Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

GUILHERME DE CLEVA FARTO

ABORDAGEM ORIENTADA A SERVIÇOS PARA IMPLEMENTAÇÃO DE UM APLICATIVO *GOOGLE ANDROID*

Assis
2010

GUILHERME DE CLEVA FARTO

ABORDAGEM ORIENTADA A SERVIÇOS PARA IMPLEMENTAÇÃO DE UM APLICATIVO *GOOGLE ANDROID*

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis,
como requisito do Curso de Graduação

Orientadora: Profª Drª Marisa Atsuko Nitto

Área de Concentração: Informática

Assis
2010

FICHA CATALOGRÁFICA

FARTO, Guilherme de Cleva

Abordagem orientada a serviços para implementação de um aplicativo *Google Android* / Guilherme de Cleva Farto. Fundação Educacional do Município de Assis – FEMA – Assis, 2010.

83p.

Orientadora: Profª Drª Marisa Atsuko Nitto

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA

1. Arquitetura SOA **2**. Web Services **3**. Google Android

CDD: 001.6

Biblioteca da FEMA

ABORDAGEM ORIENTADA A SERVIÇOS PARA IMPLEMENTAÇÃO DE UM APLICATIVO *GOOGLE ANDROID*

GUILHERME DE CLEVA FARTO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientadora: Profª Drª Marisa Atsuko Nitto

Analisador (1): Profº Drº Almir Rogério Camolesi

Assis
2010

DEDICATÓRIA

Dedico este trabalho à minha família, amigos e todas as pessoas que acreditaram em meus sonhos e anseios, apoiando-me com a força necessária para que pudesse realizá-los

AGRADECIMENTOS

Primeiramente a Deus, pois sem Ele, nada seria possível e não estaríamos aqui reunidos, desfrutando, juntos, destes momentos que nos são tão importantes.

À minha orientadora e amiga Profª Drª Marisa Atsuko Nitto, pela orientação, não somente durante este trabalho, mas por toda a minha vida acadêmica, por acreditar e tornar possível a construção de muitos de meus sonhos.

Aos meus familiares, Cecilia Aparecida de Cleva, Mauro Pereira Farto e Gabriel de Cleva Farto, por estarem sempre ao meu lado, apoiando-me e conduzindo-me durante minha caminhada.

A minha namorada, Tamires Alves da Silva, por deixar os dias de minha vida mais alegres e belos.

E a todos que colaboraram direta ou indiretamente na execução deste trabalho.

“The mind that opens to a new idea, never comes back to it's original size […]”

Albert Einstein (1879-1955)

RESUMO

A proposta deste trabalho foi a de pesquisar e compreender os conceitos da arquitetura orientada a serviços e as ferramentas empregadas para o uso de *Web Services* na linguagem de programação *Java*, assim como o desenvolvimento de aplicativos baseados na plataforma *Google Android* capazes de consumir serviços disponíveis na *Web*.

Com a finalidade de aplicar os conceitos adquiridos na fase de pesquisa teórica, um estudo de caso foi modelado, abordando a implementação de um ambiente distribuído e orientado a serviços e de duas aplicações cliente, uma *Web*, desenvolvida utilizando-se as tecnologias *Java* e *Flex*, e uma *Mobile*, desenvolvida utilizando-se a plataforma *Google Android*.

Ao concluir a etapa de implementação, foi possível testar e validar os produtos gerados, integrando-os como parte deste trabalho acadêmico para expor os resultados alcançados e contribuir com a sociedade acadêmica e científica em um assunto novo, recente e, crescentemente, estudado pelas empresas e grupos corporativos.

Palavras-chave: *Java*; XML; Arquitetura SOA; Arquitetura Orientada a Serviços; Orientação a Serviços; *Web Services*; *Google Android*.

ABSTRACT

The purpose of this study was to investigate and understand the concepts of service oriented architecture and tools employed for the use of Web Services in the Java programming language, as well as developing applications based on the Google Android platform capable of consuming the services available Web.

In order to apply the concepts acquired in the phase of theoretical research, a case study was modeled, addressing the implementation of a distributed environment and service-oriented and two client applications, a Web, developed using the Java and Flex and a Mobile, developed using the Google Android platform.

By completing the implementation stage, it was possible to test and validate the products generated by integrating them as part of this work to expose academic achievements and contribute to the academic and scientific society in a new subject, a recent and increasingly studied by companies and corporate groups.

Keywords: Java; XML; SOA Architecture; SOA; Service Oriented Architecture; Web Services; Google Android.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 – INTRODUÇÃO

Na última década, têm-se visto grandes melhorias em produtividade, na maioria dos setores da economia; as pessoas estão realizando mais por menos. Trabalhadores de fábricas produzem artefatos mais rapidamente, resolvem problemas mais rapidamente, até mesmo bancos liquidam cheques mais rapidamente. O processo de integrar aplicações de *software* corporativas, no entanto, tem resistido a essa tendência positiva: parece tão lento e improdutivo quanto sempre foi fazer com que um *software* fale com outro (PULIER; TAYLOR, 2008).

A arquitetura orientada a serviços ou SOA (*Service-Oriented Architecture*) é um paradigma para a realização e manutenção dos processos corporativos que se encontram em grandes sistemas distribuídos. Ela se baseia em três conceitos técnicos principais: serviços (pedaço de funcionalidade corporativa independente), barramento corporativo de serviços (infraestrutura que possibilita a alta interoperabilidade entre sistemas distribuídos para serviços) e acoplamento fraco (conceito de redução de dependências do sistema) (JOSUTTIS, 2008).

Apesar de a orientação a serviços, como paradigma, e da SOA, como uma arquitetura tecnológica, serem neutras em relação à implementação, sua associação com os *Web Services* se tornou tão comum, que os principais fornecedores de SOA modelam suas respectivas plataformas em torno da utilização da tecnologia de *Web Services* (ERL, 2009). Conceitualmente, um *Web Service* fornece um caminho alternativo para expor a lógica da aplicação para um conjunto de clientes heterogêneos, permitindo que entidades do negócio realizem transações pela *Internet*, minimizando tanto o investimento em infraestrutura quanto as regras para comunicação entre os atores que necessitam se relacionar (SUN MICROSYSTEMS, 2003). Embora a orientação a serviços permaneça um paradigma totalmente abstrato, trata-se de um paradigma historicamente influenciado pelas plataformas de SOA produzidas por esses fornecedores. Por esse motivo, o *framework* dos *Web Services* inspirou e promoveu vários princípios da orientação a serviços, incluindo os da abstração, do baixo acoplamento e da composição de serviços (ERL, 2009).

Com o emprego cada vez maior de dispositivos móveis, como *smartphones* (pequenos computadores pessoais) e celulares, o número de plataformas e ambientes de desenvolvimento cresce proporcionalmente. Os dispositivos móveis oferecem a vantagem da conectividade e poder de uso, em qualquer lugar e momento, tornando-se importante para uso não apenas pessoal, mas também profissional (ABLESON; COLLINS; SEN, 2009).

A escolha de uma plataforma ideal para o desenvolvimento de um projeto significa optar por uma solução que propicie os melhores benefícios, em termos de custos, eficiência e tempo de desenvolvimento esperados para a sua finalização. Neste projeto de pesquisa, será usada a plataforma *Android* SDK, lançada recentemente pelo *Open Handset Alliance* (OHA), que funciona como um sistema operacional como os já existentes *Symbian* e *Windows® Mobile*, com a diferença de possuir o código fonte aberto. O grupo OHA é formado por mais de 30 empresas, tendo como objetivo consolidar uma plataforma única, com diversos recursos que os usuários procuram em um celular moderno, e ainda revolucionar a maneira de construir aplicações para dispositivos móveis, acirrando a disputa no mercado corporativo (FARIA, 2008). As aplicações para essa plataforma são escritas empregando a linguagem de programação *Java* e executadas sobre o *Dalvik*, uma máquina virtual customizada para dispositivos com restrições de recursos, com pouca capacidade computacional, baixa capacidade de armazenamento e baterias com baixo nível de energia. Além disso, ela fornece uma abordagem bastante simples para a modelagem de aplicações que venham a utilizar essa plataforma (ABLESON; COLLINS; SEN, 2009), (FARIA, 2008), (DIMARZIO, 2008) e (LECHETA, 2009).

A proposta deste projeto de pesquisa consiste em desenvolver um ambiente orientado a serviços capaz de expor funcionalidades que serão acessadas por uma aplicação cliente desenvolvida com a plataforma *Google Android*, onde o usuário irá interagir com um celular inteligente que, comunicando-se com um servidor de aplicação, permitirá consumir soluções lógicas remotas. O desenvolvimento do projeto de pesquisa será dividido em duas fases. A primeira fase consiste em adquirir conhecimento das tecnologias e ferramentas envolvidas para o desenvolvimento do ambiente e a modelagem do problema. A segunda fase consiste em colocar em prática todos os conhecimentos adquiridos para a implementação do

ambiente. Nesta apresentação, será abordada apenas a primeira fase de execução do projeto, sendo que a segunda fase será explicitada somente ao final da realização do projeto.

1.1 OBJETIVOS

O objetivo geral deste trabalho é pesquisar e compreender os conceitos da arquitetura SOA e as ferramentas empregadas para o desenvolvimento de *Web Services*, na linguagem de programação *Java*, com a finalidade de definir um estudo de caso sobre a implementação de aplicações orientadas a serviços, baseadas na plataforma *Google Android*.

Com o desenvolvimento deste estudo de caso, será possível definir a estrutura de um ambiente provedor de serviços a ser acessado por um equipamento *Mobile*, demonstrando que as tecnologias utilizadas nesta pesquisa são capazes de solucionar problemas relacionados à interoperabilidade de sistemas com dispositivos móveis, como a demora na centralização e na atualização de informações, comumente realizadas através da importação e exportação de dados, e a capacidade limitada de obter e manipular grande quantidade de dados armazenados em dispositivos como *smartphones* e celulares. Para atingir os objetivos traçados, a execução do projeto foi dividida em duas fases:

1ª fase: pesquisar e analisar a arquitetura SOA; pesquisar e analisar a implementação de *Web Services*, usando a linguagem de programação *Java*; pesquisar e analisar a tecnologia *Google Android* e especificar um estudo de caso: modelar o problema.

2ª fase: desenvolver a arquitetura do ambiente; desenvolver a documentação do *software*; realizar a implementação; testar, validar e descrever os resultados obtidos.

1.2 JUSTIFICATIVAS

A arquitetura orientada a serviços é uma abordagem que ajuda os sistemas a permanecerem escaláveis e flexíveis enquanto crescem, auxiliando também a resolver a lacuna negócio/TI (Tecnologia de Informação), em que as pessoas de negócio e as de tecnologia parecem falar e pensar em línguas totalmente diferentes. A abordagem SOA consiste em três elementos: serviços, infraestrutura e políticas e processos, aproximando as áreas de negócio e TI, quando devidamente empregada (JOSUTTIS, 2008).

A orientação a serviços representa um estado evolutivo importante, na história da TI, porque combina elementos de *design* bem-sucedidos de abordagens antigas com novos elementos de *design*, tirando proveito da inovação conceitual e tecnológica. Algumas das vantagens obtidas por empregar a arquitetura de serviços são uma maior consistência na representação da funcionalidade de dados, menor dependência entre unidades de lógica, maior disponibilidade e capacidade de escala e maior consciência da lógica disponível (ERL, 2009).

Um *Web Service* é hoje uma das tecnologias mais utilizadas para integrar aplicações. A tecnologia *Google Android*, ainda que não possua nenhuma *Application Programming Interface* (API) nativa para acessar um *Web Service*, permite o desenvolvimento de protocolos e mecanismos para a busca e consumo de soluções baseadas em serviços.

1.3 MOTIVAÇÃO

Apesar de as tecnologias que envolvem arquiteturas orientadas a serviços estarem sendo muito utilizadas, existem poucos trabalhos científicos e técnicos abordando o desenvolvimento de ambientes distribuídos que fazem uso de serviços disponíveis na *Web* para dispositivos móveis.

Aplicando os conceitos da arquitetura SOA, implementados por meio de *Web Services*, para o desenvolvimento de uma aplicação *Google Android*, será possível

obter um ganho na qualidade e agilidade aos serviços prestados por empresas e grupos corporativos.

Outra motivação são as consideráveis chances de atuar no mercado de trabalho, baseando-se neste tema.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está estruturado nas seguintes partes:

- **Capítulo 1 – Introdução**
- **Capítulo 2 – Arquitetura Orientada a Serviços – SOA**
- **Capítulo 3 – *Web Services***
- **Capítulo 4 – Tecnologia *Google Android***
- **Capítulo 5 – Modelagem do Problema**
- **Capítulo 6 – Conclusão Parcial**
- **Referências Bibliográficas**

2 – ARQUITETURA ORIENTADA A SERVIÇOS – SOA

O intuito deste capítulo é apresentar Service-Oriented Architecture (SOA) ou Arquitetura Orientada a Serviços como um conceito, destacando os aspectos fundamentais e mostrando as circunstâncias apropriadas para seu uso.

Para entender totalmente a arquitetura orientada a serviços, é necessário familiarizar-se com o que constitui essa plataforma, além de compreender o significado e a importância de seu bloco fundamental de construção: o serviço. E, para adquirir essa compreensão, é preciso saber como uma unidade lógica pode ser transformada em algo que se pode chamar de “orientado a serviços”.

Neste capítulo, encontram-se os conceitos e o paradigma da orientação a serviços, fornecendo-se o ponto de partida para essa exploração e estabelecendo-se os conceitos de alto nível e a respectiva terminologia.

2.1 DEFINIÇÕES DE SOA

Existem muitas definições diferentes para o termo SOA, no entanto, todas elas concordam que é um paradigma para melhorar a flexibilidade.

Em computação, o termo “Arquitetura Orientada a Serviços” (SOA) expressa um conceito de arquitetura de *software* que define o uso de serviços para suportar os requisitos dos usuários. Em um ambiente orientado a serviços, os nós de uma rede tornam os recursos disponíveis a outros participantes, na forma de serviços independentes, sendo acessados pelos usuários de maneira padronizada. A maioria das definições de SOA identificam o uso de *Web Services* na sua implementação. Todavia, SOA pode ser implementada com o uso de qualquer tecnologia baseada em serviços (ERL, 2005) e (ERL, 2009).

Diferente das arquiteturas tradicionais, SOA compreende os serviços de aplicação de forma independente e com uma alta capacidade de integrar-se. Esses serviços operam em conjunto livremente de plataforma e das linguagens de programação. A

definição da interface encapsula o fornecedor e a implementação específica da linguagem, tornando-se independente de tecnologia de desenvolvimento, como *Java* e *.NET*. Quando a interface é definida de forma padronizada, os componentes de *software* se tornam bastante reutilizáveis (ERL, 2009).

2.1.1 SOA como paradigma

A arquitetura orientada a serviços, mesmo que não seja uma arquitetura concreta, conduz a uma arquitetura concreta, podendo ser chamada de estilo, paradigma, conceito, perspectiva ou representação, de modo que SOA não é um *framework* ou ferramenta que possa ser comprado ou usado. Na verdade, SOA é uma abordagem, um modo de pensar, um conjunto de valores que leva a certas decisões, quando se projeta uma arquitetura de *software* (JOSUTTIS, 2008).

2.2 CONCEITOS DE SOA

Nesta seção, serão apresentados os principais conceitos técnicos de SOA, permitindo lidar com as características de sistemas.

2.2.1 Serviços

O desenvolvimento de *software* inicia-se com a abstração. Para isso, é necessário abstrair a realidade de tal forma que apenas sejam tratados os aspectos relevantes do problema. Contudo, sabe-se que é possível abstrair de perspectivas diferentes. A arquitetura SOA objetiva abstrair, concentrando-se no aspecto corporativo do problema. O termo fundamental aqui apresentado é “serviço”. Na essência, um serviço é uma representação de alguma funcionalidade de negócio. O objetivo de SOA é estruturar grandes sistemas distribuídos baseados em abstração das regras

e das atividades de negócio, fornecendo uma clara estrutura aos sistemas que são projetados e desenvolvidos (PULIER; TAYLOR, 2008) e (JOSUTTIS, 2008).

Embora eles ainda sejam sistemas técnicos, as interfaces externas devem ser projetadas de tal maneira que as pessoas do negócio possam entendê-las. A consequência inteligente desta abordagem é que, neste nível de abstração, os detalhes específicos da plataforma não importam, de sorte que as plataformas podem ser heterogêneas (JOSUTTIS, 2008).

Como princípio básico, no entanto, pode-se considerar um serviço como sendo uma representação da TI de uma funcionalidade de negócio independente, tal como “cadastrar um funcionário”, “obter contratos de um cliente”, “solicitar extrato de contas”, “calcular a melhor rota para o carro” e assim por diante.

2.2.2 Alta interoperabilidade

O primeiro objetivo, devendo estar presente em sistemas heterogêneos, é a capacidade de se conectar a outros sistemas facilmente, exercendo a característica de “alta interoperabilidade”. A alta interoperabilidade não é uma ideia nova, pois, antes mesmo de SOA, já existia o conceito de “integração de aplicações corporativas” ou *Enterprise Application Integration* (EAI), e, no que diz respeito à interoperabilidade, SOA não é uma novidade (ERL, 2005), (PULIER; TAYLOR, 2008) e (JOSUTTIS, 2008).

Para SOA, a alta interoperabilidade é o começo e não o fim, sendo a base a partir da qual é possível implementar as funcionalidades de negócio (serviços) e compartilhar pelos múltiplos sistemas distribuídos.

2.2.3 Acoplamento fraco

Durante um projeto, nem sempre é possível ter tempo para analisar, modelar e implementar com os cuidados necessários, pois o mercado exige respostas rápidas

e, normalmente, a flexibilidade é mais valorizada do que a qualidade, levando a diversos problemas.

Considere-se o fato de que, cada vez mais, sistemas são integrados e novos processos de negócio são implementados, distribuindo-os pelas diferentes aplicações. A princípio, os dados fluem de tal forma que os processos são executados com sucesso em todos os sistemas afetados, porém, com tudo isso, o menor dos problemas pode parar todo o negócio. Isso tem que ser evitado, fornecendo-se mecanismos de tolerância a falhas para os sistemas integrados (JOSUTTIS, 2008).

A chave para alcançar os objetivos de flexibilidade, escalabilidade e tolerância a falhas é fazer uso do acoplamento fraco. O acoplamento fraco é o conceito de minimizar as dependências. Quando as dependências estão minimizadas, as modificações têm os efeitos minimizados e os sistemas ainda executam, mesmo quando partes deles estão quebradas ou indisponíveis.

Além disso, o acoplamento fraco leva à escalabilidade, fazendo com que grandes sistemas tendam a desafiar os limites. Todos os grandes sistemas funcionam apenas se os negócios comuns puderem ser feitos de forma mais descentralizada possível. Uma maneira de introduzir o acoplamento fraco é evitar utilizar mais centralização do que o necessário (JOSUTTIS, 2008).

2.3 FUNDAMENTOS DE *DESIGN*

Antes de começar a explorar os detalhes da computação orientada a serviços, é preciso estabelecer uma terminologia básica de *design*. Alguns dos termos relacionados a *design* são: característica, princípio, paradigma, modelo, linguagem de modelo, padrão e boa prática.

2.3.1 Característica

Uma característica de algo é apenas um atributo ou uma qualidade. Em uma solução de negócios automatizada, existem inúmeras características específicas, estabelecidas durante seu projeto inicial. O tipo das características de *design*, consequentemente, é uma qualidade ou um atributo específico de um corpo da lógica que se documenta em uma especificação, de modo a ser entendida no desenvolvimento (ERL, 2009).

