedida, além de serem os mais caros e trabalhosos no momento da manutenção.

Visando atender a preocupação por parte dos engenheiros de *software* em obter *softwares* de qualidade é que tanto as comunidades acadêmicas quanto industrial focaram seus esforços em propor novas normativas e ferramentas para auxiliar nesta difícil tarefa. Neste trabalho iremos utilizar a normativa NBR ISO/IEC 9126 e o *NFR Framework* proposto por Chung que é composto de catálogo de RNFs e uma ferramenta que auxilia no esboço destes requisitos, visando satisfazer grande parte de requisitos não-funcionais possíveis.

2.2.1 NBR ISO/IEC 9126

A ISO/IEC 9126 é uma norma criada pela ISO (*International Organization for Standardization*) com o propósito de ser um modelo de qualidade de *software*, que posteriormente foi traduzida, revisada e melhorada pela ABNT (Associação Brasileira de Normas Técnicas) resultando na norma NBR ISO/IEC 9126 [32].

Segundo [32], este modelo é intitulado como "Engenharia de *software* - Qualidade do produto", consistindo das seguintes partes:

- Parte 1: Modelo de qualidade;
- Parte 2: Métricas externas;
- Parte 3: Métricas internas;
- Parte 4: Métricas de qualidade de uso.

A parte 1 denominada NBR ISO/IEC 9126-1 é composta por um conjunto de atributos utilizados para definir um modelo de qualidade que podem ser usados em qualquer produto de *software*. Este modelo é dividido em duas partes: modelo de qualidade interna e qualidade externa e modelo de qualidade de uso.

As partes 2, 3 e 4 denominadas NBR ISO/IEC 9126-2, NBR ISO/IEC 9126-3 e NBR ISO/IEC 9126-4 respectivamente, são métricas utilizadas para a avaliação dos modelos de qualidades abordadas pela NBR ISO/IEC 9126-1.

O NBR ISO/IEC 9126-1 possui um catalogo de RNFs resumidos abaixo. Mais detalhes sobre estes RNFs serão apresentados na seção 2.2.3, a qual apresenta no nosso entendimento os RNFs críticos para aplicações *Web*.

Em relação à qualidade interna e qualidade externa:

1. Funcionalidade
    1.1. Adequação
    1.2. Acurácia
    1.3. Interoperabilidade
    1.4. Segurança de acesso
    1.5. Conformidade relacionada à funcionalidade
2. Confiabilidade
    2.1. Maturidade
    2.2. Tolerância a falhas
    2.3. Recuperabilidade
    2.4. Conformidade relacionada à confiabilidade
3. Usabilidade
    3.1. Inteligibilidade

3.2. Apreensibilidade
3.3. Operacionalidade
3.4. Atratividade
3.5. Conformidade relacionada à usabilidade

4. Eficiência
4.1. Comportamento em relação ao tempo
4.2. Utilização de recursos
4.3. Conformidade relacionada à eficiência

5. Manutenibilidade
5.1. Analisabilidade
5.2. Modificabilidade
5.3. Estabilidade
5.4. Testabilidade
5.5. Conformidade relacionada à manutenibilidade

6. Portabilidade
6.1. Adaptabilidade
6.2. Capacidade para ser instalado
6.3. Coexistência
6.4. Capacidade para substituir
6.5. Conformidade relacionada à portabilidade

Em relação à qualidade de uso:

1. Eficácia
2. Produtividade
3. Segurança
4. Satisfação

Esta normativa por ser de âmbito internacional, teve um rigoroso comitê julgador que validou a proposta dando a esta uma grande confiabilidade nos seus dados, além de poder ser utilizada em aplicações de qualquer natureza. Assim, na seção 2.2.3 estes requisitos não-funcionais com suas respectivas subdivisões serão descritos detalhadamente. A seguir será abordada uma ferramenta que auxiliará na representação dos RNFs. Esta ferramenta é

denominada *NFR Framework*, e dispõe de uma estrutura de grafos, permitindo representar de forma gráfica os RNFs e as satisfações que as arquiteturas e tecnologias envolvidas para a concepção do produto de *software*, permitindo que posteriormente possa ser feito o relacionamento das operacionalizações e RNFs, além de possibilitar a avaliação de suas interdependências, tendo em vista que a NBR ISO/IEC 9126-1 não dispõem de nenhuma ferramenta que possibilite esse tipo de operação.

2.2.2 *NFR Framework*

Através de vários trabalhos desenvolvidos sobre requisitos não funcionais [30], [31], [2], tornou-se possível o desenvolvimento de um *framework* chamado *NFR Framework*, em que os requisitos não-funcionais são tratados como metas, podendo ser conflitantes entre si, devendo ser identificados na sua forma mais geral e refinados em conjuntos até que se alcance a satisfação do requisito geral.

O *NFR framework* disponibiliza uma estrutura para representar e registrar o raciocínio de cada processo em grafos, chamado *Softgoal Interdependency Graph* (SIG), além de oferecer um catálogo de conhecimento. Este grafo baseia-se em árvores de E/OU usadas na resolução de problemas, estes itens que constituem a árvore representam metas que ilustram RNFs que podem ser raramente considerados como totalmente "satisfeitos". A avaliação de cada meta dentro de um projeto determinará quais operacionalizações podem contribuir positivamente ou negativamente, além de definir quais os RNFs estão efetivamente sendo alcançados.

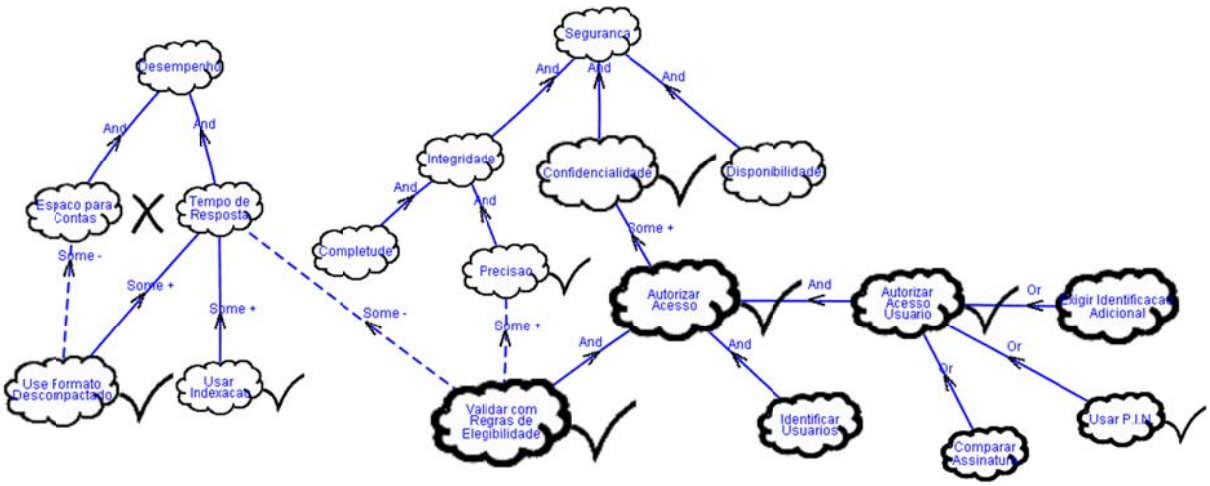


Figura 2.2: *Softgoal Interdependency Graph* (SIG).

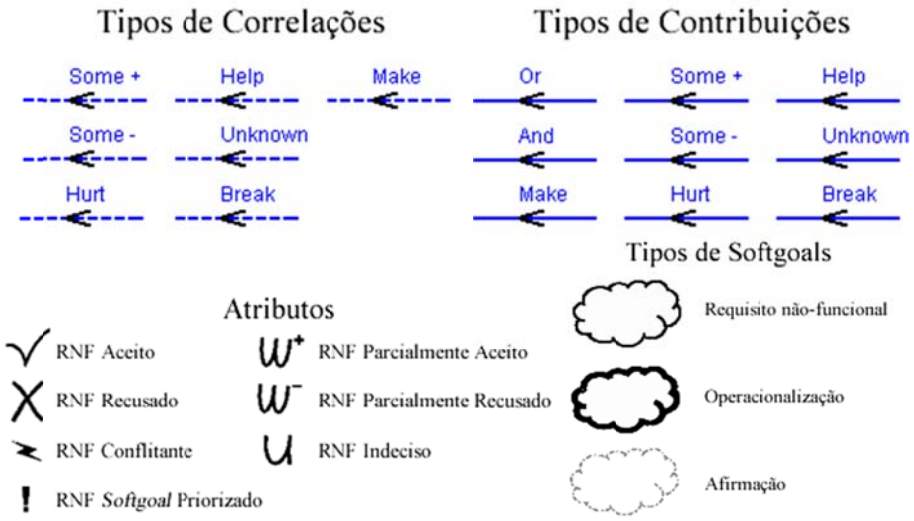


Figura 2.3: Legenda SIG.

No grafo SIG apresentado na figura 2.2, temos os RNFs segurança e desempenho com suas ramificações, espaço para contas e tempo de resposta, refinamento do RNF desempenho e completude e precisão, refinamento de integridade, que juntamente com confidencialidade e disponibilidade, pertencem ao refinamento do RNF segurança, todos eles representados pelas nuvens de borda fina. As nuvens de borda grossa, representam as operacionalizações que podem ser de influência positiva, como no caso de Autorizar acesso em relação a confidencialidade, ou de influência negativa, como no caso de Validar com Regras de Elegibilidade em relação a tempo de resposta, ambas representadas por setas de linhas tracejadas. Já as setas de linhas contínuas, como por exemplo, refinamentos em relação à segurança, determinam que para se chegar a determinado grau de qualidade relacionado a segurança, deve-se satisfazer seus refinamentos, ou pode-se satisfazer qualquer uma das opções, por exemplo, Autorizar Acesso Usuário que pode ser satisfeito através da autorização utilizando o uso de P.I.N. (*Personal Identification Number*), Comparar Assinaturas ou Exigir Identificação Adicional.

O *NFR Framework* contribui no contexto de modelagem de processos de negócios, para a melhoria na qualidade da especificação, modelagem, análise e decisões tomadas no decorrer do processo de desenvolvimento do sistema de informação, tornando explícitas as ligações e as soluções entre *softgoals*, permitindo desta forma uma visualização clara dos objetivos, auxiliando na análise do sistema de quanto o projeto está sendo fiel as especificações de qualidade que o usuário deseja.

2.2.3 RNF Críticos para Aplicações *Web*

Através da análise do catálogo do *NFR Framework* e da norma NBR ISO/IEC 9126-1, percebeu-se que ambos tinham RNFs em comum, além de descreverem outros RNFs que julgavam importantes. Com isso, houve a necessidade de elaborar um catálogo que englobasse os RNFs abordados por cada um deles, resultando no seguinte Grafo SIG ilustrado na figura 2.2. Esta junção resultará em um catálogo que terá uma abrangência maior, em relação aos catálogos utilizados, possibilitando satisfazer uma série de RNFs aumentando a qualidade do *software* a ser desenvolvido.

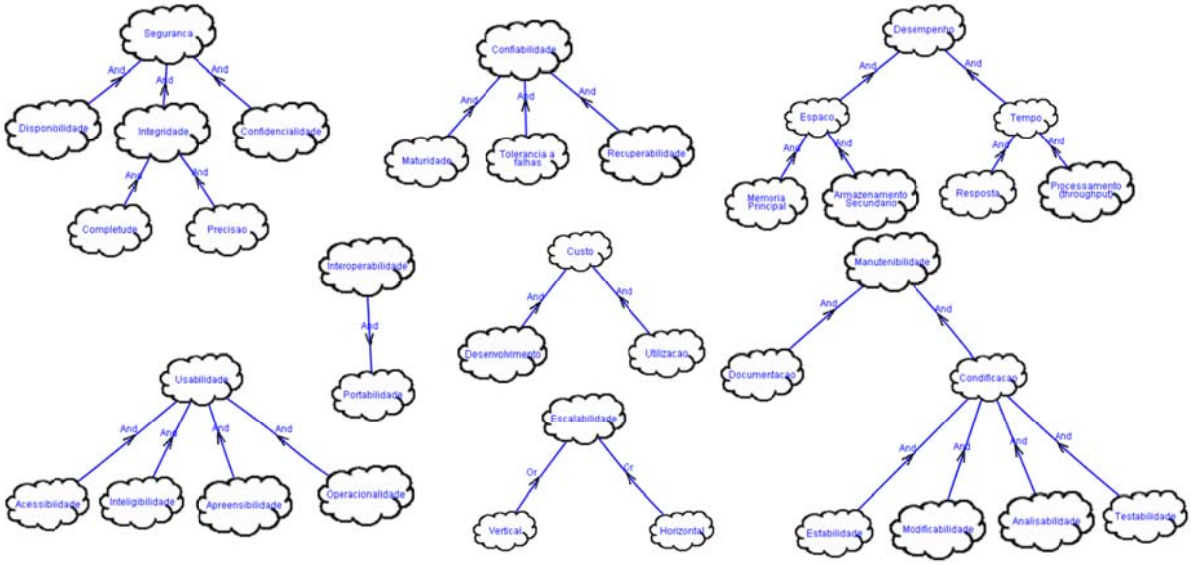


Figura 2.4: Grafo SIG dos RNFs críticos.

A seguir serão descritos os RNFs mencionados na figura anterior:

1. Segurança: Capacidade do produto de *software* de proteger informações e dados, de forma que pessoas ou sistemas não autorizados não possam lê-los nem modificá-los e que não seja negado o acesso às pessoas ou sistemas autorizados. É subdividido em:

    1.1. Disponibilidade: Capacidade de um produto de *software* de estar pronto para executar uma função requisitada num dado momento, sob condições especificadas de uso.

    1.2. Integridade: Capacidade de um produto de *software* de disponibilizar informações confiáveis e inalteradas.

1.2.1. Precisão: Capacidade de um produto de *software* de dispor determinadas informações com certo grau de precisão;

1.2.2. Completude: Capacidade do produto de *software* de disponibilizar as informações com certo grau de completude;

1.3. Confidencialidade: Capacidade do produto de *software* de restringir o acesso a dados da aplicação que são considerados confidenciais por usuários não autorizados.

2. Confiabilidade: Capacidade do produto de *software* de manter um nível de desempenho especificado, quando usado em condições especificadas. É subdividido em:

2.1. Maturidade: Capacidade do produto de *software* de evitar falhas decorrentes de defeitos no *software*.

2.2. Tolerância a falhas: Capacidade do produto de *software* de manter um nível de desempenho especificado em casos de defeitos no *software* ou de violação de sua interface especificada.

2.3. Recuperabilidade: Capacidade do produto de *software* de restabelecer seu nível de desempenho especificado e recuperar os dados diretamente afetados no caso de uma falha.

3. Desempenho: Capacidade do produto de *software* de manter um nível de desempenho especificado, quando usado em condições especificadas. É subdividido em:

3.1. Espaço

3.1.1. Memória Principal: Capacidade do produto de *software* de utilizar uma quantidade especifica de espaço em memória volátil para ser executado.

3.1.2. Armazenamento Secundário: Capacidade do produto de *software* de utilizar uma quantidade específica de espaço em disco para ser armazenado.

3.2. Tempo

3.2.1. Resposta: Capacidade do produto de *software* responder para que alguma operação seja concluída, dentro de um tempo de reposta aceitável pelo usuário.

3.2.2. Processamento (Throughput): Capacidade do produto de *software* de processar uma quantidade especifica de dados em um intervalo pré-definido de tempo.

4. Manutenibilidade: Capacidade do produto de *software* ser modificado após sua entrega (ao cliente) para corrigir erros, melhorar seu desempenho ou qualquer outro atributo ou para adaptar o produto a um ambiente modificado. É subdividido em:

4.1. Documentação: Capacidade do produto de *software* de conter diagramações, descrições e detalhamentos de todos os processos de concepção e aperfeiçoamento do *software*.

4.2. Codificação: Refere-se à parte de implementação do produto de *software*, sendo subdividido em:

4.2.1. Estabilidade: Capacidade do produto de *software* de evitar efeitos inesperados decorrentes de modificações no *software*.

4.2.2. Modificabilidade: Capacidade do produto de *software* de permitir que uma modificação especificada seja implementada.

4.2.3. Analisabilidade: Capacidade do produto de *software* de permitir o diagnóstico de deficiências ou causas de falhas no *software*, ou a identificação de partes a serem modificadas.

4.2.4. Testabilidade: Capacidade do produto de *software* de permitir que seja validado quando modificado.

5. Usabilidade: Capacidade do produto de *software* de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas. É subdividido em:

5.1. Acessibilidade: Capacidade do produto de *software* de possibilitar ao usuário com limitações físicas possa utilizar a aplicação.

5.2. Inteligibilidade: Capacidade do produto de *software* de possibilitar ao usuário compreender se o *software* é apropriado e como ele pode ser usado para tarefas e condições de uso específicas.

