UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

JONASTA DOS SANTOS

DEBITAR: GERENCIAMENTO DE DÉBITOS EM ANDROID COM SINCRONIZAÇÃO WEB/JSF ATRAVÉS DE WEB SERVICE

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA

2012

JONASTA DOS SANTOS

DEBITAR: GERENCIAMENTO DE DÉBITOS EM ANDROID COM SINCRONIZAÇÃO WEB/JSF ATRAVÉS DE WEB SERVICE

Trabalho de conclusão de Pós-Graduação apresentado ao Programa de Pós Graduação em Tecnologia JAVA da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Pós-Graduado em Tecnologia JAVA” – Área de Concentração: Tecnologia JAVA. Orientador : Paulo Roberto Bueno.

CURITIBA

2012

RESUMO

SANTOS, Jonasta. Debitar: Gerenciamento de débitos em Android com sincronização web/JSF através de Web service. 2012. Monografia (Especialização em Tecnologia Java) – Departamento acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

Este trabalho apresenta o desenvolvimento de uma aplicação Android para gerenciamento de pequenos gastos que sincroniza as informações com a internet. O autor utilizou um Web service para disponibilizar a outra aplicação web gastos previamente cadastrados em um *smartphone* com sistema operacional Android, permitindo o gerenciamento e a classificação dessas informações.

Palavras-chave

Java, Sincronização, Android, JavaServer Faces, web, gerenciamento de débitos.

ABSTRACT

SANTOS, Jonasta. Debitar: Application for Android for debt management with web synchronization / JSF through a Web service. 2012. Monografia (Especialização em Tecnologia Java) – Departamento acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

This work presents the development of an Android application for small expenses management which synchronizes the information to the internet. The author applied a Web service in order to provide to another web application the expenses previously registered in an Android smartphone, allowing its management and classification.

Keywords

Java, synchronization, Android, JavaServer Faces, web, management of debts.

LISTA DE SIGLAS

3GP	Third Generation Partnership Project
A2DP	Advanced Audio Distribution Profile
AAC	Advanced Audio Coding
AMR	Adaptive Multi-Rate
API	Perfil de Distribuição Avançada de Áudio
AVRCP	Audio/Video Remote Control Profile
BMP	Bitmap
CDMA	Code Division Multiple Access
DBA	Database Administrator
EDGE	Enhanced Data rates for GSM Evolution
EV-DO	Evolution Data Only
GIF	Graphics Interchange Format
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HE-AAC	High Efficiency - Advanced Audio Coding
HTTP	Hyper Text Transfer Protocol
IDEN	Integrated Digital Enhaced Networks
JDK	Java Development Kit
JSF	JavaServer Faces
JPEG	Joint Picture Expert Group
LTE	Long Term Evolution
MIDI	Musical Instrument Digital Interface
MMS	Multimidia Messaging Service
MP2, 3 e 4	Media Player 2 - 3 e 4
MPEG	Moving Picture Experts Group
PC	Personal Computer
PNG	Portable Network Graphics
RAM	Random Access Memory
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SO	Sistema Operacional
SOAP	Simple Object Access Protocol
TI	Tecnologia da Informação
UMTS	Universal Mobile Telecommunications System
W3C	World Wide Web Consortium
WAV	Waveform Audio
WEB	World Wide Web
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WSDL	Web Services Description Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

As pessoas estão alterando a forma com que consomem e compartilham conteúdo, além de interagirem de uma forma diferente e com mais frequência entre si. Boa parte dessa mudança está relacionada aos aparelhos móveis. Tarefas que antes eram apenas possíveis em computadores, começaram a ser suportadas em dispositivos portáteis. Trazendo um processador semelhante aos encontrados nos computadores, memória RAM e até um sistema operacional, eles começaram a se tornar cada vez mais complexos, robustos e populares.

Taurion (2011, p. 4) afirma que “A geração digital não consegue conceber a vida sem a Internet, assim como hoje os oriundos do mundo analógico já não conseguem imaginar uma vida sem eletricidade”.

No ano de 2011 a venda de *smartphones* superou pela primeira vez a de computadores. Foram vendidos 487 milhões de *smartphones*, 18% a mais que o montante de 414 milhões formados por *desktops*, *notebooks* e *netbooks*, conforme dados da empresa de pesquisa Canalys (CANALYS, 2012).

Juntamente com o número de vendas de aparelhos cresce o de aplicações. Somente em uma semana em dezembro de 2011 foram baixados 1.2 bilhão de aplicativos para Android e iOS, de acordo com dados da Flurry Analytics. Isso mostra a força com que esses aparelhos estão entrando no mercado, agora não só como meio de comunicação, mas também de entretenimento e banco de dados pessoal.

De acordo com uma pesquisa realizada pela Ericsson, 40% dos entrevistados usam com frequência o *smartphone* para navegar na internet e 33% o utilizam diariamente para acessar redes sociais e publicar informações na *web (*TELECOMPAPER, 2012).

Para que haja esta conectividade e interação direta e de forma ágil entre dispositivos móveis, servidores e computadores pessoais, um recurso se torna imprescindível: sincronização de dados.

> “À medida que mais e mais pessoas começam a depender da Internet para acessar informações, usar e-mails ou realizar transações, a própria característica de mobilidade das pessoas físicas e dos funcionários das empresas nos leva a buscar soluções que permitam explorar essas facilidades.” (TAURION, 2002. p.XV)

Algumas redes sociais, como Facebook e Twitter, têm módulos nativos para dispositivos móveis e utilizam a sincronização de dados com a *web* para manter tanto o módulo móvel quanto a *web* atualizados.

Outras empresas estão optando por interação física direta entre dispositivos. Como exemplo pode-se citar o “PadFone” da Asus. Trata-se de um *tablet* e um *smartphone* que são vendidos em conjunto. Para que os dois sejam conectados

existe um compartimento atrás do *tablet* onde o *smartphone* pode ser inserido. Uma vez conectado, tem-se acesso a todas as informações e funcionalidades do *smartphone* com a conveniência de uma tela maior.

Um aplicativo que opta por sincronização de dados é o Evernote, recentemente avaliado em um bilhão de dólares pela consultoria americana Meritech Capital Partners. Ele tem como objetivo manter na *web* textos, fotos e até páginas da *web* que foram escritos ou marcados pelo usuário em vários dispositivos distintos, fazendo com que estas informações possam ser acessadas de qualquer lugar e gerenciadas posteriormente.

Outro dado a ressaltar é uma recente pesquisa realizada pela Cisco, que estimou que em 2016 haja 1.4 *gadgets* por habitante no planeta (CISCO, 2012). Com mais de um dispositivo por pessoa a necessidade de convergir as informações se tornará ainda mais interessante para organizá-las e centralizá-las (TAURION 2011, p. 4).

Taurion (2009, p. 5) relata que “Depois de uma era tecnológica caracterizada pela ascensão do computador pessoal (PC), estamos vendo o ressurgimento da centralização”. Ele ainda afirma que “A era que se aproxima deverá trazer maior consolidação do poder de computação, integrados em uma rede massiva de servidores, a Computação em Nuvem”.

Neste estudo, será desenvolvido um aplicativo que faz a sincronização de informações de um *software* para registro de débitos para o SO Android e uma aplicação *web*. Assim poderá ser mostrado de forma prática como é feita a implementação desse tipo de aplicação e quais são as dificuldades, vantagens e desvantagens envolvidas.

1.1 PROBLEMA

Pequenos gastos diários, como cafezinhos, cigarros ou guloseimas passam despercebidos por planilhas, anotações manuscritas e outros recursos que são utilizados para controlar o orçamento.

Martins (2010, p.10), alerta “Contudo, nunca despreze pequenos valores, cuidado com gastos exagerados em alimentos, transporte e gastos domésticos. Economizar um pouco no dia a dia faz a diferença no final do mês”.

Tabela 1 - Gasto de R$ 3,00 ao dia (em um maço de cigarros) aplicado em investimento com taxa de retorno de 1% ao mês

1 ano	5 anos	10 anos	20 anos	30 anos	45 anos	60 anos
1.153	7.424	20.911	89.923	317.692	1.950.232	11.738.614

Fonte: Adaptado de Martins (2010, p. 10).

Ter a mão um meio de registrar esses gastos de maneira rápida e fácil, e posteriormente gerenciá-los pode ser uma solução interessante para este problema.

Já existe no Google Play, canal oficial da Google para baixar aplicações para dispositivos com sistema operacional Android, inúmeros softwares que permitem fazer estes registros e até apresentam alguns gráficos com os gastos dos usuários por categoria entre outras funcionalidades. Podemos citar os que estão em destaque até o presente momento no canal, são eles o Super Budget, My Money e o EasyMoney, único dessa lista com tradução para o português.

O fato é que os dados armazenados por estes aplicativos ficam somente no telefone, tornando menos cômodo o gerenciamento ágil da informação. De acordo com o W3C uma interface para dispositivos móveis de qualidade deve minimizar ao máximo a entrada de dados (W3C, 2012), e nesses softwares temos telas com vários campos a serem preenchidos, e vários cadastros que são pré-requisitos para inserir lançamentos que complicam sua utilização.

Aqui é apresentada uma possível solução para este problema levando em consideração os aspectos de sincronização de dados citados anteriormente. Para isso o usuário poderá inserir os registros de seus gastos em aparelhos com sistema operacional Android e posteriormente consultá-los na *web*.

Espera-se que desta forma a aplicação se torne ágil e simples, para ser usada no celular e ao mesmo tempo mais completa, com funcionalidades extras na *web*.

1.2 OBJETIVOS

Nesta seção são detalhados os objetivos gerais e específicos deste estudo.

1.2.1 Objetivo Geral

A objetivo geral do estudo é criar um software de gerenciamento de débitos móvel, que envie informação para internet para que essas sejam gerenciadas ou acessadas de outros dispositivos posteriormente.

1.2.2 Objetivos Específicos

Os objetivos específicos consistem em:

- solucionar a falta de uma ferramenta móvel para controle de pequenos gastos;
- criar uma aplicação Android para dispositivos móveis;
- criar um provedor de serviços que funcione tanto para dispositivos móveis quanto para páginas *web*;
- criar uma aplicação *web* que acesse um provedor de serviços e gerencie as informações;
- realizar a integração de aplicações distintas, desenvolvidas em plataformas diferentes;
- usar as ferramentas mais atuais no tocante da tecnologia Java.

1.3 JUSTIFICATIVA

Computação em nuvem e mobilidade são duas fortes tendências para o mercado de TI, de acordo com a multinacional fabricante de equipamentos de redes Cisco (COMPUTERWORLD, 2012). Utilizar estes dois conceitos para resolver um problema diário real é uma forma de explorar um contexto maior a partir de um estudo com implementação prática.

Seguindo essas tendências, o Debitar centraliza os dados que podem ser inseridos em aparelhos móveis, desde que possuam sistema operacional Android. Isso faz com que o gerenciamento das informações seja mais ágil e prático.

Outra característica é que além de ficarem armazenadas no smartphone, as informações são enviadas para um servidor. Assim, podem ser acessadas de qualquer local, e a probabilidade de perda de dados ao trocar de aparelho, por exemplo, é minimizada.

Além disso, como as funções do sistema estão disponibilizadas através de serviços (*Web service*), é possível que se desenvolvam aplicações em outras plataformas (iOS por exemplo) para acessar e inserir dados. Isso amplia a capacidade de ramificação para outros sistemas operacionais.

Tendo em vista os aspectos observados, este projeto visa implementar um software que agrega a praticidade de uma plataforma móvel e a robustez de aplicações *Web* utilizando as tecnologias Java mais recentes.