A orientação a serviços enfatiza a criação de características bem específicas do *design* e, ao mesmo tempo, relega outras a um segundo plano. É importante observar que praticamente cada característica de *design* que se explora é atingível até certo ponto, significando que, em geral, não se trata de a solução lógica possuir ou não certa característica; a questão, quase sempre, é em que medida uma característica pode ou deve ser descrita (ERL, 2005), (JOSUTTIS, 2008) e (ERL, 2009).

Embora cada sistema possa dispor de suas próprias características, o interesse é principalmente no estabelecimento de características comuns de *design*, assegurando um grau maior de coerência, tornando mais parecidos diferentes tipos de lógica. Quando as coisas são parecidas, elas ficam mais previsíveis. As características previsíveis de um *design* levam a um comportamento previsível. Isso, por sua vez, conduz a maior confiabilidade e à oportunidade de tirar proveito da lógica de diferentes maneiras (ERL, 2009).

A Figura 1 ilustra três *designs* distintos de aplicativo (A, B e C). Cada pequeno quadrado representa uma unidade de solução lógica; as flechas sólidas, o reuso ou o acesso compartilhado; e as flechas tracejadas, a transferência de dados de estado.

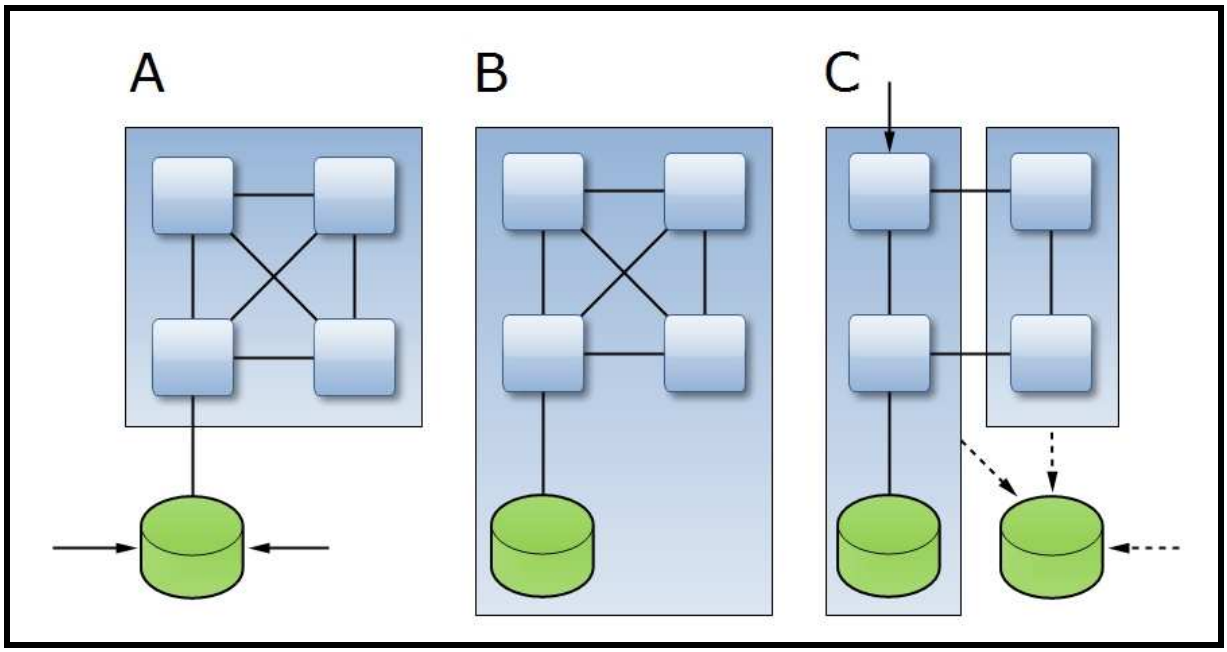

Figura 1 – Característica de *Design* (In: ERL, 2009)

Na Tabela 1, são apresentadas as características de *design*.

Aplicativo A	Aplicativo B	Aplicativo C
• Componentizado • Fortemente acoplado • Banco de dados compartilhado • Dependência de estado moderada	• Componentizado • Fortemente acoplado • Banco de dados dedicado • Dependência de estado alta	• Componentizado e distribuído • Fracamente acoplado • Reuso visado • Banco de dados dedicado • Dependência de estado mínima (adiamento de estado via banco de dados de estado externo compartilhado)

Tabela 1 – Característica de *Design* (In: ERL, 2009)

2.3.2 Princípio

Um princípio é uma prática generalizada e aceita pelo mercado. Em outras palavras, é algo que os outros fazem ou promovem em associação com um objetivo comum. Pode-se comparar um princípio com uma boa prática, pelo fato de que ambos propõem um meio de alcançar algo com base na experiência ou na aceitação por todo o mercado (ERL, 2005) e (ERL, 2009).

Quando chega o momento de construir soluções, um princípio de *design* representa um conceito altamente recomendável para dar forma à lógica de solução, da maneira certa e com os objetivos certos.

Por exemplo, pode-se ter um princípio afirmando que a lógica deve ser distribucional. Aplicar esse princípio resulta na partilha lógica em unidades individualmente distribucionais. Assim se estabelece a característica distinta do *design* da lógica, que passa a ser formada por componentes. Isso não é apenas o exemplo de um princípio de *design* muito amplo, mas é, também, o ponto de partida para a orientação a serviços (ERL, 2009).

Os princípios de *design* fornecem regras e diretrizes que ajudam a determinar exatamente como a lógica deve ser decomposta e modelada em unidades distribucionais.

A aplicação repetida dos princípios de *design* aumenta a quantidade de características comuns do *design*. Na Figura 2, o acoplamento entre as unidades de lógica A e B foi diminuído, como indicado por uma redução dos pontos de conexão.

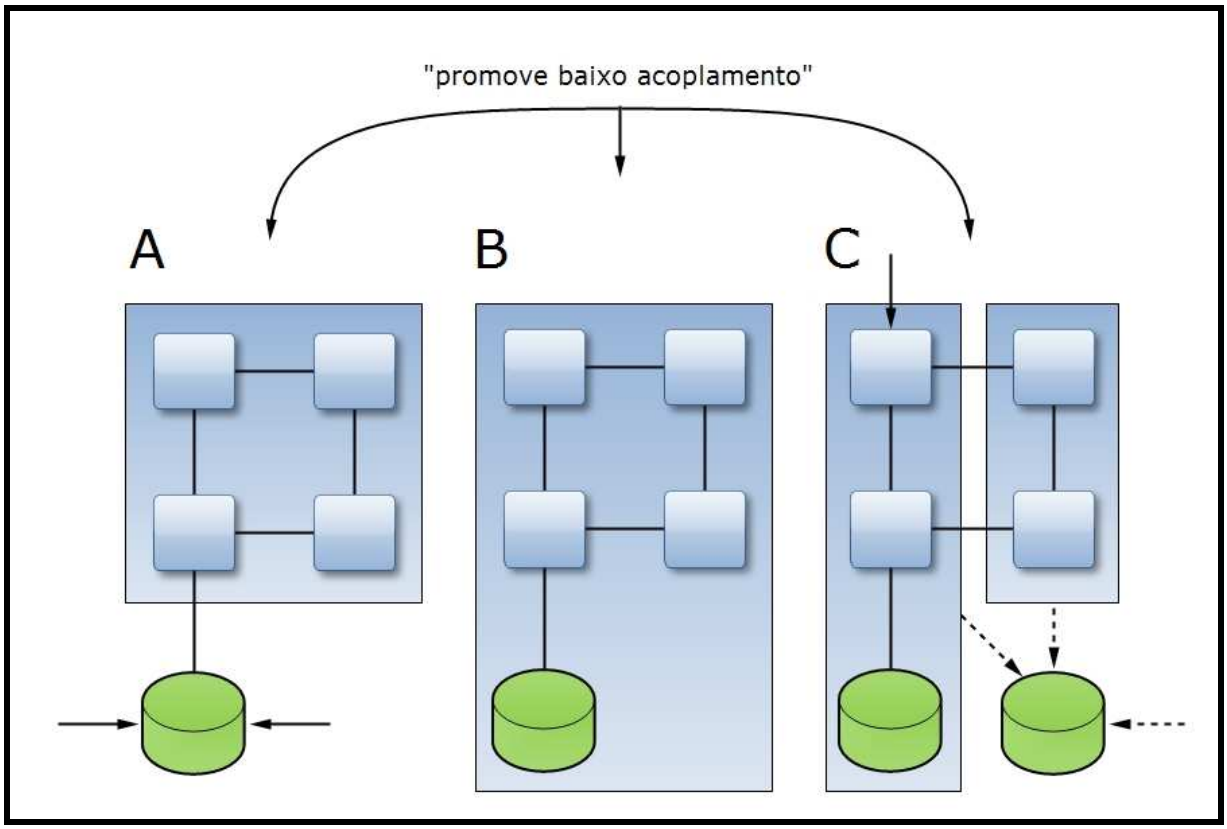

Figura 2 – Princípio de *Design* (In: ERL, 2009)

2.3.3 Paradigma

Há muitos significados associados ao termo “paradigma”, que pode ser uma abordagem de algo ou um conjunto combinado de regras aplicadas dentro de um limite predefinido.

Um paradigma de *design*, no contexto da automação de negócios, geralmente, é considerado uma abordagem que rege o *design* da lógica. Esse paradigma consiste em um conjunto de regras ou princípios complementares que definem a abordagem ampla representada pelo paradigma (ERL, 2009).

A orientação a objetos (ou *design* orientado a objetos) é um exemplo clássico de um paradigma aceito. Ela fornece um conjunto de princípios que modela, de determinada maneira, a lógica formada por componentes, a fim de que seja possível alcançar um conjunto específico de objetivos.

Como ocorre com a orientação a objetos, a orientação a serviços é um paradigma que se aplica à lógica distribuída; contudo, seus princípios se diferenciam daqueles associados à orientação a objetos, resultando na criação de diferentes tipos de características do *design* (ERL, 2009).

Como um paradigma de *design* representa uma coleção dos princípios de *design*, o grau de semelhança em todos os diferentes corpos da lógica é aumentado ainda mais. Na Figura 3, a quantidade de reuso em A e B aumentou.

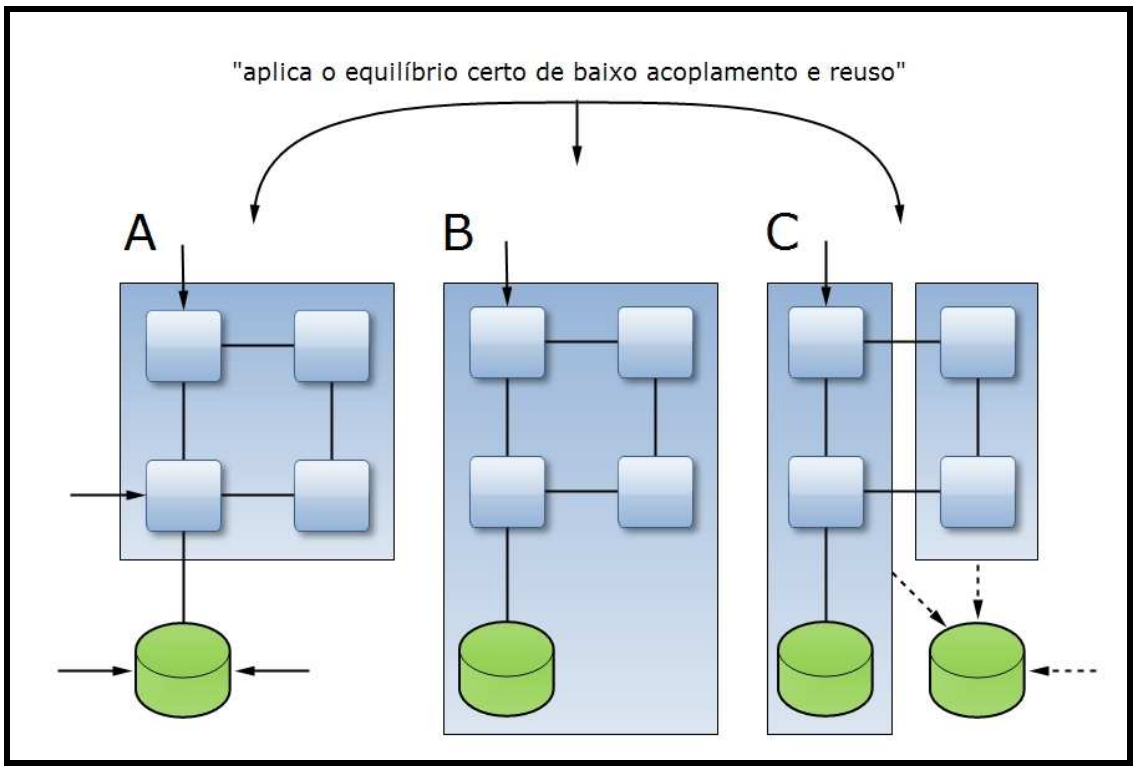

Figura 3 – Paradigma de *Design* (In: ERL, 2009)

2.3.4 Modelo

A orientação a serviços é um paradigma de *design* que abrange um conjunto de princípios de *design*, cada um dos quais fornece uma regra ou diretriz generalizada, para entender certas características de *design*. Apesar de o próprio paradigma ser

bem completo, aplicá-lo com sucesso no dia-a-dia requer mais do que uma simples compreensão teórica de seus princípios.

Segundo (ERL, 2009), os *designers* de serviços enfrentam desafios, ao tentar aplicar um paradigma de *design* a situações cotidianas. Isso ocorre porque a compreensão das características desejadas do *design* costuma ser complicada por vários fatores, incluindo:

- Restrições impostas pela tecnologia que é utilizada para construir ou hospedar as unidades lógicas;
- Restrições impostas pela tecnologia ou por sistemas que residem junto às unidades lógicas implementadas da lógica;
- Restrições impostas pelos requisitos e prioridades do projeto que entrega as unidades de lógica.

Um modelo de *design* descreve um problema comum e fornece uma solução correspondente. Essencialmente, o modelo documenta a solução no formato de um modelo genérico, a fim de que possa ser aplicado repetidamente. Os conhecimentos dos modelos de *design* não apenas fornecem uma compreensão dos potenciais problemas que podem aparecer nos *designs*, mas também disponibilizam respostas sobre como lidar melhor com tais dificuldades (ERL, 2005) e (ERL, 2009).

Os modelos oferecem soluções recomendáveis a problemas comuns de *design*. Na Figura 4, um modelo sugere que se reduza o acesso externo a um banco de dados compartilhado, para aumentar a autonomia do aplicativo.

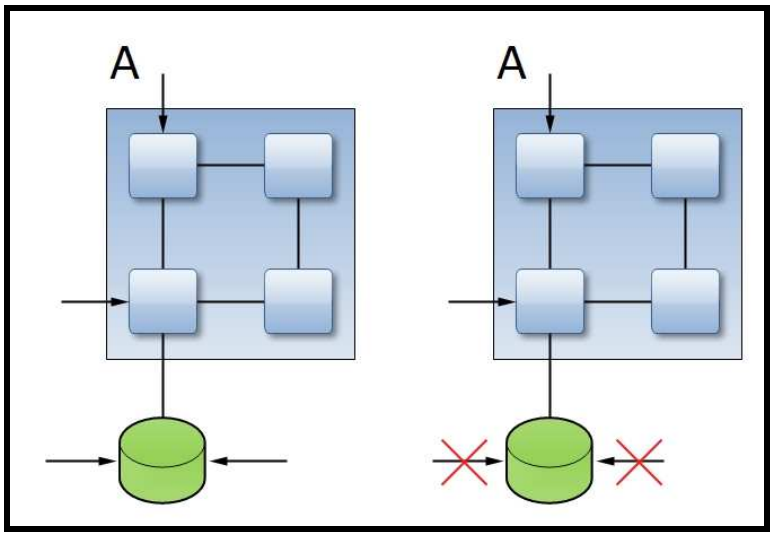

Figura 4 – Modelo de *Design* (In: ERL, 2009)

Os modelos de *design* nascem da experiência, sendo necessário passar por ciclos de tentativas e erros para aprender com base naquilo que não funcionou e, por fim, desenvolver abordagens que tornam possível alcançar seus objetivos. Quando um problema e a solução correspondente forem identificados como suficientemente comuns, a base de um modelo de *design* é formada. Os modelos de *design* podem ser combinados em modelos compostos, que resolvem problemas maiores, e uma série de modelos pode formar a base de uma linguagem de modelo (ERL, 2009).

2.3.5 Linguagem de modelo

A aplicação de um modelo de *design* pode levantar novas questões ou problemas para os quais, talvez, seja necessário adotar outro padrão. Uma coleção de modelos relacionados pode estabelecer uma expressão formalizada de um processo de *design*, pelo qual cada um resolve uma questão primária de decisão. A combinação de modelos, dessa maneira, forma a base de uma linguagem de modelo (ERL, 2009).

Essencialmente, uma linguagem de modelo abrange uma série de modelos de *design* relacionados, que estabelecem uma sequência configurável à qual os

modelos podem ser aplicados. Ela proporciona um meio altamente eficaz de comunicar aspectos fundamentais de uma dada abordagem de *design*, pois fornece uma documentação detalhada de cada passo principal no processo que modela as características da solução lógica (ERL, 2005) e (ERL, 2009).

Na Figura 5, a lógica no *design* do aplicativo B é decomposta como resultado de um padrão e, por conseguinte, decomposta ainda mais como resultado de outro modelo. Os próximos modelos continuam a modelar a lógica.

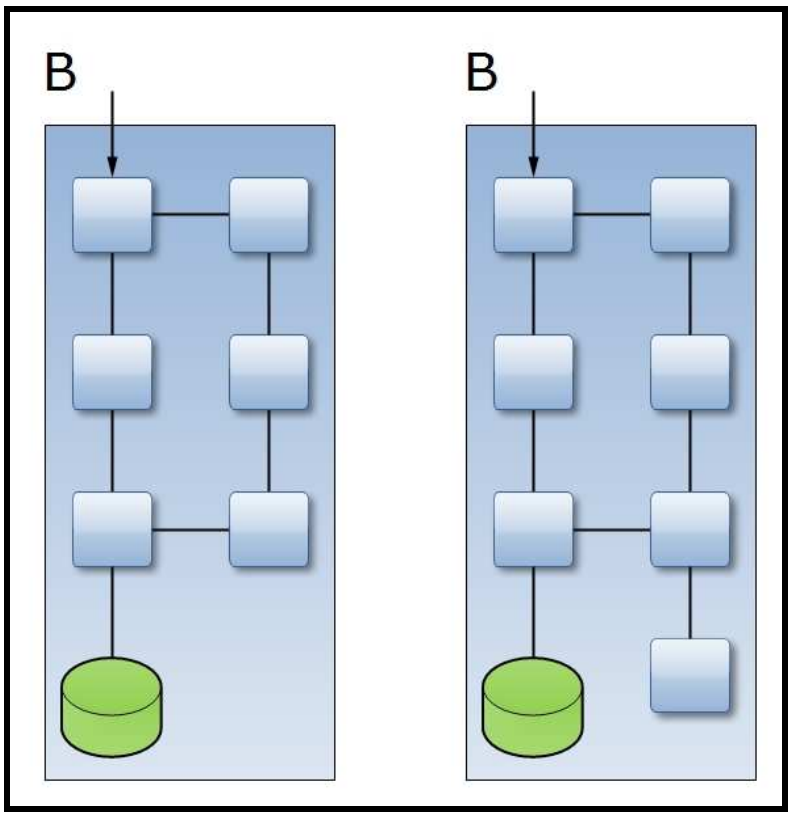

Figura 5 – Linguagem de Modelo de *Design* (In: ERL, 2009)

2.3.6 Padrão

Para que uma organização aplique com sucesso um paradigma de *design*, é necessário mais do que obediência aos princípios associados de *design* e conhecimento dos padrões de suporte de *design* (ERL, 2009). Cada organização

tem objetivos estratégicos e ambientes corporativos exclusivos, que formam um conjunto distinto de requisitos e restrições que precisa ser acomodado dentro dos *designs* da solução (ERL, 2005).