5.3. Apreensibilidade: Capacidade do produto de *software* de possibilitar ao usuário aprender sua aplicação.

5.4. Operacionalidade: Capacidade do produto de *software* de possibilitar ao usuário operá-lo e controlá-lo.

6. Escalabilidade: Capacidade do produto de *software* de suportar aumentar o poder computacional de um sistema, tanto no aumento na quantidade de recursos, quanto no melhoramento dos recursos existentes. É subdividido em:

6.1. Vertical: Capacidade do produto de *software* de possibilitar a adição de recursos em um único nó (computador) do sistema.

6.2. Horizontal: Capacidade do produto de *software* de possibilitar a adição de outros nós (computadores) ao sistema.

7. Interoperabilidade: Capacidade em que dois sistemas ou conjuntos destes poderem trocar informações entre si independentes da tecnologia utilizada. É subdividido em:

7.1. Portabilidade: Capacidade de um *software* funcionar corretamente em qualquer sistema operacional (aplicação *Desktop*), navegador (aplicação *Web*) e/ou dispositivo móvel (aplicação *Mobile*).

8. Custo: Capacidade do produto de *software* de possuir baixo custo de desenvolvimento e recursos para posterior utilização. É subdividido em:

8.1. Desenvolvimento: Capacidade do produto de *software* de possuir baixo custo de aquisição de tecnologias e treinamentos necessários para a concepção do produto de *software*.

8.2. Utilização: Capacidade do produto de *software* de necessitar por parte do usuário final, a compra de *softwares* e/ou *hardwares* necessários para garantir o seu correto funcionamento.

Nesta seção foram abordados os RNFs contidos no catálogo do *NFR Framework* e na norma NBR ISO/9126-1 a fim de expô-los para melhor compreensão e de como cada um deles contribui para designar se uma aplicação, de modo geral, pode ser considerada de qualidade. Estes RNFs serão considerados como base para a elaboração da proposta de arquitetura de Aplicações *Web*, pois desta forma poderá validar quais RNFs que a arquitetura proposta pode atender e com o auxílio do *NFR Framework*, representar de forma gráfica as interdependências que as operacionalizações e RNFs possam ter, auxiliando os engenheiros de *software* na compreensão do modelo arquitetural proposto.

2.3 Integrando Arquitetura e RNFs

O surgimento de novas arquiteturas de *software* foi decorrente da procura pela satisfação de requisitos não-funcionais, tendo como meta principal a busca de uma técnica que permitisse aos engenheiros de *software* projetar aplicações com alto nível organizacional, permitindo maior facilidade de manutenção do sistema, além de construir componentes que pudessem ser facilmente reutilizados.

No início da programação, os primeiros sistemas criados utilizavam linguagens procedurais, as quais implicavam em códigos extensos e de difícil compreensão, o que tornava trabalhoso a manutenção de qualquer aplicação desenvolvida na época. Com a

evolução das arquiteturas de *software*, houve a necessidade de criar um novo paradigma na programação, conhecido atualmente como OO (Orientação a Objetos).

O surgimento da OO implicou em uma série de benefícios tais como [26]:

- Manutenibilidade: Os programas criados utilizando o conceito de orientação a objetos possuem facilidade na leitura e compreensão, além de encapsularem a complexidade do programa, tornando visível apenas os detalhes mais relevantes;
- Modificabilidade: Facilidade na inclusão ou exclusão de operações no sistema, basta adicionar, excluir ou modificar objetos;
- Reutilização: Os Objetos se forem devidamente projetados, podem ser reutilizados em diferentes projetos;
- Confiabilidade: As aplicações orientadas a objetos, por serem amplamente definidas e testadas através da utilização de objetos, tendem a ser confiáveis.

2.4 Considerações Finais

Neste capítulo foi abordado a evolução das arquiteturas de software, desde o início da programação em que se usavam mainframes, até a atualidade onde se dispõem de arquiteturas elaboradas que visam a separação dos componentes presentes nas aplicações em camadas. Além disso, foi discutido sobre a normativa NBR ISO/IEC 9126 que é uma normativa de âmbito nacional, e contém uma lista de todos os RNFs necessários para designar se uma aplicação possui qualidade, que juntamente com catálogo do NFR Framework, pode-se elaborar um catálogo com maior abrangência de RNFs afim de aumentar o nível de qualidade de uma aplicação.

Dentre as arquiteturas existentes, percebe-se que há uma preocupação em dividir e separar partes importantes do sistema de forma a facilitar, entre outros aspectos, a manutenção, reuso, escalabilidade, desempenho. Desta forma, defende-se neste trabalho que a abordagem proposta por SOA permite a separação de cada funcionalidade em camadas, definidas como serviços pela arquitetura, permitindo uma fácil manutenção das regras de negócios e reutilização.

Detalhes de como o SOA é organizado e seu funcionamento será mencionado na próxima seção.

Capítulo 3

SOA

SOA é uma dos modelos arquiteturais que nos últimos anos, vem sendo acolhido pelos engenheiros de *software*, sendo na maioria das vezes, utilizada na concepção de produtos de *software* corporativos. Esta arquitetura permite que corporações constituídas de uma infraestrutura fragmentada sejam integradas através da utilização de serviços.

O intuito de discutir SOA neste trabalho é de desvendar os motivos por trás da SOA que conduzem engenheiros de *software* a sua adesão, expondo as vantagens e desvantagens através do relacionamento com os RNFs existentes no catálogo demonstrado no capítulo anterior, facilitando o compreendimento dos engenheiros de *software* que possuem um breve ou nenhum conhecimento dessa arquitetura.

Esta proposta tem como objetivo não ser específica somente para aplicações de pequeno, médio ou grande porte, mas permitir que com a utilização da SOA, o seu crescimento seja flexível independente do porte da aplicação desenvolvida.

Este capítulo apresenta os principais conceitos associados a SOA. A seção 3.1 descreve os conceitos básicos associados a organização estrutural da SOA. Logo em seguida na seção 3.2, será mostrado como a SOA funciona e quais os componentes que integram a mesma. Na seção 3.3, será exposto um exemplo prático de um sistema utilizando esses conceitos e na seção 3.4 são realizadas as considerações finais do capítulo.

3.1 Conceitos Básicos

SOA é, segundo a OASIS [12] (*Organization for the Advancement of Structured Information Standards*) um consórcio que definiu um modelo de referência para SOA, "um

paradigma para organização e utilização de competências distribuídas que estão sob controle de diferentes domínios proprietários".

Segundo Dias [80], "SOA é uma forma de se projetar uma arquitetura baseada na composição de serviços interoperáveis e reutilizáveis". SOA permite que corporações que possuem infraestrutura de aplicações fragmentadas, sendo administradas por diversos departamentos, possam ser integradas em nível de serviço.

Para W3C (*World Wide Web Consortium*) [13], a orientação a serviços modulariza os recursos de TI, criando os processos de negócios interligados e que integram informações entre sistemas. Com isso, os serviços constituem pequenas partes de um software, por serem de baixo acoplamento de unidades e funcionalidades, desta forma, os serviços são pequenas porções de software, construídas de tal forma que possam ser vinculadas a outros componentes de software.

Um serviço, no contexto de SOA, é um mecanismo que permite ter acesso a um conjunto de regras de negócio, através de uma interface que descreve as restrições e políticas na qual é especificada pelo serviço [12]. Os serviços são utilizados para desenvolver aplicações que suportam processos de negócios, permitindo o alinhamento da TI com as necessidades da corporação [80].

Segundo a OASIS [12], SOA é escalável por que ela faz a menor suposição possível sobre a rede e também minimiza qualquer suposição de confiança as quais são frequentemente feitas em sistemas de escala menor. Além disso, SOA reduz os custos e agiliza a produção por promover o reuso, crescimento e interoperabilidade das aplicações.

Como a SOA não necessita que o usuário tenha conhecimento de como foi implementado o serviço, é preciso ter uma descrição suficientemente clara de como o serviço funciona para que o usuário consuma o serviço, podendo habilitá-lo e saber sobre o que ele dispõe, permitindo assim o seu reuso [12].

A interface do serviço deve ser sintaticamente especificada para que o consumidor passe as informações no formato padrão necessárias para obter acesso ao serviço, assim sendo obrigado o provedor a descrever com detalhes a forma de seu uso para que não haja nenhum problema de interpretação por parte do consumidor, evitando falha no consumo do serviço [12].

3.2 Modelo Arquitetural Proposto pelo SOA

No modelo MVC [14], temos os controladores que fazem a comunicação com a camada de visão e a camada de persistência. Dessa maneira, o controlador conterá as regras de negócio e será vinculado a uma única camada de visão. Com a SOA substituiremos o controlador pelo serviço, desta forma poderá reutilizar o serviço para mais de uma camada de visão, além de poder integrar com outros softwares que são de plataformas diferentes, tornando possível a integração com sistemas heterogêneos.

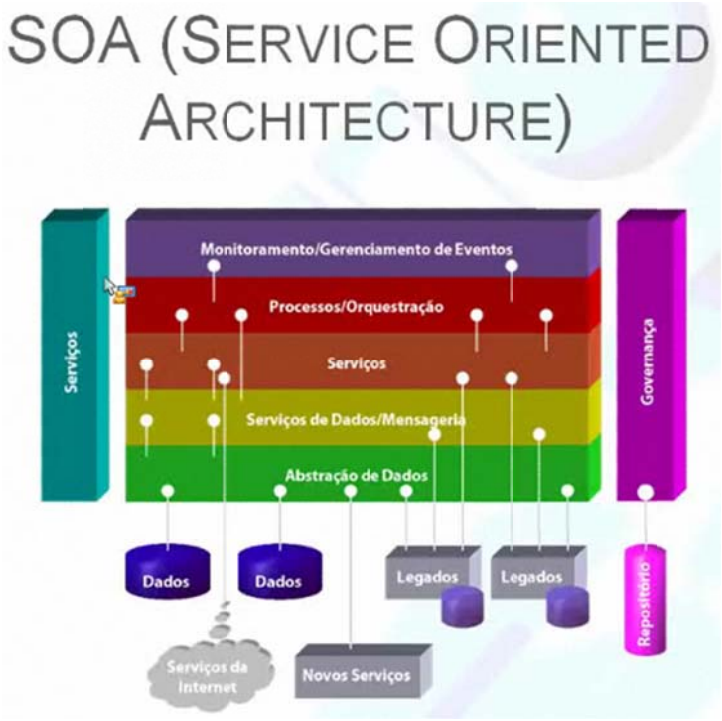


Figura 3.1: Esquema de SOA (extraído de [15]).

A SOA não necessita que o consumidor tenha conhecimento de como foi implementado o serviço, desta forma é preciso ter uma descrição suficiente de como o serviço funciona para que o consumidor possa habilitá-lo e saber sobre o que ele dispõe, permitindo assim o seu reuso [12].

Para que seja possível disponibilizar os serviços na internet em que a SOA se propõem em organizar, a existência de tecnologias que permitam a sua implementação se faz necessária,

tais como: *Web Service*, CORBA e Jini. Porém, atualmente, *Web Service* é uma das tecnologias que melhor atendem os objetivos de SOA.

Segundo Reckziegel [16], *Web Service* possui as seguintes vantagens:

- Interface abstrata: os *Web Services* fornecem uma interface abstrata para acesso aos métodos disponibilizados, ocultando detalhes de implementação do usuário do serviço;
- Semântica acompanha os dados: Ao invés de trafegarem somente os dados, a comunicação entre o servidor e o cliente carrega consigo metadados;
- Portabilidade: Por se tratar de um padrão aberto, baseado em XML, garante-se a portabilidade das mensagens mesmo sob diferentes plataformas de operação;
- Segurança: Opcionalmente, as informações trafegadas podem ser criptografadas;
- Utilização de recursos: Os *Web Services* são sistemas não invasivos, pois não consomem recursos de comunicação enquanto em estado de espera.

Existem duas maneiras de se visualizar a arquitetura *Web Service*. A primeira é examinar os papéis de cada ator em um *Web Service*; e a segunda maneira é verificar a pilha de protocolos.

Web Service é uma tecnologia baseada em SOA e por esta razão possui os mesmos papéis, tais como: Fornecedor do serviço, consumidor do serviço e registro de serviço. O Fornecedor desenvolve o serviço e o publica no registro de serviço para que o consumidor possa ser capaz de localizar o serviço no registro e consumi-lo diretamente no fornecedor.

Existem 4 camadas distintas que constituem a pilha de protocolo do *Web Service*:

- Transporte: é responsável por transportar as mensagens entre as aplicações, utilizando um dos protocolos de transporte tais como HTTP, SMTP, FTP, entre outros;
- Mensagem XML: é responsável pelo encapsulamento das mensagens de forma que as partes se entendam, nesta camada utiliza-se o SOAP;
- Descrição do Serviço: é responsável por descrever as interfaces publicas para o *Web Service*, o WSDL é usado para esta finalidade;
- Descobrimento do serviço: é responsável por utilizar um sistema de registro comum para a centralização dos serviços, a fim de fornecer funcionalidades de publicação e busca, para isso é utilizado o UDDI.

O SOAP é um protocolo de comunicação baseado em XML que permite a troca de informações, utilizando-se de um ambiente computacional distribuído, permitindo a resolução

de problemas de interoperabilidade de linguagem e plataforma, através do empacotamento de mensagens codificadas em XML.

O WSDL (*Web Services Desiption Language*) é utilizada para descrever um *Web Service*. Baseada em XML, define interfaces e tipos de dados para os serviços, representando um contrato entre o fornecedor e o consumidor do serviço.

O UDDI (*Universal Descripton, Discovery and Integration*) é uma forma padronizada de publicar e descobrir informações sobre *Web Services*. Possuem informações dos fornecedores e serviços disponíveis, podendo ser categorizados.

Na figura a seguir mostra os dois pontos de vista de SOA para uma melhor visualização da forma de como se relacionam.

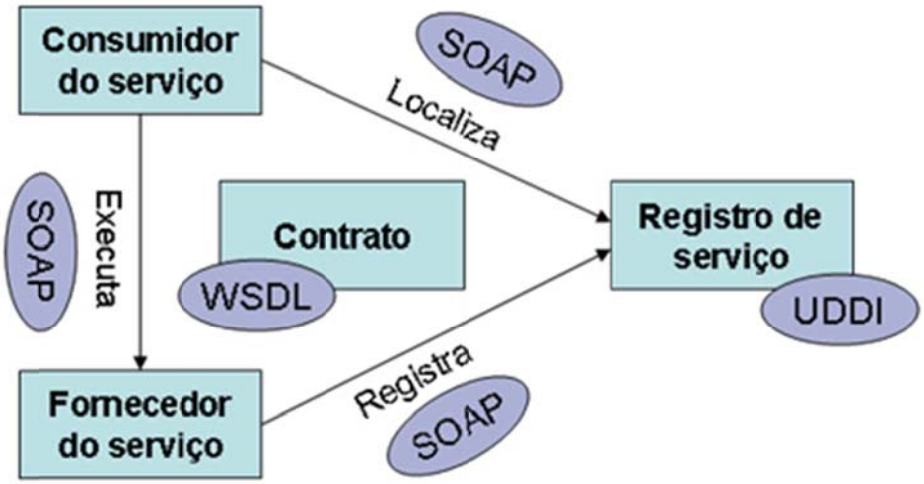


Figura 3.2: Papéis e protocolos em *Web Service* [39].

3.3 Exemplo

Nesta seção será apresentado um estudo de caso realizado por Roberto & Daniel [19], utilizando SOA, demonstrando a funcionalidade do *Web Services* . A aplicação tem como objetivo exibir uma lista de noticias de um *feed* RSS do site do Terra (www.terra.com.br) de acordo com uma lista de preferências cadastradas no sistema. A figura a seguir mostra a organização da aplicação.

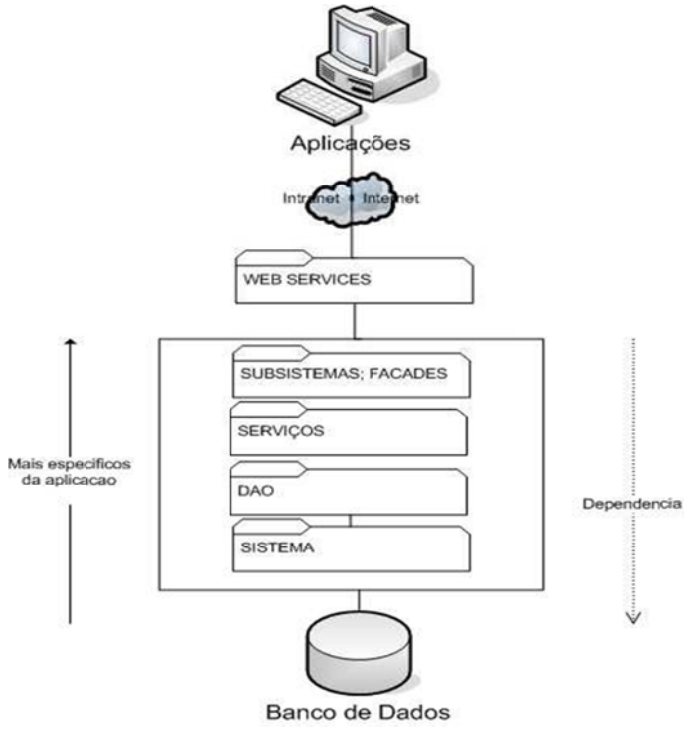


Figura 3.3: Arquitetura da Aplicação (Extraído de [19]).