Espera-se ainda, que grande parte do que foi aprendido no decorrer do curso possa ser aplicado no projeto, visto que ele abrange um grande número de conceitos, como desenvolvimento Android, uso de Web services, banco de dados e criação de aplicações com JavaServer Faces(ver cap. 2.6).

2. REVISÃO TECNOLÓGICA

2.1 Java

Claro e Sobral (2008, p. 12) descrevem Java como a linguagem de programação orientada a objetos, desenvolvida pela Sun Microsystems, capaz de criar tanto aplicativos para desktop, aplicações comerciais, softwares robustos, completos e independentes e aplicativos para a Web. A Oracle, atual proprietária do Java, a define em seu site como uma linguagem de programação e uma plataforma de computação, que foi lançada em 1995.

Ela foi baseada nas duas linguagens de programação mais utilizadas no mundo na época, C e C++, isso fez com que tivesse de imediato uma enorme base de programadores (DEITEL 2003, p. XI). Deitel acrescenta ainda que ela é uma melhoria dessas duas linguagens, mais concisa e portável. Era suportando, então, a implementação de aplicativos para internet dentre outros “recursos que as pessoas realmente precisam”, como strings, interface gráfica, tratamento de exceções, multimídia entre outros. Desde seu lançamento foram liberadas diversas versões com novas funcionalidades e falhas corrigidas, fazendo com que a linguagem se estabilizasse (HORSTMANN E CORNELL, 2000 p.25).

Hoje o pacote de desenvolvimento, ou JDK - *Java Development Kit*, conjunto de utilitários que permite criar softwares para a plataforma Java, pode ser baixado em três versões no site da Oracle: Java SE, *Standard Edition*, é a versão padrão que contem as bibliotecas básicas. Java EE, *Enterprise Edition*, voltado para web e gerenciamento e J2ME, *Micro Edition*, utilizado em dispositivos móveis.

2.2 Android

Android é um sistema operacional móvel baseado em uma versão modificada do Linux. Foi originalmente desenvolvida por uma *startup* de mesmo nome, Android Inc. Em 2005, como parte de sua estratégia para entrar no mercado de dispositivos móveis, a Google comprou o Android e assumiu o seu trabalho de desenvolvimento e a equipe que o fazia (LEE, 2011 p.2).

Lee (2011 p.2) relata que a Google queria um SO livre e aberto, por isso o Android foi lançado sob a *Open-Source Apache License* (traduzindo livremente seria Licença Apache de código aberto). Assim quem quisesse poderia baixar seu código fonte e alterá-lo como fosse conveniente.

Então, adotar uma solução pronta e robusta no mercado e fazer versões proprietárias foi a solução para alguns fabricantes como Motorola e Sony Ericsson (LEE, 2011, p.2).

A principal vantagem da adoção do Android pelos desenvolvedores então seria que ele oferece uma abordagem unificada para desenvolvimento de aplicações, ou seja, um aplicativo produzido para este sistema operacional será capaz de rodar em inúmeros dispositivos de vários fabricantes.

A tabela 2 ilustra suas principais versões e datas de lançamento.

Tabela 2 – Versões do Android

Versão do Android	Data do Lançamento	Apelido
1.1	09 de fevereiro de 2009	
1.5	30 de abril de 2009	Cupcake
1.6	15 de setembro de 2009	Donut
2.0/2.1	26 de outubro de 2009	Eclair
2.2	20 de maio de 2010	Froyo
2.3	6 de dezembro de 2010	Gingerbread
3.0	* Somente Tablets - Janeiro de 2011	Honeycomb
* 4.0	* 19 de outubro de 2011	* Ice Cream Sandwich

Fonte: Adaptado de Lee (2011, p. 2).

(*) dados adicionados pelo autor para que as informações sejam as mais atuais possíveis.

2.2.1 Arquitetura do Android

Lee (2011 p.4) explica que o sistema Android é dividido em cinco sessões localizadas dentro de quatro camadas principais. São elas:

- Kernel do Linux (*Linux Kernel*): núcleo Linux no qual o Android é baseado, nesta camada estão todos os *drivers* de baixo nível para os componentes de *hardware* do aparelho.

- Bibliotecas (*Libraries*): contém todo o código que disponibiliza as principais características do SO Android. Nesta camada estão, por exemplo, a biblioteca do SQLite que fornece suporte a banco de dados e o WebKit, utilizado para se ter acesso à navegação na *web*. Essas bibliotecas podem ser consumidas pelas aplicações para se ter as respectivas funcionalidades.

- Tempo de execução (*Runtime*): localizado na mesma camada que as bibliotecas, fornece um conjunto de instruções que permite que sejam escritas aplicações para Android em Java. Também inclui a máquina virtual Dalvik, projetada especificamente para dispositivos alimentados com bateria e que tenham poder de processamento limitado, que permite que cada aplicação rode seu próprio processo, com sua própria instancia da máquina virtual.

- *Framework* de Aplicação (*Application Framework*): disponibiliza funcionalidades do SO para que possam ser utilizadas pelos desenvolvedores em suas aplicações.

- Aplicações (*Applications*): camada superior, contém tanto as aplicações nativas do aparelho (contatos, navegador, etc.) como as instaladas posteriormente no aparelho.

A figura 1 representa graficamente o que foi explicado acima:

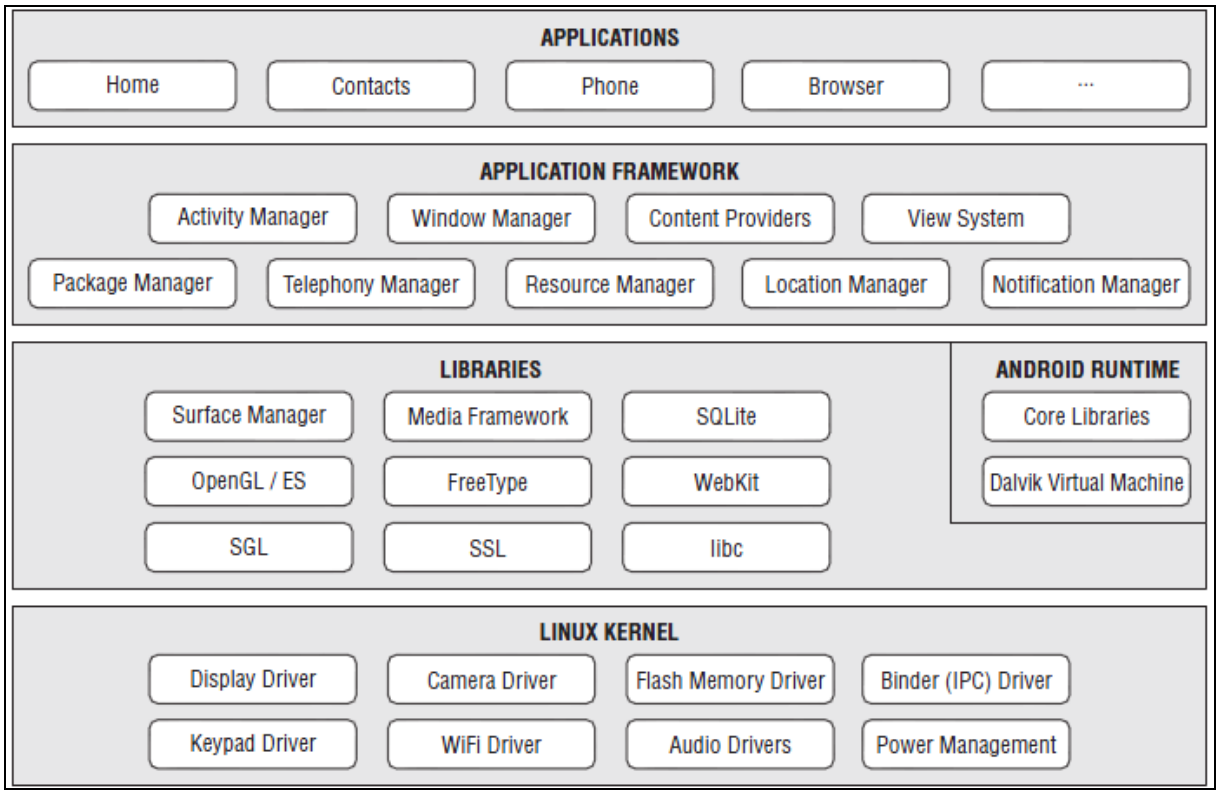


Figura 1: Arquitetura do Android
Fonte: Adaptado de Lee (2011, p. 3).

2.2.2 Características do Android

Como o Android tem código aberto e é disponibilizado gratuitamente para as fabricantes modificarem, não possui configurações fixas de *software* e *hardware.* No entanto, nativamente tem as seguintes características (Lee, 2011 p.3):

- Armazenamento - usa o SQLite, um banco de dados relacional leve, para armazenamento de dados.

- Conectividade - suporta GSM / EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (inclui A2DP e AVRCP), WiFi, LTE e WiMAX.

- Mensagens - suporta SMS e MMS.

- Navegador da Web – baseado no projeto de código aberto WebKit, e melhorado com o mesmo suporte a JavaScript utilizado na versão 8 do Chrome.
- Suporte Multimídia - inclui suporte para os seguintes formatos: H.263, H.264 (em 3GP ou MP4), MPEG-4 SP, AMR, AMR-WB (em 3GP), AAC, HE-AAC (MP4 ou em 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF e BMP
- Características físicas - acelerômetro, câmera, bússola digital, sensor de proximidade e GPS
- Multi-toque - suporta telas multi-touch.
- Multi-tarefa – suporta aplicações multi-tarefa.
- Flash – a partir do Android 2.3 suporta Flash 10.1.
- *Tethering* - suporta compartilhamento de conexões com a Internet como um *hotspot* com fio / sem fio.

2.2.3 Android Market

Como citado anteriormente, um dos fatores mais importantes para definir o sucesso de uma plataforma móvel é o número de aplicações disponíveis para ela.

Em 2008 o Google anunciou o Android Market (atual Google Play), uma loja online de aplicativos para dispositivos Android. A partir daí também passou a vir nativamente um aplicativo nos aparelhos pelo qual os usuários podiam facilmente baixar aplicações de terceiros (LEE, 2011, p.6).

Hoje, qualquer desenvolvedor pode abrir uma conta no Google Play e disponibilizar suas aplicações para *download,* sendo elas pagas ou não. Mediante o pagamento de uma taxa de registro de U$25 o aplicativo é distribuído pelo Market e sua disponibilidade pode ser gerenciada pelo desenvolvedor.

2.2.4 Android SDK – Kit de Desenvolvimento de Software para Android

O Kit de Desenvolvimento de Software para Android fornece as ferramentas e APIs necessárias para desenvolver aplicações na plataforma Android usando a linguagem de programação Java.

Ele é composto por exemplos de projetos com código fonte, ferramentas, como emuladores de SO Android para que as aplicações sejam testadas, e as bibliotecas necessárias para construção de aplicativos.

O SDK conta ainda com a Dalvik, uma máquina virtual personalizada para uso em dispositivos móveis que possuiu um núcleo baseado no Kernel do Linux.

2.3 SQLite

SQLite é um banco de dados relacional embutido e de código aberto. Originalmente lançado em 2000, foi projetado para fornecer uma forma mais leve para os aplicativos gerenciarem dados sem a sobrecarga que geralmente é uma característica de um banco de dados dedicado. Ele tem reputação por ser altamente portátil, fácil de usar, compacto, eficiente e confiável (OWENS, 2006, p.1).