Padrões de *design* são convenções de *design* personalizadas, cuja função é possibilitar que as características possam ser predeterminadas no suporte aos objetivos organizacionais para ambientes corporativos específicos (ERL, 2009). Por meio do uso de padrões internos de *design*, as organizações podem entregar, com consistência, soluções personalizadas para seus ambientes, recursos, objetivos e prioridades, como demonstrado na Figura 6, em que um padrão de *design* requer que o *design* original do aplicativo, no caso C, seja alterado para remover o acesso a um banco de dados compartilhado e externo, por conta de requisitos específicos de segurança e privacidade.

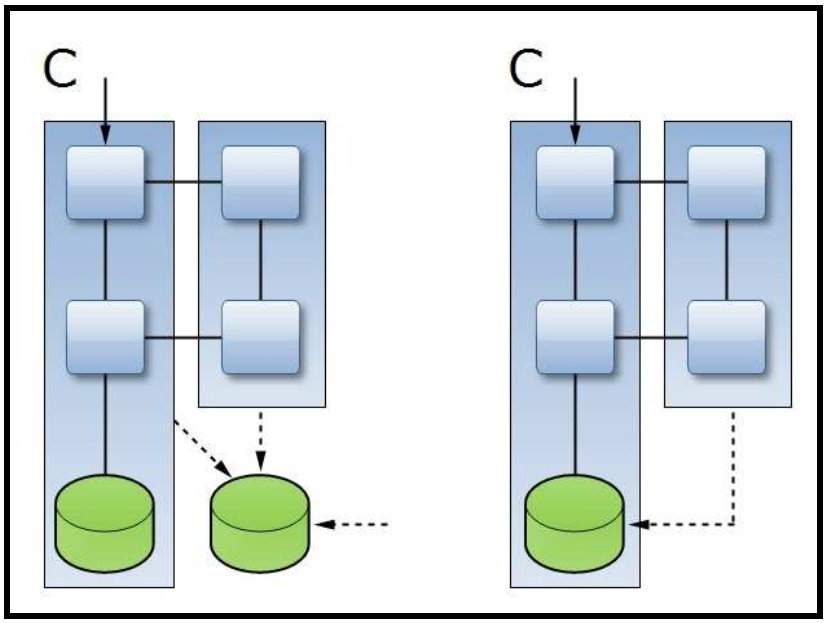

Figura 6 – Padrão de *Design* (In: ERL, 2009)

Como ocorre com os princípios de *design*, a aplicação de padrões de *design* resulta na criação de características específicas de *design*. Como acontece com os modelos de *design*, os padrões de *design* refinam essas características, a fim de evitar problemas e fortalecer o *design* geral da solução. Na realidade, é recomendável que

os padrões de *design* se baseiem ou mesmo derivem dos princípios e modelos de *design* do mercado (ERL, 2009).

2.3.7 Boa prática

Uma boa prática, geralmente, é considerada uma técnica ou abordagem para resolver ou evitar certos problemas. Ela, normalmente, consiste em uma prática que tem o reconhecimento do mercado e que surgiu da experiência.

Uma boa prática se diferencia de um princípio de *design*, pois o princípio está limitado apenas ao *design*, enquanto uma boa prática pode se relacionar a algo desde a entrega do projeto até questões organizacionais, de governança ou de processos (ERL, 2009).

As boas práticas fornecem uma orientação na forma de “lições aprendidas”. Na Figura 7, é sugerido que a manutenção contínua das unidades reusáveis da solução lógica de todos os aplicativos esteja sob a responsabilidade de uma única pessoa.

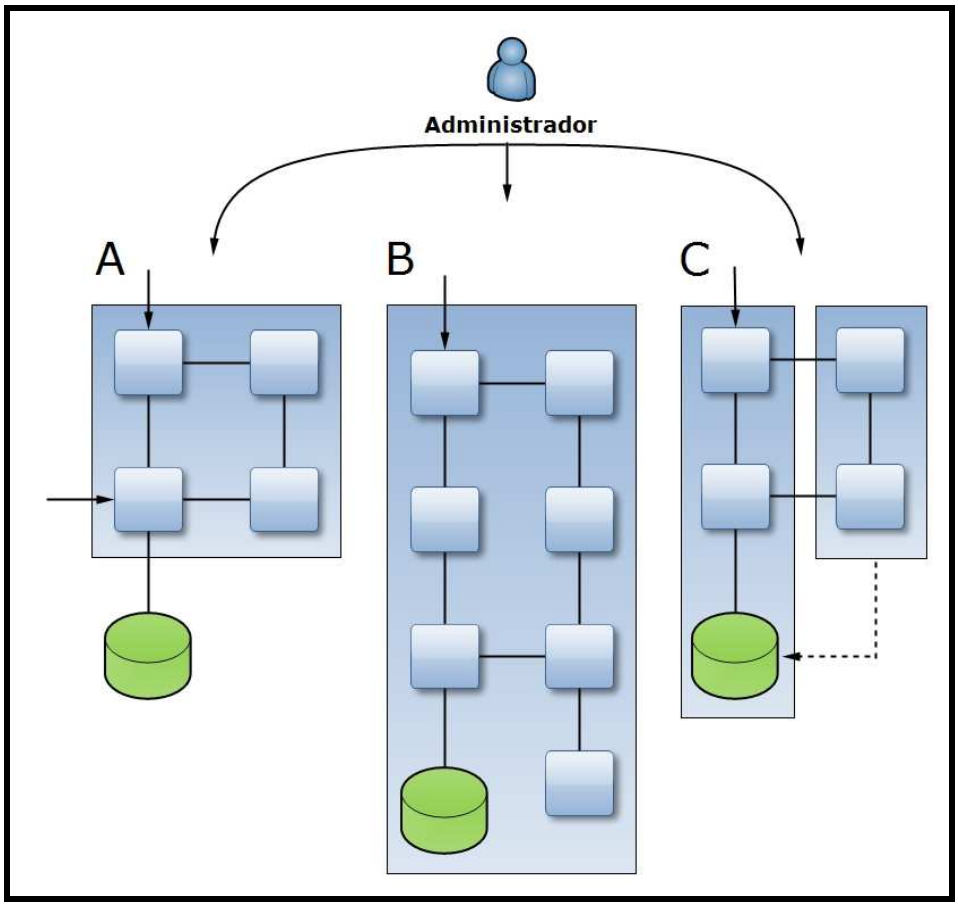

Figura 7 – Boa Prática de *Design* (In: ERL, 2009)

2.3.8 Um *framework* de *design* fundamental

Cada uma das seções anteriores descreveu uma parte da inteligência que pode atuar como uma entrada para o processo de um *design*. É, portanto, importante entender como elas se relacionam entre si, para que se possa ter uma previsão de como e onde elas são mais bem utilizadas.

A Figura 8 mostra como algumas partes mais comuns de um *framework* de *design* costumam se relacionar, destacando a importância que o uso dos princípios de *design* pode ter, além de fornecer uma “dica” de como algumas partes do *framework* básico de *design* podem se relacionar entre si.

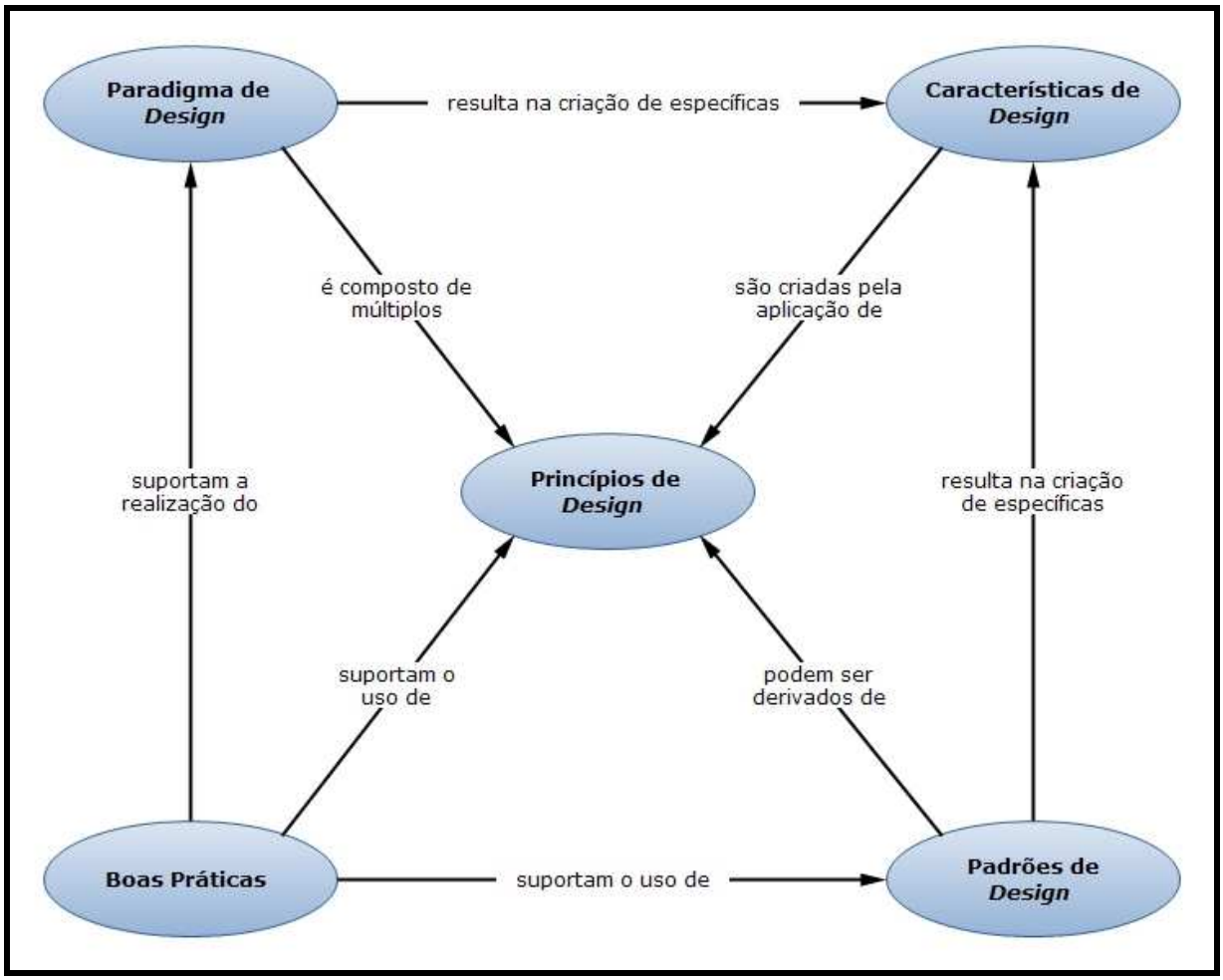

Figura 8 – *Framework* de *Design* Fundamental (In: ERL, 2009)

Na Figura 9, é possível entender essa perspectiva, já que se ilustra como o uso dos modelos de *design* pode suportar e estender ainda mais um *framework* básico de *design*.

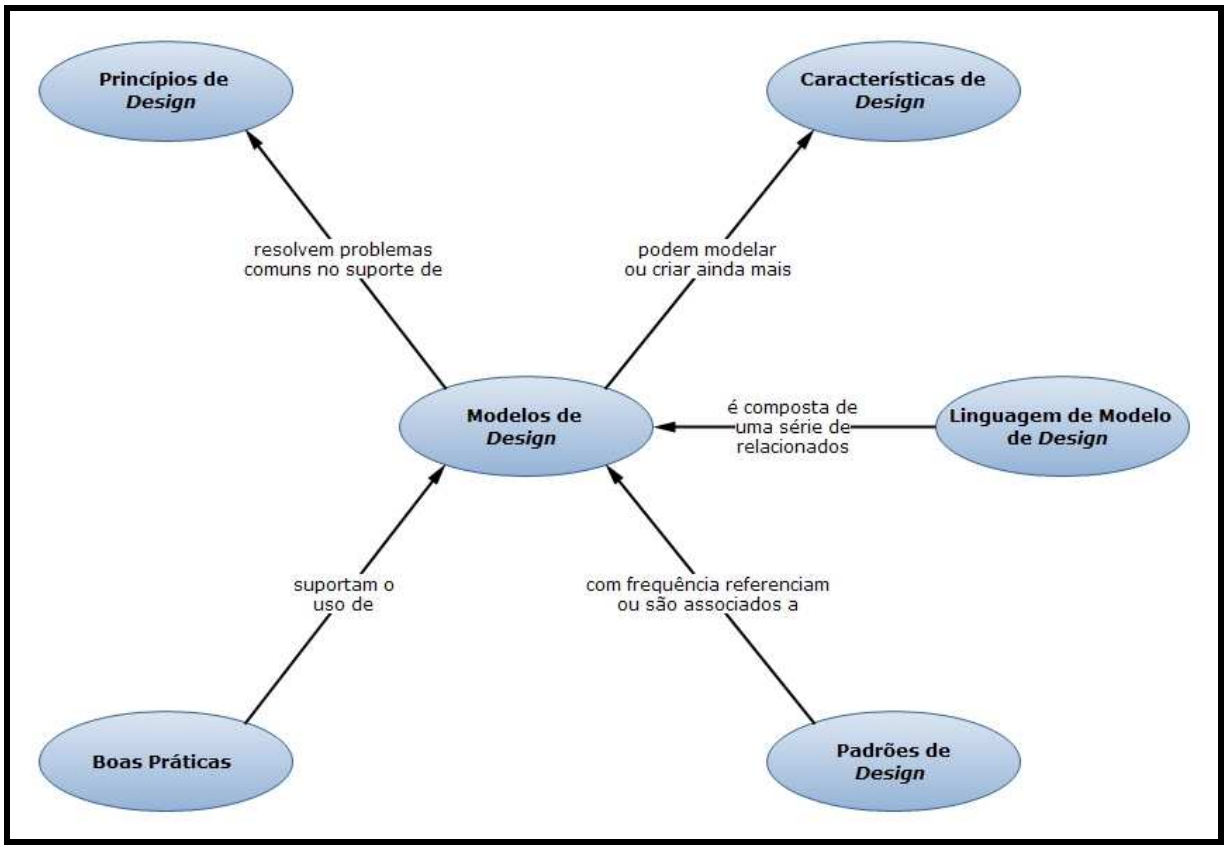

Figura 9 – *Framework* de *Design* Fundamental (In: ERL, 2009)

Por fim, a Figura 10 mostra como as partes do *framework* de um *design* podem, em última análise, ajudar a compreender a aplicação de um paradigma de *design* abrangente.

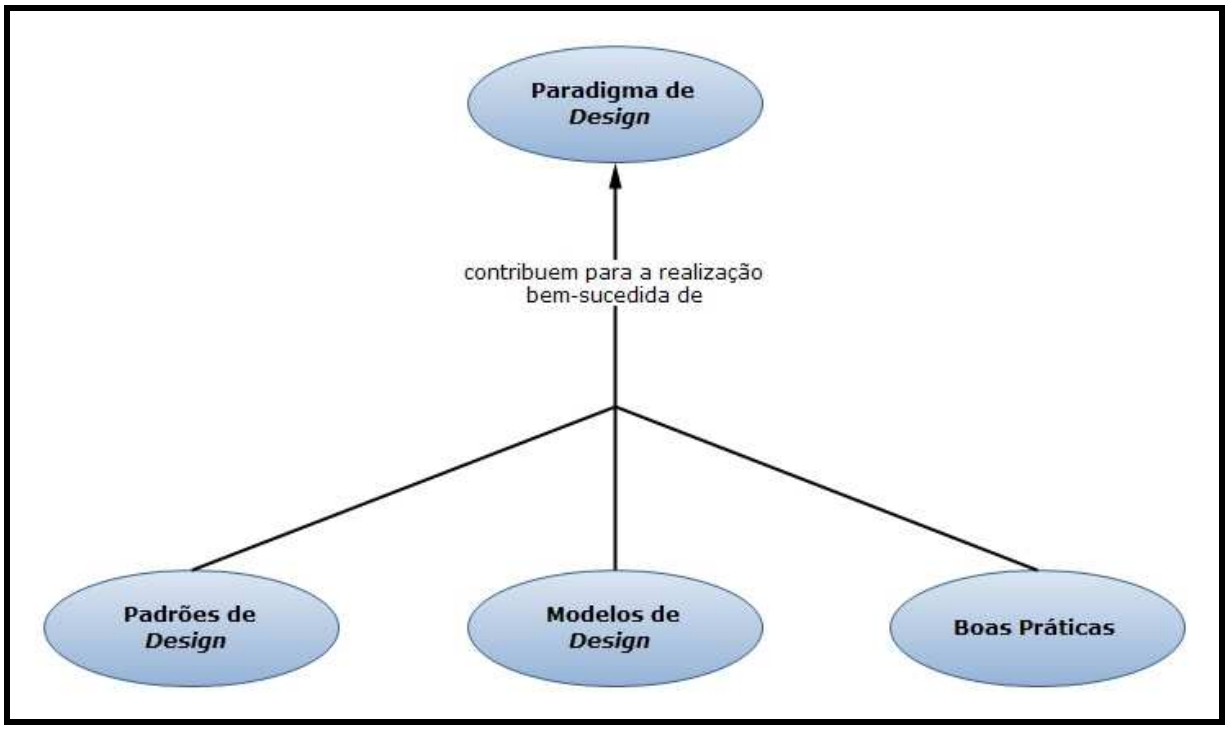

Figura 10 – *Framework* de *Design* Fundamental (In: ERL, 2009)

2.4 INTRODUÇÃO À COMPUTAÇÃO ORIENTADA A SERVIÇOS

A computação orientada a serviços representa uma nova geração da plataforma da computação distribuída, abrangendo muitos conceitos, incluindo seu próprio paradigma de *design*, princípios, catálogo de modelos, linguagens-padrão, um modelo arquitetônico, conceitos, tecnologias e *frameworks* relacionados.

Com a orientação a serviços, as antigas plataformas da computação distribuída foram aprimoradas e novas camadas adicionadas ao *design*. Por isso, deve-se investir o tempo necessário para assimilar sua estrutura, antes de ir em frente com o *design* real e as fases de construção do projeto distribuído (ERL, 2009).

Para entender melhor a complexidade fundamental de uma típica plataforma de computação orientada a serviços, é preciso descrever cada uma de suas partes primárias, as quais serão referidas como elementos: arquitetura orientada a serviços, orientação a serviços, lógica orientada a serviços, serviços, composições de serviços e inventário de serviços. Esses elementos serão descritos

detalhadamente, e será feita também uma abordagem de como eles podem se relacionar.

2.4.1 Arquitetura orientada a serviços

A SOA estabelece um modelo arquitetônico que visa a aprimorar a eficiência, a agilidade e a produtividade de uma empresa, posicionando os serviços como os principais meios para que a solução lógica seja representada (PULIER; TAYLOR, 2008) e (JOSUTTIS, 2008).

Em essência, a plataforma de computação orientada a serviços tem a ver com o paradigma de *design* da orientação a serviços e seu relacionamento com a arquitetura orientada a serviços. Na realidade, o termo “arquitetura orientada a serviços” e sua sigla associada são empregados tão amplamente pela mídia e na literatura de *marketing* dos fornecedores, que se tornou quase um sinônimo para a própria computação orientada a serviços (ERL, 2005) e (ERL, 2009).

Como forma de arquitetura de tecnologia, uma implementação SOA pode consistir em uma combinação de tecnologias, produtos, APIs e várias outras partes.

A face real de uma arquitetura orientada a serviços implementada é caracterizada pela introdução de novas tecnologias e plataformas que suportam especificamente a criação, a execução e a evolução das soluções orientadas a serviços (ERL, 2009).