Foram usados neste estudo de caso o SGBD *SqlServer* 2005 e o servidor Web IIS, ambos da Microsoft, para disponibilizar os serviços. Para o desenvolvimento da aplicação foi utilizado Turbo *Delphi for DotNET* da Borland.

Os serviços implementados possuem a capacidade de gerenciar (listar, cadastrar, atualizar, remover) usuários, gerenciar preferências de notícias e listar as notícias de um determinado usuário.

Para consumir esses serviços, foram criadas duas interfaces, uma Web que foi testada utilizando o navegador *Internet Explorer* 7 e uma *Desktop* testada no sistema operacional *Windows* XP, ambos da Microsoft.

Em relação à organização interna dos serviços, foram usadas 4 camadas: A primeira camada é a fachada que é responsável por gerenciar todas as solicitações recebidas e chamar a operação correta para ser executada, será com esta camada que o *Web Service* irá se comunicar. A segunda camada é a de serviços que contém as regras de negócios necessárias para fazer as operações necessárias. A terceira camada é a de sistemas que tem como função gerenciar as persistências de dados independentes do SGBD e a quarta e última camada que é a de acesso a dados que utiliza o padrão de projeto DAO, responsável pela manipulação da estrutura física de dados.

3.4 Considerações Finais

A utilização de SOA propicia alguns benefícios devido às suas características. A seguir serão expostos alguns dos benefícios que essa plataforma arquitetural pode oferecer [38] [39]:

- Interoperabilidade: É a capacidade de sistemas de *software* de poderem se comunicar independente da linguagem de programação, como já mencionado anteriormente. Com a utilização da tecnologia *Web Service* que dispõe do SOAP, pode-se desenvolver um sistema, que caso necessite de uma camada interoperável, precisa somente implementar a especificação SOAP;
- Desempenho: É uma barreira que a SOA tenta superar, tal barreira é imposta por algumas razões. Primeiro, SOA envolve computação distribuída, desta forma, os serviços estão em diferentes máquinas. Além disso, os serviços estão em sua grande maioria na Internet, onde não existe garantia determinística de latência. Segundo, a existência de níveis intermediários dentro de uma arquitetura. Um exemplo é o diretório de serviço em que o consumidor necessita encontrar os serviços para que possa utilizá-lo, levando ao aumento do tempo total de execução de uma transação. Terceiro, a utilização de XML para a padronização de mensagens, aumenta o tempo de processamento por exigir a execução de três operações: *parsing*, validação e transformação. Estas operações produzem um *overhead* no desempenho. Para reverter isso algumas técnicas podem auxiliar nisso, tais como: aumentar a granularidade dos serviços para aumentar o desempenho, pois mensagens com granularidade fina resultam no aumento de trafego na rede, tornando o tratamento de erros mais difícil. E a replicação de serviços que permite uma divisão do processamento em vários servidores, balanceando a carga;
- Segurança: Em aplicações SOA a segurança não é garantida usando somente um protocolo seguro como o HTTPS, pois este garante somente um nível de segurança somente na camada de transporte, pondo em risco as informações que chegam ao destinatário ou um intermediário, exigindo assim uma proteção no destinatário, em vista que os dados podem ser processados por intermediários. Outro problema é a impossibilidade de selecionar quais informações que poderiam ser criptografadas e quem poderá vê-las. Foi através destas aberturas que empresas como IBM,

Microsoft e Verisign, propuseram um modelo de segurança para *Web Service*, surgindo o WS-Security (*Web Services Security*), que inclui alguns padrões de segurança disponibilizados pela W3C, garantindo a integridade, autenticação e criptografia dos dados;

- Confiabilidade: Um serviço para ser considerado confiável, é necessário que haja uma forma de que o consumidor do serviço saiba se o fornecedor executou a solicitação ou não. Desta forma, a confiabilidade de um *Web Service* esta relacionada o protocolo de transporte, pois é neste protocolo que ocorre a troca de mensagens entre fornecedor e consumidor. O protocolo HTTP é o protocolo padrão utilizado pela *Internet* que é baseado no mecanismo de "melhor esforço", ou seja, não há garantia de que a requisição será executada. Para contornar esse problema é que existem protocolos como REST e HTTPR, que são protocolos confiáveis que podem ser utilizados pelas aplicações SOA;
- Disponibilidade: Para um *Web Service*, a disponibilidade pode garantir um sucesso ou fracasso de uma aplicação SOA, pois a indisponibilidade de um serviço pode ser um prejuízo não somente para o consumidor que necessita executar operações de seu interesse, mas para o fornecedor que poderá ter sua reputação ferida. Geralmente, fornecedores concordam em fornecer aos consumidos um conjunto de serviços com um SLA (*Service-Level-Agreement*), que estipula as obrigações das partes envolvidas, além de especificar as medidas a serem consideradas em caso de falha ou desvio, tal como o tempo de resposta e disponibilidade. Existe uma especificação chamada WSLA, redigida pela IBM, que contem três seções: uma seção descrevendo as partes, uma seção contendo um ou mais definições de serviço e uma seção definindo as obrigações das partes. Outra forma de garantir a disponibilidade é a implementação de um mecanismo de replicação, afim de que caso um servidor fique indisponível, outro assuma o seu lugar.
- Escalabilidade: O aumento do número de consumidores pode ser um grande problema para *Web Service* pelo fato de que o UDDI ser centralizado. A centralização pode provocar uma sobrecarga no servidor, impossibilitando o processamento de todas as requisições. Por este motivo, pesquisadores investigam maneiras de utilizar registros distribuídos, através da utilização de *Grid computing*

e P2P. Estas tecnologias permitem a descentralização do sistema de registros, permitindo a escalabilidade.

- Manutenibilidade: O grau de acoplamento afeta diretamente a manutenibilidade de um sistema. Em sistemas SOA o ideal é que os serviços sejam projetados a terem fraco acoplamento e dependerem somente das interfaces de outros serviços e não de implementações. Desta forma, um bom planejamento pode garantir que uma mudança em um serviço não afete o consumidor;
- Flexibilidade: A necessidade de ser apto a suportar novos requisitos, sendo eles por motivos de mudança nas leis, reorganização de negócio ou alguma necessidade do mercado. A SOA possui característica que permitem o acrescimento de novos serviços, bem como o reuso dos serviços a fim de responder as mudanças.

Com base nesses aspectos, a SOA permite que sejam reaproveitadas as regras de negócio definidas nos serviços. A forma como o sistema é arquitetado continua mantendo estrutura do MVC, mas sem a necessidade de ter que fazer adaptações no código para poder atender as necessidades exigidas dos novos projetos. O Flex exige a necessidade de se implementar um controlador que será usado para permitir o acesso aos dados que irão alimentar a interface, fugindo um pouco da arquitetura MVC, que é organizado em 3 camadas sendo uma delas o controlador que é encarregado de fazer esse tipo de trabalho. Assim a SOA permite a criação de camadas de acordo com a necessidade do projeto, possibilitando acomodar o Flex por causa desta divisão, podendo ser criada uma camada específica para permitir o funcionamento do Flex no sistema, e outras vantagens já mencionadas anteriormente.

Capítulo 4

Frameworks RIAS

Entre os anos de 1990 e 2000 houve um crescimento de usuários acessando a internet e consigo o aumento do numero de aplicações para a *Web*. Nesta época o HTML (*Hyper Text Markup Language*) era a linguagem utilizada para a confecção de *softwares* para a *Web*, por ser uma linguagem de marcação de fácil aprendizagem, utilização e baixo custo de desenvolvimento, alavancando o crescimento das aplicações para a internet [33] [34].

Essas aplicações *Web* tradicionais possuem seu código centralizado e seu processamento realizado no servidor, disponibilizando para o usuário uma tela estática, desta forma, a aplicação precisa enviar dados para o servidor (através de *clicks* em *hyperlinks* ou submissões de formulários), para que mande uma resposta e faça com que a tela de visualização do usuário seja recarregada.

A RIA permite maior interação com o usuário, carregando toda a interface para o computador do usuário na primeira vez que é acessada, possibilitando a transmissão de dados que são vitais para o funcionamento da aplicação *Web*, separando a interface da sua parte lógica e não viabilizando a necessidade de recarregar a tela de exibição, tornando as resposta das operações efetuadas mais rápidas.

Nas subseções seguintes serão explanados os principais *frameworks* do mercado que possibilitam a criação de tais aplicações para a *Web*. Na seção 4.1 discute-se a respeito das vantagens e desvantagens do *framework* da Microsoft Silverlight e na seção 4.2, apresenta-se o *framework* Oracle JavaFX Na seção 4.3 apresenta-se o *framework* Adobe Flex/Air e na seção 4.4 a tecnologia HTML 5.

Framework é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma determinada categoria de *software* [36].

4.1 Microsoft Silverlight

O Silverlight é um *framework* desenvolvido pela Microsoft em 2007, com o intuito de desenvolver aplicações ricas para a *Internet.*

O Silverlight é constituído do .NET Framework para Silverlight, que contém os componentes e bibliotecas utilizados para integração de dados, classes bases, funcionalidades de rede e a CLR (*Common Language Runtime*), esta permite o gerenciamento de memória, coletor de lixo (*garbage collector*), checagem de tipos e manuseamento de exceções e também possui o Núcleo de Apresentação que é constituído de componentes e serviços focado na interface de usuário.

O XAML (*eXtensible Application Markup Language*) é a linguagem de codificação de interfaces do Silverlight que permite a declaração de componentes visuais através do mesmo mecanismo de *tags* como no XML, sem a necessidade de controle de fluxo. Quando compilado código, o compilador se encarrega de fazer a instanciação dos componentes declarados, gerando arquivo XAP que são executados pelo plug-in Silverlight.

```
<UserControl x:Class="HelloWorld3.Page"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignHeight="50" d:DesignWidth="400">

        <Canvas x:Name="LayoutRoot">
                <TextBlock Canvas.Left="8" Canvas.Top="8"Text="Hello World!"/>
        </Canvas>
</UserControl>
```

Figura 4.1: *Hello world* em Microsoft Silverlight.

Para o desenvolvimento de apicações Silverlight existe o Microsoft Expression Blend que permite a criação de interfaces sem a necessidade de codificar em XAML, além de ser parte integrante da principal IDE (Integrated Development Environment) de desenvolvimento da Microsoft, o Microsoft Visual Studio.

O Silverlight se integra facilmente com as linguagens da plataforma .Net, mas pode se integrar com outras linguagens de programação *Web* através da utilização de um plug-in

chamado WebORB distribuída de forma gratuita pela Midnight Coders, este plug-in implementa um protocolo de comunicação conhecido por AMF (*Action Message Format*), que faz a serialização dos dados em formato binário. Desta forma, não existe a transferência de dados redundantes como ocorre na utilização de XML através de consume de *Web Services*. A seguir, na figura 4.1, mostra um gráfico de desempenho utilizando o AMF, *Web Service* e HTTP *Service* que esta disponível em [37] juntamente como código-fonte.

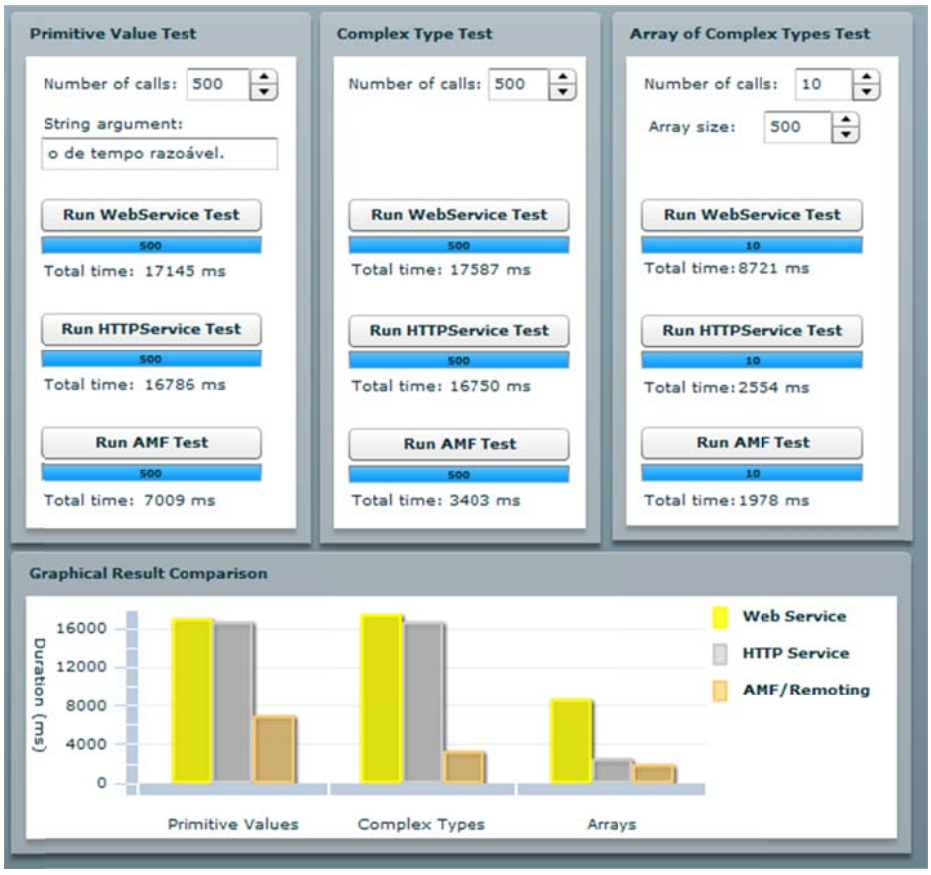


Figura 4.2: Avaliação na transferência de dados possíveis no Microsoft Silverlight.

4.2 Oracle JavaFX

O JavaFX é uma tecnologia desenvolvida pela Sun Microsystems por Chris Oliver com o nome F3 (*Form Follows Function*), sendo rebatizado para o nome atualmente conhecido em 2007, com o propósito de criar aplicações ricas para a *Internet*, sendo adquirida posteriormente pela Oracle [35].

A plataforma JavaFX é composta de três componentes: JavaFX SDK que contém o compilador e ferramentas de tempo de execução, gráficos, mídias e *Web Services*, além de ferramentas que possibilita a criação de RIAs para *desktops*, navegadores, dispositivos móveis, aparelhos de TV, entre outros; Um plug-in para desenvolvimento, disponível para o Eclipse através de desenvolvimento de terceiros e integrado no NetBeans 6.9; e o JavaFX Prodution Suite constituído de plug-ins que auxiliam na importação de interfaces gráficas de aplicativos como Adobe Illustrator e Photoshop, contém o JavaFX Graphics Viewer que permite a visualização da interface antes da realização do *deploy*; e uma ferramenta de conversão de arquivos gráficos de formato SVG para visualização no JavaFX.

As aplicações com interfaces criadas em JavaFX rodam diretamente em cima da JVM (*Java Virtual Machine*), tornando-o multi-plataforma assim como o Java, desta forma a JVM lêem os arquivos .class, arquivos que contém as instruções para a máquina virtual, repassando as instruções para poderem ser executadas no JavaFx Runtime. Esta ferramenta é parte integrada do JRE (*Java Runtime Environment*) e, portanto desnecessária a instalação de algum plug-in adicional para seu funcionamento.

Utilizando-se de uma linguagem declarativa de script, compilada e estaticamente tipada (necessita a de declaração explícita das variáveis antes de serem utilizadas), o JavaFX tenta propiciar uma forma fácil de se criar telas, levando programadores a pensarem como designers, visualizando a composição da tela declarando os objetos na cena (*Scene*). No quadro a seguir mostra um exemplo de código para criação de tela com o JavaFX:

```
import javafx.scene.Scene;
import javafx.scene.text.*;
import javafx.stage.Stage;

Stage {
        title: "Hello World JavaFX"
        scene: Scene {
                content: Text {
                        content: "Hello World!"
                        font: Font { size: 30 }
                        layoutX: 114
                        layoutY: 45
                }
        }
        width:400 height:100
}
```

Figura 4.3: *Hello World* em Oracle JavaFX.

É possível a customização visual dos componentes existentes no JavaFX através da utilização de CSS (*Cascading Style Sheets*). O CSS é uma folha de estilo composta de um conjunto de configurações visuais que é padronizada pela W3C, este padrão é utilizado pelas páginas *Web* tradicionais, implementada em todos navegadores, sendo integrada no JavaFX para permitir que desenvolvedores que tenham conhecimento em CSS possam alterar as partes visuais de seus componentes como tamanhos das fontes, cores das letras, cores das bordas, entre outros.

Através de componentes comuns (*Common Elements*) é possível integrar bibliotecas desenvolvidas por terceiros com a linguagem Java, além de permite o acesso as APIs que esta linguagem possui. A integração de outras linguagem como PHP, ASP.NET, Rails, entre outras, se dá através da utilização de *Web Services.*

4.3 Adobe Flex

O Flex é um *framework* que surgiu com o nome de Macromedia Flex Server, após a compra dos direitos da Macromedia pela Adobe Systems em 2005, foi rebatizado como Adobe Flex.