É utilizado no sistema operacional Mac OS X da Apple, no navegador Safari, nos projetos Mozilla Firefox, Thunderbird e SoundBird e no sistema operacional Symbian (OWENS, 2006, p.4).

Apesar de ser leve, oferece uma surpreendente variedade de recursos, dentre elas os principais são (OWENS, 2006, p.8):

- Configuração Zero (*Zero Configuration*): desde o princípio foi projetado para que não necessite de um DBA. Configurar e administrar o SQLite é extremamente simples, um só programador é capaz de fazê-lo.

- Portabilidade (*Portability*): foi projetado com o objetivo de ser portátil, ele compila e funciona em Windows, Linux, BDS, Mac OS X, sistemas comerciais Unix como Solaris, HPUX, e AIX. Também funciona em muitas plataformas embarcadas como QNX, VxWorks, Symbian, Palm OS, Windows CE e atualmente também vem junto com o Android.

- Compacidade *(Compactness):* foi projetado para ser leve e autossuficiente, sendo assim, não necessita de um servidor de banco de dados instalado, basta apenas um arquivo de cabeçalho e uma biblioteca com menos de 180 *kilobytes* para que possa ser usado.

- Simplicidade (*Simplicity*): as APIs do banco são simples e fáceis de usar, também são bem documentadas e intuitivas. Foram feitas para ajudar a customizar o SQLite da maneira necessária, como por exemplo criar uma função SQL em C. A própria comunidade de desenvolvimento disponibiliza bibliotecas que facilitam o acesso a ele em várias linguagens de programação como : Java, PHP, Delphi, Ruby entre outras.

- Flexibilidade (*Flexibility*): vários fatores tornam o SQLite muito flexível. Como um banco de dados embarcado, oferece o melhor de dois mundos: o poder e flexibilidade de banco de dados relacional com a simplicidade e compacidade de uma *B-tree.* Assim, não há necessidade de se preocupar com configuração, problemas de rede ou limitações de plataforma.

- Código Aberto (*Liberal Licensing*): todo o código do SQLite está sob domínio público. Não há licença ou impedimentos sobre o uso: pode ser modificado, incorporado, distribuído, vendido e usado para qualquer propósito - comercial ou não - sem quaisquer restrições ou pagamento de *royalties*.

- Confiança (*Reliability*): o código não é somente aberto, mas também bem escrito. Consiste em 30.000 linhas de ANSI C, que é limpo, modular e bem comentado. O objetivo é que seja fácil de entender e customizar.

- Confortável *(Convenience):* devido a algumas facilidades nativas do SQLite é bem confortável trabalhar com esta ferramenta. Isso inclui tipagem dinâmica, resolução de conflitos e a capacidade de "agrupar" vários bancos de dados em uma só sessão.

A pesquisa sobre SQLite não será aprofundada por não ser o objetivo deste estudo, mas o fato dele vir como banco "padrão" do Android e ser utilizado no projeto fez com que fossem citadas algumas de suas características.

2.4 Netbeans

Netbeans é uma IDE, ou seja, um ambiente integrado de desenvolvimento que inicialmente foi projetado para uso com Java. Suas versões mais recentes suportam também JavaFX, C, C++, Groovy e PHP. Scala, Ruby e outras linguagens são suportadas através da instalação de plugins (HEFFELFINGER, 2011, p.7).

É mais fácil descrevê-lo através de seus recursos. Os principais são listados a seguir (KOGENT, 2011, p. 2).

- Idiomas adicionais: já vem com suporte a vários idiomas, como inglês, japonês, chinês simplificado e português do Brasil.

- Editor de código aperfeiçoado: completa automaticamente palavras-chave, campos e variáveis. A navegação entre os códigos é facilitada e um bloco pode ser salvo como *template* para uso futuro.

- *Profiler* do Netbeans: usado para verificar se a aplicação tem problemas com uso de memória ou falhas de desempenho.

- Aplicação *desktop*, com *Swing* e banco de dados: facilita a criação deste tipo de aplicação através de *templates*.

- Usa *Beans:* torna mais rápido a implementação de Beans na aplicação, que por sua vez facilita a retenção de dados recuperados do banco de dados.

- Controle de versão: o sistema de controle de versão (VCS) pode ser utilizado para compartilhamento de projetos e para unificar o trabalho de vários desenvolvedores, sendo muito útil quando se trabalha com desenvolvimento em grupo.

2.5 *Plugin* NBAndroid para Netbeans

NBAndroid é um conjunto de módulos para o Netbeans que fornece suporte para desenvolvimento de aplicativos Android. Ele é distribuído sobre a licença Apache 2.0 (www.nbandroid.org).

É composto por vários módulos que desempenham diferentes tarefas, são listados a seguir (www.nbandroid.org).

- Suporte ao *kit* de desenvolvimento do Android (Core Android SDK Support): faz uma ponte entre a IDE, dispositivos conectados e todas as ferramentas do Android SDK.

  Para que funcione corretamente deve ser configurado o caminho do SDK que deve ser utilizado. Para que seja parametrizado no NetBeans deve-se navegar até o menu ferramentas >> opções >> Seção Miscelânea >> Aba Android.

- Suporte a criação de projetos Android (Android Project Support) : o suporte a projetos foi projetado para ser compatível com as ferramentas de linha de comando do Android SDK. Isto significa que o leiaute e o sistema de construção da aplicação é o mesmo utilizado em outras plataformas que utilizem este mesmo princípio (Eclipse por exemplo) e não é muito trabalhoso trocá-las se necessário.

  Ele disponibiliza ainda um assistente para criação de novos projetos Android, que ajuda na seleção da versão do Android para qual se destina a aplicação entre outros detalhes. Ele ainda disponibiliza os projetos de teste do Android SDK, que são exemplos que vem junto com o SDK, e um ambiente integrado de depuração.

A figura 2 mostra a estrutura padrão de um projeto criado no Netbeans utilizando o *plugin* NBAndroid.

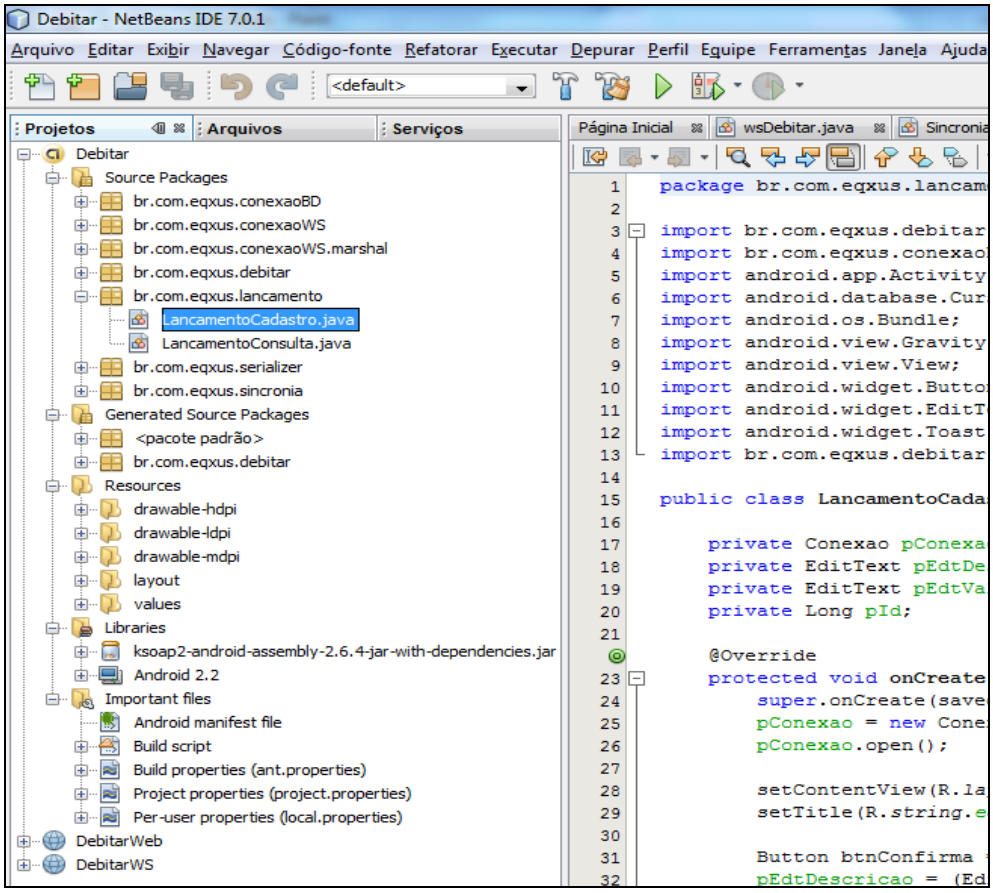


Figura 2: Estrutura de um projeto Android na IDE Netbeans com NBAndroid
Fonte : Arquivos do autor.

- Visualizador do LogCat (LogCat viewer): LogCat é o sistema de *log* de eventos do Android, ele permite que a aplicação lance mensagens com vários níveis de prioridade (erro, aviso, etc.) (ZECHNER, 2011 p.46). O NBAndroid tem um visualizador para LogCat que se integra com a interface do NetBeans.

- Editores aperfeiçoados para XML do Android (*Improved editors for Android XML files*): reconhece e completa as *tags* utilizadas no ambiente Android, como as referentes ao AndroidManifest por exemplo.

2.6 JSF

JavaServer Faces é um framework Java para construção de interface para aplicações web. A sua principal vantagem é que ele simplifica a criação da interface para o usuário, que geralmente é uma das partes mais complexas no desenvolvimento de aplicações *web* (BURNS;SCHALK, 2010, p 3).

Ele evita problemas de desenvolvimento ou manutenção que poderiam ocorrer se fossem usados apenas *sevlets* ou *JavaServer Pages,* pois é um

framework robusto, que possui padrões de desenvolvimento bem definidos e foi construído com base em vários outros pré-existentes (BURNS;SCHALK, 2010, p. 3).

A construção é facilitada das seguintes maneiras (BURNS;SCHALK, 2010 p 4).

- O framework possui uma abordagem centralizadas em componentes (*componente-centric*) e independente de clientes específicos (*cliente-independent*) para construção de interface. Isso aumenta a produtividade e torna-o mais fácil de usar.
- Simplifica o acesso e gerenciamento de dados da aplicação através da interface.
- Gerencia automaticamente o *status* da interface durante múltiplas requisições de múltiplos clientes de uma maneira simples e transparente
- É amigável tanto para desenvolvedores inexperientes quanto para os que precisam de recursos mais específicos.
- Ele descreve um conjunto de padrões de arquitetura para uma aplicação web.

Para BURNS e SCHALK (2010, p. 4) ele é resultado de anos de experiência *web* unificados em um único ambiente, e torna o desenvolvimento altamente produtivo e facilitado sem sacrificar a performance e flexibilidade.

2.7 PostgreSQL

PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional (SGBDOR). É um projeto de código aberto que começou com o nome Ingres na universidade da Califórnia, em Berkeley. Ele é largamente difundido com o SGBD de código aberto mais avançado do mundo e é rico em recursos, dentre eles (DRAKE;WORSLEY, 2002, p.6):

- Sistema de gerenciamento objeto-relacional (*Object-Relational DBMS*): tem uma abordagem objeto-relacional com os dados, e é capaz de gerenciar rotinas e regras complexas. Tem suporte a consultas SQL declarativas, controle de concorrência multi-versão, multi-usuário, transações, otimização de consultas, herança e vetores.
- Altamente extensível (*highly extensible*): suporta operadores definidos pelo usuário, funções, métodos de acesso e tipagem de dados.