2.4.2 Orientação a serviços

A orientação a serviços é um paradigma de *design* que abrange um conjunto específico de princípios de *design*. A aplicação desses princípios ao *design* da lógica resulta em uma lógica orientada a serviços. A unidade mais fundamental da lógica orientada a serviços é o serviço (ERL, 2009).

Os serviços existem como programas de *software* independentes, com características de *design* distintas, que dão suporte à obtenção dos objetivos

estratégicos associados à computação orientada a serviços. Cada serviço recebe seu próprio contexto funcional e possui um conjunto de capacidades relacionadas a esse contexto. Essas capacidades adequadas para a invocação por programas externos são comumente expressas via um contrato de serviços públicos, quase como uma API tradicional (PULIER; TAYLOR, 2008), (JOSUTTIS, 2008) e (ERL, 2009).

A Figura 11 apresenta o símbolo usado neste trabalho, para representar um serviço.

Figura 11 – Símbolo de Serviço (In: ERL, 2009)

2.4.3 Composição de serviços

Uma composição de serviços consiste em um conjunto coordenado de serviços, e é comparável a um aplicativo tradicional, pois seu escopo funcional normalmente se associa à automação de um processo de negócio da empresa (JOSUTTIS, 2008) e (ERL, 2009).

A Figura 12 demonstra a composição de três serviços, representada por três esferas conectadas.

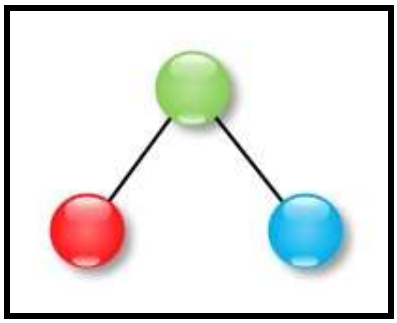

Figura 12 – Composição de Serviços (In: ERL, 2009)

A aplicação coerente dos princípios da orientação a serviços leva à criação de serviços com um contexto funcional independente de qualquer processo de negócio. Esses serviços independentes são, portanto, capazes de participar de múltiplas composições de serviços. A capacidade de um serviço ser natural e, repetidamente, parte de uma composição é fundamental à obtenção de vários objetivos estratégicos da computação orientada a serviços (ERL, 2009).

2.4.4 Inventário de serviços

Um inventário de serviços é uma coleção padronizada e governada de maneira independente dos serviços que se complementam, dentro de um limite que representa uma empresa ou um segmento significativo de uma empresa. A Figura 13 estabelece o símbolo utilizado neste projeto, para representar um inventário de serviço (JOSUTTIS, 2008) e (ERL, 2009).

Figura 13 – Inventário de Serviços (In: ERL, 2009)

2.4.5 Compreendendo os elementos da computação orientada a serviços

Compreender os elementos da computação orientada a serviços individualmente é tão importante quanto entender como eles podem se relacionar entre si, uma vez que esses relacionamentos estabelecem alguns dos conceitos mais fundamentais

da computação orientada a serviços. Para isso, esses elementos serão enfatizados, explicando-se o modo como cada um se associa aos demais: (JOSUTTIS, 2008), (ERL, 2005) e (ERL, 2009)

- A arquitetura orientada a serviços representa uma forma distinta da arquitetura de tecnologia projetada no suporte à lógica orientada a serviços, que é formada por serviços e composições de serviço modeladas e projetadas em conformidade com a orientação a serviços;
- A orientação a serviços é um paradigma de *design* que abrange os princípios de *design* da orientação a serviços. Quando aplicados a unidades de lógica, esses princípios criam serviços com características distintas de *design*, fornecendo suporte aos objetivos gerais e à visão da computação orientada a serviços;
- A computação orientada a serviços representa uma nova geração da plataforma de computação, abrangendo o paradigma da orientação a serviços e a arquitetura orientada a serviços, com o objetivo fundamental de criar e montar um ou mais inventários de serviços.

A Figura 14 ilustra detalhadamente os relacionamentos entre os elementos, fornecendo uma visão conceitual de como esses elementos da computação orientada a serviços podem se relacionar.

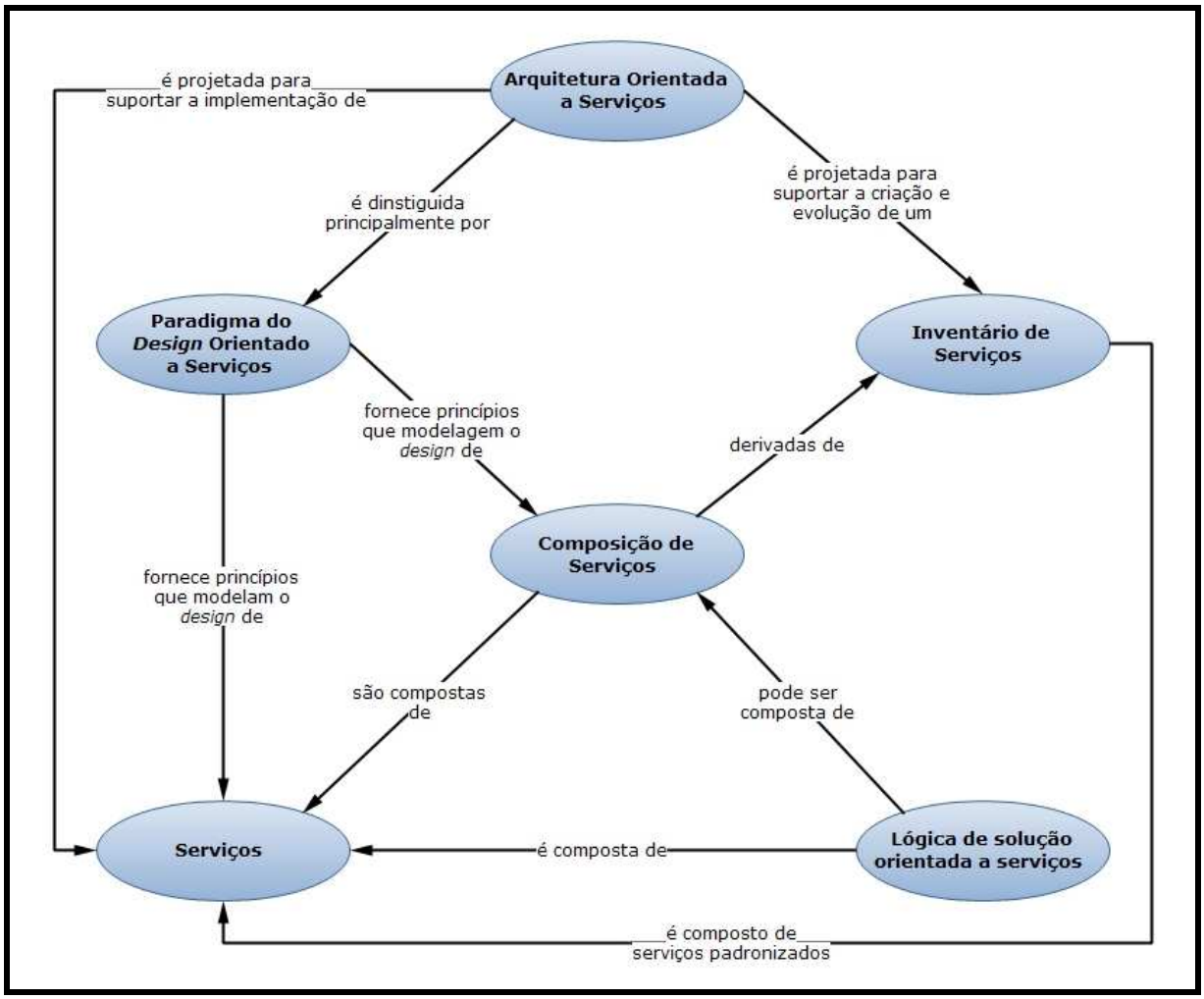

Figura 14 – Computação Orientada a Serviços (In: ERL, 2009)

Para entender totalmente como esses elementos são utilizados, torna-se necessário explorar a maneira como eles podem ser convertidos em exemplos práticos. Para fazer isso, é preciso distinguir com clareza o papel e a posição de cada elemento dentro de uma perspectiva implementação, como descrito a seguir: (ERL, 2005) e (ERL, 2009)

- A lógica orientada a serviços é implementada como serviços e composições de serviços projetados, conforme os princípios de *design* da orientação a serviços;

- Uma composição de serviços é composta de serviços montados, a fim de fornecer as funcionalidades requeridas para automatizar uma tarefa ou um processo específico de negócios;
- Como a orientação a serviços modela muitos serviços independentes, um serviço pode ser invocado por vários programas para o consumidor, e cada um deles pode envolver esse mesmo serviço em uma composição de serviços diferente;
- Uma coleção de serviços padronizados pode formar a base de um inventário de serviços, a qual pode ser administrada de maneira independente, dentro de um ambiente de implementação;
- Diversos processos de negócio podem ser automatizados pela criação de composições de serviços desenhadas a partir de uma série de serviços independentes existentes, que residam em um inventário de serviços;
- A arquitetura orientada a serviços é uma forma de arquitetura de tecnologia otimizada para o suporte a serviços, composições e inventários de serviços.

A Figura 15 ilustra como um inventário de serviços estabelece um grupo de serviços, muitos dos quais são projetados para serem reusados dentro de várias composições de serviços.

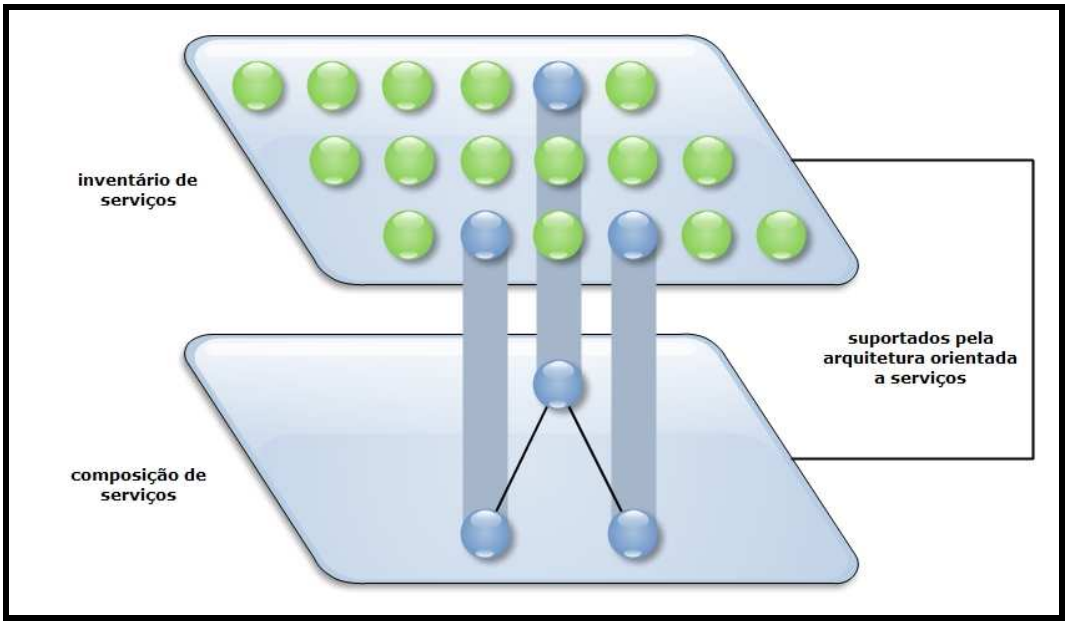

Figura 15 – Inventário e Composição de Serviços (In: ERL, 2009)

3 – *WEB SERVICES*

É muito importante visualizar a arquitetura orientada a serviços como modelo arquitetônico que seja independente a qualquer plataforma de tecnologia. Fazendo isso, uma empresa tem a liberdade de buscar constantemente os objetivos estratégicos associados à computação orientada a serviços, tirando proveito dos futuros avanços da tecnologia. No mercado atual, a plataforma de tecnologia mais associada à SOA é a de *Web Services*.

3.1 INTRODUÇÃO À *WEB SERVICES*

Os *Web Services* são baseados no conceito da arquitetura orientada a serviços, que é a última evolução da computação distribuída, tornando possível que componentes de *software*, incluindo funcionalidades de aplicações, objetos e processos de diferentes sistemas, sejam expostos como serviços (SUN MICROSYSTEMS, 2003).

Conceitualmente, um *Web Service* provê um meio alternativo de expor a lógica de uma aplicação para um conjunto de clientes heterogêneos, através de interfaces programáveis e protocolos de rede, criando caminhos para encontrar, descrever e invocar esses serviços. Por serem construídos de maneira padronizada, clientes que executam diferentes aplicações são capazes de utilizar esses serviços, independentemente de detalhes de implementação ou ambientes de execução (SUN MICROSYSTEMS, 2003) e (NEWCOMER, 2002).

Baseados no padrão XML (*Extensible Markup Language*), os *Web Services* podem ser desenvolvidos como componentes de aplicação encapsulados, empregando qualquer linguagem de programação, qualquer protocolo ou qualquer plataforma (SUN MICROSYSTEMS, 2003).

Há cinco elementos básicos que compõem um *Web Service*: serviços, registro de serviços, cliente ou consumidor de serviços, protocolo de transferência e mensagens baseadas em XML (SUN MICROSYSTEMS, 2003) e (ERL, 2005).

A Figura 16 ilustra o relacionamento entre os elementos básicos de um *Web Service*.

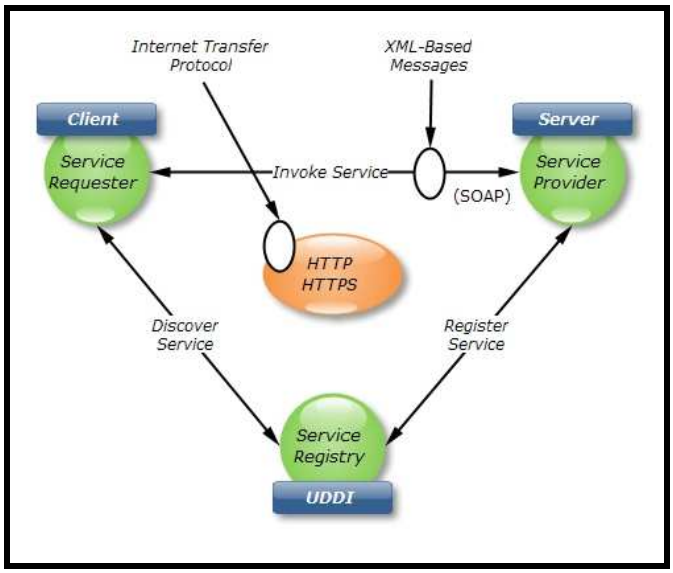

Figura 16 – Elementos básicos de um *Web Service* (In: ERL, 2005)

A Tabela 2 descreve os cinco elementos, que, executados como um todo, definem o modelo *Web Service*.

Elemento	Descrição
Serviço	Uma função de aplicação ou um processo acessível da lógica de negócios
Registro de Serviços	Um local onde é possível localizar informações sobre serviços disponíveis, não sendo obrigatório consultá-lo para consumir um *Web Service*
Cliente ou consumidor de serviços	Um cliente que acessa um serviço
Mensagens baseadas em XML	Informações transmitidas entre clientes e serviços para acessar um serviço e retornar o resultado de um processo
Protocolo de transferência	Um protocolo de comunicação, como HTTP ou *Hypertext Transport Protocol Secure* (HTTPS), usado entre um cliente e um serviço

Tabela 2 – Descrição de elementos básicos de um *Web Service* (In: *Sun Microsystems*, 2003)

A Figura 17 mostra um exemplo de como um serviço de reserva de viagens pode ser disponibilizado em forma de serviços com *Web Services*.

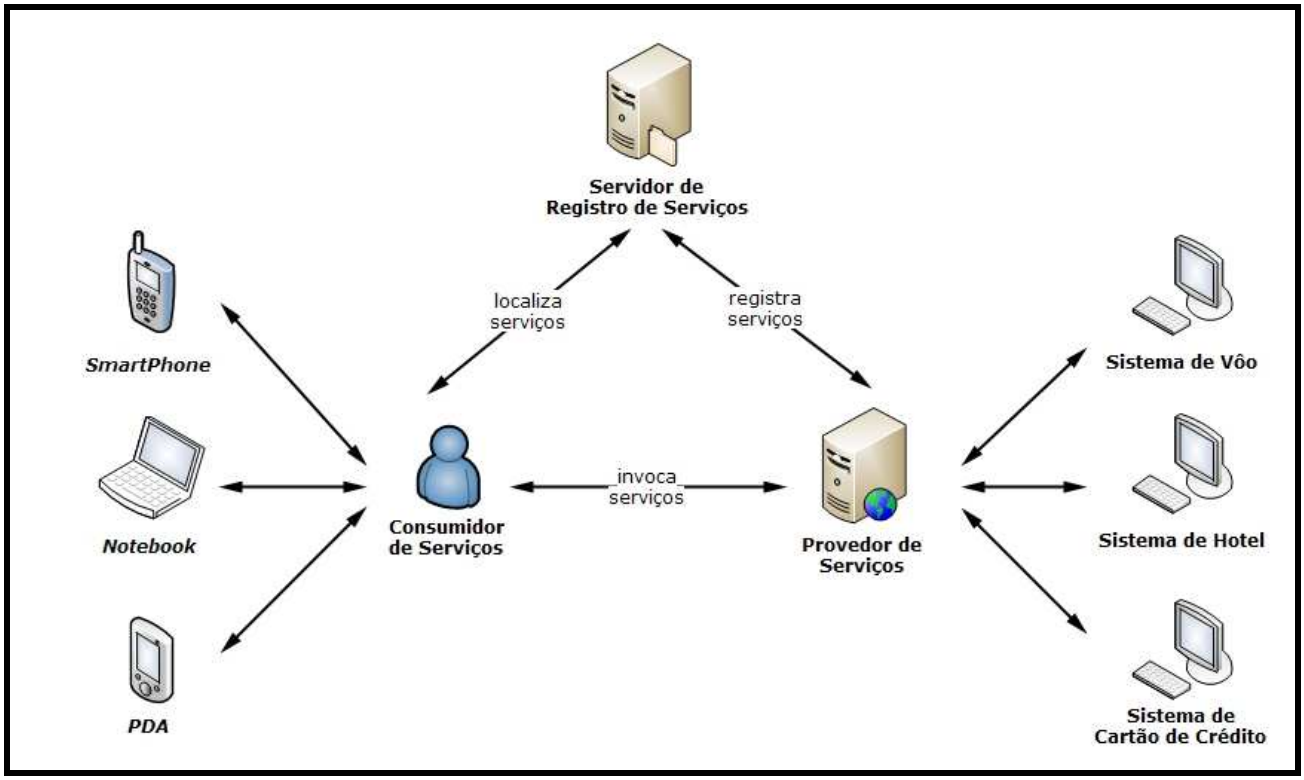

Figura 17 – Exemplo de Cenário *Web Service* (In: NAGAPPAN et al, 2003)

No exemplo da Figura 17, existem sistemas expostos como serviços baseados na arquitetura SOA que são visíveis e alcançáveis por diversas aplicações e dispositivos. O cenário pode ser dividido em etapas, como segue:

- O provedor de serviços contém *Web Services* expostos a partir de aplicações já existentes, obtidas de diversos locais, como sistema de reserva de vôo e pagamento por cartão de crédito;
- O servidor de registro de serviços armazena as informações de registros públicos e privados sobre os serviços expostos pelo provedor de serviços;
- O consumidor de serviços localiza um *Web Service* por meio de um sistema de busca ou diretamente por um registro de serviço e o invoca, utilizando qualquer dispositivo de comunicação com um *software* capaz de interpretar e

consumir o serviço descrito com XML, além da independência de linguagem de programação, tecnologia e plataforma.

Esse exemplo provê um simples cenário de como as funcionalidades dos negócios da organização podem ser expostos como *Web Services* e invocados pelos consumidores ou usuários, utilizando uma gama de diferentes aplicações como clientes.