O Adobe Flex dispõe de um SDK disponível gratuitamente, que contém todas as bibliotecas, documentação, ferramentas e um compilador para que os desenvolvedores possam criar suas aplicações.

Para que as aplicações desenvolvidas com o *framework* Adobe Flex possam rodar corretamente, é utilizado o Adobe Flash Player para rodar as aplicações no navegador e o Adobe Air Runtime para executar as aplicações em dispositivos móveis e *desktops*, permitindo que os arquivos swf, gerados após a compilação, rodem em qualquer sistema operacional, navegador e dispositivo que os contenham.

O MXML (sem definição por parte da Adobe sobre o significado da origem da extensão) é a linguagem de codificação de interfaces do Flex que permite a declaração de componentes visuais através do mesmo mecanismo de tags como no XML, sem a necessidade de controle de fluxo. Quando compilado código, o compilador se encarrega de fazer a instanciação dos componentes declarados.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                        xmlns:s="library://ns.adobe.com/flex/spark"
                        xmlns:mx="library://ns.adobe.com/flex/mx">
        <fx:Declarations>
                <!-- Place non-visual elements (e.g., services, value objects) here -->
        </fx:Declarations>
        <fx:Script>
                <![CDATA[
                ]]>
        </fx:Script>
        <s:Label text="Hello Wolrd!"/>
</s:Application>
```

Figura 4.4: *Hello World* em Adobe Flex.

O ActionScript 3 é uma linguagem de programação OO (Orientada a Objetos), que declaradas dentro da tag *fx:Script*, possibilita criar mecanismos de controle dos dados que são exibidas pelas interface, além de possibilitar a codificação de componentes do mesmo modo que outras linguagens de programação OO permitem.

Os componentes visuais existentes na biblioteca do Adobe Flex podem ser customizados utilizando CSS, para mudar cores, tamanhos de letras, fontes, entre outros atributos, também permite a modificação dos componentes utilizando notações MXML, desta forma permite que o desenvolvedor modifique o estilo visual dos componentes baseados nos existentes, podendo adicionais ou remover características existentes de forma mais fácil do que utilizando ActionScript.

Para o desenvolvimento de aplicações utilizando Adobe Flex é possível através da IDE Adobe Flash Builder, baseado na IDE Eclipse, ou através do *pluig-in* criado para a IDE Eclipse, sendo necessária a compra para a utilização comercial.

Com o Adobe Flex é possível integrar outras linguagens de programação *Web* através da utilização do protocolo AMF, através do componente RemoteObject que está disponível no SDK do Adobe Flex, ou através do consumo de serviços (SOAP, REST), através do componente WebService.

A imagem a seguir mostra um comparativo utilizando o Adobe Flex e as suas diversas formas de transmissão de dados:

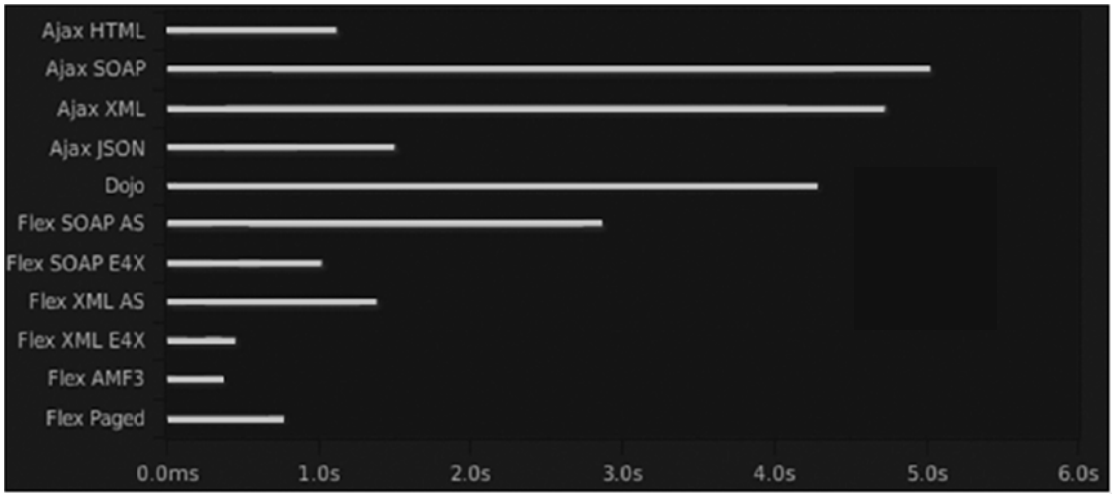


Figura 4.5: Velocidades dos Protocolos Binários VS Ajax VS XML (Extraído de [24]).

4.4 HTML5

O HTML5 é a nova atualização da principal linguagem de desenvolvimento *Web* que até o momento utiliza a versão 4.0.1, aprovada em 1999 e especificada pela W3C. A especificação desta nova versão está sendo feita por dois grupos, a W3C e a WHATWG (Web Hypertext Application Technology Working Group). Essa nova versão permitirá que os desenvolvedores *Web* tenham em mãos as mesmas funcionalidades que os outros *frameworks* RIA possuem, tendo em vista a disponibilidade da versão final para o ano de 2012 [25].

A forma de editar as interfaces com o HTML5 continua sendo através de uma linguagem de marcação, porém dispondo de novas *tags* criadas especificamente para a modelagem do *layout* de uma página *Web*, sendo elas: Header, Articule, Nav, Section, Footer e Aside. Isso se deve ao fato de que na sua versão anterior, os *Web Designers* necessitavam utilizar *tags* DIV excessivamente para conseguir o resultado esperado.

Umas das vantagens que o HTML5 tem em relação a sua versão atualmente usada, é em relação a arquivos multimídias de vídeos e áudio que terão *tags* específicas, sendo elas: <vídeo> e <audio> respectivamente. Existe a possibilidade de manipular gráficos vetoriais, utilizando a *tag* <canvas> e a biblioteca gráfica disponível para a criação e manipulação de objetos em 2D.

Outra novidade nessa nova versão é a possibilidade de executar uma aplicação no navegador *Web* sem a necessidade de estar conectado a *Internet*, permitindo efetuar quaisquer alterações nos dados do sistema, as quais são armazenadas na memória *cache* do navegador e,

automaticamente sincronizadas todas as informações no momento em que existir acesso a *Internet*.