- Abrangente suporte a SQL (*comprehensive SQL support*): suporta a especificação SQL99 e inclui recursos avançados como por exemplo as junções do SQL92.
- Integridade referencial (*referential integrity*): suporta integridade referencial, que garante a solidez dos dados do banco.
- API flexível (*flexible API*): a flexibilidade da API permitiu que os fornecedores tornassem o desenvolvimento com PostgreSQL mais fácil. Dentre eles Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, TCL, C/C++ e Pike.
- Linguagens procedurais (*procedural languages*): possui uma linguagem procedural própria e nativa chamada PL/pgSQL, que pode ser comparada a PL/SQL do Oracle. Outra vantagem é que Perl, Python e TCL podem ser usadas como linguagens procedurais embutidas.
- CCMV - controle de concorrência multi-versão (*MVCC - multiversion concurrency control*): faz o gerenciamento de quem está lendo ou gravando dados no banco. Assim evita os travamentos na leitura que são ocasionados por outros usuários que estão inserindo dados ao mesmo tempo.
- Cliente/servidor (*cliente/server*): usa um processo por cliente/servidor, similar ao método de gerenciamento de processo do *Apache 1.3*.x. E há um processo maior que fornece as conexões.
- LTWA - Log de transações *write-ahead (write ahead logging):* aumenta a segurança do banco de dados através da tecnologia WAL, que faz o log das alterações antes que sejam gravadas no banco

2.8 Primefaces

Primefaces é uma suíte de componentes para JSF, possui código aberto e é distribuído sob licença Apache V2. Foi criado e é mantido pela Prime Teknoloji, uma empresa de desenvolvimento turca especializada em consultoria JAVA EE e Agile (ÇIVICI, 2012, p.11).

Para que seja utilizado basta o *download* e configuração de uma única biblioteca, no entanto nas versões mais recentes do Netbeans ele já vem como um componente nativo não necessitando configurações (ÇIVICI, 2012, p.12).

Suas principais vantagens, de acordo com Çivici (2012, p.11), autor do guia oficial do usuário, são:

- Possui um rico conjunto de componentes, tais como diálogos, gráficos, *AutoCompletes* entre outros.
- Ajax embutido baseado nas APIs padrões do JSF.
- Leve, composto de apenas uma biblioteca, não precisa de configuração ou de dependências.
- Suporte nativo a Ajax Push via *websockets*.
- Possui um *Mobile UI Kit,* para criação de aplicações *web* em dispositivos móveis.
- Vasta documentação.
- Comunidade de usuários grande, participativa e ativa.

A figura 3 ilustra alguns de seus componentes.

Figura 3: Alguns componentes Primefaces
Fonte : Adaptado do guia do usuário disponibilizado pelo fabricante.

2.9 Glassfish

Glassfish é um servidor de aplicações Java EE, possui código aberto e é disponibilizado gratuitamente sob as licenças Common Development and Distribution License (CDDL) e General Public License (GPL) versão 2 (HEFFELFINGER, 2007, p.6).

Ele fornece todas as bibliotecas necessárias para que se desenvolva e implante aplicativos compatíveis com as especificações Java EE (HEFFELFINGER, 2007, p.6). Foi desenvolvido pela Sun Microsystems, responsável pelo Java, em 2005 baseado no servidor Tomcat. Além da versão gratuita possui uma licenciada, que conta com suporte e ferramentas adicionais, fazendo com que os desenvolvedores tenham uma maior confiança nessa ferramenta (GONCALVES, 2010, p. 34).

2.10 MVC

O MVC é um dos padrões de projetos mais importantes na área da computação. Enquanto alguns padrões procuram resolver problemas específicos ele descreve toda a arquitetura do sistema com objetos, podendo ser aplicado em parte ou em todo o projeto (BUCANEK, 2009, p.353).

Seus fundamentos estão ilustrados na figura 4, a seguir.

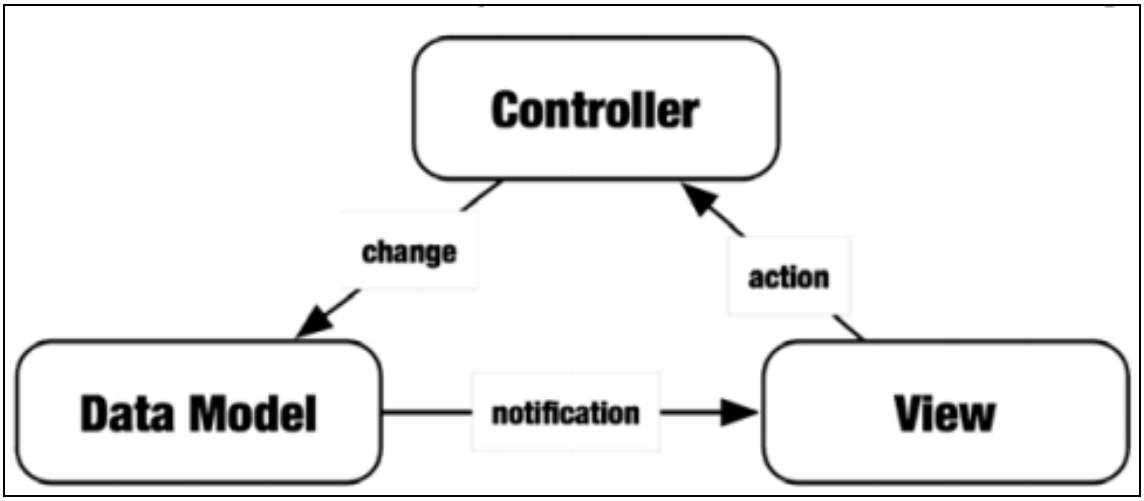


Figura 4: Comunicação fundamental do MVC
Fonte : Adaptado de Bucanek (2009, p.354).

Ele não é tão rígido quanto outros padrões, isso deixa margens para implementações alternativas de acordo com a necessidade, é mais uma filosofia que uma receita (BUCANEK, 2009, p.354).

Bucanek (2009, p.354) o define da seguinte maneira:

- Objetos do modelo de dados (*Model*): mantém, encapsulam e abstraem os dados, eles não contém métodos com lógica para a aplicação funcionar. Por exemplo, um modelo deve implementar um método que serializa os dados, mas não pode ter um método “Salvar”. Este segundo deve ser implementado no *controller*.

- Objetos de visão (*View*): mostra as informações do modelo para o usuário. Também captura as interações dele, como cliques e pressionamento de teclas, e os converte em “ações” que são passados para o *controller* processar.

- Objetos de controle (*Controller*): implementa as funções e métodos da lógica da aplicação.

Quando o usuário faz uma interação, como digitar um atalho do teclado, a camada de visão o interpreta e envia uma mensagem de ação para a camada de controle. Ela executa a ação solicitada, que geralmente envolve processar o modelo de dados. Quando há mudança no modelo de dados, observadores notificam a visão, que altera o que está sendo exibido (BUCANEK, 2009 p.355).

2.11 Web service

Para SNELL (2001, p.6) Web service é uma aplicação que pode ser acessada através de uma interface disponível na *Web*, feita com tecnologias padrões para *internet*. Ou seja, se um aplicativo pode ser acessado usando uma combinação de protocolos como HTTP, XML, SMTP ou Jabber, então é um *Web service*.

O esquema funcional de um Web service pode ser visto na figura 5.

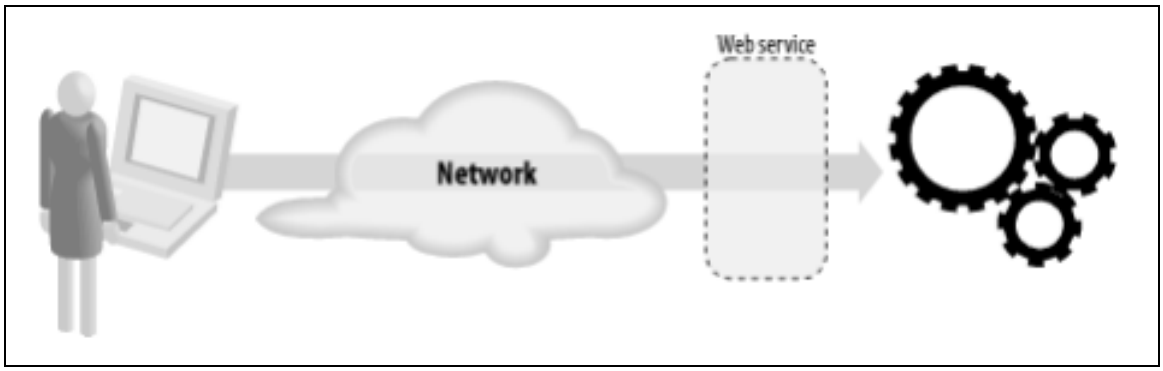


Figura 5: Web service permite acesso à aplicação usando tecnologias padrões de internet. Fonte : Adaptado de Snell (2001, p.6).

Ele fornece um padrão para integração de diferentes aplicações desenvolvidas em diferentes plataformas através da disponibilização da descrição de seus serviços (WSDL). Através desta descrição outros sistemas sabem como o Web service funciona e podem interagir com ele através de mensagens SOAP, que geralmente são transportadas usando HTTP com XML e outras tecnologias web (W3C.org).

2.12 kSOAP

O site oficial do kSOAP o descreve como uma biblioteca usada para criar clientes para *Web services* SOAP, ele foi projetado para ser usado ambientes Java restritos como Applets, J2ME e Android (ksoap2.sourceforge.net).

SOAP é o protocolo padrão para compartilhamento de mensagens entre *softwares* distintos. Ele é uma aplicação da especificação XML, e se baseia fortemente em seus padrões *XML Schema* e *XML Namespaces* para a sua definição e função. A especificação o define como nada mais do que um simples envelope baseado em XML para a informação que está sendo transferida, e um conjunto de regras para a tradução de tipos específicos de dados de aplicações e plataformas para uma representação XML (SNELL, 2001, p.15).

As bibliotecas padrões do Java que oferecem suporte a SOAP são muito pesadas para dispositivos móveis, por isso kSOAP surge como uma boa opção. O problema é que por ser reduzido não suporta toda a especificação SOAP (YUAN, 2004, p.309).

Ele faz a conversão de dados de mensagens SOAP em objetos Java, de forma que o desenvolvedor trabalhe com essa tecnologia de forma transparente. O programador apenas alimenta um escritor SOAP com objetos Java, envia a mensagem, aguarda a resposta do servidor e lê objetos Java diretamente do conversor SOAP da biblioteca (YUAN, 2004, p.310).

Para enviar uma mensagem usando kSOAP basta seguir os seguintes passos (YUAN, 2004, p.311):

- Preparar os argumentos para passar para o método remoto. Instancia-se um *SoapObject* e adiciona-se os argumentos de chamada através do método *addProperty().*
- Preparar a chamada. Instancia-se um objeto *HTTPTransport* com a URL da interface SOAP.
- Faz a chamada do método remoto. É passado o *SoapObject* instanciado para a chamada (método *call())* do *HTTPTransport,* o valor do retorno do método fica armazenado na propriedade *return* do método *call().*

A figura 6, a seguir, mostra em detalhes como fazer uma chamada SOAP usando kSOAP.

```
// The URL of the SOAP interface
String endPointURL = "http://api.google.com/search/beta2"
String licenseKey = "Register with Google to get it";

public String spellCheck (String query) throws Exception {
  // Prepare request SOAP message in a memory object
  SoapObject method = new SoapObject("urn:GoogleSearch",
                                     "doSpellingSuggestion");
  method.addProperty("key", licenseKey);
  method.addProperty("phrase", query);

  // Prepare SOAP RPC call object.
  HttpTransport rpc = new HttpTransport(endPointURL, "\"\"");
  // Google uses 1999 SOAP standards.
  ClassMap cm = new ClassMap(Soap.VER10);
  rpc.setClassMap (cm);
  // Conduct RPC call through HTTP and
  // directly get results
  String spellSugg = (String) rpc.call (method);
  return spellSugg;
}
```

Figura 6: Verificação Ortográfica do Google usando kSOAP
Fonte : Adaptado de Yuan (2004 p.311).