A Figura 18 mostra soluções orientadas a serviços, exemplificando ambientes distribuídos que podem ser compostos de serviços construídos por *Web Services*, componentes ou combinações de ambos (ERL, 2009).

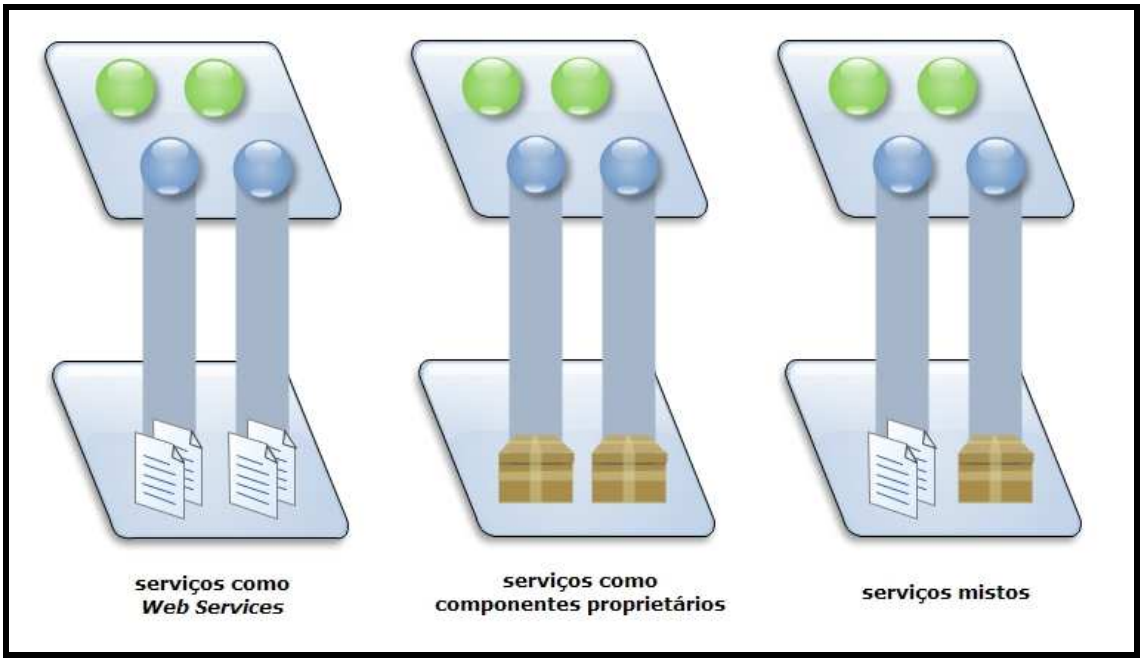

Figura 18 – Soluções orientadas a serviços (In: ERL, 2009)

3.2 PADRÃO DE *WEB SERVICES*

A plataforma de *Web Services* é definida por vários padrões da indústria, podendo ser distribuída em duas gerações claramente identificáveis, cada uma associada a uma coleção de padrões e especificações: (ERL, 2005), (ERL, 2009) e (SUN MICROSYSTEMS, 2003).

- **Plataforma de *Web Services* de primeira geração**
    - A plataforma original da tecnologia de *Web Services* é composta das principais especificações e tecnologias abertas a seguir: *Web Services Description Language* (WSDL), *XML Schema Definition Language* (XSD), *Simple Object Access Protocol* (SOAP), *Universal Description, Discovery and Integration* (UDDI) e o *WS-I Basic Profile*.

      Essas especificações são usadas há algum tempo e foram adotadas por toda a indústria de TI. Contudo, a plataforma que elas representam coletivamente não contém vários recursos da qualidade exigidos pelo serviço para entregar funcionalidades de missão crítica de nível corporativo.
- **Plataforma de *Web Services* de segunda geração (extensões WS-*)**
    - Algumas das maiores lacunas relacionadas a serviços de qualidade, na plataforma de primeira geração, residem nas áreas de segurança no nível de mensagens, transações de serviços cruzados e troca de mensagens confiáveis; esta, juntamente a várias outras extensões, está sendo fornecida pela plataforma de *Web Services* de segunda geração. Consistindo em inúmeras especificações que aprimoram o *framework* fundamental da troca de mensagens de primeira geração, esse conjunto de tecnologias *Web Services* (em geral, rotulado como “WS-*”) fornece um rico conjunto de recursos muito mais complexo, tanto em termos da tecnologia quanto de *design*.

3.3 ARQUITETURA DE *WEB SERVICES*

Um *Web Service* é composto por: (SUN MICROSYSTEMS, 2003) e (ERL, 2009).

- **Contrato de Serviço Técnico:** consiste em uma definição WSDL, uma definição do esquema XML e, possivelmente, uma definição *WS-Policy*. Esse contrato de serviço expõe funções públicas (chamadas operações) e,

portanto, é comparável a uma tradicional interface de programação de aplicativo (API);

- **Corpo da Lógica de Programação:** a qual pode ser desenvolvida de uma maneira personalizada para um *Web Service* ou pode existir como lógica legada, que é empacotada por um *Web Service*, para que sua funcionalidade possa ser disponibilizada via padrões de comunicação *Web Services*. Se a lógica for desenvolvida de maneira personalizada, geralmente será criada como componente e chamada “lógica de serviço principal” (ou lógica do negócio);
- **Lógica do Processamento de Mensagens**: que existe como uma combinação de *parsers*, processadores e agentes de serviços. Os programas que executam processamento relacionado a mensagens têm, principalmente, fundamentos em eventos e, por isso, podem interceptar uma mensagem após a transmissão, ou antes do recebimento.

Um *Web Service* pode ser associado a papéis temporários, dependendo de sua utilização em tempo de execução. Por exemplo, um serviço *Web* pode atuar como fornecedor de serviços, quando recebe e responde a mensagens de solicitação, mas também pode assumir o papel de consumidor de serviços, quando for necessário o envio de mensagens de solicitação a outros *Web Services*.

Quando *Web Services* são posicionados dentro de composições de serviço, é comum que mudem para papéis de provedor de serviços. Observe-se que programas normais, componentes e sistemas legados também podem atuar como consumidores de um *Web Service*, desde que sejam capazes de se comunicar, empregando os padrões dos *Web Services*.

A Figura 19 apresenta os símbolos utilizados neste projeto para ilustrar representações físicas dos *Web Services*.

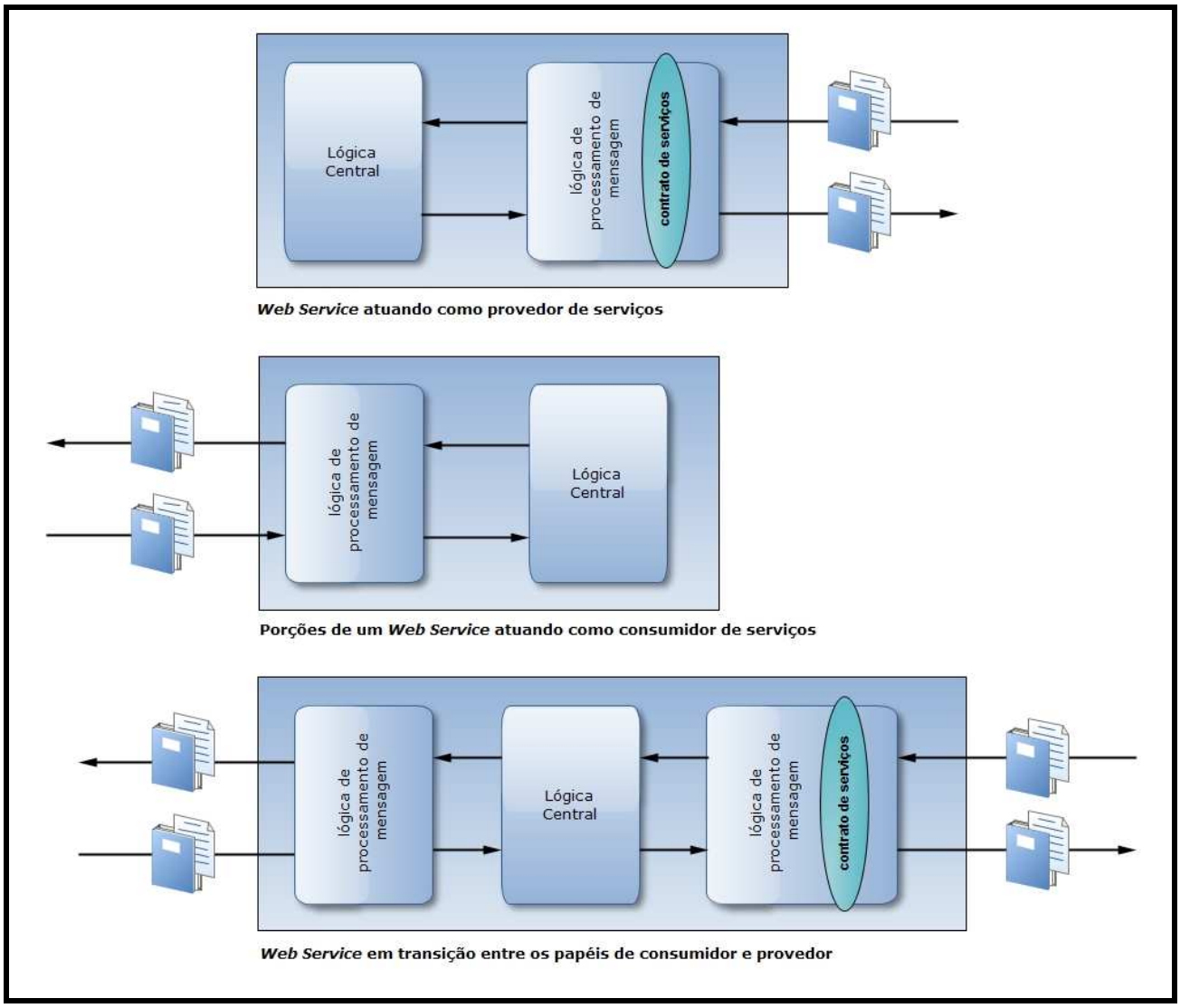

Figura 19 – Variações de *Web Services* (In: ERL, 2009)

3.4 *SIMPLE OBJECT ACCESS PROTOCOL* (SOAP)

O protocolo baseado em XML, utilizado para troca de informações entre computadores em uma arquitetura orientada a serviços, é o *Simple Object Access Procotol* ou SOAP. Embora o SOAP possa ser usado em uma variedade de sistemas de mensagens e possam ser entregues através de uma grande variedade de protocolos de transporte, o foco inicial do SOAP é chamado de procedimento remoto ou *remote procedure call* (RPC), transportado via HTTP (SNELL, 2001).

O protocolo SOAP, portanto, permite que aplicativos clientes possam se conectar facilmente a serviços remotos e invocar métodos remotos. Essa possibilidade

representa um marco na arquitetura de serviços *Web*, permitindo diversos pedidos de serviços e a fácil troca de dados (CHAPPEL; JEWELL, 2002) e (SNELL, 2001).

O envelope SOAP define regras específicas para encapsular dados que estão sendo transferidos entre computadores. Isso inclui dados específicos do aplicativo, como o nome do método a invocar, parâmetros de método ou retorno de valores, podendo também incluir informações sobre quem deve processar o conteúdo do envelope e, em caso de fracasso, como codificar mensagens de erro (NEWCOMER, 2002) e (SNELL, 2001).

3.5 *WEB SERVICES DESCRIPTION LANGUAGE* (WSDL)

A *Web Services Description Language (*WSDL) é uma especificação criada para descrever e publicar os formatos e protocolos de um *Web Service* de maneira padronizada, funcionando como um contrato de um serviço disponibilizado na *Web*, especificando como acessá-lo e quais operações ou métodos podem ser acessados. Ela também descreve os serviços *Web* em uma linguagem XML padronizada. A WSDL descreve quatro tipos críticos de dados de um serviço: (NEWCOMER, 2002), (CHAPPEL; JEWELL, 2002) e (SNELL, 2001).

- Informação descrevendo todas as funções ou métodos publicamente disponíveis;
- Informação dos tipos de dados para todos os pedidos de requisição e resposta;
- Informação sobre a vinculação do protocolo de transporte a ser utilizado;
- Informação do endereço para localizar o serviço especificado.

Em suma, a WSDL representa um contrato entre o solicitante do serviço e o provedor do serviço, sendo uma plataforma independente de linguagem e é utilizada principalmente, mas não exclusivamente, para descrever serviços SOAP

(NEWCOMER, 2002). A Figura 20 apresenta um documento WSDL típico que terá a seguinte estrutura de alto.

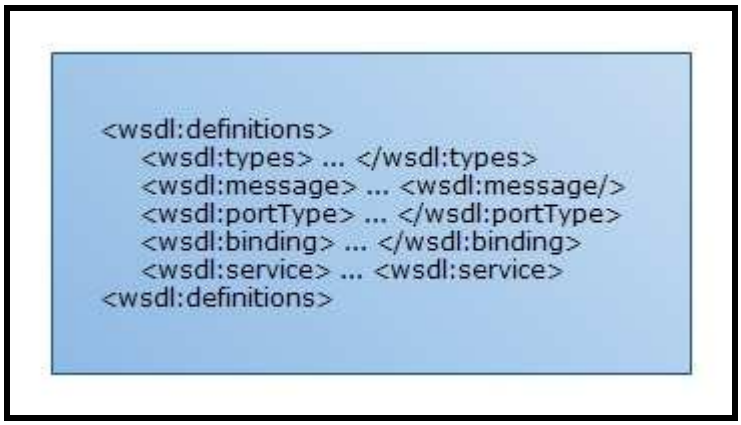

Figura 20 – Arquivo WSDL (In: NEWCOMER, 2002)

A Tabela 3 descreve as funcionalidades de cada elemento que compõe um arquivo WSDL.

Elemento	Descrição
wsdl:definitions	Utilizado para definir o elemento raiz do documento WSDL, possuindo o nome do *Web Service.*
wsdl:types	O elemento contêiner das definições do tipo de dados feitas, usando XSD ou outro sistema semelhante para tipos de dados.
wsdl:message	Definição dos dados de mensagem comunicada. A mensagem pode ser composta por várias partes e cada uma delas pode ser de um tipo diferente.
wsdl:portType	Conjunto de operações abstrato para o qual um ou mais pontos de extremidade oferecem suporte.
wsdl:binding	Protocolo e especificação do formato de dados concretos de um tipo de porta particular.
wsdl:service	Coleção de pontos de extremidade relacionados.

Tabela 3 – Elementos de um arquivo descrito em linguagem WSDL (In: NEWCOMER, 2002)

3.6 *UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION* (UDDI)

Para que seja possível efetuar uma chamada a um *Web Service*, torna-se necessário localizá-lo, descobrir a interface e semântica de sua chamada, escrever e configurar o aplicativo cliente para colaborar com o serviço.

O *Universal Description, Discovery and Integration* ou UDDI é um protocolo aprovado como padrão pela *Organization for the Advancement of Structured Information Standards* (OASIS), e define um método padrão para publicar e descobrir os componentes de software, baseado em uma arquitetura orientada a serviços (SOA). Um serviço de registro UDDI é um *Web Service* que gerencia informação sobre provedores, implementações e metadados de serviços. Os provedores de serviços podem utilizar UDDI para publicar os serviços que são oferecidos, enquanto os usuários de serviços podem usá-lo para descobrir serviços que lhes interessem e obter os metadados necessários para utilizar esses serviços (SNELL, 2001), (NEWCOMER, 2002) e (ERL, 2005).

Na versão 2.0, a UDDI define quatro elementos de dados principais dentro do modelo de dados:

- ***Bussiness Entity***: na estrutura *business entity* reside a *business information*, componente UDDI que corresponde às páginas que contêm dados gerais sobre um negócio e os produtos e serviços oferecidos pelo negócio. A estrutura *business entity* categoriza negócios pelos seus identificadores únicos;
- ***Bussiness Service***: o componente *business service information* corresponde às páginas que contêm dados técnicos sobre os produtos e serviços oferecidos por um determinado negócio. A informação de serviço do negócio reside na estrutura *business service*;
- ***tModel***: descreve especificação, classificação ou identificação de um serviço. O *tModel* é uma abstração de uma especificação técnica de um tipo de serviço, que organiza a informação do tipo de serviço e torna acessível o registro de dados;

- ***Binding Template***: o componente *binding information* também corresponde às páginas que contêm informação técnica relativa a um serviço na *Web*. Tal informação especifica como consumidores de *Web Services* podem se conectar a um ponto de entrada de um determinado serviço.

A Figura 21 apresenta graficamente o modelo de dados do protocolo UDDI.

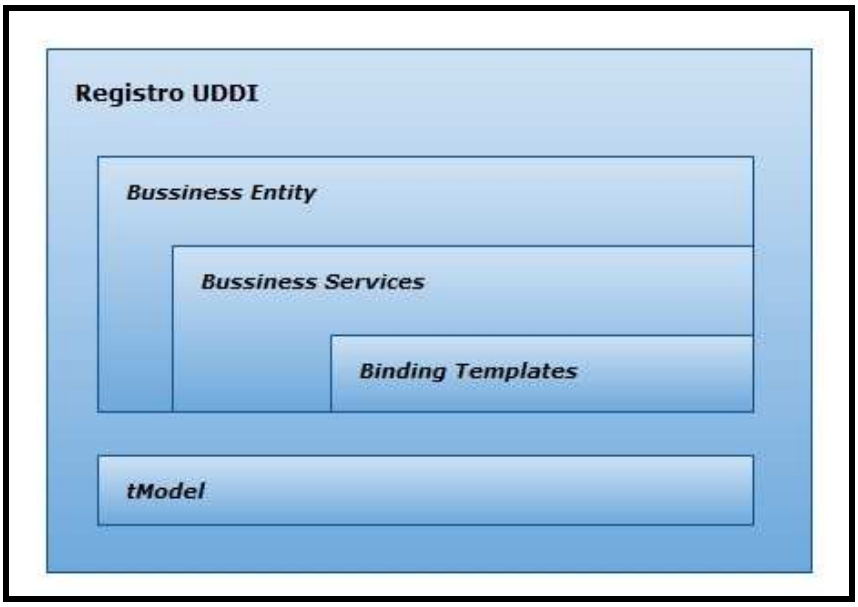

Figura 21 – Modelo do protocolo UDDI (In: NEWCOMER, 2002)

4 – TECNOLOGIA *GOOGLE ANDROID*

O *Android* é uma plataforma de código-aberto criada inicialmente pelo Google e mantida pela *Open Handset Alliance* (OHA), grupo de empresas de TI e telefonia. O objetivo desse projeto é criar uma plataforma de desenvolvimento de aplicativos para dispositivos móveis. Atualmente, é a mais nova sensação, revolucionando o desenvolvimento de aplicações. Possui um sistema operacional baseado no *kernel* do *Linux* e diversos aplicativos, com uma rica interface gráfica, um navegador de *Internet*, integração com a API do *Google Maps*, suporte à multimídia, GPS, banco de dados *Mobile* e entre outras características.

A moderna plataforma de desenvolvimento do *Google Android* torna possível o desenvolvimento e a integração de maneira simplificada, utilizando a linguagem de programação *Java* e uma *Integrated Development Enviromnent* (IDE) ou ambiente de desenvolvimento de alto nível e produtividade.

4.1 INTRODUÇÃO AO *GOOGLE ANDROID*

O mercado de celulares está crescendo cada vez mais. Estudos mostram que, atualmente, mais de três bilhões de pessoas possuem um aparelho celular, correspondendo a mais ou menos metade da população mundial (LECHETA, 2009).

O mercado corporativo também está crescendo muito, e diversas empresas estão buscando incorporar aplicações móveis a seu dia-a-dia, com a finalidade de agilizar negócios e integrar aplicações móveis com seus sistemas de *back-end*. Empresas visam lucro, e os celulares e *smartphones* podem ocupar um importante espaço em um mundo onde a palavra “mobilidade” está cada vez mais conhecida.