```
<!DOCTYPE html>
<html>
	<head>
		<title>HTML5</title>
		<meta charset="utf-8">
	</head>
	<body>
		<article>
			Hello World!
		</article>
	</body>
</html>
```

Figura 4.6: *Hello World* em HTML5.

Além disso, apresenta novos componentes para criação de formulários como: DATE, COLOR, DATETIME, E-MAIL, NUMBER, RANGE, entre outros, facilitando a manipulação de determinados tipos de dados, sem a necessidade de modificar componentes e também criar mecanismos de validação de formulários através da utilização de código em JavaScript, pois essas operações poderão ser feitas através do HTML.

Atualmente, a maioria dos navegadores possui as funcionalidades do HTML5 implementadas, também as ferramentas de desenvolvimento de páginas *Web* estão sendo atualizadas para permitirem a confecção, como é o caso do Adobe Dreamweaver e Microsoft Expression.

4.5 Considerações finais

Neste capítulo foram abordados os principais *frameworks* que permitem o desenvolvimento de aplicações RIA, tais como: Adobe Flex, Oracle JavaFX e Microsoft Silverlight e a versão 5 do HTML.

Entre todos os *frameworks* citados, o Adobe Flex é o que está a mais tempo no mercado, o JavaFX e o Silverlight surgiram em 2007, mas tiveram repercussões diferentes, pelo fato do forte marketing existente por parte da Microsoft para alavancar seu *framework* a fim de

concorrer igualmente com o Adobe Flex. O HTML 5 não está na sua versão final, pois nem todos navegadores de *Internet* permitem a sua interpretação. Como foi dito anteriormente, provê as mesmas características que os outros *frameworks* possuem. A figura 4.3 contém um comparativo a respeito dos *players* que permitem a execução de alguns *frameworks* citados em relação à porcentagem de adesão por parte dos usuários *Desktop*.

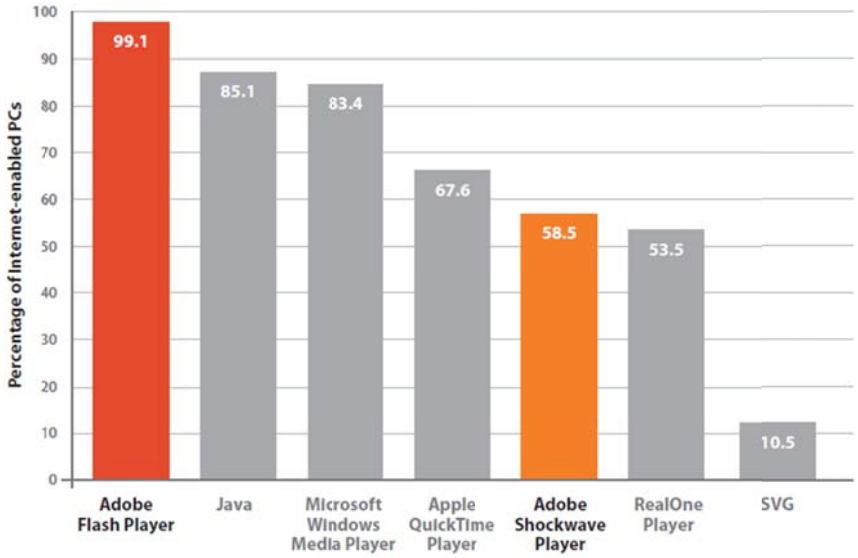


Figura 4.7: Alcance do *Flash Player* (Extraído de [22]).

Pode-se concluir que grande parte dos usuários, cerca de 99.1%, possuem em seus computadores *Desktop* o Adobe Flash Player instalado, seguido de 85.1%, que possuem o JVM, o mesmo não ocorre com o Silverlight que não chegou ao mesmo nível de adesão como os outros *frameworks*, exigindo a instalação do *plug-in* nos navegadores.

Os *frameworks* Adobe Flex e Microsoft Silverlight, possuem seus pacotes de desenvolvimento disponíveis gratuitamente, porém os seus editores oficiais são pagos, algo que não ocorre com o JavaFX que possui as IDEs Eclipse e NetBeans disponíveis de forma gratuita. A respeito do HTML5, existe a possibilidade de edição nos novos editores *Web* atuais.

Capítulo 5

Proposta

Neste trabalho propõe-se o modelo arquitetural ilustrado na figura 5.1. A ilustração demonstra detalhadamente o esboço da tecnologia e do *framework* que serão utilizados no estudo de caso, facilitando a compreensão de como essas ferramentas estão dispostas na arquitetura.

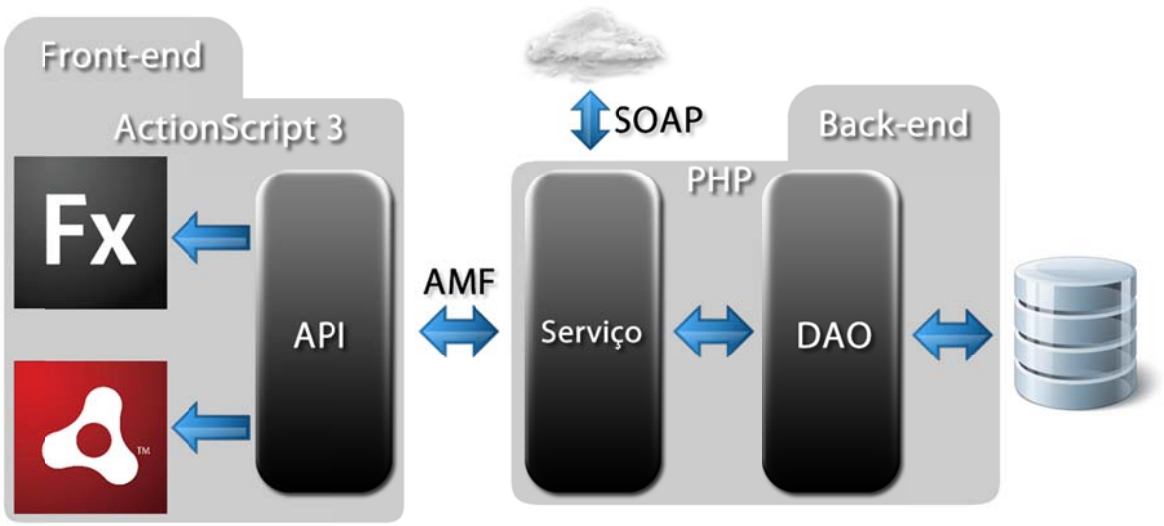


Figura 5.1: Arquitetura da Aplicação *Web* proposta.

Normalmente as aplicações *Web* são divididas em *front-end* e *back-end*. *Front-end* em uma aplicação se refere à parte em que o usuário terá contato com o *software* projetado, desta forma, todas as estruturas existentes que possibilitam a criação e gerenciamento de dados das telas de visualização ficam armazenadas nesta camada, no caso das aplicações RIA, a parte de visão será executada na máquina do usuário.

No *back-end* existem estruturas responsáveis pelo funcionamento do *software* e as mesmas podem implementar mecanismos para acesso ao banco de dados, gerenciamento de arquivos,

regras de negócios complexas, entre outras. Esta parte da aplicação fica armazenada e executa suas operações na máquina servidor. Quando se utilizam tecnologias diferentes, como é a que propomos, existe a necessidade de usar mecanismos para conciliá-las. O AMF e os *Web Services* são pontes de comunicação que fazem esse papel, pois surgiram com essa finalidade.

Esta proposta não tem finalidade de prender o engenheiro de software nas tecnologias que serão usadas neste estudo de caso, mas sim propor uma arquitetura que seja flexível para usar os *frameworks* anteriormente abordados, para que se encaixe nas exigências que o usuário possa exigir do *software*.

A figura 5.1 representa a estrutura da implementação deste trabalho, que constitui de duas camadas, em alto nível, que representam o *front-end* quadro referente ao *ActionScript 3* e *back-end* quadro referente ao PHP.

A camada de *front-end* é constituída por uma API que contém todas as implementações referentes ao acesso e manipulação de dados para sua exibição. Utiliza-se o AMF para a comunicação com o *back-end*. Esta API poderá ser usada nas interfaces criadas em Flex (visualização em navegadores) e AIR (visualização em Desktops e Mobiles), desta forma, havendo a necessidade de mudar parte da codificação para resolver algum problema existente no momento de adquirir algum dado do *back-end*, será feito somente na API e essa alteração se refletirá para as interfaces em Adobe Flex e AIR. A implementação das interfaces separadamente permite que seja desenvolvida para garantir um melhor aproveitamento da tela a serem visualizadas, como no caso dos *Mobiles*, que possuem uma tela menor do que de um computador, exigindo um ajuste dos componentes de visualização para melhor visualização.

A camada de *back-end* contida nesta proposta possui uma camada interna denominada DAO (*Data Access Object*), que é responsável pela persistência de dados, utilizada para centralizar todo em qualquer tipo de acesso ao banco de dados. A camada Serviço contém a parte vital do sistema, a implementação das regras de negócio da aplicação *Web*. Em comparação com a arquitetura MVC esta camada é representada pelos controladores. Os Serviços podem ser consumidos de duas formas: utilizando o protocolo AMF que é uma forma de comunicação que utiliza a serialização de dados, sendo a forma mais rápida e eficiente de comunicação com um *front-end* em Adobe Flex, e utilizando o protocolo SOAP (*Simple Object Access Protocol*) que permite que outros sistemas, implementados em qualquer outra tecnologia, possam consumir esses serviços. Os serviços que serão disponibilizados podem ser controlados pelo desenvolvedor do sistema, podendo ser liberados

os acessos a determinados serviços diante de um contrato firmado entre as duas partes caso haja necessidade.

Através dos estudos realizados pode-se construir o seguinte diagrama SIG com as operacionalizações dos componentes que constituem a arquitetura proposta com os RNFs do catálogo criado a partir da junção do catálogo do *NFR Framework* e a normativa NBR ISO/IEC 9126. A partir deste diagrama o engenheiro de software poderá averiguar que ao desenvolver uma aplicação utilizando a arquitetura proposta, irá satisfazer os RNFs por ele representados. Abaixo o respectivo grafo SIG, ressaltado que não contem o HTML 5 por ainda não ser suportado totalmente nos navegadores atuais.

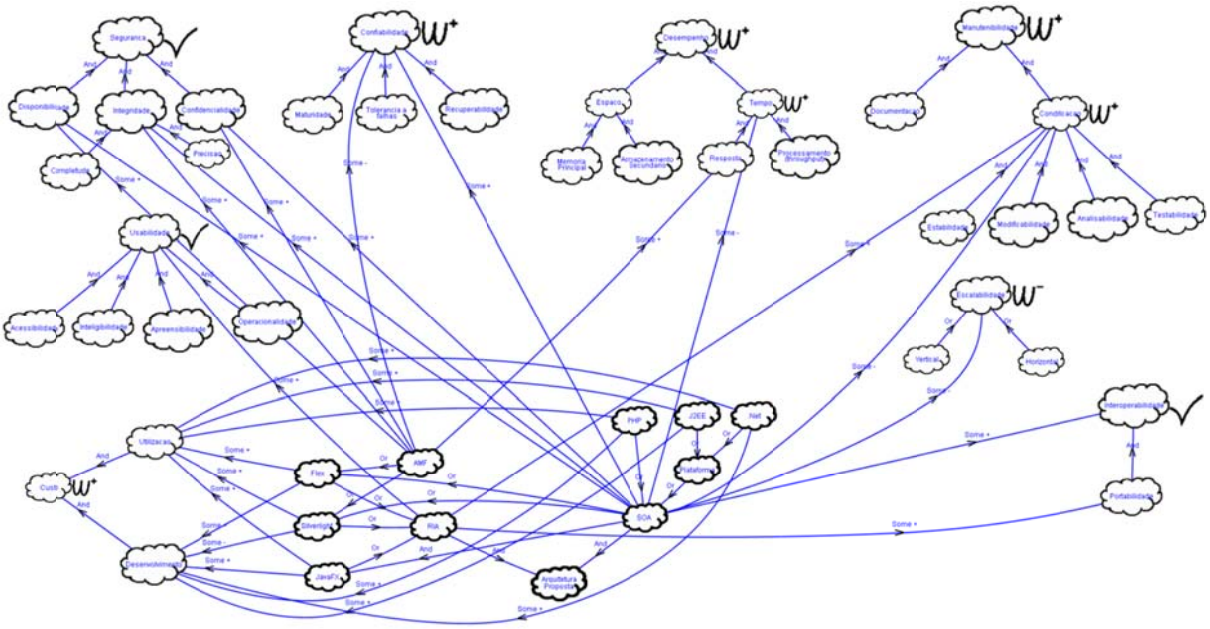


Figura 5.2: Grafo SIG com as operacionalizações entre RNFs e arquitetura proposta.

Para um melhor entendimento da figura 5.2, segue uma descrição detalhada de cada RNF e suas respectivas subdivisões. Desta forma, estas descrições permitem que o engenheiro de *software* compreenda qual o impacto que a arquitetura proposta exerce em relação aos RNFs que constituem o catálogo criado com base na normativa NBR ISO/IEC 9126 e o catálogo do *NFR Framework*. A seguir serão apresentados os RNFs do catálogo elaborado com as respectivas descrições das influências existentes com os elementos da arquitetura proposta.

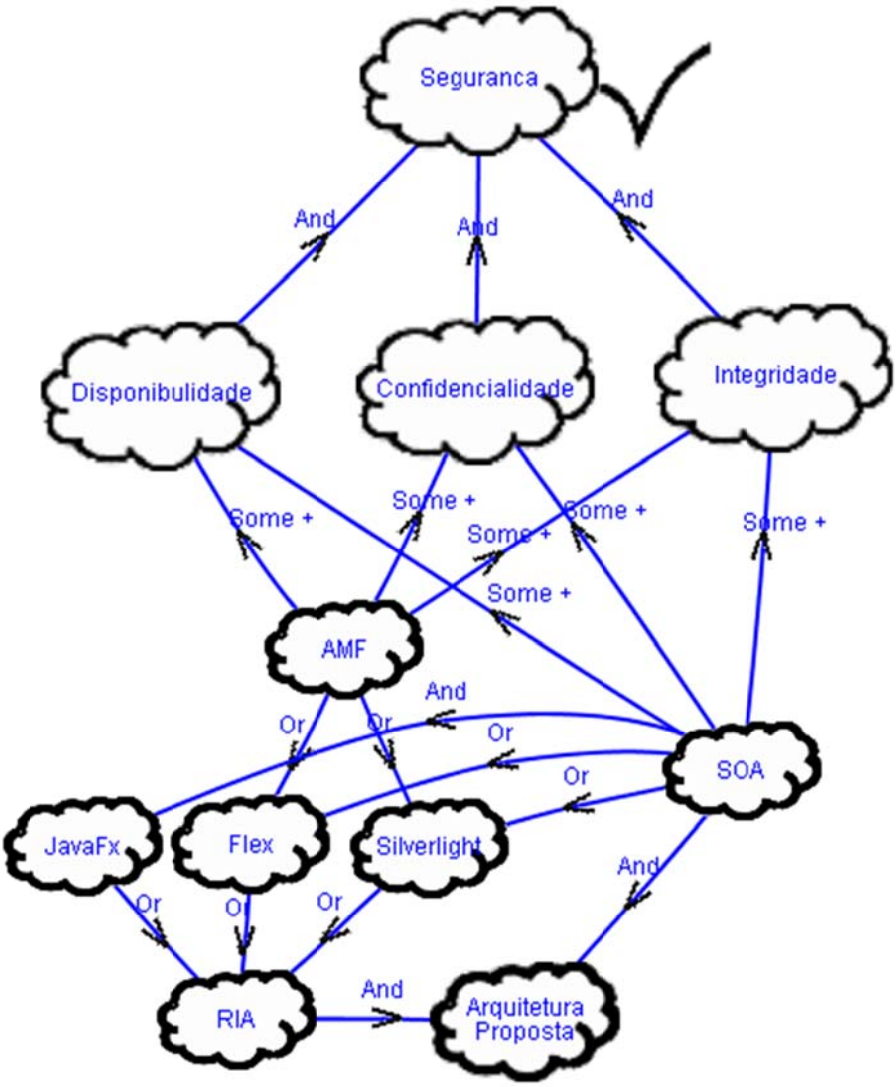


Figura 5.3: Grafo SIG do RNF segurança e influências da arquitetura.

Segurança: A segurança é uma das preocupações dos engenheiros de *software*, pelo fato de ser crucial perder informações internas da empresa que são utilizadas pelo sistema. A SOA possui o WS-Security que é um padrão de segurança desenvolvido para garantir a integridade, autenticação e criptografia dos dados. As mensagens que são enviadas possuem cabeçalhos contendo informações de segurança necessárias para o descriptografia de dados. Através do AMF é possível utilizar o protocolo HTTPS (*HyperText Transfer Protocol Secure*) podendo ser utilizado juntamente com outros métodos de criptografia como AES (*Advanced Encryption Standard*) e RSA para melhorar a segurança. Desta forma, o JavaFX utiliza somente o protocolo SOAP que o SOA dispõe, representado pela seta AND, o Flex e o

Silverlight podem utilizar tanto o AMF quanto o SOAP, sendo representado pelas setas OR. Assim, a influência destes dois protocolos são positivas e são aceitos pelas razões já citadas.