3. DESENVOLVIMENTO

A figura 7 ilustra uma visão geral da solução proposta, ela foi resultado da análise dos objetivos citados no primeiro capítulo e atinge todos os requisitos propostos.

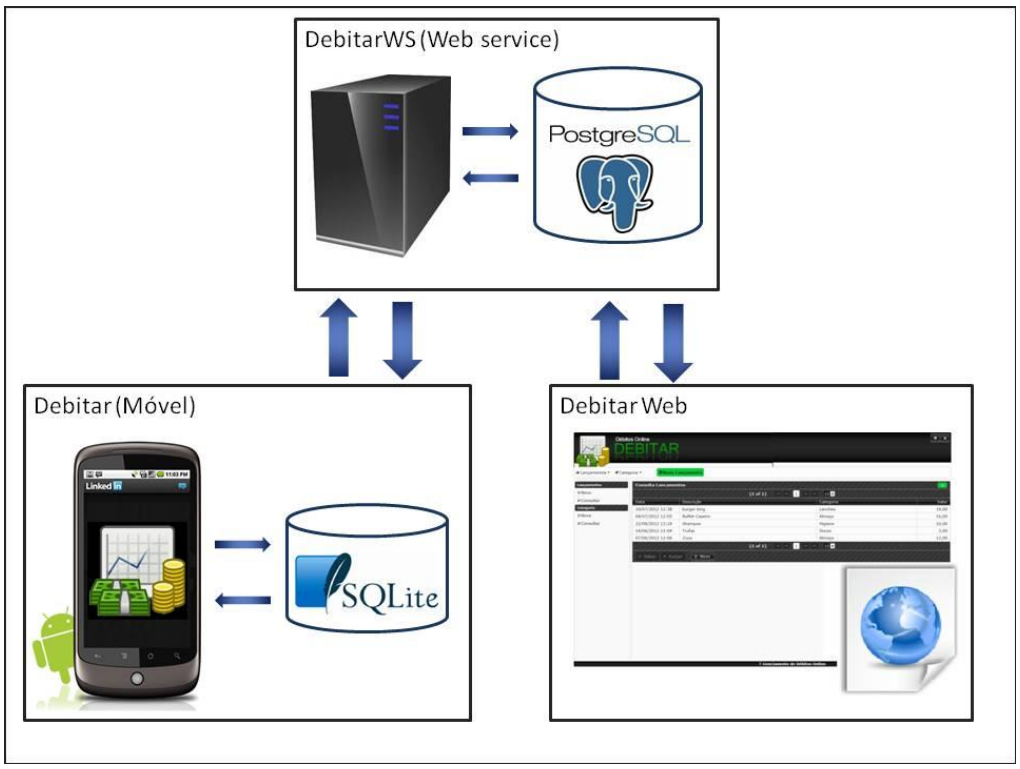


Figura 7: Visão geral do software
Fonte: Arquivo do autor.

Como visto na figura 7 acima, o software foi dividido em três módulos:

- Módulo Móvel – Debitar para Android: consiste na aplicação nativa para sistema operacional Android. Nele será possível inserir os gastos, editá-los e sincronizar com o módulo web.

- Módulo Web – Debitar Web: este módulo é responsável por gerenciar as informações previamente inseridas no Debitar para Android. Ele foi implementado utilizando o framework JSF – Java Server Faces, baseado em Java.

  O usuário pode visualizar as informações que foram sincronizadas dos dispositivos móveis e classificá-las em categorias. As categorias são gerenciadas exclusivamente neste módulo.

- Módulo Centralizador – WebService: é um conjunto de serviços que capta as informações digitadas no Debitar e grava na base de dados do servidor, que posteriormente é acessada pelo Debitar Web. Também manda as alterações para o módulo móvel, quando solicitado pelo usuário.

O fluxo mais simples de uso seria o usuário registrar os gastos pelo módulo móvel conforme fosse necessário, e posteriormente sincronizá-los. A sincronização faria com que fossem gravados os dados no servidor através de funções do Web service. Depois disso os lançamentos poderiam ser acessados, gerenciados e classificados pelo módulo *web*.

3.1 Diagrama de Casos de Uso geral

Na figura 8, abaixo, pode ser visualizado o diagrama de Casos de Uso geral do sistema. Os diagramas mais detalhados de cada módulo constam nos apêndices 6.1 a 6.3.

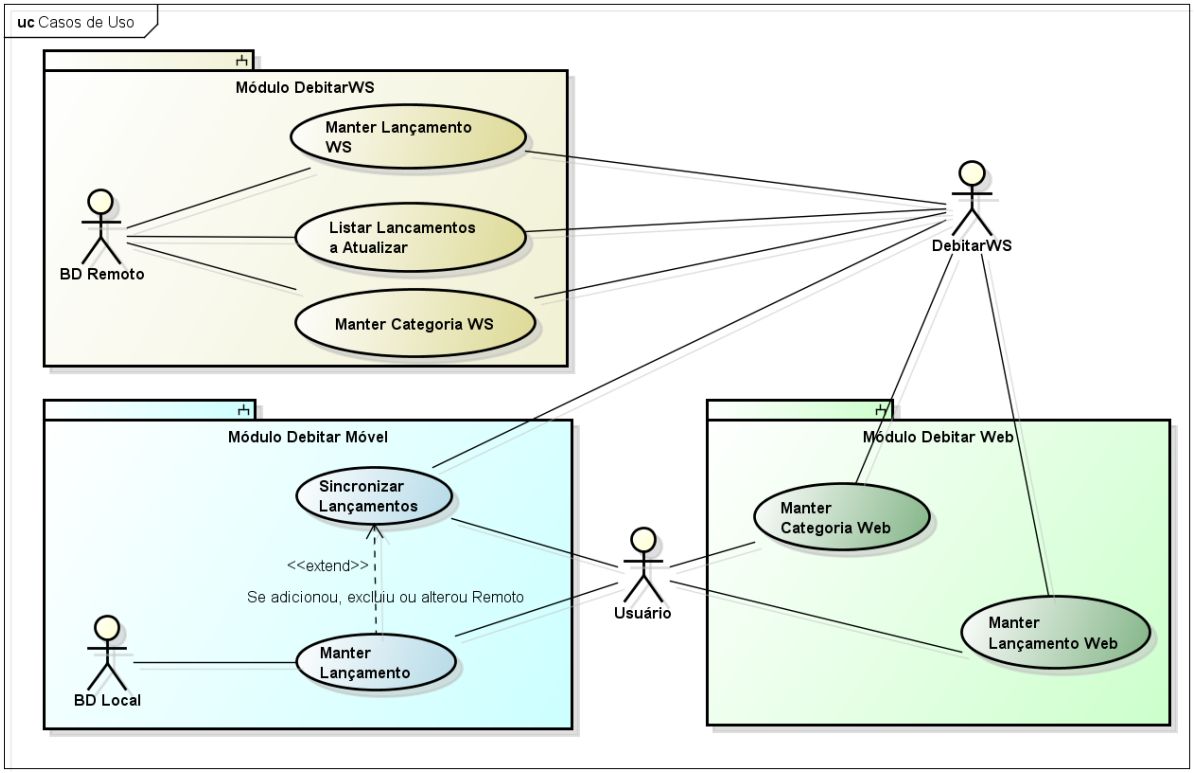


Figura 8: Diagrama de Casos de Uso geral do sistema
Fonte: Arquivo do autor.

Os atores foram definidos da seguinte forma:

- Usuário: pessoa que irá interagir com o sistema;
- BD Local: banco de dados local do dispositivo móvel, foi utilizado SQLite;
- DebitarWS: Web service que faz a integração do sistema;
- BD Remoto: banco de dados remoto, fica no servidor junto com o Web service, foi utilizado PostgreSQL.

Os casos de uso do módulo Debitar Móvel são os seguintes:

- Manter Lançamento: este caso de uso visa o cadastro, alteração, exclusão e consulta de lançamentos no banco de dados local;

- Sincronizar Lançamentos: referente à sincronização, ou seja, troca de dados com o Web service. Se eventualmente um lançamento foi alterado, excluído ou incluído remotamente, ele será atualizado no banco de dados local e vice-versa.

Para o módulo Debitar Web foram atribuídos os casos de uso abaixo:

- Manter Lançamento Web: este caso de uso faz com que as funções remotas do Web service sejam utilizadas pelo módulo web, para que sejam adicionados, excluídos, alterados ou consultados os lançamentos do banco de dados remoto.

- Manter Categoria Web: permite que sejam utilizadas, pelo módulo web, as funções remotas do Web service para que sejam adicionadas, excluídas, alteradas ou consultadas as categorias do banco de dados remoto.

E finalizando, os casos de uso do módulo DebitarWS:

- Manter Lançamento WS: faz o cadastro, alteração, exclusão e consulta dos lançamentos que estão no banco de dados remoto;

- Manter Categoria WS: referente ao cadastro, alteração, exclusão e consulta das categorias armazenadas no banco de dados remoto;

- Listar Lançamentos A Atualizar: permite que sejam listados os lançamentos que devem ser atualizados no dispositivo móvel.

Para maior esclarecimento, cada um dos três módulos será explanado detalhadamente nos capítulos a seguir.

3.2 Módulo Móvel – Debitar para Android

Para desenvolvimento deste módulo foi utilizado o Netbeans 7.0 em conjunto com o plugin NBAndroid. A estrutura básica do projeto foi baseada no “Notepad tutorial”, um guia oficial do Google para introdução ao desenvolvimento Android disponível em “http://developer.android.com/training/notepad/index.html”.

A aplicação foi desenvolvida tendo como alvo (*Runtime target*) a versão 2.2 do Android, por ser a versão mais estável na época do início do projeto. É composta de três telas: consulta, cadastro e sincronização, que são visualizadas na figura 9, a seguir.

Figura 9: Interfaces do software Debitar, consulta, cadastro e sincronização
Fonte: Arquivo do autor.

Quando o usuário abre a aplicação, a tela exibida é a de consulta, ela mostra os lançamentos cadastrados em ordem decrescente de data. Para que um lançamento seja editado basta que se clique nele, direcionando assim para a tela de cadastro. Com relação à exclusão, o usuário deverá manter pressionado o lançamento a excluir, o sistema mostrará um diálogo para confirmar a exclusão.

Para cadastrar um novo lançamento, deve ser pressionado o botão “Adicionar”, o sistema vai para a tela de cadastro já com a data e hora preenchida, sugerindo o momento da inclusão, e com *hints* para preenchimento dos outros campos. Após inserido os dados deve-se clicar em “Confirmar” para completar a operação.

Quando for mais conveniente, desde que tenha conexão com a internet, ele pode abrir a aba de sincronização para efetuar o envio dos lançamentos para web. Após pressionado o botão “Sincronizar” o sistema buscará no banco de dados local os lançamentos que foram alterados, excluídos ou adicionados para que sejam sincronizados, esse procedimento foi denominado sincronização local. Após isso são solicitados todos os lançamentos que foram atualizados no módulo web para o smartphone, a chamada sincronização remota.