Dessa forma, aplicações que executam em um celular podem estar literalmente conectadas e *online*, sincronizando informações diretamente de um servidor confiável da empresa. Hoje, diversos bancos oferecem serviços aos seus usuários, mediante os quais é possível pagar suas contas e visualizar o extrato diretamente de

um celular. Países mais desenvolvidos já permitem que celulares seja utilizados em supermercados para ler os códigos de barras dos produtos e realizar a compra, como se fosse um cartão de crédito. Para isso, todos deveriam possuir uma plataforma poderosa e flexível, para tornar tudo isso mais viável e cada vez mais uma realidade para todos (LECHETA, 2009).

Enquanto as empresas e os desenvolvedores buscam uma plataforma moderna e ágil para o desenvolvimento de aplicações corporativas que lhes auxiliem em seus negócios e lucros, os usuário comuns buscam um celular com um visual elegante e moderno, de fácil navegação e uma infinidade de recursos. O *Android* é a resposta do *Google* para ocupar esse espaço. Consiste em uma nova plataforma de desenvolvimento para aplicativos móveis, baseada em um sistema operacional *Linux*, com diversas aplicações já instaladas e um ambiente de desenvolvimento bastante poderoso e flexível (ABLESON; COLLINS; SEN, 2009) e (LECHETA, 2009).

O *Android* causou um grande impacto quando foi anunciado, atraindo a atenção de muita gente. E isso se deve ao fato de ter sido o *Google* o responsável por esse projeto. Entretanto, não é apenas o *Google* que está investindo nessa área, e sim um grupo formado por empresas líderes do mercado de telefonia, como a *Motorola*, *LG*, *Samsung*, *Sony Ericsson* e muitas outras. Esse grupo, chamado de *Open Handset Alliance* (OHA), foi criado com a intenção de padronizar uma plataforma de código aberto e livre para celulares, justamente para atender a todas as expectativas e tendências do mercado atual (FARIA, 2008) e (LECHETA, 2009).

A plataforma do *Google* é a primeira para aplicações móveis completamente livre e de código aberto (*open-source*), o que representa uma grande vantagem para sua evolução, uma vez que diversos programadores do mundo poderão contribuir para melhorar a plataforma (FARIA, 2008) e (LECHETA, 2009).

Para os fabricantes de celulares, isso também é uma grande vantagem, pois possibilita utilizar o sistema operacional *Android* em seus celulares sem ter que pagar por isso. Além disso, a licença *Apache Software Foundation* (ASF) permite que alterações sejam feitas no código-fonte para criar produtos customizados sem precisar compartilhar as alterações com ninguém (LECHETA, 2009).

4.2 *OPEN HANDSET ALLIANCE* E O *ANDROID*

A *Open Handset Alliance* (OHA) é um grupo formado por gigantes do mercado de telefonia de celulares, liderados pelo *Google*. Entre alguns integrantes do grupo, estão nomes consagrados como *HTC*, *LG*, *Motorola*, *Samsung*, *Sony Ericsson*, *Toshiba*, *Sprint Nextel*, *China Mobile*, *T-Mobile*, *ASUS*, *Intel*, *Garmin* e muitos mais (FARIA, 2008).

No *site* da OHA (OPEN HANDSET ALLIANCE, 2010) existe uma ótima descrição do que seria essa aliança. O texto, traduzido do inglês, enfatiza:

> Hoje, existem 1,5 bilhão de aparelhos de televisão em uso em todo o mundo e um bilhão de pessoas têm acesso à *Internet*. No entanto, quase três bilhões de pessoas têm um telefone celulares, tornando o aparelho um dos produtos de consumo mais bem-sucedidos do mundo. Dessa forma, construir um aparelho celular superior melhoraria a vida de inúmeras pessoas em todo o mundo. A *Open Handset Alliance* é um grupo formado por empresas líderes em tecnologia móvel que compartilham essa visão para mudar a experiência móvel de todos os consumidores [...].

Assim, o objetivo do grupo é definir uma plataforma única e aberta para celulares, deixando os consumidores mais satisfeitos com o produto final. Outro objetivo dessa aliança é criar uma plataforma moderna e flexível para o desenvolvimento de aplicações coorporativas. O resultado dessa união de empresas foi a criação do *Android* (LECHETA, 2009) e (OPEN HANDSET ALLIANCE, 2010).

O *Android* é a nova plataforma de desenvolvimento de aplicativos móveis e contém um sistema operacional baseado em *Linux*, uma interface visual rica, GPS, diversas aplicações já instaladas e um ambiente de desenvolvimento bastante poderoso,

inovador e flexível. Outra boa notícia é que podemos utilizar a linguagem *Java* para desenvolver as aplicações, usufruindo de todos os recursos que ela nos proporciona (DIMARZIO, 2008) e (LECHETA, 2009).

O mundo da tecnologia está sempre em evolução, e a OHA foi criada justamente para manter uma plataforma-padrão em que todas as novas tendências do mercado estejam englobadas em uma única solução (LECHETA, 2009).

Para os fabricantes de celulares, o fato de existir uma plataforma única e consolidada é uma grande vantagem, no momento de criar novos aparelhos. A licença do *Android* é flexível e permite que cada fabricante possa realizar alterações no código-fonte, customizando seus produtos, sem a necessidade de compartilhar essas alterações. O fato de o *Android* ser de código aberto contribui muito para seu aperfeiçoamento, uma vez que desenvolvedores de todos os lugares do mundo podem cooperar para melhorar seu código-fonte, adicionando novas funcionalidades ou simplesmente corrigindo falhas (ABLESON; COLLINS; SEN, 2009).

Já os desenvolvedores de aplicações podem desfrutar de uma plataforma de desenvolvimento moderna, com diversos recursos disponíveis, com tudo o que há de mais moderno.

A Figura 22 ilustra o logotipo escolhido para o *Android*.

Figura 22 – Logotipo *Google Android* (In: *Google Projects for Android*, 2010)

4.3 SISTEMA OPERACIONAL *LINUX*

O sistema operacional do *Android* foi baseado no *kernel* 2.6 do *Linux*, e é responsável por gerenciar a memória, os processos, *threads* e a segurança dos arquivos e pastas, além de redes e *drivers* (ABLESON; COLLINS; SEN, 2009), (FARIA, 2008) e (LECHETA, 2009).

Cada aplicativo no *Android* dispara um novo processo, no sistema operacional. Alguns deles podem exibir uma tela para o usuário, enquanto outros podem ficar em execução em segundo plano, por tempo indeterminado. Diversos processos e aplicativos podem ser executados simultaneamente, enquanto o *kernel* do sistema operacional é o responsável por realizar todo o controle de memória. Caso necessário, o próprio sistema operacional pode decidir encerrar algum processo para liberar memória e recursos, e talvez até reiniciar o mesmo processo, quando a situação estiver controlada (LECHETA, 2009).

4.4 MÁQUINA VIRTUAL *DALVIK*

A linguagem *Java* é usada para construir aplicações para o *Android*, porém, o fato é que no sistema operacional *Android* não existe uma máquina virtual *Java* ou *Java Virtual Machine* (JVM). Na verdade, o que existe é uma máquina virtual chamada *Dalvik*, que é otimizada para a execução em dispositivos móveis (ABLESON; COLLINS; SEN, 2009) e (LECHETA, 2009).

Ao desenvolver as aplicações para o *Android*, é possível empregar a linguagem *Java* e todos os seus recursos normalmente, porém, o *bytecode* (*.class*) é compilado e convertido para o formato *.dex* (*Dalvik Executable*), representando a aplicação do *Android* compilada (ABLESON; COLLINS; SEN, 2009) e (DIMARZIO, 2008).

Após esse processo, os arquivos *.dex* e outros recursos, como imagens, são compactados em um único arquivo com a extensão *.apk* (*Android Package File*), representando a aplicação final, pronta para ser distribuída e instalada (ABLESON; COLLINS; SEN, 2009).

4.5 DESENVOLVIMENTO DE APLICAÇÕES *ANDROID*

É possível desenvolver uma aplicação para o *Android*, utilizando a linguagem de programação *Java* em seu ambiente de desenvolvimento preferido, como o *Eclipse* ou *Netbeans*.

O *Eclipse* é o ambiente de desenvolvimento preferido pelo *Google*, e há um *plug-in* chamado *Android Development Tools* (ADT) para facilitar o desenvolvimento, os testes e a compilação do projeto.

Usando o *plug-in* ADT, é possível executar o emulador do *Android* diretamente no *Eclipse*, usufruindo de todos os seus recursos. Também, diretamente do *Eclipse*, é possível controlar o emulador, visualizando *logs* e simulando o envio de uma mensagem *Short Message Service (*SMS) ou uma ligação telefônica, além da capacidade de visualizar e enviar arquivos para o emulador, executar o *Garbage Collector* (GC), entre outras funcionalidades (DIMARZIO, 2008) e (LECHETA, 2009).

4.6 *ANDROID* SDK

O *Android* SDK é o *software* adotado para desenvolver aplicações no *Android*, possuindo um emulador para simular o celular, ferramentas utilitárias e uma API completa para a linguagem *Java*, com todas as classes necessárias para desenvolver as aplicações (DIMARZIO, 2008).

Embora o SDK tenha um emulador que pode ser executado como um aplicativo comum, existe um *plug-in* para o *Eclipse* que visa a integrar o ambiente de desenvolvimento *Java* com o emulador (LECHETA, 2009).

A Figura 23 ilustra o emulador do *Android*.

Figura 23 – Emulador *Android*

Com o *plug-in*, é possível iniciar o emulador diretamente dentro do *Eclipse*, instalando a aplicação automaticamente e, com o *debug* do *Eclipse* integrado, é possível depurar o código-fonte como qualquer outra aplicação *Java*.

4.7 PLATAFORMA *GOOGLE ANDROID*

A Figura 24 evidencia as camadas da plataforma *Google Android*. Será feita uma descrição detalhada de cada uma dessas camadas, pois é de fundamental importância um bom entendimento de como os processos funcionam, para contribuir com o desenvolvimento de aplicativos mais eficientes.

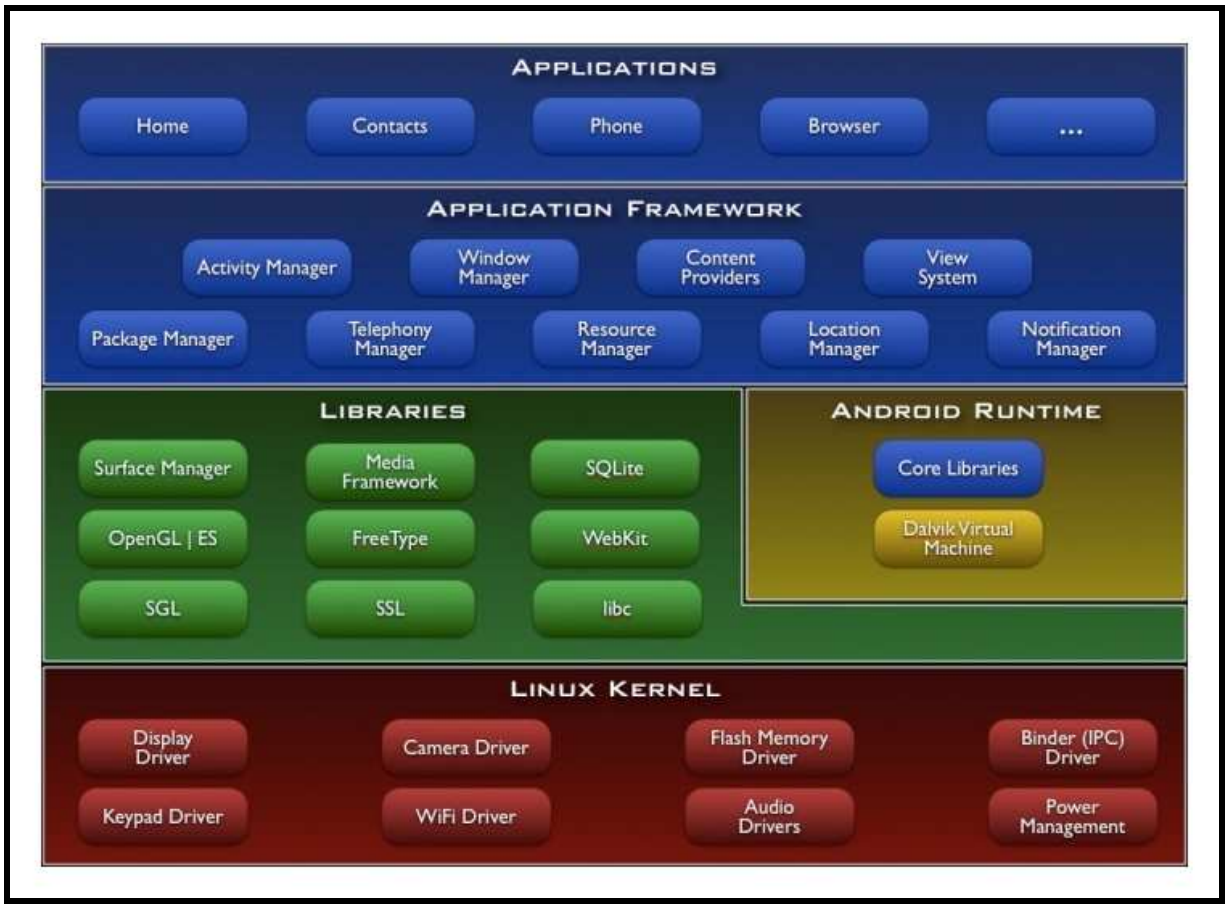

Figura 24 – Camadas da plataforma *Android* (In: FARIA, 2008)

4.7.1 Camada *applications*

É a camada em que, como o próprio nome revela, encontram-se todos os aplicativos do *Android*, como cliente de *e-mail*, navegador *Web*, contatos, entre outros. Todos os projetos desenvolvidos e instalados no emulador estarão presentes nessa camada da plataforma (FARIA, 2008).

4.7.2 Camada *application framework*

É a camada responsável por disponibilizar todas as APIs e recursos necessários para os pacotes e aplicativos, como classes visuais, *Content Providers* (provedores de conteúdo, que permitem a troca de informações entre aplicações), gerenciador de recursos, ciclo de vida de uma aplicação e gerenciador de pacotes (FARIA, 2008).

4.7.3 Camada *libraries*

O *Android* inclui um conjunto de bibliotecas C/C++ utilizadas por vários componentes do sistema. Tais capacidades são expostas para os desenvolvedores através do *framework*. Na Tabela 4, é possível visualizar as principais bibliotecas (FARIA, 2008).

Biblioteca	Descrição
System C Library	Uma implementação derivada da biblioteca C padrão sistema (libc) do BSD sintonizada para dispositivos rodando Linux
Media Libraries	As bibliotecas de mídia suportam os mais populares formatos de áudio, vídeo e imagem
Surface Manager	Subsistema de exibição, bem como múltiplas camadas de aplicações 2D e 3D
LibWebCore	Um *Web Browser Engine* utilizado tanto no *Android Browser* quanto para exibições *Web*
SGL	*Engine* de gráficos 2D
3D libraries	Implementação baseada no *OpenGL* As bibliotecas empregam aceleração 3D via *hardware* (quando disponível) ou o *software* de renderização 3D altamente otimizado
FreeType	Renderização de fontes *Bitmap* e *Vector*
SQLite	Um poderoso e leve *Engine* de banco de dados relacional disponível para todas as aplicações *Android*

Tabela 4 – Principais bibliotecas da camada *Libraries* (In: FARIA, 2008)

4.7.4 Camada *android runtime*

O *Android* possui um grupo de bibliotecas, fornecendo a maioria das funcionalidades nas principais bibliotecas da linguagem de programação *Java*.

Toda aplicação *Android* roda em seu próprio processo, com sua própria instância da máquina virtual *Dalvik*. O *Dalvik* foi escrito de forma a executar várias máquinas virtuais eficientemente. Ele executa arquivos .*dex*, que são otimizados para consumo mínimo de memória. A máquina virtual é baseada em registros e executa classes compiladas pela linguagem *Java* que foram transformadas em arquivos .*dex*, por meio da ferramenta *dx*, incluída no SDK (FARIA, 2008).

4.7.5 Camada *linux kernel*

Utiliza a versão 2.6 do *kernel* do *Linux* para os serviços centrais do sistema, tais como segurança, gestão de memória, gestão de processos, entre outras tarefas. O *kernel* também atua como uma camada de abstração entre o *hardware* e o resto do *software*, exercendo a funcionalidade de *middleware* entre as camadas da plataforma *Google Android* (FARIA, 2008).

5 – MODELAGEM DO PROBLEMA

Neste capítulo serão apresentadas a especificação e implementação do modelo proposto neste trabalho. O problema que fora abordado consistiu em desenvolver um ambiente orientado a serviços, capaz de expor funcionalidades que podem ser acessadas por uma aplicação cliente, desenvolvida com a plataforma *Google Android*, permitindo que um usuário interaja com um celular inteligente, que, comunicando-se com um servidor de aplicação, permitirá consumir soluções lógicas remotas.

5.1 DEFINIÇÃO DO PROBLEMA

A definição de um problema consistiu em um estudo de caso sobre a implementação de aplicações orientadas a serviços, baseadas na plataforma *Google Android*. Para o desenvolvimento deste estudo de caso foram utilizados os conceitos da arquitetura SOA e as ferramentas empregadas para implementar *Web Services* na linguagem de programação *Java*. Este desenvolvimento foi aplicado em um ambiente acadêmico, permitindo que administradores, professores e alunos obtenham informações a partir de uma página *Web* e de um aplicativo móvel, resultando em duas aplicações cliente independentes de qualquer regra de negócio. Para facilitar o desenvolvimento deste estudo de caso, ele foi dividido em dois módulos.

1º Módulo: Ambiente orientado a serviços

Neste módulo será desenvolvido o modelo responsável pela exposição das regras de negócio no formato de *Web Services*, comunicando-se com um servidor de aplicação e uma base de dados;

2º Módulo: Aplicação cliente

Neste módulo serão desenvolvidos todos os componentes responsáveis por conectar o servidor de aplicação, buscar os serviços disponíveis e consumi-los, bem como ser capaz de obter a resposta e tratá-la, criando uma interação com o usuário.

5.2 ARQUITETURA DO AMBIENTE DISTRIBUÍDO

A modelagem do problema a ser abordado é ilustrada na Figura 25 e representa a arquitetura do ambiente distribuído com as entidades e seus relacionamentos. Esta arquitetura mostra claramente a complexidade do problema que será abordado neste trabalho.