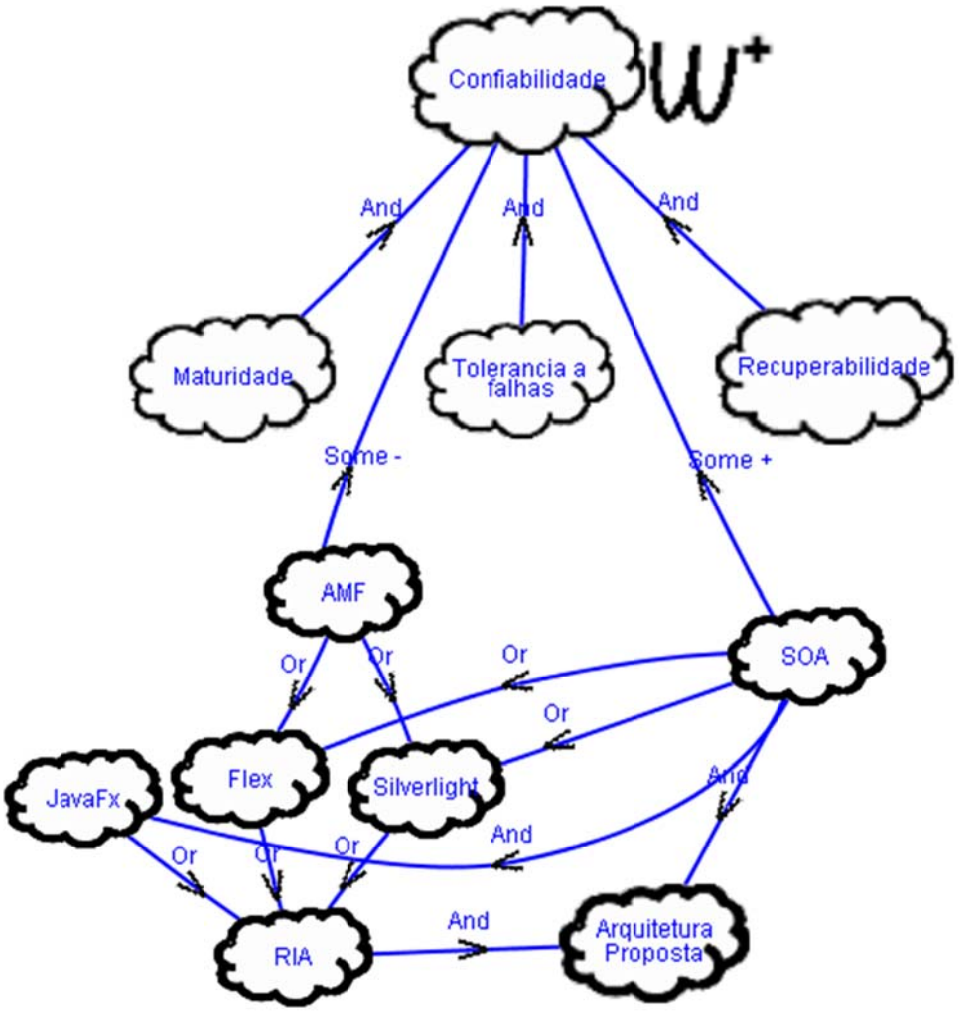


Figura 5.4: Grafo SIG do RNF confiabilidade e influência da arquitetura.

Confiabilidade: Garantir que as informações serão entregues na *Internet* é algo impossível. O tráfego na rede é por muitas vezes uma grande complicação no momento de envio de alguma informação. Foi com o intuito de contornar esse problema que protocolos como o REST surgiram para resolver tal problema. O REST é um conjunto de princípios que definem como o protocolo HTTP e URI (*Uniform Resource Identifier*) devem ser usados. Em contra partida o AMF utiliza o protocolo HTTP para troca de mensagens, com isso não se pode garantir que as mensagens enviadas serão entregues. Com relação ao protocolo AMF que pode ser usado pelo Flex e Silverlight, exerce uma influência negativa sobre a

confiabilidade, mas a SOA por permitir a utilização do protocolo REST para implementar os Web Services exerce uma influência positiva permitindo ser parcialmente satisfeito este RNF.

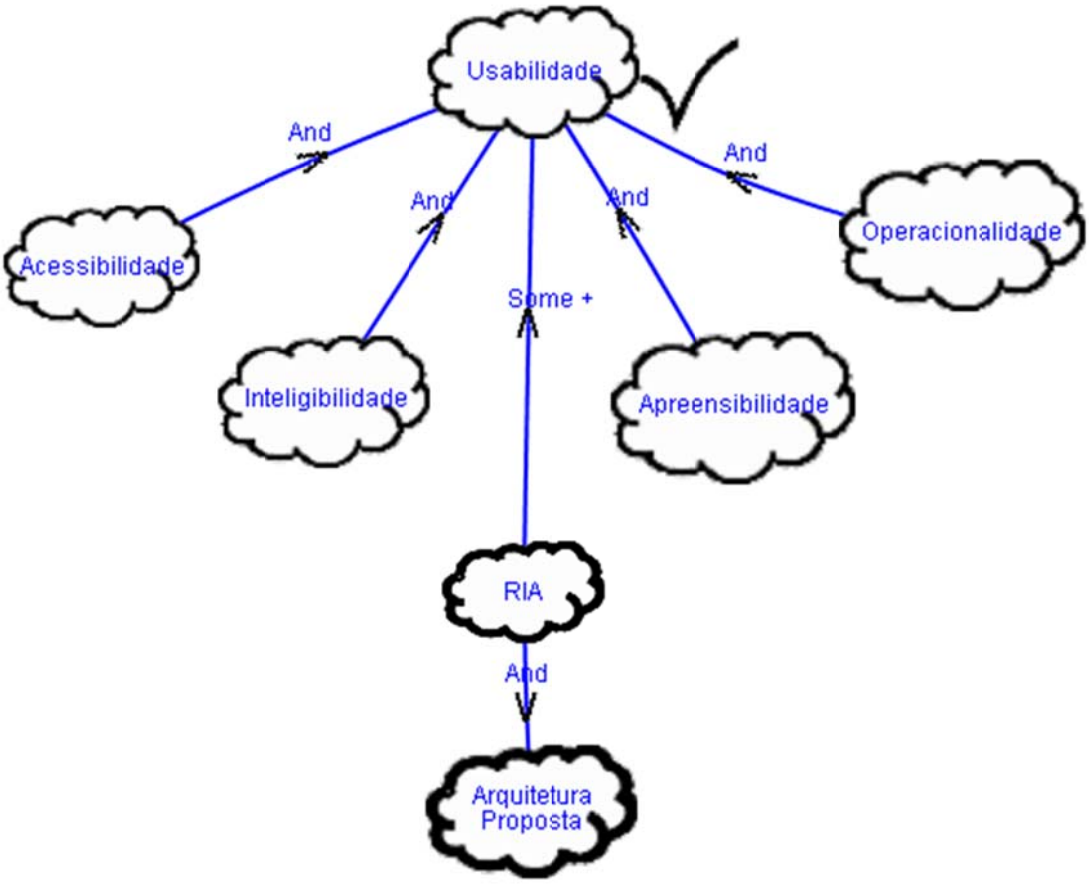


Figura 5.5: Grafo SIG do RNF usabilidade e influência da arquitetura.

Usabilidade: É a capacidade da aplicação de possuir características que permitam facilidade na realização das operações. Nas aplicações *Web*, as formas em que são realizadas as operações são diferentes das existentes em ambiente *Desktop*, mas com o surgimento de *frameworks* RIA, pode-se importar para a *Web* todas as formas de interação que existiam nos ambientes *Desktops* para a *Web*, além de possuírem ferramentas que permitem a criação de mecanismos para garantir a acessibilidade e organizar a interface de forma que o usuário possa ter facilidade em aprender a usar o sistema, bem como poder realizar operações com maior eficiência.

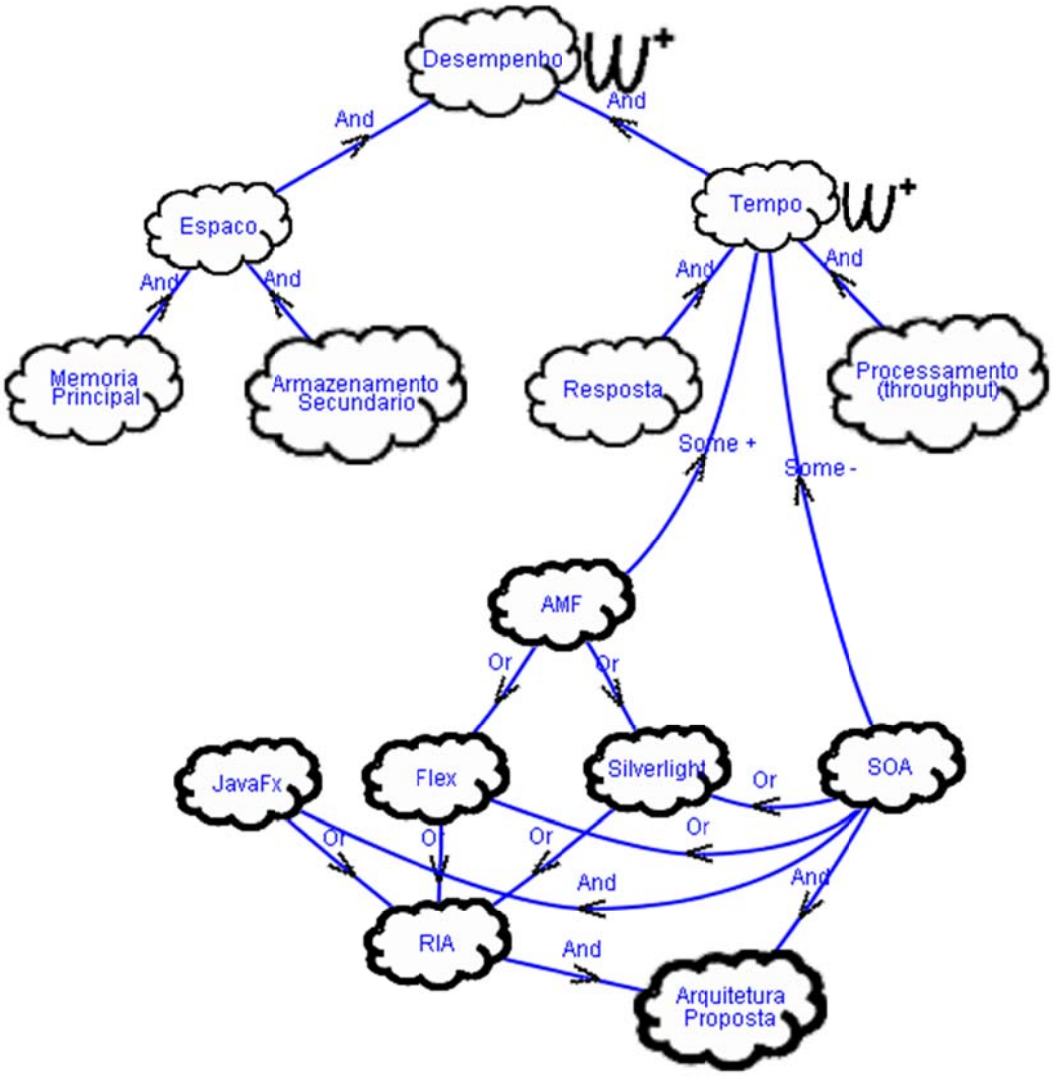


Figura 5.6: Grafo SIG do RNF desempenho e influências da arquitetura.

Desempenho: A velocidade em que as operações são realizadas em uma aplicação *Web* é medida a partir do momento em que um dado é enviado até o momento em que a resposta referente é recebida. A SOA, que utiliza mensagens no padrão XML para prover a comunicação, tem uma perda significativa no desempenho por necessitar de *tags* para identificação do conteúdo, aumentando a quantidade de dados a serem transmitidos e a necessidade de processamento extra para interpretar a mensagem XML. Isto deve ser observado na relação entre a operacionalização SOA e RNF Tempo. O AMF tem um ganho em desempenho por disfrutar da serialização de dados, enviando somente dados de interesse, mas o que pode interferir é a latência da *Internet* que pode variar de forma imprevisível, sendo assim, viável a utilização do SOAP quando se necessita transmitir uma grande quantidade de

informações, para compensar os dados extras que são enviados nas mensagens. Com isso, na figura 5.5 ilustra a ligação AND da SOA com o JavaFx por permitir somente essa forma de comunicação com outras linguagens e ligações OR entre AMF e SOA com relação aos *frameworks* Flex e Silverlight. O AMF possui, conforme comentado anteriormente influência positiva ao contrário da SOA que possui influência negativa, tornando este RNF parcialmente satisfeito.

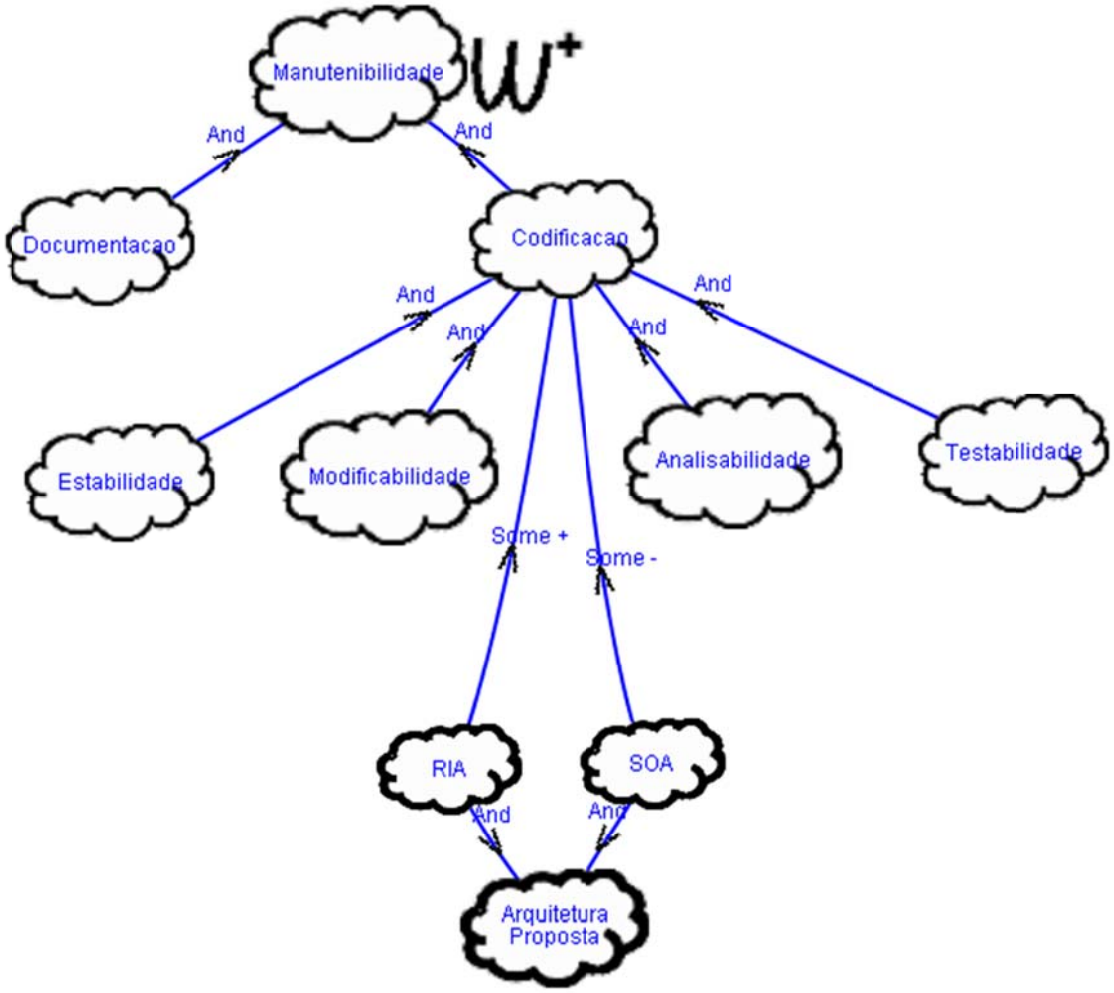


Figura 5.7: Grafo SIG do RNF manutenibilidade e influências da arquitetura.

Manutenibilidade: A Manutenção faz parte do ciclo de vida de qualquer aplicação. Para SOA a manutenibilidade pode ser um fator negativo dependendo do grau de acoplamento dos serviços, quanto menor for o acoplamento, menor será o esforço para prover a manutenção. Com a utilização da arquitetura proposta, o *front-end* da aplicação, na qual os *frameworks* RIA atuam, provê uma facilidade na manutenção por existir uma separação entre os

controladores que gerenciam as ações que as telas de apresentação sofrem e as transações de dados com o *back-end*, e os componentes de visualização da tela, desta forma estes elementos não ficarão situados na mesma camada, que dependendo da complexidade da interface pode aumentar a dificuldade de compreensão do código. Deste modo, dependendo da forma da composição dos serviços, a SOA pode ser uma influência negativa, mas caso os serviços não tenham dependência de outro serviço disponível na *Internet*, esta influência passa a ser positiva. Os *frameworks* RIA, de acordo com a estruturação da arquitetura proposta, exercem uma influência positiva em relação ao RNF codificação.

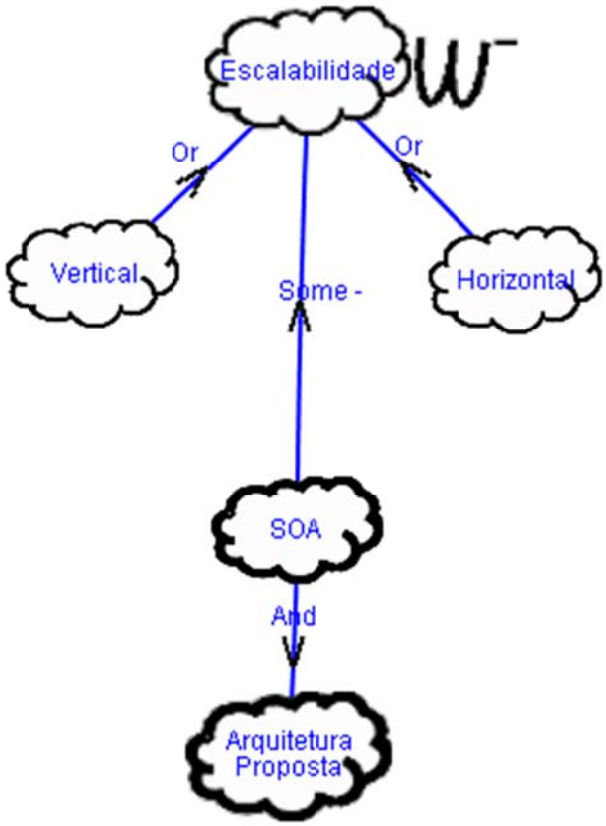


Figura 5.8: Grafo SIG do RNF escalabilidade e influência da arquitetura.

Escalabilidade: A escalabilidade é a capacidade de um servidor expandir seu poder computacional. Para a SOA isso acaba sendo algo complicado, pelo fato da UDDI ser centralizada impossibilitando a escalabilidade horizontal. É possível efetuar a escalabilidade vertical, mas dependendo da tecnologia dos componentes de *hardware* utilizados acaba se tornando uma alternativa cara. Em relação a este aspecto, a SOA exerce uma influência negativa, como ilustrado na figura 5.7, tornando este RNF parcialmente insatisfeito.

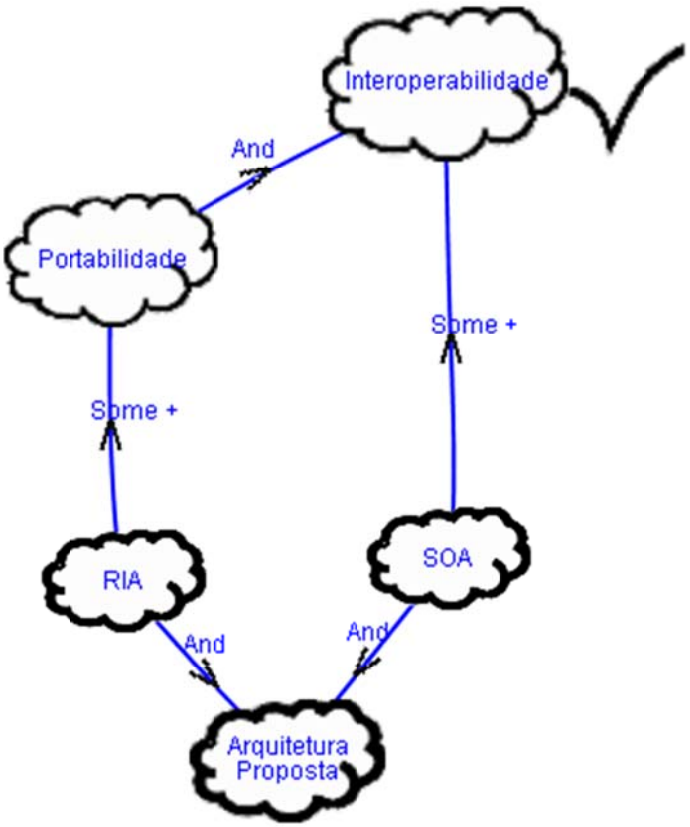


Figura 5.9: Grafo SIG do RNF interoperabilidade e influências da arquitetura.

Interoperabilidade: É a capacidade de varias aplicações implementadas em diferentes linguagens poderem trocar mensagens entre si. Essa barreira é quebrada através das mensagens XML utilizada pela SOA, as mensagens são blocos de textos, formato utilizados pelas páginas padrões da *Internet*, podendo ser interpretados por qualquer linguagem de programação. Os *frameworks* RIA, são por definição, portáveis por varias plataformas, sendo elas *Desktop*, *Mobile* ou *Web*. Assim, a SOA e os *frameworks* RIA exercem influência positiva sob o RNF Interoperabilidade e Portabilidade respectivamente, como ilustrado na figura 5.8, sendo então totalmente aceita a satisfação deste RNF.

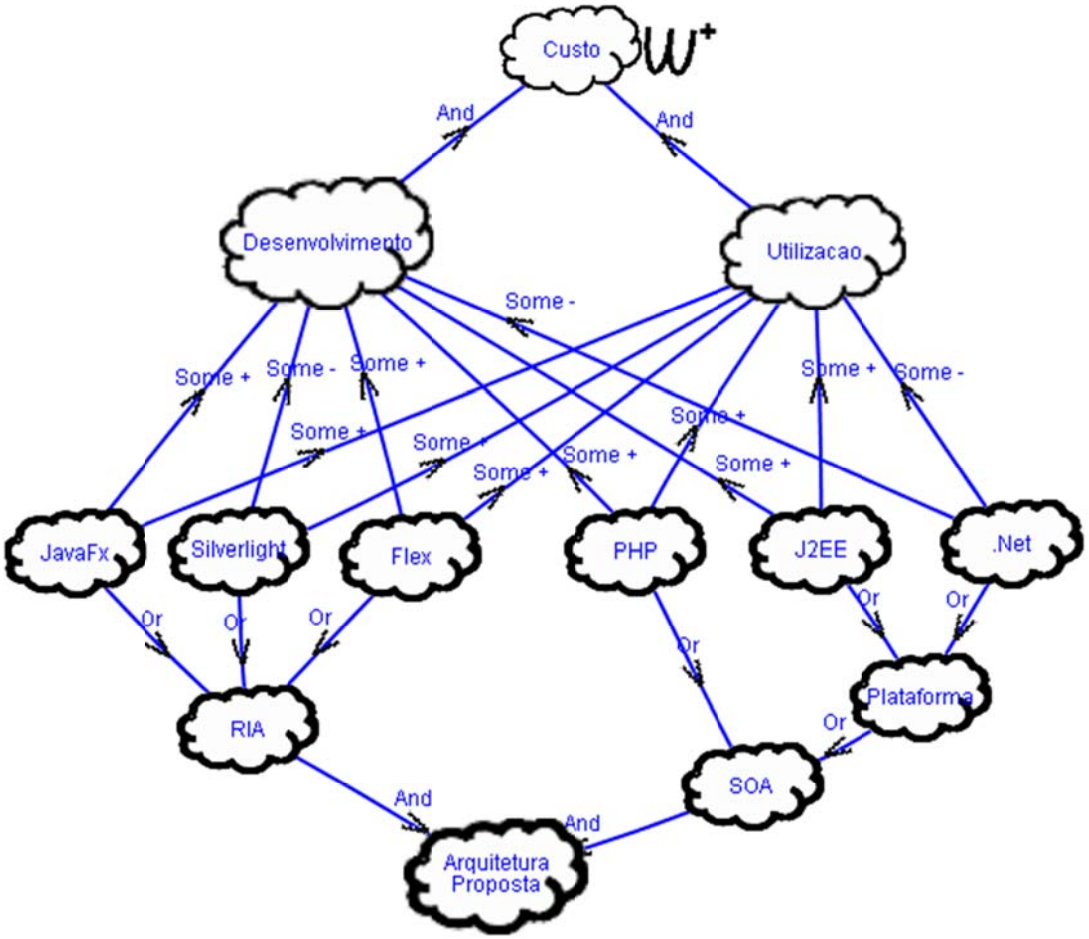


Figura 5.10: Grafo SIG do RNF custo e influências da arquitetura.

Custo: No planejamento de qualquer *software*, os custos relacionados à aquisição de ferramentas, treinamento, licenças e equipamentos necessários para o desenvolvimento e funcionamento da aplicação acabam sendo fatores importantes no momento da escolha. No grafo SIG ilustrado na figura 5.9, está relacionado às plataformas e *frameworks* estudados com suas respectivas operacionalizações. O JavaFx é uma contribuição positiva tanto no RNF de desenvolvimento quanto no de utilização, por ser um *framework* que se utiliza de ferramentas gratuitas para o desenvolvimento. Em relação aos outros *frameworks*, o Flex e o Silverlight são de utilização gratuita, mas ambos possuem ferramentas de desenvolvimento proprietárias, sendo o Flex dentre os três, o *framework* que possui uma ampla e disponível documentação. O JavaFx pode interagir diretamente com a linguagem de programação Java e pode integrar-se a outras linguagens através do consumo de serviços, enquanto o Silverlight e Flex podem utilizar o protocolo AMF.

O PHP é uma linguagem de programação Web que funciona em servidores e pode ser

desenvolvido em ferramentas gratuitas, é uma linguagem que está há muito tempo no mercado e não necessita de licença. Entre as plataformas, a plataforma .Net possui contribuições negativas por ser proprietária, já a J2EE é uma plataforma gratuita, sendo este um requisito decisivo para determinar qual contribuição exerce sobre os RNFs. No entanto cada uma destas plataformas possui suas particularidades e cabe ao engenheiro de *software* estudar e decidir qual delas melhor se adequa ao projeto a ser desenvolvido.

Capítulo 6

Estudo de Caso

Para o estudo de caso foi implementado um sistema para uma corretora de sementes. Este sistema foi construído utilizando do *framework* Adobe Flex, referente à parte de apresentação do sistema, e a linguagem de programação PHP tendo seus serviços disponíveis de duas maneiras:

- Utilizando AMF para prover a comunicação entre o PHP e o Flex.
- Utilizando SOAP para disponibilizar alguns dos serviços na *Internet* para acesso a outras aplicações.

O *back-end* da aplicação foi construído utilizando a linguagem PHP, contendo duas camadas. A primeira camada é um DAO que possui a responsabilidade de prover todas as transações necessárias entre o sistema e o SGBD MySQL. A segunda camada conterá todas as regras de negócios da aplicação, estas compõem os serviços necessários para o funcionamento da aplicação, sendo disponibilizados de duas formas: uma será usando o ZendAMF [26], utilizando as vantagens da serialização de dados e outra forma será utilizando o NuSOAP [27] que é uma implementação do SOAP para ser utilizado no PHP.