Essas operações são compreendidas em detalhes nos diagramas de sequência constantes nos apêndices 6.9, 6.10 e 6.11.

Com relação às ferramentas utilizadas, o Netbeans se mostrou bastante competente no que tange o suporte ao desenvolvimento e integração com o NBAndroid. Esse ambiente de desenvolvimento, diferente do proposto pelo Google que sugere o uso do Eclipse, foi bastante satisfatório.

Após a instalação do *plugin,* que se integra com a interface do Netbeans, é dado suporte completo a criação de projetos Android, e uma estrutura básica é criada quando se inicia um projeto, contendo uma *Activity* e uma interface. No quadro de depuração do Netbeans é criada mais uma aba referente ao LogCat (ver capítulo 2.5) e na guia “serviços” são mostrados os dispositivos conectados para que sejam gerenciados.

Com relação às desvantagens deste ambiente, o NBAndroid ainda não oferece suporte ao desenvolvimento de interfaces, necessitando que os XMLs sejam criados manualmente. E apresenta certa instabilidade, fazendo com que o projeto tenha que ser desnecessariamente limpo (ou “construído” no Netbeans) várias vezes quando se faz uma alteração nas interfaces ou no arquivo de configuração “Android Manifest.xml”.

Outro problema encontrado foi o fato do visualizador do LogCat ter parado de funcionar, prejudicando assim a depuração da aplicação visto que não podiam ser logadas mensagens de erro em tempo de execução.

Para que fossem consumidas as funções do Web service, necessárias para a sincronização de dados, foi utilizada a biblioteca kSoap. Embora ela seja bastante estável e completa possui pouca documentação, o que fez com que o desenvolvimento da função de sincronização demorasse mais que o previsto.

Apesar dos imprevistos, o desenvolvimento deste módulo ocorreu de forma satisfatória. Com ele foram atingidos os dois primeiros objetivos deste estudo, a criação de uma ferramenta para controle de gastos móveis e desenvolvimento de uma aplicação Android.

3.2.1 Estrutura do projeto Debitar

A estrutura foi dividida da seguinte forma:

- conexãoBD: pacote com as classes para conexão ao banco de dados do telefone, o SQLite;
- conexãoWS: possui as classes que fazem o acesso as funções do Web service, são elas que utilizam a biblioteca do kSOAP. Possui um sub-pacote “marschal” para tipos de dados complexos;
- debitar: tem a classe principal do projeto: “DebitarActivity”, que herda da classe para abas “TabActivity”. É a primeira classe a ser chamada quando a aplicação é compilada, e possui dentro dela as *activities* de consulta, cadastro e sincronização de lançamentos;
- lancamento: classes que tratam da consulta e cadastro do lançamento, elas acessam as classes do pacote conexaoDB;

- sincronia: classe para sincronização de dados com o Web service, ela acessa tanto o pacote conexaoWS quanto o conexaoDB;
- layout: pasta padrão em projetos Android que contem os XMLs que descrevem as telas.

Na figura 10 é possível visualizar a estrutura do projeto Debitar.

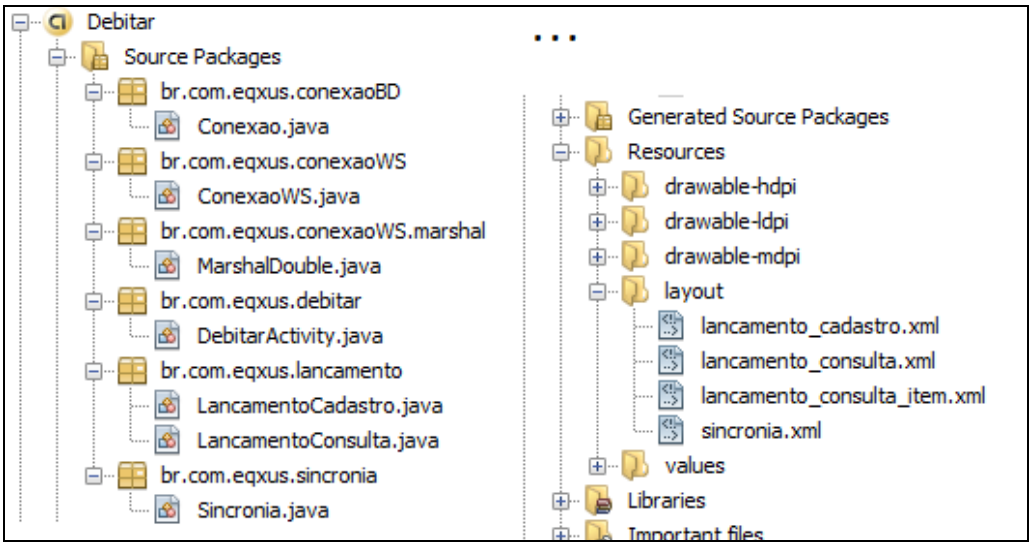


Figura 10: Estrutura do projeto Debitar
Fonte: Arquivos do autor.

3.3 Módulo Web – Debitar Web

Para desenvolvimento deste módulo foi utilizado o Netbeans 7.0, servidor Glassfish 3.0 e o framework JSF 2.0 com Primefaces.

Aqui é feito o gerenciamento das informações que foram inseridas e sincronizadas pelo dispositivo móvel. Tudo o que foi enviado para o Web service pelo dispositivo móvel pode ser alterado ou excluído e atualizado novamente no servidor. Assim, quando for efetuada uma nova sincronização os dados atualizados serão enviados para o módulo móvel.

Um diferencial é que nesta aplicação podem ser criadas categorias, e os lançamentos podem ser divididos em grupos.

Figura 11: Cadastro de nova categoria
Fonte: Arquivos do autor.

Para se cadastrar uma nova categoria o usuário deve clicar no link “Nova” na guia “Categoria”, o sistema redirecionará para a tela de cadastro de categoria, visualizada na figura 11. Depois de preenchida a descrição, deve ser pressionado o botão “Salvar” para completar a operação.

Elas podem ser consultadas e alteradas através da tela “consulta categorias”, disponível no link “Consultar” da guia “Categoria”. Não podem ser excluídas as que possuam lançamentos vinculados. A figura 12 mostra a tela de consulta de categorias.

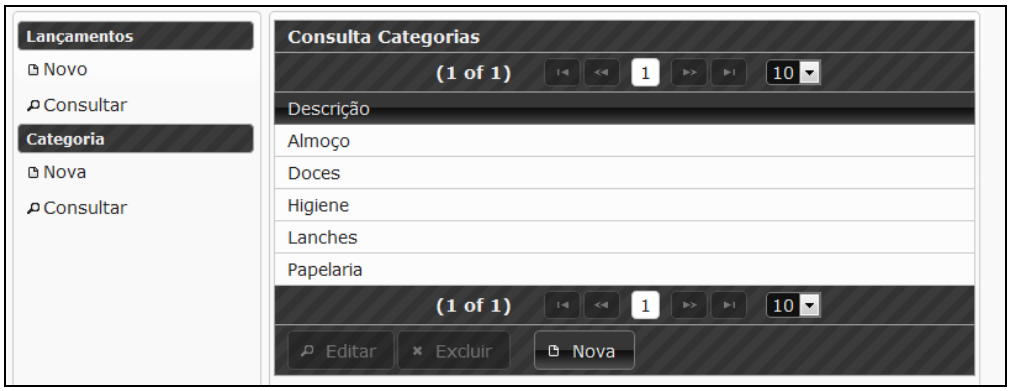


Figura 12: Tela de consulta de categorias
Fonte: Arquivos do autor.

O funcionamento das telas foi padronizado de forma que o uso do sistema ocorra de forma intuitiva, para fazer o gerenciamento dos lançamentos, por exemplo, os passos a seguir são semelhantes aos do cadastro de categorias.

Para inserção de um novo lançamento deve-se clicar no link “Novo” na guia “Lançamento”, o sistema redirecionará para a página de cadastro de lançamentos. Após o preenchimento dos campos deve ser pressionado o botão “Salvar” para completar a operação. Esta tela também é utilizada para categorizar os valores. É ilustrada na figura 13.

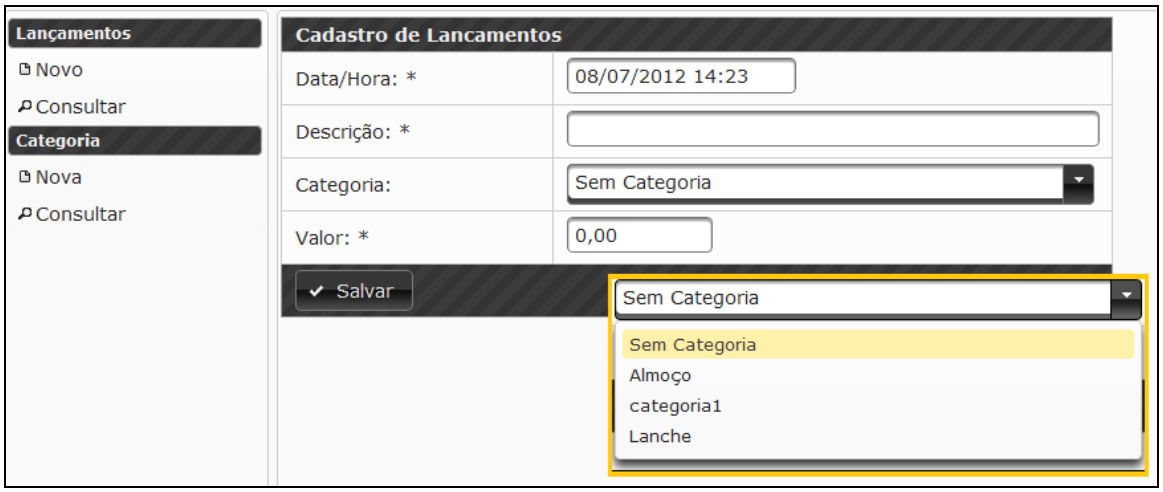


Figura 13: Cadastro de lançamento
Fonte: Arquivos do autor.

Para consulta-los basta abrir a interface de consulta de lançamentos, disponível na guia “Lançamento”, no link “Consultar”. Esta página também é a interface inicial do sistema, ela mostra os dados do lançamento e em que categoria ele se enquadra.

Com relação as ferramentas utilizadas, o Netbeans se mostrou bastante satisfatório. Ele possui um “wizard” para a criação de projetos que nos permite selecionar JSF 2.0 e Primefaces, assim não há necessidade de procurar por bibliotecas externas. O servidor Glassfish também pode ser vinculado automaticamente ao projeto.

Outra funcionalidade que foi extremamente vantajosa foi o guia para criação de clientes para serviço *web* disponibilizada pelo Netbeans. Indica-se o descritor dos serviços (WSDL, ver capítulo 2.11) do web service e ele gera automaticamente o cliente para ser utilizado na aplicação. No projeto apresentado, foi criada uma classe chamada “ServiçoWS” que encapsula o acesso ao Web service, ela pode ser vista em detalhes no apêndice 6.8.

O Primefaces possui componentes bastante elegantes e faz com que o visual da aplicação seja muito atraente. Ele conta com uma documentação bem completa e inúmeros temas para escolher. Todos os componentes de interface que compõem o Debitar Web são integrantes da suíte Primefaces. Foi necessário apenas o *download* da biblioteca do tema, *black tie* neste caso, para que ele pudesse ser incorporado a aplicação. Como citado anteriormente, ele pode ser escolhido como componente padrão na hora da criação do projeto.