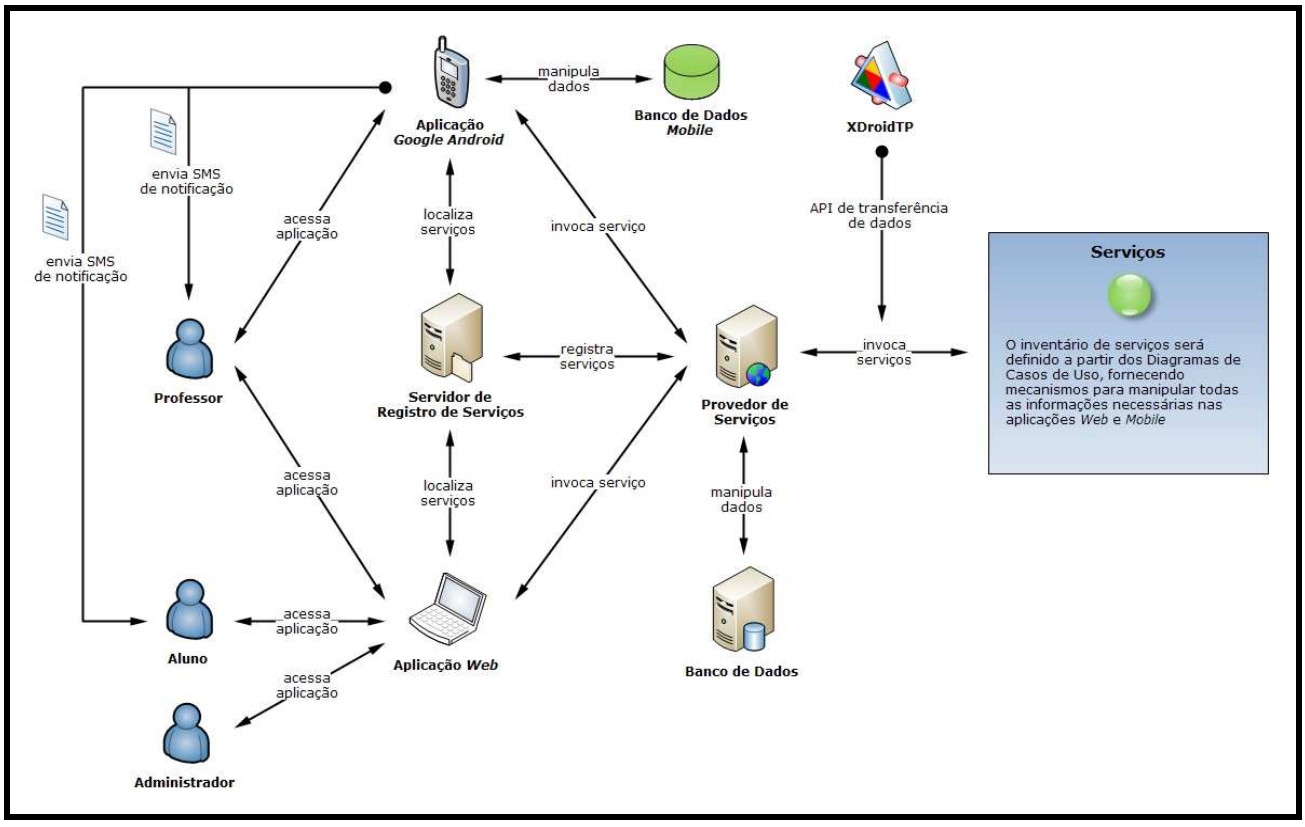

Figura 25 – Modelo da arquitetura do ambiente distribuído

O modelo arquitetado possui professor, aluno e administrador como entidades usuárias do ambiente distribuído. As entidades aluno e administrador irão se comunicar com uma aplicação *Web*, enquanto a entidade professor, além de utilizar essa mesma aplicação, também usará um aplicativo *Mobile*, desenvolvido na plataforma *Google Android*.

O ambiente conta com um servidor de registro de serviços, que é responsável pela verificação de novos *Web Services* e, consequentemente, pelo cadastramento utilizando o protocolo UDDI. Uma vez registrado um novo serviço *Web*, este poderá ser invocado e consumido por uma das aplicações cliente, *Web* ou *Mobile*, por meio de um servidor de aplicação que armazena o projeto contêiner do inventário de serviços. Juntamente com o provedor de serviços, há o banco de dados como repositório das informações do ambiente distribuído.

5.3 DESENVOLVIMENTO DO ESTUDO DE CASO

O estudo de caso será a implementação de um ambiente acadêmico, permitindo que administradores, professores e alunos obtenham informações a partir de uma página *Web* e de um aplicativo móvel, resultando em duas aplicações cliente independentes de qualquer regra de negócio, já que este controle transacional será delegado ao ambiente orientado a serviços, desenvolvido no primeiro módulo.

A aplicação móvel será acessada pelos professores, devidamente autenticados por meio de uma tela de *login*, sendo possível selecionar uma de suas turmas, interagindo com o aplicativo. A partir dessas requisições, alguns serviços serão invocados no servidor de aplicação, buscando e recuperando dados diretamente para o dispositivo móvel, para que o professor possa alterá-los e retorná-los ao ambiente orientado a serviços, persistindo tais informações no banco de dados distribuído. O objetivo da aplicação móvel é fornecer mecanismos para que professores possam efetuar chamadas durante as aulas, sem a dependência de qualquer outro material além de um celular e conexão com o servidor. Além de facilitar o trabalho de verificação das presenças e ausências, será possível obter dados estatísticos e informações adicionais sobre alunos e aulas ministradas,

criando uma visão mais portátil e moderna para essa tarefa. No exato momento da realização de uma chamada, serão enviadas mensagens de notificação no formato *Short Message Service* (SMS), comunicando o próprio professor e os alunos sobre o que está sendo atualizado, em termos de frequência escolar.

5.3.1 Ambiente orientado a serviços

O ambiente orientado a serviços é composto por um servidor de aplicações e por um projeto *Web* dinâmico desenvolvido em *Java*, responsável por controlar as regras e unidades lógicas de negócio, expondo serviços que podem ser consumidos por aplicações cliente. Presente no ambiente, também está o servidor de banco de dados, usado para ser o repositório das informações que virão a ser visualizadas e manipuladas, de acordo com as operações realizadas pelas entidades usuárias do ambiente e das aplicações de *front-end*.

5.3.2 Aplicações cliente

As aplicações cliente envolvem o desenvolvimento das interfaces que permitem a interação entre os usuários e o ambiente, e para isso foram desenvolvidos dois *softwares*: uma aplicação *Web* e um aplicativo *Mobile*.

5.3.2.1 Aplicação *web*

A aplicação *Web* foi desenvolvida empregando-se *Java* como linguagem de programação, fazendo uso de da tecnologia *Adobe Flex*, um *framework* que suporta o desenvolvimento de aplicações ricas para a *Internet*. Uma aplicação do tipo *Rich Internet Application* ou RIA é uma aplicação *Web* que possui características e funcionalidades de um *software* tradicional, assim como os aplicativos *Desktop*. Esses aplicativos transferem todo o processamento da interface para o navegador da *Internet*, porém mantém a maior parte dos dados no servidor de aplicação.

Dentre os *frameworks* de RIA utilizados atualmente, o *Adobe Flex* é o que mais vem crescendo. A tecnologia *Flex* utiliza a linguagem MXML para gerar as interfaces, *ActionScript* para sua programação *front-end* e outra linguagem para *back-end*, sendo possível desenvolver com *Java*, *Ruby* ou *Python*, entre outras linguagens de programação.

Na Figura 26, é apresentada uma das telas da aplicação *Web*, desenvolvida com *Java* e *Flex*:

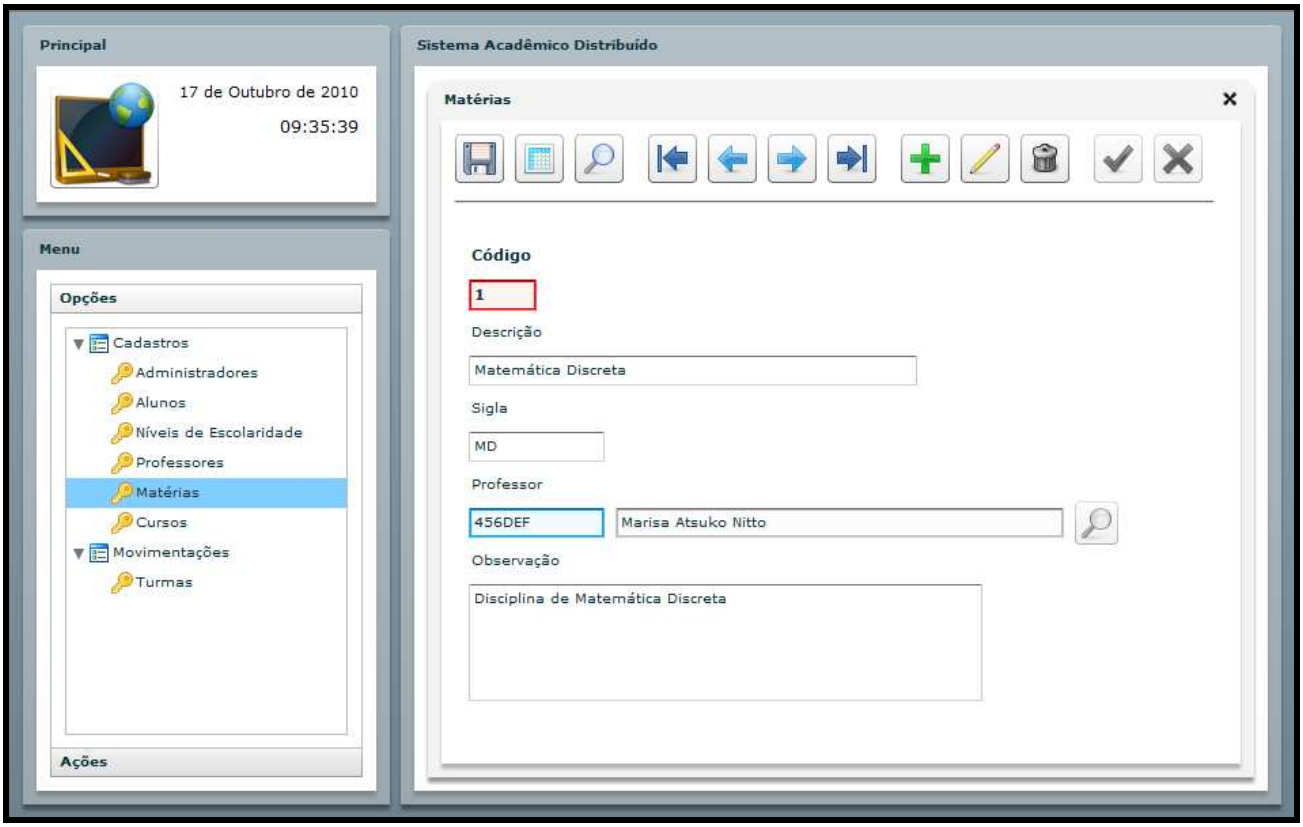

Figura 26 – Aplicação *Web*

5.3.2.2 Aplicação *mobile*

A aplicação *Mobile* foi desenvolvida adotando-se a plataforma *Google Android*, também implementada com *Java*, comunicando-se com o servidor de aplicação através da *Internet* e por um protocolo de transferência de dados.

A figura 27 apresenta a tela principal do aplicativo, onde é possível selecionar a opção de “Efetuar *Login*”, para efetuar a autenticação no sistema, além da opção de “Informações”, que exibe dados sobre o aplicativo.

Figura 27 – a) Tela principal b) Tela de autenticação c) Tela de informações

Na figura 28 são mostradas três telas sequenciais para o processo de realização de uma chamada. A primeira tela exibida, de seleção de turma, apresenta uma listagem de turmas. A segunda tela exibida, de seleção de matéria, apresenta uma listagem de matérias. A terceira tela exibida apresenta a listagem de alunos, recuperada com base nas informações selecionadas nas telas anteriores.

Figura 28 – a) Tela de seleção de turmas b) Tela de seleção de matérias c) Tela de listagem de alunos

Tanto a aplicação *Web* quanto o aplicativo *Mobile* possuem uma camada responsável pela comunicação entre as aplicações cliente e o provedor de serviços.

5.3.3 API *XDroidTP*

Para ser possível essa comunicação e a troca de mensagens XML, foi desenvolvida uma API *Java* com o nome de *XDroidTP* (simplificação de *XML Droid Transfer Protocol*), com a finalidade de ser um protocolo de transferência de dados complexos. Fez-se necessário a implementação desse conjunto de classes compiladas *Java*, porque, atualmente, a transferência de dados é feita, em sua maioria, para tipos primitivos; entretanto, com esse novo protocolo, é possível que as aplicações *Web* e *Mobile* troquem objetos complexos serializados entre elas e o servidor de aplicação.

O desenvolvimento desta API para transformação de dados foi possível graças a duas tecnologias que a linguagem de programação *Java* nos provê:

1ª Tecnologia: *Java Reflection*

A “reflexão” é geralmente utilizada por programas que exigem a capacidade de analisar ou modificar o comportamento em tempo de execução de aplicativos em uso na máquina virtual *Java*. Esta é uma característica relativamente avançada e deve ser usada apenas por desenvolvedores que têm um forte domínio dos fundamentos da linguagem. Com essa advertência em mente, a reflexão é uma técnica poderosa, permitindo que aplicativos possam executar operações que de outra forma seria impossível.

2ª Tecnologia: *Java Annotation*

Uma “anotação” é uma forma especial de acrescentar, sintaticamente, metadados ao código-fonte *Java*. Classes, métodos, variáveis, parâmetros e os pacotes podem ser anotados. Ao contrário de *tags Javadoc*, anotações *Java* podem ser reflexivas na medida em que, embutidas em arquivos de classes geradas pelo compilador, podem ser recuperadas, em tempo de execução, pela máquina virtual *Java*.

Para exemplificar suas funcionalidades, segue um exemplo da utilização da API *XDroidTP*.

Uma classe Aluno possui os atributos de RA e Nome, conforme a implementação *Java*. A classe e seus atributos são marcados pela *annotation* ou anotação *XMLElement*, pertencente a API *XDroidTP*, permitindo informar o nome da *tag* e a qual classe ela pertence:

```java
package beans;

import annotations.XMLElement;

@XMLElement( tagName="aluno" )
public class Aluno {
	@XMLElement( tagName="registroAcademico" )
	private String ra;
	@XMLElement( tagName="nomeCompleto" )
	private String nome;

	public Aluno() {
		super();
	}

	public Aluno(String ra, String nome) {
		super();
		this.ra = ra;
		this.nome = nome;
	}

	// Métodos getters e setters ocultos para facilitar visualização

}
```

No programa principal, cria-se uma nova variável de referência para um objeto do tipo Aluno, atribuindo-se alguns valores a seus atributos e, logo após, o método *generateXML* da API *XDroidTP* é invocado para realizar a transformação de um objeto em um XML, conforme a implementação *Java*:

```java
package main;

import beans.Aluno;

public class Principal {
	public static void main(String[] args) {
		Aluno guilherme = new Aluno();

		guilherme.setRa("0711270028");
		guilherme.setNome("Guilherme de Cleva Farto");

		XDroidTP xDroidTP = new XDroidTP();

		try {
			String xml = xDroidTP.generateXML(guilherme);

			System.out.println(xml);
		} catch (Exception e) {
			System.err.println("Erro ao gerar XML");
		}
	}

}
```