Para o funcionamento do AMF é necessário criar um arquivo com extensão .php que será encarregado de importar as bibliotecas necessárias para o seu funcionamento e pode-se declarar quais as classes dos serviços estão disponíveis para o seu acesso pela interface. Um fato importante é que os objetos criados tanto em PHP quanto no Flex podem ser transmitidos entre si, ou seja, um objeto criado em PHP pode ser enviado para o Flex através do AMF sem a necessidade de fazer conversões de tipos, tudo isso através de uma *metadata* que é inserido nas classes que as referenciam.

Será criado outro arquivo utilizando-se do NuSOAP para criar os serviços que disponibilizará uma lista dos produtos que a empresa tem disponível no momento da consulta.

Outros serviços não serão criados, pois esta é uma das necessidades que a empresa tem no momento, mas nada impede que outros serviços possam ser implementados.

Para a parte de apresentação do sistema foi usado Flex, através da utilização da ferramenta de desenvolvimento criada pela Adobe, o Flash Builder 4 Standard, que disponibiliza licenças gratuitas para fins não comerciais, para todos os que gostariam de conhecer esta ferramenta [28].

No Flash Builder foram criados dois projetos seguindo a ideia do Janderson [29], um projeto chamado CorretoraCore que será uma biblioteca (API), contendo todas as regras de negócio necessárias para o gerenciamento das tela de apresentação do sistema, além de conter os mecanismos necessários para os serviços disponibilizados através do ZendAMF. Dependendo da forma em que essa API for implementada, poderá ser disponibilizada para ser usada por outros desenvolvedores a utilizem ou agregá-la a outros projetos. Esse é o caso da API do GoogleMaps que disponibiliza uma biblioteca para o Flex contendo mecanismos para customizar a tela de mapas, podendo adicionar recursos de deslocamento latitudinal e longitudinal, zoom e alterar o ângulo de visualização.

Um projeto CorretoraWeb contem as telas de apresentação da aplicação *Web* e que teve incluso a biblioteca mencionada anteriormente para exibir os dados do sistema. Também foi criado um projeto CorretoraDesktop contendo as telas de apresentação da aplicação para *Desktop* que foi utilizado para consumir o serviço disponibilizado através do NuSOAP com o propósito de averiguar seu correto funcionamento. A imagem a seguir mostra os projetos criados.

Figura 6.1: Lista dos projetos criados no Adobe Flash Builder.

A partir desta sugestão de projeto, o engenheiro de *software* poderá criar projetos de acordo com a necessidade do cliente, por exemplo, uma empresa que comercializa um determinado produto e necessita de uma aplicação *Web* que gerencie as operações da empresa. Este projeto que esta sendo desenvolvido seguindo a arquitetura proposta, permitirá realizar qualquer operação, tal como: compra, venda, entre outras ações, através da *Internet*, então será criada uma infraestrutura necessária para fazer o gerenciamento utilizando-se do PHP ou qualquer

outra linguagem, disponibilizando os serviços através do protocolo AMF e SOAP, criando as telas com o *framework* Adobe Flex ou qualquer outro *framework* RIA.

Neste sentido, pode-se observar que se construída de maneira correta, permitirá agilidade na execução de algumas melhorias no sistema, como por exemplo, se este mesmo cliente venha através de uma parceria querer disponibilizar uma lista de produtos para venda a outra empresa, basta o engenheiro de *software* criar um serviço especifico para esse parceiro com a listagem dos insumos e disponibiliza-lo através do protocolo SOAP. Caso precise construir interfaces para dispositivos *Mobiles*, basta que sejam criadas as telas e integradas com a API existente, sendo necessário somente o teste para validar o funcionamento da interface, tendo em vista que a API foi devidamente testada.

O *back-end* é a parte da aplicação que contem toda a implementação do funcionamento de todo o sistema, para que fosse realizada a sua construção, foi criado um projeto no NetBeans 6.9 que é uma IDE gratuita que além ter suporte para a linguagem de programação Java, possui suporte para a linguagem PHP.Nesta parte foi criada toda a parte que será responsável pelo gerenciamento dos dados que a aplicação necessitará trabalhar, tendo nela dois arquivos importantes para que essas operações sejam possíveis, o services.php que contem a implementação do acesso de dados através da utilização do protocolo AMF e o arquivo soap_products.php que contem as métricas de consumo de serviço através da utilização do protocolo SOAP, os códigos-fontes dos arquivos citados estão em anexo no final deste trabalho. Nas figuras a seguir são mostradas as telas do sistema.

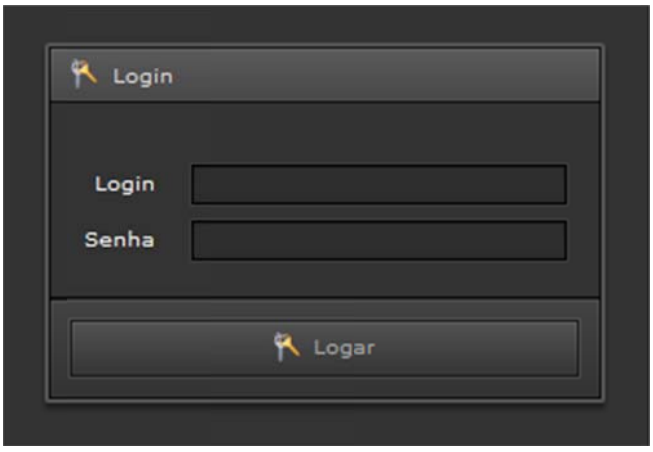


Figura 6.2: Tela de acesso ao sistema.

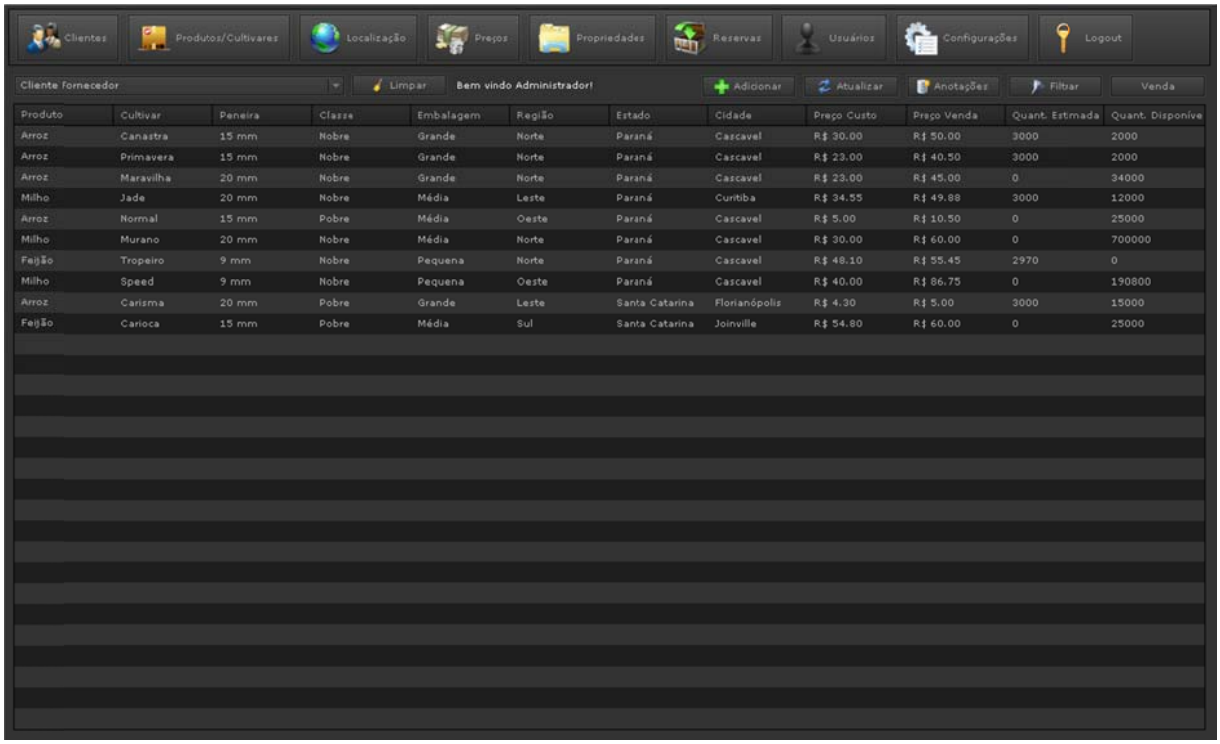


Figura 6.3: Tela principal do sistema.

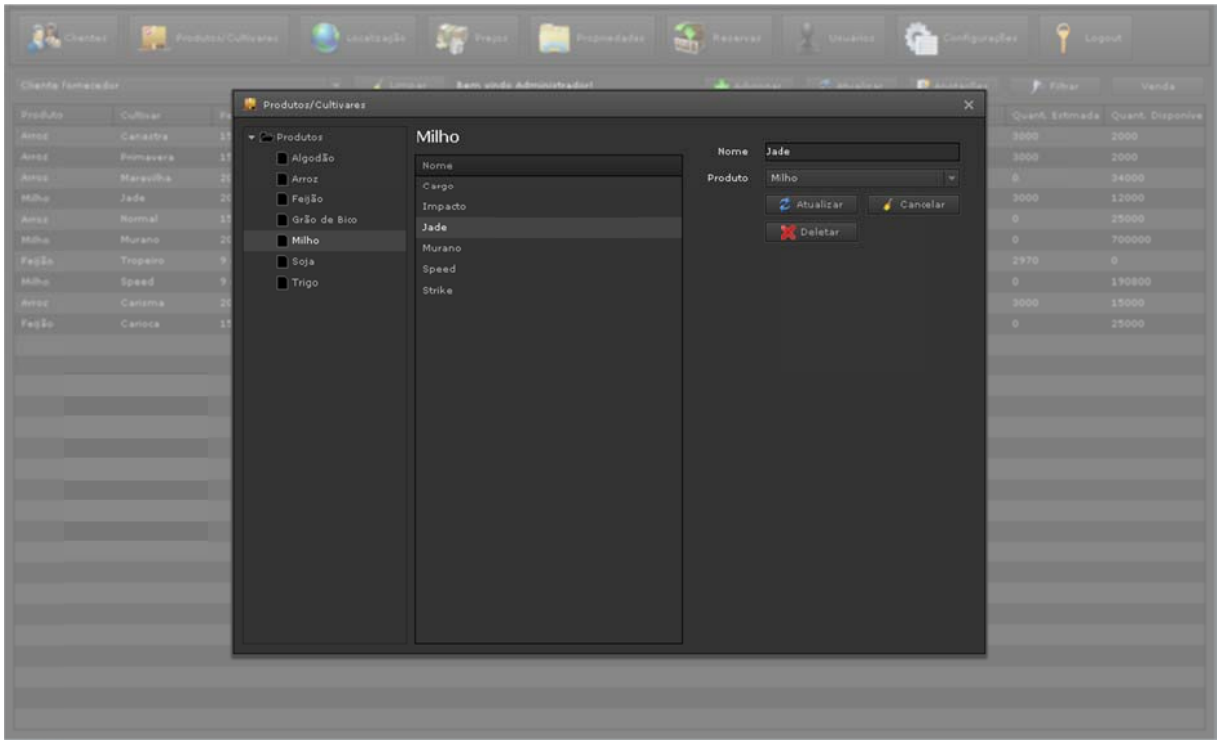


Figura 6.4: Tela de gerenciamento de produtos.

Assim observando-se a arquitetura proposta na figura 5.1 e os RNFs apresentados na figura 5.2 podemos observar que a organização arquitetural apresenta muitos benefícios, permitindo o desenvolvimento de aplicações com alto nível de qualidade, pelo fato de satisfazerem uma grande quantidade de requisitos não-funcionais. Ao atingir essa satisfação, o engenheiro de *software* tem a possibilidade de entregar ao cliente um sistema com alto grau de aceitação, com interface dinâmica, e de acordo com o nível organizacional da tela, poderá torná-la agradável e de fácil compreensão, permitindo agilidade na execução de tarefas. Além disso, facilita o trabalho do engenheiro de software nas ocasiões em que o cliente necessite adicionar novas funcionalidades sem causar drásticas mudanças no projeto.

Capítulo 7

Considerações Finais e Trabalhos Futuros

Com a realização deste trabalho pôde-se elaborar uma proposta de arquitetura que contem um alto grau de qualidade, através da satisfação de requisitos não-funcionais, inseridos no catálogo elaborado a partir da junção da normativa NBR ISO/IEC 9126 e o catálogo do *NFR Framework*.

O catálogo elaborado neste trabalho além de auxiliar na validação da proposta de arquitetura, também permite que o engenheiro de *software* use-o como base para a avaliação do nível de qualidade das aplicações por ele elaboradas, ajudando na percepção dos êxitos e fracassos possibilitando o estudo de técnicas que permitam o melhoramento das partes que afetam de forma negativa os RNFs mapeados.

Esta proposta tem como objetivo utilizar *frameworks* RIA e a arquitetura SOA, possibilitando maior flexibilidade no gerenciamento das funcionalidades do sistema e disponibilizar ao usuário interfaces com maior interatividade, já que a maioria dos sistemas *Web* não possui esta característica.

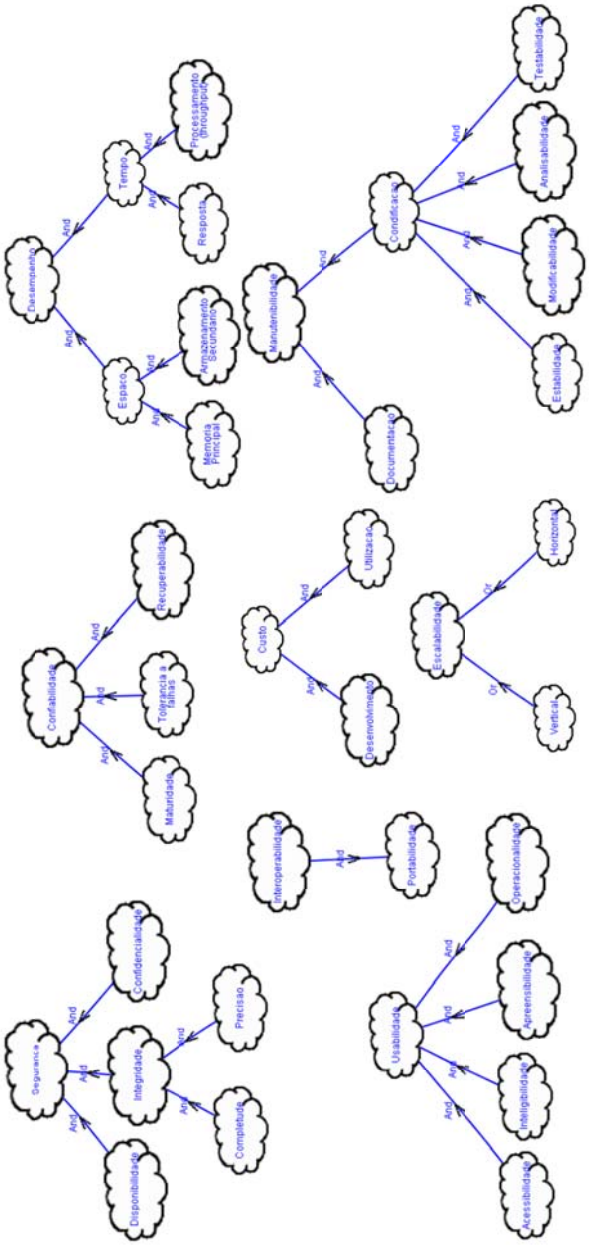
Como limitação da arquitetura, temos a dificuldade no aumento da escalabilidade da aplicação, isso se deve ao fato de que todos os serviços são centralizados. Além disso, esta proposta pode ser aprimorada através de um estudo aprofundado visando melhorar a satisfação de cada RNF apresentado.

A arquitetura apresentada não teve uma validação eficiente para garantir com segurança os resultados apresentados, que foram em grande parte obtidos através de análises teóricas de cada requisito não-funcional em relação a arquitetura e *frameworks* RIA, sendo necessário um estudo aprofundado de cada RNF para obter resultados mais precisos do que os apresentados neste trabalhos. Portanto, esta seria uma sugestão de trabalho futuro.

Apêndice A

Grafo SIG com RNFs

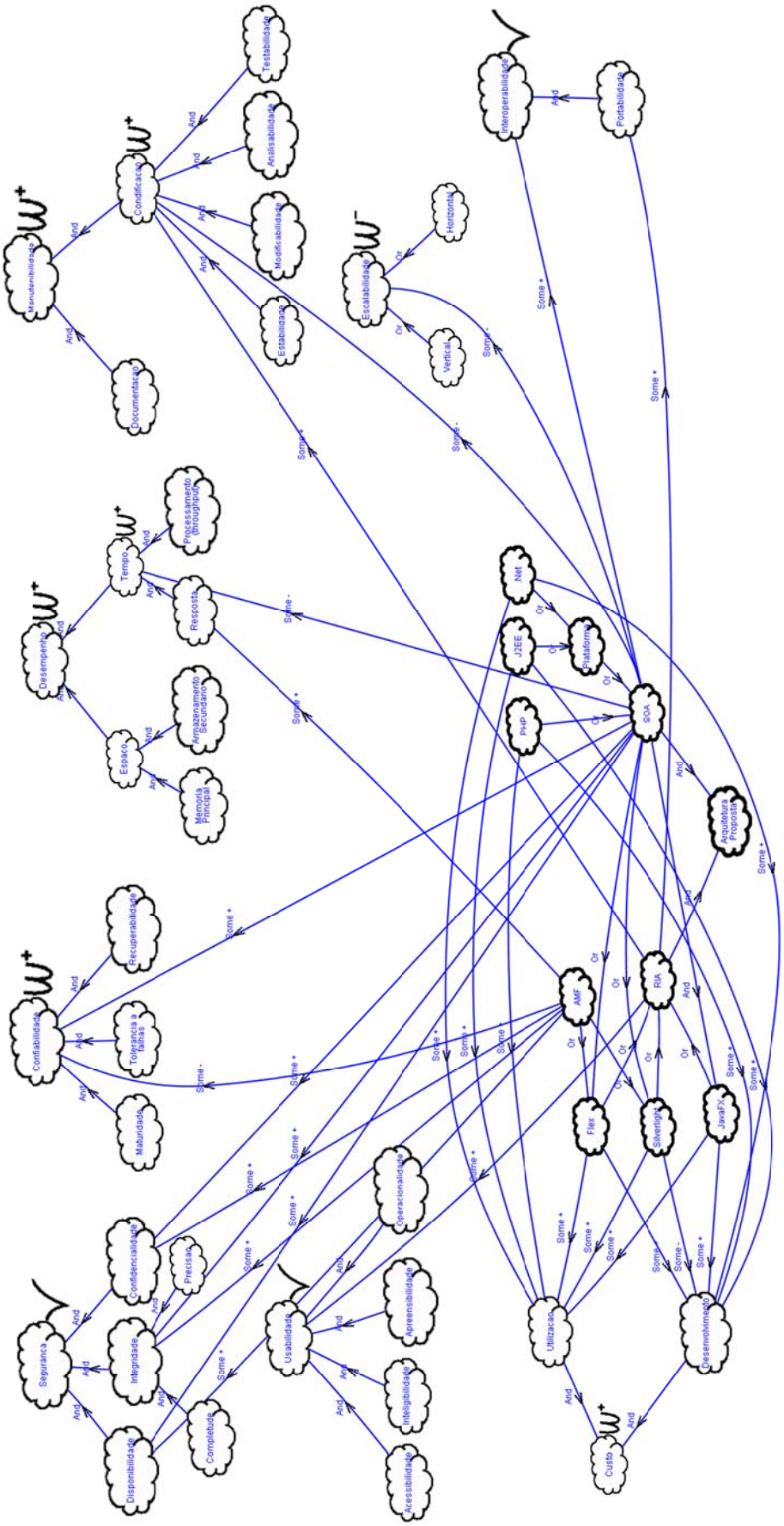
Grafo SIG referente à imagem 2.2, do capítulo 2, seção 2.2.3



Apêndice B

Grafo SIG com Operacionalizações

Grafo SIG referente à imagem 6.1 do capítulo 6.


Manutenibilidade
Condificação
Testabilidade
Analisabilidade
Modificabilidade
Estabilidade
Documentação
And
Escalabilidade
Horizontal
Vertical
Or
Interoperabilidade
Portabilidade
Some +
Some -
Desempenho
Tempo
Processamento (throughput)
Resposta
Espaço
Armazenamento Secundário
Memória Principal
Net
J2EE
Plataforma
PHP
SOA
Arquitetura Proposta
Confiabilidade
Recuperabilidade
Tolerância a falha
Maturidade
AMF
RIA
Flex
Silverlight
JavaFX
Segurança
Confidencialidade
Integridade
Disponibilidade
Precisão
Completude
Usabilidade
Operacionalidade
Apreensibilidade
Inteligibilidade
Acessibilidade
Utilização
Desenvolvimento
Custo

Apêndice C

Implementação protocolo AMF

Implementação do protocolo SOAP contido no arquivo services.php.