O servidor Glassfish apresentou certa instabilidade, muitas vezes eram feitas alterações na aplicação que não apareciam depois que era implantada no servidor. Para resolver este problema ele precisava ser reinicializado, mas nada que tenha causado grande atraso no desenvolvimento.

Devido a experiência acadêmica adquirida com as ferramentas utilizadas neste módulo, a dificuldade na sua construção foi minimizada. Com ele é atingido o objetivo de criação de interface web para gerenciamento de informações de um serviço web.

3.3.1 Estrutura do projeto Debitar Web

A estrutura do projeto Debitar Web foi dividida da seguinte forma:

- Páginas Web : pasta que contém os arquivos xhtml com o leiaute das páginas da aplicação, dentro dela estão outras pastas com recursos utilizados para que fiquem visualmente melhores e mais dinâmicas. Representam a camada de visão;

- Pacote beans: camada de controle da aplicação, contém as regras de negócio e acessam a camada DAO;
- conexaoWS: contem a classe ServicoWS citada anteriormente, ela encapsula o acesso as funções do Web service;
- dao: neste caso, a camada DAO não acessa diretamente os dados da aplicação, e sim o pacote ServicoWS. Foram mantidas estas classes por uma questão de convenção;
- model: camada de modelo, classes abstraem os dados utilizados na aplicação;
- util: possui funções de conversão que são utilizadas por toda a aplicação.

Na figura 14 é ilustrada a estrutura do projeto Debitar Web.

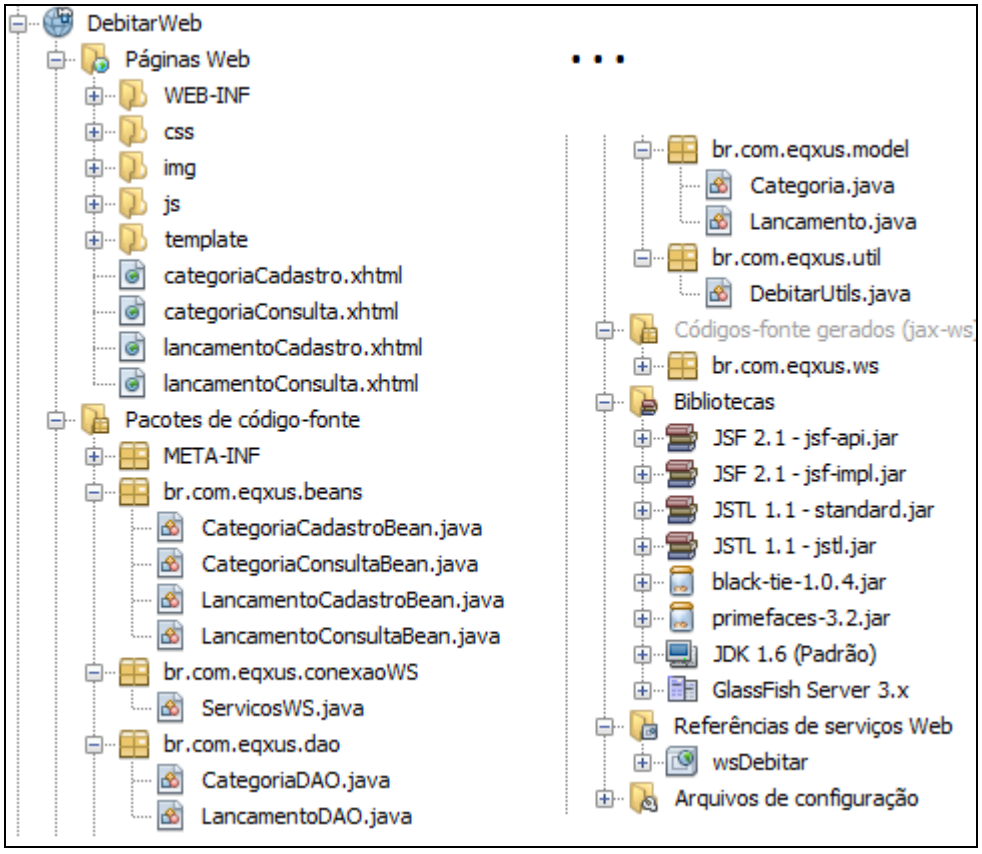


Figura 14: Estrutura do projeto Debitar Web
Fonte: Arquivos do autor.

Podem ser vistas ainda a pasta de “Códigos-fonte gerados” que contém as classes geradas automaticamente pelo guia de criação de clientes de serviço web e as bibliotecas utilizadas pelo sistema.

3.4 Módulo Centralizador (WebService) – DebitarWS

Trata-se de uma aplicação web que disponibiliza serviços para gerenciamento e cadastramento de lançamentos e categorias no banco de dados do servidor. Ela fica armazenada num *container* Glassfish 3.0.

Junto com ela fica o banco de dados remoto do sistema, construído com PostgreSQL 9. Quando é feito a inserção ou alteração por qualquer um dos outros dois módulos, este é o responsável por atualizar o banco remoto.

Para que os lançamentos sejam identificados e atualizados, foram implementados dois conceitos : chave e status. Chave é um número único criado na hora de gravar o lançamento, tanto no módulo móvel quanto web; e status é o estado do lançamento, são eles:

- 0: Não sincronizado - o lançamento foi inserido em apenas um módulo e necessita ser sincronizado no outro;
- 1: Sincronizado - lançamento está gravado da mesma forma nos dois módulos;
- 3: A excluir – quando for efetuada a próxima sincronização, deverá ser excluído;
- 4: Excluído – lançamento excluído nos dois módulos.

Para sua criação, foi utilizado o guia de criação de serviço Web do netbeans. Seus métodos são:

- gravarLanc: faz a inserção ou alteração de um lançamento;
- pesquisarLanc: retorna uma lista de *Strings* com os lançamentos cadastrados;
- pesquisarLancPorId: retorna uma s*tring* com os dados do lançamento que possui o id passado por parâmetro;
- pesquisarLancAtualizar: retorna uma lista de *strings* com os lançamentos que precisam ser atualizados no dispositivo móvel;
- gravarCat: insere ou altera uma categoria;
- excluirCat: exclui uma categoria;
- pesquisarCat: retorna uma lista de *strings* com as categorias cadastradas;
- pesquisarCatPorId: retorna uma *string* com os dados da categoria que possui o id passado por parâmetro.

3.4.1 Estrutura do projeto DebitarWS

Na figura 15 é possível visualizar a estrutura do projeto Debitar, ele foi dividido da seguinte forma:

- dao: pacote com as classes que acessam o banco de dados PostgreSQL do servidor;
- model: camada de modelo, classes que abstraem os dados utilizados na aplicação;
- util: possui funções de conversão que são utilizadas por toda a aplicação;
- ws: contém a classe WsDebitar, que possui todos os serviços disponibilizados pelo Web service.

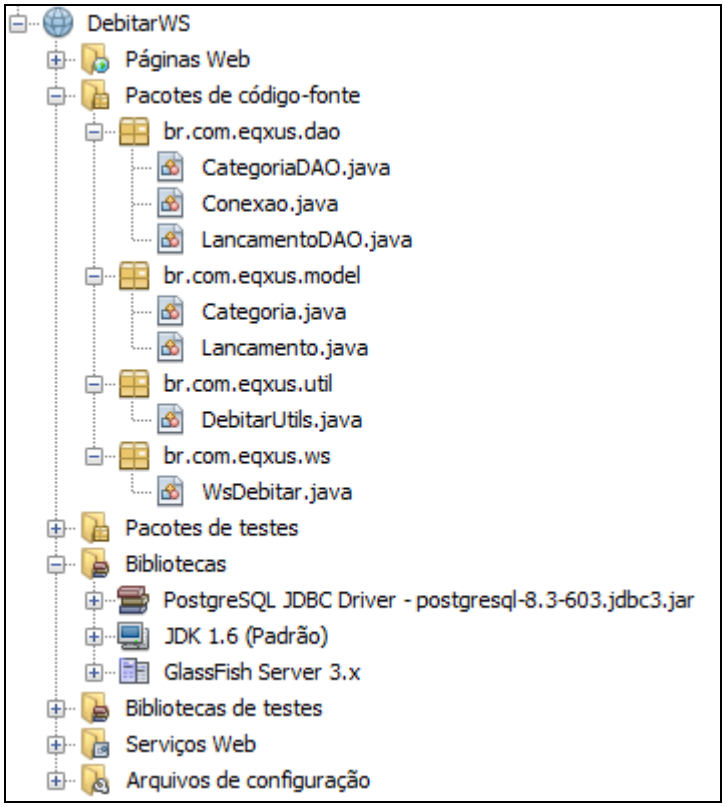


Figura 15: Estrutura do projeto DebitarWS
Fonte: Arquivos do autor.

Com esta aplicação atingimos o objetivo de criação de um provedor de serviços que funcione para dispositivos móveis e web.

4. CONSIDERAÇÕES FINAIS

4.1 Conclusões

O objetivo geral de construção de um software que fizesse gerenciamento de débitos e sincronizasse com a web foi atingido. Foi uma excelente oportunidade para aprofundar os conhecimentos sobre as tecnologias Android, Web serviços e JSF.

Embora alguns imprevistos e a falta de experiência com a maior parte das tecnologias tenham sido obstáculos durante o desenvolvimento, superá-los e conseguir criar uma ferramenta útil e que pode ser expandida posteriormente foi muito satisfatório.

Além disso, estudar conceitos que estão atualmente em pauta no mercado de TI, como mobilidade, Android e computação em nuvem pode ser bastante proveitoso, embora o software desenvolvido seja apenas um embrião de uma idéia maior a ser desenvolvida.

Por fim, puderam ser aplicados muitos dos conceitos estudados no decorrer do curso de forma efetiva, contribuindo para fixação e um maior aproveitamento de tudo que foi aprendido.

4.2 Projetos Futuros

O *software* que foi desenvolvido neste projeto é uma versão simplificada, focada na sincronização de dados entre os dispositivos. As funcionalidades relacionadas ao gerenciamento dos débitos e gestão de informação são básicas e podem ser aprimoradas substancialmente.

A princípio, seria necessário implementar um controle de conta de usuário, para que várias pessoas pudessem criar seus perfis e utilizar o sistema. Uma solução interessante seria integrar o cadastro do usuário com um *login* no Facebook, por exemplo, uma prática comum atualmente.

Gráficos, relatórios, filtros por data e categoria também seriam recursos que auxiliariam num controle mais detalhado do que foi lançado. Outra proposta interessante é criar uma exportação de dados, assim seria mais flexível a manipulação das informações do sistema. Seria possível, por exemplo, integrar os dados com uma planilha de orçamento doméstico.

Algumas melhorias já estão sendo estudadas, uma delas é a integração do Debitar móvel com sistemas bancários. É possível que seja configurado, no site de algumas instituições, o envio de SMS para cada débito que seja lançado na conta. A ideia é capturar os dados dessas mensagens e lançar automaticamente no sistema.

E finalmente, poderiam ser desenvolvidos módulos móveis em outras plataformas para que o software atinja o maior número de usuários possível e sua capacidade de sincronização multiplataforma seja totalmente explorada.

5. REFERÊNCIAS

BUCANEK, James. **Learn Objective-C for Java Developers**. Nova Iorque: Apress, 2009.

BURNS, Ed; SCHALK, Chris. **JavaServer Faces 2.0**: The Complete Reference. Nova York: McGray-Hill, 2010.