No console da IDE *Java*, a saída é um XML formatado contendo as informações do objeto passado como parâmetro, conforme mostrado:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
    <metadata class="beans.Aluno" tagname="aluno">
        <field name="ra" tagname="registroAcademico" class="java.lang.String" />
        <field name="nome" tagname="nomeCompleto" class="java.lang.String" />
    </metadata>
    <aluno>
        <registroAcademico><![CDATA[0711270028]]></registroAcademico>
        <nomeCompleto><![CDATA[Guilherme de Cleva Farto]]></nomeCompleto>
    </aluno>
</root>
```

Toda a comunicação da aplicação *Mobile* com o servidor provedor de serviços foi realizada utilizando-se a API *XDroidTP*, facilitando o acesso, recuperação e transformação dos registros armazenados na base de dados.

5.4 MODELAGEM DO NEGÓCIO

Para que seja possível obter uma visão geral sobre o funcionamento e as atividades pertinentes a cada entidade, são apresentados os diagramas de casos de uso para administrador, professor e aluno.

O administrador é o responsável por gerenciar as principais informações do sistema. Entre suas tarefas, estão o cadastramento de outros administradores e o controle de dados dos alunos, professores, matérias e turmas.

A Figura 29 apresenta o diagrama de casos de uso para o administrador.

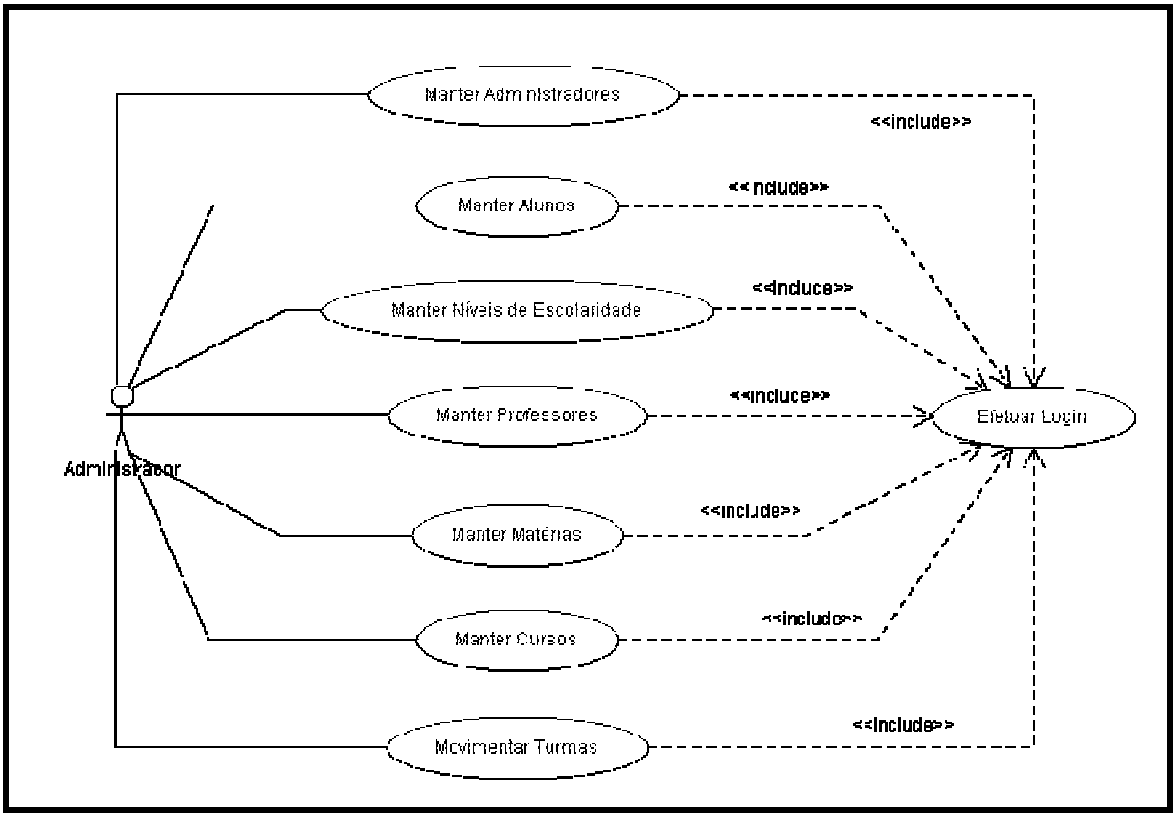

Figura 29 – Diagrama de Casos de Uso – Administrador

Pertencem à entidade professor as tarefas de solicitação da lista de chamada e o processo de realização da chamada, no aplicativo *Android*, além da possibilidade de visualizar as chamadas realizadas por meio da aplicação *Web*.

A Figura 30 evidencia o diagrama de casos de uso para o professor.

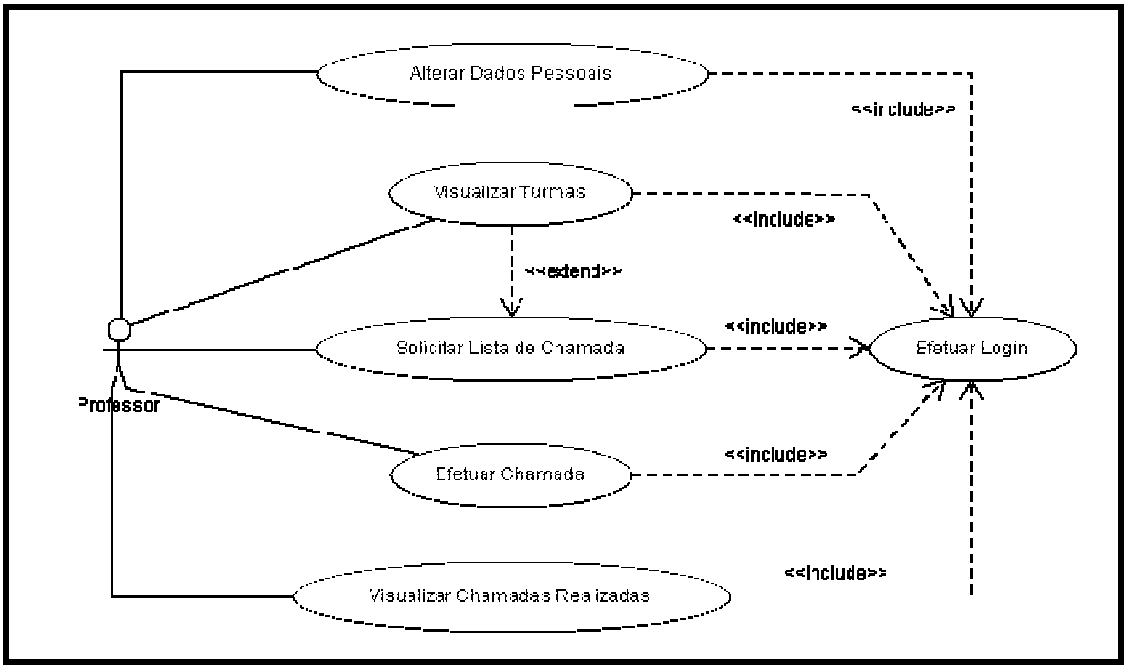

Figura 30 – Diagrama de Casos de Uso – Professor

Com a aplicação *Web*, a entidade aluno é capaz de visualizar a frequência, informação essa obtida das chamadas realizadas pelo professor, bem como a alteração de seus dados pessoais e a visualização de sua agenda de aulas.

A Figura 31 apresenta o diagrama de casos de uso para o aluno.

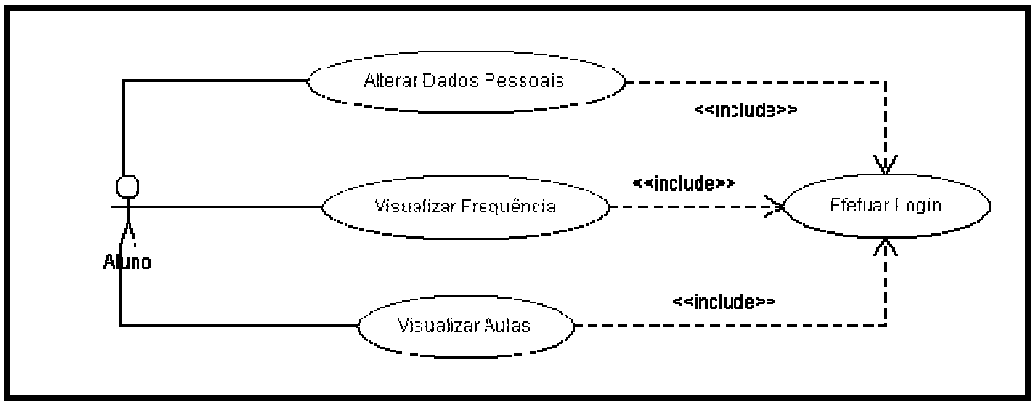

Figura 31 – Diagrama de Casos de Uso - Aluno

Para que qualquer atividade seja efetivada por uma das entidades do sistema (administrador, professor ou aluno), existe a premissa de estarem devidamente autenticados nos sistemas *Web* e *Mobile*, como forma de assegurar que as informações sejam válidas e restritas às pessoas que possuem acesso aos aplicativos.

Para especificar o processo de realização da chamada, foram desenvolvidos os seguintes modelos de processos, descrevendo todas as fases que serão utilizadas para executar essa tarefa.

A modelagem dos processos foi realizada utilizando-se o *software BizAgi Process Modeler*, uma ferramenta que fornece mecanismos para a criação de diagramas e fluxogramas, organizando graficamente vários processos e as relações existentes em cada etapa.

A Figura 32 ilustra o processo de efetuar *login*:

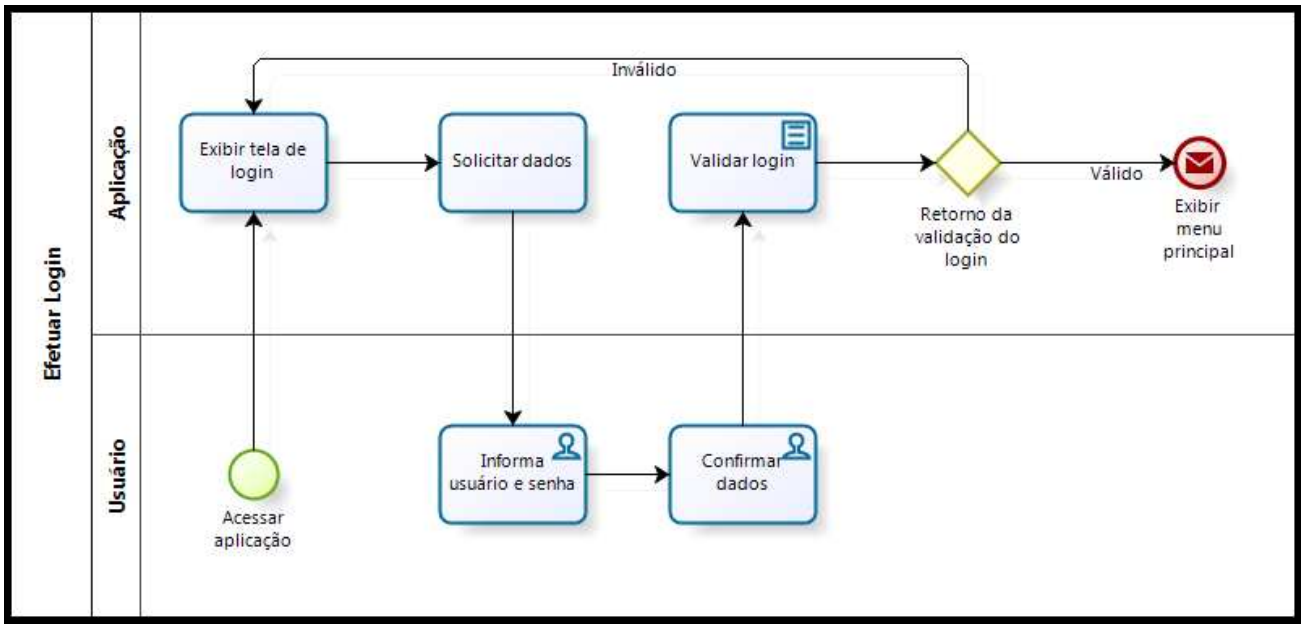

Figura 32 – Modelagem de Processo – Efetuar *login*

Após a validação do *login* do professor, as informações referentes as turmas também são recuperadas, como ilustra a Figura 33:

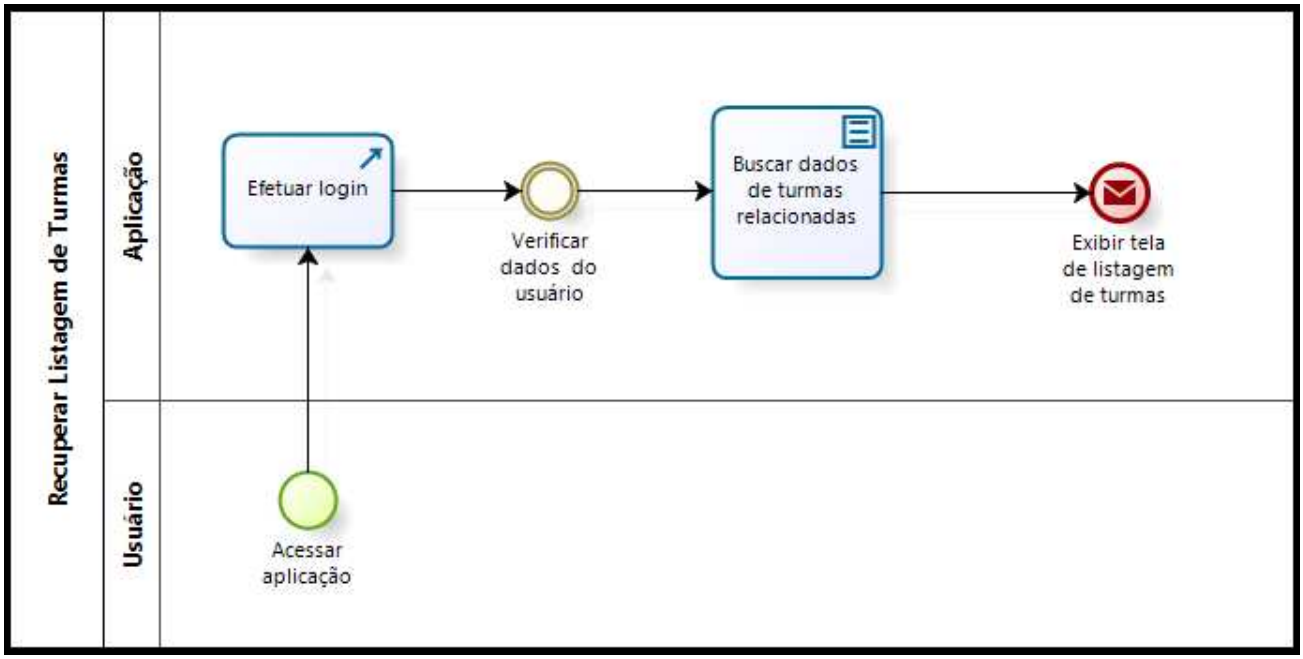

Figura 33 – Modelagem de Processo – Recuperar listagem de turmas

Após a recuperação das informações das turmas, o professor deve selecionar uma delas para recuperar uma listagem de matérias, como ilustra a Figura 34:

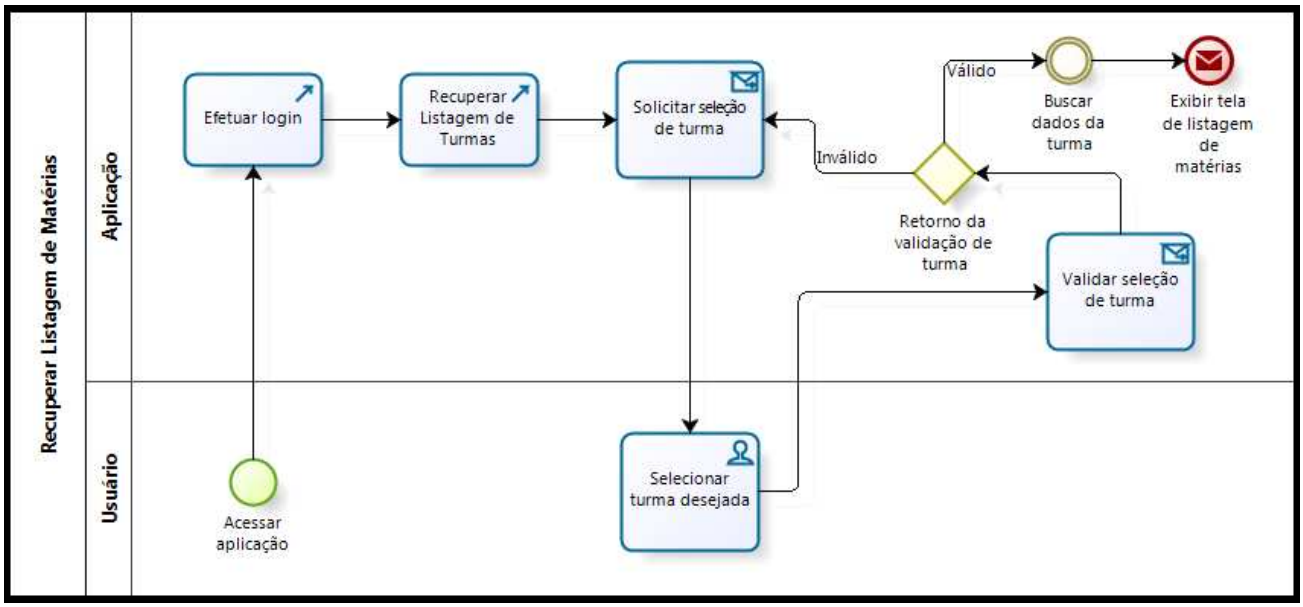

Figura 34 – Modelagem de Processo – Recuperar listagem de matérias

Após a recuperação das informações das matérias, o professor deve selecionar uma delas para recuperar uma listagem de alunos, como ilustra a Figura 35:

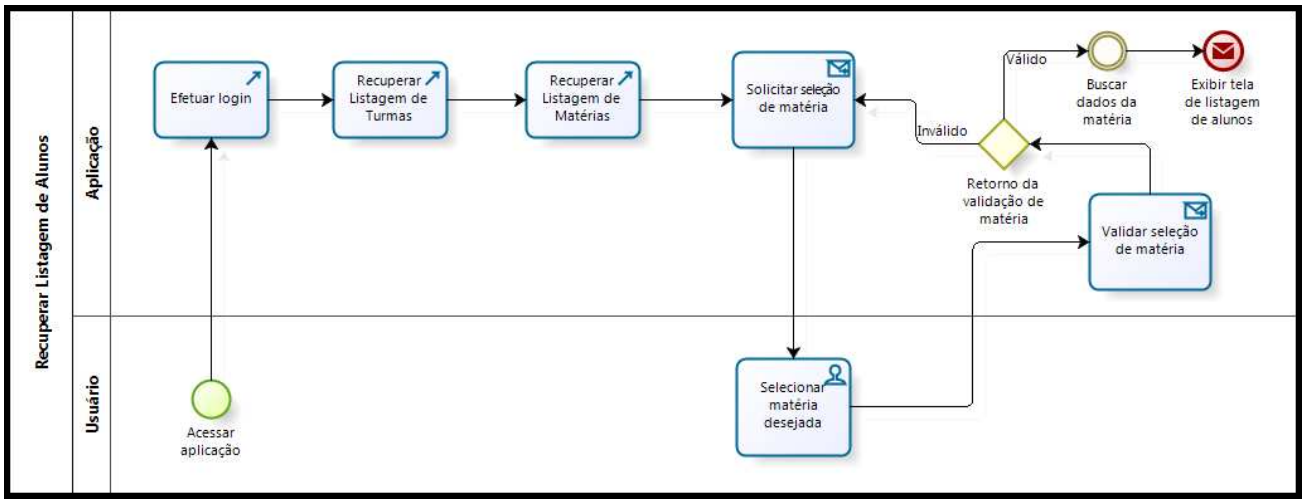

Figura 35 – Modelagem de Processo – Recuperar listagem de alunos

Ao concluir a etapa de implementação, foi possível testar e validar os produtos gerados, integrando-os como parte deste trabalho acadêmico para expor os resultados alcançados e contribuir com a sociedade acadêmica e científica em um assunto novo, recente e, crescentemente, estudado pelas empresas e grupos corporativos.

6 – CONCLUSÃO

O interesse pelo tema escolhido se deve ao fato do grande avanço tecnológico na área de sistemas distribuídos, proposta que vem sendo aceita pela grande maioria das empresas e grupos corporativos, criando-se possibilidades inovadoras para disponibilizar serviços e informações, que, há algum tempo, estavam restritas a páginas *Web* estáticas e em aplicações centralizadas, acessadas por meio de terminais previamente configurados.

A pesquisa na área de sistemas distribuídos com a arquitetura orientada a serviços, apesar de ser recente, tem contribuído muito para a interoperabilidade entre sistemas e para a disponibilização de dados e operações, permitindo que as unidades de soluções lógicas de uma empresa sejam expostas em um servidor que, acessado por qualquer dispositivo, podem executar determinados processos, facilitando a obtenção e a manipulação de informações de uma maneira mais portátil, leve e independente de tecnologia.

A proposta do projeto foi a de desenvolver os estudos teóricos com a finalidade de adquirir os conhecimentos necessários sobre a arquitetura orientada a serviços, bem como a sua implementação por meio de *Web Services*. Juntamente com a pesquisa, foi criada a modelagem do problema de um estudo de caso, permitindo aplicar os conceitos obtidos na análise teórica, tornando possível realizar a implementação de um ambiente orientado a serviços, bem como o desenvolvimento de uma ferramenta na plataforma *Google Android*, capaz de interagir com os serviços expostos.

Algumas dificuldades surgiram em virtude do desafio proposto, todavia, novas metas foram visadas e satisfatoriamente alcançadas, fornecendo um novo patamar de conhecimento e uma maior evolução, quanto aos conceitos estudados inicialmente, ampliando a visão sobre o que estava sendo arquitetado e desenvolvido. Como destaque, inclui-se o desenvolvimento de uma API *Java* que fornece funcionalidades para a troca de mensagens XML entre as aplicações *Web* e *Mobile* e o servidor de aplicação, visto que este é um problema ocasionado pela escassez de mecanismos capazes de transmitir dados complexos pelos *Web Services*.

Como objetivo final deste projeto, foi realizado o desenvolvimento do ambiente orientado a serviços, responsável pela parte lógica do negócio e por expor os serviços que serão acessados e consumidos pelo aplicativo móvel. Com o desenvolvimento da aplicação *Mobile*, foi possível obter um produto que igualmente será usado na apresentação final deste trabalho acadêmico, reforçando os conceitos de modelagem e implementação de sistemas distribuídos por meio da orientação a serviços.

REFERÊNCIAS BIBLIOGRÁFICAS

ABLESON, W. Frank; COLLINS, Charlie; SEN, Robi. **Unlocking Android.** A Developer's Guide. 1. ed. Greenwich: Manning, 2009.

CHAPPEL, David A.; JEWELL, Tyler. **Java Web Services** – Using Java in Service-Oriented Architectures. São Paulo: O'Reilly, 2002.

DIMARZIO, Jerome F. **Android –** A Programmer's Guide. 1. ed. New York City: McGraw-Hill, 2008.

ERL, Thomas. **Service-Oriented Architecture:** Concepts, Technology and Design. São Paulo: Pearson/Prentice Hall, 2005.

ERL, Thomas. **SOA – Princípios de *Design* de Serviços**. 1. ed. Tradução de Edson Furmankiewicz e Carlos Schafranski. São Paulo: Pearson/Prentice Hall, 2009.

FARIA, Alessandro de Oliveira. Programe seu Andróide. **Linux Magazine**, Volume 1, Número 43, p. 73-77, 2008.

GOOGLE PROJECTS FOR ANDROID. Disponível em: <http://code.google.com/android/>. Acesso em 29 mai. 2010.

JOSUTTIS, Nicolai M. **SOA na Prática** – A Arte de Modelagem de Sistemas Distribuídos. 1. ed. Tradução de Ivan Bosnic. Rio de Janeiro: Alta Books, 2008.

LECHETA, Ricardo R. **Google Android** – Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 1. ed. São Paulo: Novatec, 2009.

NAGAPPAN, Ramesh; SKOCZYLAS, Robert; SRIGANESH, Rima Patel. **Developing Java Web Services –** Architecting and developing secure Web Services using Java. 1. ed. Wiley: Agency for Instructional Technology, 2003.

NEWCOMER, Eric. **Understanding Web Services** – XML, WSDL, SOAP and UDDI. São Paulo: Addison-Wesley Professional, 2002.

OPEN HANDSET ALLIANCE. Disponível em: <http://www.openhandsetalliance.com/>. Acesso em: 24 mai. 2010.

PULIER, Eric; TAYLOR, Hugh. **Compreendendo SOA Corporativa**. 1. ed. Tradução de Marcelo Trannin Machado. Rio de Janeiro: Ciência Moderna, 2008.

SNELL, James. **Programming Web Services with SOAP**. São Paulo: O'Reilly, 2001.

SUN MICROSYSTEMS, Inc. **Web Services for Java™ Technology Programmers DWS-310**. 1. ed. Broomfield: Sun Microsystems, 2003.