```
<?php
// Start session
session_cache_expire(10);
session_start();
// Set up debug
error_reporting( E_ALL | E_STRICT );
ini_set('display_errors', "off");

// Set up include path for Zend Framework, this path is assuming a
frameworks
// folder contains the Zend package on the same level as your root folder.
ini_set('include_path', ini_get('include_path').'\Zend');

// Require Zend_Amf_Server
require_once('Zend/Amf/Server.php');
require_once('Zend/Loader/Autoloader.php');
Zend_Loader_Autoloader::getInstance()->setFallbackAutoloader(true);

// Require mapped classes
require_once('Control/Control.Address.class.php');
require_once('Control/Control.Bank.class.php');
require_once('Control/Control.BusinessAcumen.class.php');
require_once('Control/Control.City.class.php');
require_once('Control/Control.Class.class.php');
require_once('Control/Control.Client.class.php');
require_once('Control/Control.Employee.class.php');
require_once('Control/Control.Farm.class.php');
require_once('Control/Control.ManagerTables.class.php');
require_once('Control/Control.Operation.class.php');
require_once('Control/Control.Package.class.php');
require_once('Control/Control.Personal.class.php');
require_once('Control/Control.Price.class.php');
require_once('Control/Control.Product.class.php');
require_once('Control/Control.Products.class.php');
require_once('Control/Control.Region.class.php');
require_once('Control/Control.Sieve.class.php');
require_once('Control/Control.State.class.php');
require_once('Control/Control.Stock.class.php');
require_once('Control/Control.TypeCrop.class.php');
require_once('Control/Control.TypeOperation.class.php');
require_once('Control/Control.User.class.php');
require_once('Control/Control.Timer.class.php');
```

```
// Start Zend AMF Server
$server = new Zend_Amf_Server();

// Register ZendAMF Service classes
$server->setClass('ControlAddress');
$server->setClass('ControlBank');
$server->setClass('ControlBusinessAcumen');
$server->setClass('ControlCity');
$server->setClass('ControlClass');
$server->setClass('ControlClient');
$server->setClass('ControlEmployee');
$server->setClass('ControlFarm');
$server->setClass('ControlManagerTables');
$server->setClass('ControlOperation');
$server->setClass('ControlPackage');
$server->setClass('ControlPersonal');
$server->setClass('ControlPrice');
$server->setClass('ControlProduct');
$server->setClass('ControlProducts');
$server->setClass('ControlRegion');
$server->setClass('ControlSieve');
$server->setClass('ControlState');
$server->setClass('ControlStock');
$server->setClass('ControlTypeCrop');
$server->setClass('ControlTypeOperation');
$server->setClass('ControlUser');
$server->setClass('ControlTimer');

// Handle the AMF request
echo($server->handle());
?>
```

Apêndice D

Implementação protocolo SOAP

Implementação do protocolo SOAP contido no arquivo soap_products.php.

```
<?php
require_once 'SOAP/nusoap.php';
require_once 'DAO/DAOFactory.class.php';
require_once 'Object/Product.class.php';
require_once 'Object/TypeCrop.class.php';

$server = new soap_server();
$server->configureWSDL('listproducts_wsdl', 'urn:listproducts_wsdl');
$server->wsdl->schemaTargetNamespace = 'urn:server.listproducts_wsdl';
$server->soap_defencoding = 'UTF-8';

$server->wsdl->addComplexType('Product',
    'complexType',
    'struct',
    'all',
    '',
    array(
            'id' => array('name' => 'id', 'type' => 'xsd:int'),
            'name' => array('name' => 'name', 'type' => 'xsd:string'),
            'type_crop' => array('name' => 'type_crop', 'type' =>
'xsd:string')
    )
);
$server->wsdl->addComplexType('AllProducts',
    'complexType',
    'array',
    '',
    'SOAP-ENC:Array',
    array(),
    array(
            array('ref'=>'SOAP-
ENC:arrayType','wsdl:arrayType'=>'tns:Product[]')
    ),
    'tns:Product'
);
$server->register('listAllProducts',
    array(),
    array('return' => 'tns:AllProducts'),
    'listproducts_wsdl',
    'listproducts_wsdl#list',
    'rpc',
    'encoded',
```

```
    'Return list all products'
);
$HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA :
'';
$server->service($HTTP_RAW_POST_DATA);

function listAllProducts() {
      $dao = DAOFactory::getDAOFactory ( DAOFactory::MYSQL );
      $products = $dao->getProduct ()->listAll ();
      $results = array ();
      foreach ( $products as $product ) {
            $typecrop = $dao->getTypeCrop()->select ((int) $product-
>getClsTypeCrop()->getIntId() );

            $tempArray = array ('id' => $product->getIntId (), 'name' =>
$product->getStrName (), 'type_crop' => $typecrop->getStrName() );
            array_push ( $results, $tempArray );
      }

      return $results;
}
?>
```

Referências Bibliográficas

[1] STALEY, Ted. *Planning for RIA Success*. Las Vegas. Adobe Systems Incorporated. 2007.

[2] CHUNG, L., Nixon, B.A., Yu, E., Mylopoulos, J. *Non-Functional Requirements in Software Engineering (Monograph)*. Kluwer Academic Publishers, p. 472, Springer, 2000.

[3] ISAKOWITZ, Tomás, STOHR, Edward A., BALASUBRAMANIAN, P. *RMM: A Methodology for Structured Hypermedia Design*. Comm ACM, p. 34-48, August 1995.

[4] PAOLINI, P., GARZOTTO, F., MAINETTI, L. *HDM - A Model-Based Approach to Hypertxt Application Design*. ACM Transaction on Information Systems, v. 11, n. 1, p. 1-26, January 1993.

[5] SCHWABE, Daniel; ROSSI, Gustavo; BARBOSA, Simone D. J. *Systematic Hypermedia Application Design with OOHDM*. Pontifícia Universidade Católica. Rio de Janeiro. 1996.

[6] SANTANDER, V. F. A.; VASCONCELOS, A. L. *Estudo de Princípios de Qualidade em Aplicações Web*. In: IDEAS'00 - Terceiro Workshop Ibero-Americano de Engenharia de Requisitos e Ambientes de Software, 2000, Cancún, México, 5-7 de abril.

[7] GARRET, Jesse J. *Ajax: A New Approach to Web Applications*. Disponível em: http://adaptivepath.com/ideas/essays/archives/000385.php. Acesso: 23 de Março de 2010.

[8] MACDONALD, Matthew. *Pro Silverlight 3 in C#*. 1. ed. Reading: Apress. 2009.

[9] STEINMANN, Lalo. *Changing the world with SOA?*. Congresso Ibero-americano sobre "Engenharia de software" (CIbSE). Universidad del Azuay, Cuenca, Ecuador. 2010.

[10] FILHO, Antonio Mendes da Silva. *Arquitetura de Software*. 1. ed. Reading: Editora Campus; 2002.

[11] GARLAND, Jeff; ANTHONY, RICHARD. *Large-Scale Software Architecture - A Practical Guide Using UML*. 1. ed. Reading; John Wiley & Sons, 2003.

[12] OASIS. *Reference Model for Service Oriented Architecture*. Disponível em: http://www.pcs.usp.br/~pcs5002/oasis/soa-rm-csbr.pdf. Acessado: 15 de Julho de 2010.

[13] W3C. *Working Group Note - Web Services Architecture*. Disponível em: http://www.w3.org/TR/ws-arch. acessado: 15 de Julho de 2010.

[14] SUNDSTED, Todd. *MVC meets Swing*. Disponível em: http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html. Acesso: 23 de Julho de 2010.

[15] MIYASHIRO, Paulo. *SAP e Flex: Soluções com Usabilidade para Sistemas Corporativos*. Disponível em: http://www.flexmania.com.br/gravacoes.php. Acesso: 20 de Julho de 2010.

[16] RECKZIEGEL, M. *Protocolo de Transporte Padrão - SOAP*. Disponível em:

http://imasters.uol.com.br/artigo/4379/webservices/protocolo_de_transporte_padrao_-_soap. Acesso: 14 de Julho de 2010.

[17] ALVES, Rodrigo Sanger, HECHT, Fábio Victora, TOLÖKEN, Rafael, GRANVILLE, Lisandro Zambenedetti, ALMEIDA, Maria Janilce B., TAROUCO, Liane Margarida Rochenbach; Comparação e Avaliação dos Protocolos NETCONF e SOAP para Configuração de Dispositivos; [WGRS] X : 2005 mai. 10 : Fortaleza/CE

[18] ZAVALIK, Claudimir; LACERDA, Guilherme; OLIVEIRA, José Palazzo M.; Implementando Web Services com Software Livre; Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS); 2004.

[19] CALIENDO, Roberto Felipe, NOTARI, Daniel Luís; Desenvolvimento de uma Aplicação Utilizando SOA: Um Estudo de Caso; Publicação: [ERBD] 5ª Edição, Abril. 2009.

[20] GASSNER, D. *Flex 3 Bible*. 1. ed. Reading: Wiley; 2008.

[21] HARDING, John. *Flash and the HTML5 <video> tag*; Disponível em: http://apiblog.youtube.com/2010/06/flash-and-html5-tag.html; Acesso: 15 de Julho de 2010.

[22] COLE, Alaric. *Learning Flex 3*. 1. ed. Reading: O'Reilly. 2008.

[23] CLAUDIO, Pedro; Seu back-end é tão produtivo quanto seu MXML?; Disponível em: http://www.flexmania.com.br/gravacoes.php; Acesso: 20 de Julho de 2010.

[24] AHMED, Tariq, ORLANDO, Dan, John C. Bland II, HOOKS, Joel; Flex 4 in Action; Manning; 2009.

[25] KEITH, Jeremy. HTML5 for Web Designers. 1. ed. Reading: A Book Apart. 2010.

[26] Zend Framework; Disponível em: http://framework.zend.com/download/; Acesso: 23 de Julho de 2010.

[26] ALARCÓN, Raúl. Diseño Orientado a Objetos con UML. 1. ed. Reading: Grupo Eidos. 2000.

[27] NuSOAP - SOAP Toolkit for PHP. Disponível em: http://sourceforge.net/projects/nusoap/. Acesso: 23 de Julho de 2010.

[28] Adobe Developer Connection. Disponível em: http://www.adobe.com/devnet/flex/free/index.html. Acesso: 12 de Julho de 2010.

[29] CARDOSO, Janderson F. *Flex 4 - Desenvolvendo com Portabilidade (Web, Desktop e Mobile)*. Disponível em: http://www.flexmania.com.br/gravacoes.php. Acesso: 20 de Julho de 2010.

[30] CHUNG, L., NIXON, B. A., MYLOPOULOS. *Representing and Using Non-Functional Requirements: a process oriented approach*. IEEE Trans. on Software Engineering, v. 18, n. 6, p. 483 – 497. 1992.

[31] CHUNG, L., NIXON, B.A. *Dealing with Non-Functional Requirements: three experimental studies of a process-oriented approach*. Proc. of the 17th International Conference on Software Engineering (ICSE ’95), ACM, p. 25–37, New York, NY, USA, 1995.

[32] ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBRISO/IEC9126-1 Engenharia de software - Qualidade de produto - Parte 1: Modelo de qualidade. 2003.

[33] PRESSMAN, Roger S. *Engenharia Web*. 1. ed. Reading: LTC. 2010.

[34] PRECIADO, Juan Carlos; FIGUEROA, Fernand Sanchez. On the Implementation of Multiplatform RIA User Interface Components. Workshop on Web-Oriented Software Technologies. San Sebastian. Spain. 2008.

[35] ANDERSON, Gail; ANDERSON, Paul; *Essential JavaFX*. 1. ed. Reading: Prentice Hall. 2009.

[36] GAMMA, Erich; HELM Richard; JOHNSON, Ralph; VLISSIDES, John. *Padrões de projeto*. 1. ed. Reading: Bookman. 2000.

[37] THE MIDNIGHT CODERS. Disponivel em: http://www.themidnightcoders.com/products/weborb-for-net/developer-den/technical-articles/amf-vs-webservices.html. Acesso: 01 de Novembro de 2010.

[38] DIAS, Jorge. Arquitetura Orientada a Serviços - Sobre o que você precisa refletir para adotá-la em um contexto empresarial. *Engenharia de Software Magazine*, Rio de Janeiro, v. 2, ed. 22, p. 30-35, Março, 2010.

[39] DIAS, Jorge. SOA e seus Atributos de Qualidade - Entenda os atributos de qualidade em uma Arquitetura Orientada a Serviços. *Engenharia de Software Magazine*, Rio de Janeiro, v. 2, ed. 23, p. 31-35, Abril, 2010.