CANALYS, **Smart phones overtake client PCs in 2011**. Disponível em < http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011>, acesso em : 17 abr 2012, 23:34:00.

CISCO, **Global Mobile Data Traffic Forecast Update, 2011–2016**. Disponível em < http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html>, acesso em 29 mai 2012, 22:30.

ÇIVICI, Çağatay. **Primefaces User's Guide 3.3**.Turquia, 2012. 475 p.

CLARO Daniela B.; SOBRAL João B. M. **Programação em Java**. Santa Catarina: Copyleft Pearson Education, 2008.

COMPUTERWORLD, **Vídeo, cloud e mobilidade vão liderar salto da TI, diz Cisco**. Disponível em < http://computerworld.uol.com.br/tecnologia/2012/04/18/video-cloud-e-mobilidade-vao-liderar-salto-da-ti-diz-cisco/>, acesso em 30 mai 2012, 20:35.

DEITEL, H.M. **Java: como programar**. Porto Alegre: Bookman, 2003.

DRAKE, Joshua D.; WORSLEY, John C. **Practical PostgreSQL**. Califórinia: O'Reilly, 2002.

GONCALVES, Antonio. **Beginning Java™ EE 6 Platform with GlassFish™ 3**. Nova Iorque: Apress, 2010.

HEFFELFINGER, David R. **Java EE 5 Development using GlassFish Application Server**. Birmingham, UK: PACKT Publishing, 2007.

HEFFELFINGER, David R. **Java EE6 Development with NetBeans7**. Birmingham, UK: PACKT Publishing, 2011.

HORSTMANN S. C. E CORNELL G. **Core Java™ 2**: Volume I – Fundamentals. Prentice Hall PTR, 2000.

JAVA, **O que é a tecnologia Java e por que é necessária?**. Disponível em < http://www.java.com/pt_BR/download/faq/whatis_java.xml>, acesso em : 02 jun 2012, 14:22.

LEE, W. M. **Beginning Android™ Application Development**. Indianapolis, Indiana : Wiley Publishing Inc, 2011.

MARTINS, Leandro. **Aprenda a Investir:** Saiba onde e como aplicar seu dinheiro. São Paulo: ATLAS, 2010.

MSN, **Pequenos gastos, grandes furos no orçamento** . Disponível em < http://estilo.br.msn.com/tempodemulher/dinheiro/artigo.aspx?cp-documentid=31841009>, acesso em 29 mai 2012, 22:36:00.

MSNBC**, Report: 1 billion apps downloaded in a week**. Disponível em < http://www.technolog.msnbc.msn.com/technology/technolog/report-1-billion-apps-downloaded-week-118210>, acesso em : 18 abr 2012, 00:43:00.

OWENS, Michael. **The Definitive Guide to SQLite**. New York: APRESS, 2006.

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

TAURION, Cezar. **Internet móvel**: Tecnologia e modelos. Rio de Janeiro: Campus, 2002.

TAURION, Cezar. **Cloud Computing: Computação em Nuvem** : Transformando o mundo da tecnologia da informação. Rio de Janeiro : Brasport, 2009.

TAURION, Cezar. **Mídias Sociais** : Como estão transformando nossos relacionamentos pessoais e profissionais, 2010. *Ebook* disponível em <http://www.smashwords.com/books/download/66646/1/latest/0/0/midias-sociais-e-seus-impactos-na-nossa-vida.pdf> , acesso em : 29 abr 2012, 22:12:00.

TECMUNDO, **Como escolher um smartphone ?**. Disponível em <http://www.tecmundo.com.br/celular/915-como-escolher-um-smartphone-.htm>, acesso em: 21 abr 2012, 20:14:00.

TELECOMPAPER, **Ericsson charts difference in men, women smartphone users.** Disponível em **<**http://www.telecompaper.com/news/ericsson-charts-difference-in-men-women-smartphone-users>, acesso em 10 abr 2012, 21:30:00.

UNIVERSOSMART, **Como o smartphone influencia as relações sociais no dia a dia**. Disponível em < http://www.universosmart.com.br/tecnologia/comportamento-como-o-smartphone-influencia-as-relacoes-sociais-no-dia-a-dia-58.html>, acesso em: 17 abr 2012, 19:35:00.

VEJA, **Evernote pode ser o próximo Instagram.** Disponível em <http://veja.abril.com.br/noticia/vida-digital/evernote-pode-ser-o-proximo-instagram>, acesso em 29 mai 2012, 22:26:00.

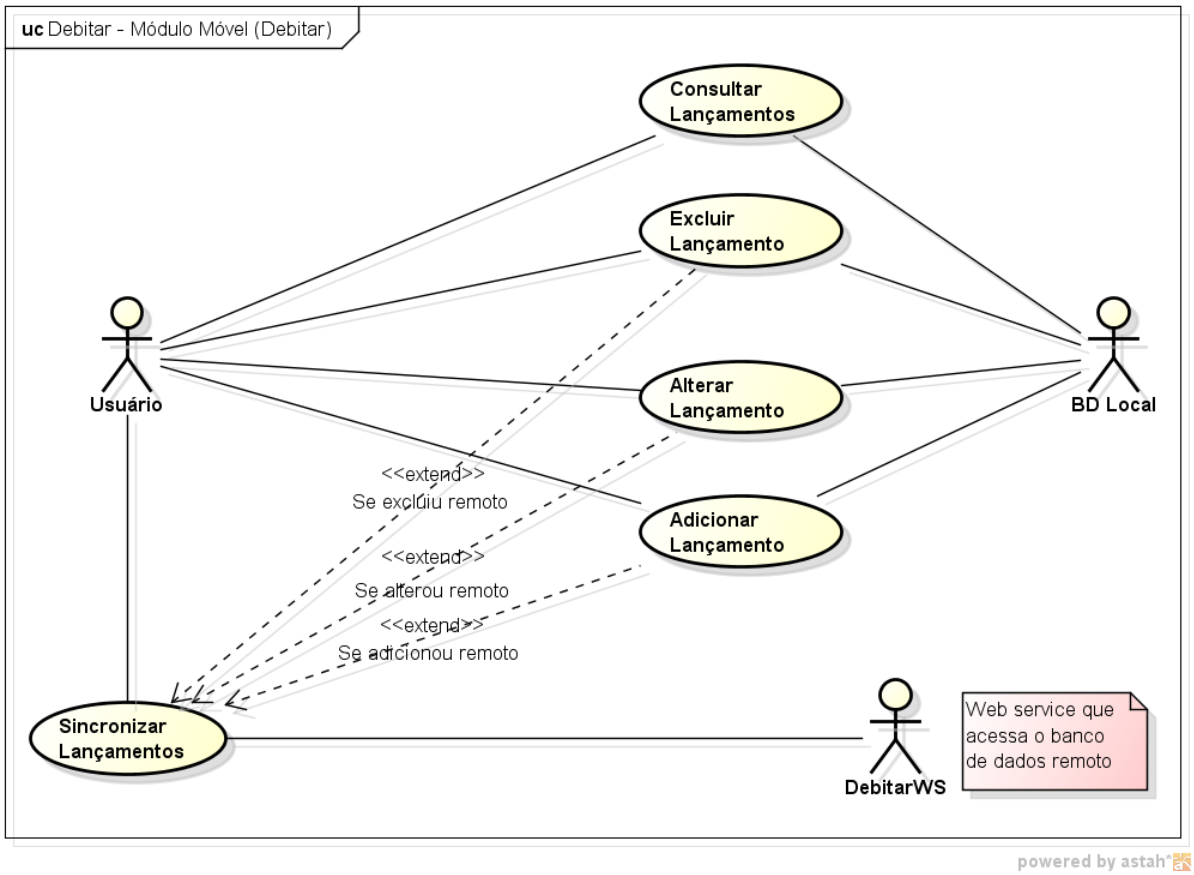
W3C, **Mobile Web Best Practices 1.0**. Disponível em <http://www.w3.org/TR/2006/CR-mobile-bp-20060627/>, acesso em 29 mai 2012, 22:40:00.

YUAN, Michael J. **Enterprise J2Me**: Developing Mobile Java Applications. Nova Jersey: Prentice Hall, 2004.

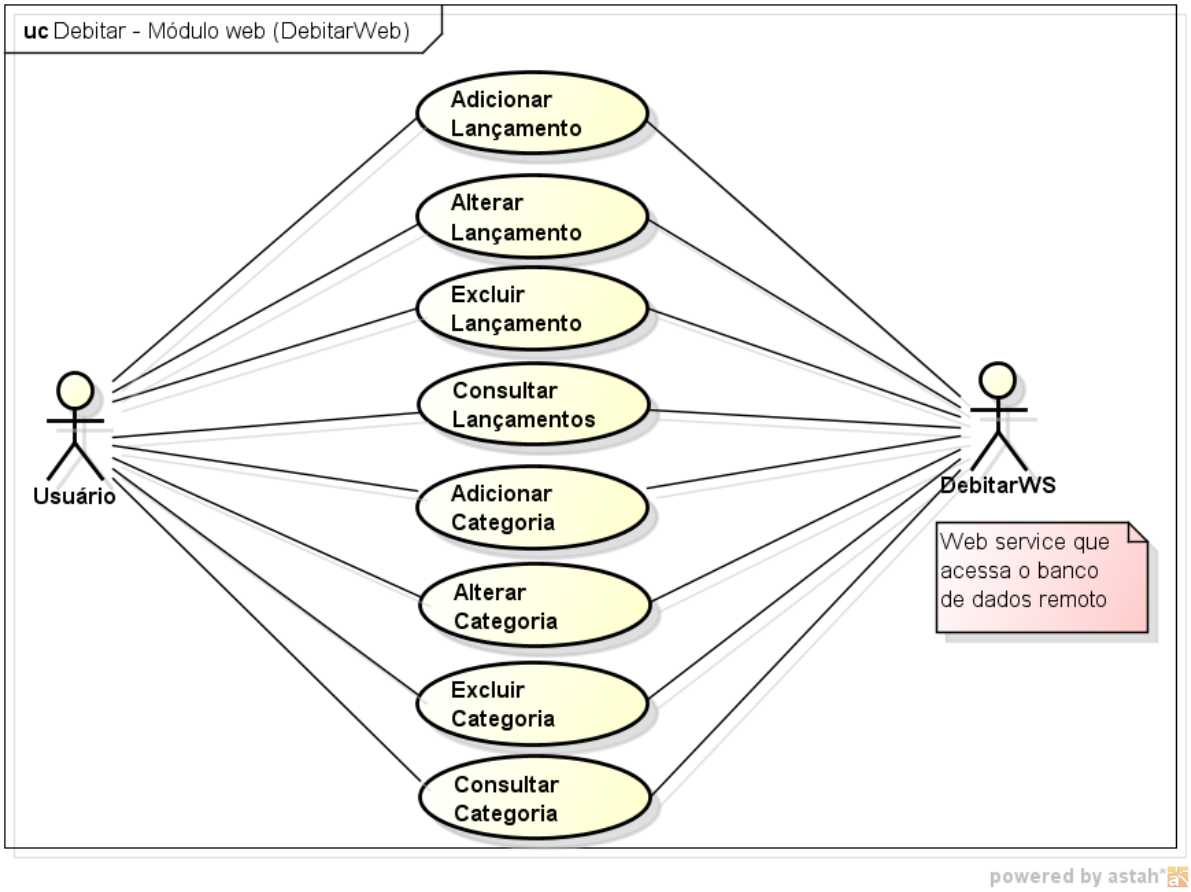
ZECHNER, Mario. **Beginning Android Games.** Nova Iorque: Apress, 2011.

6. APÊNDICES

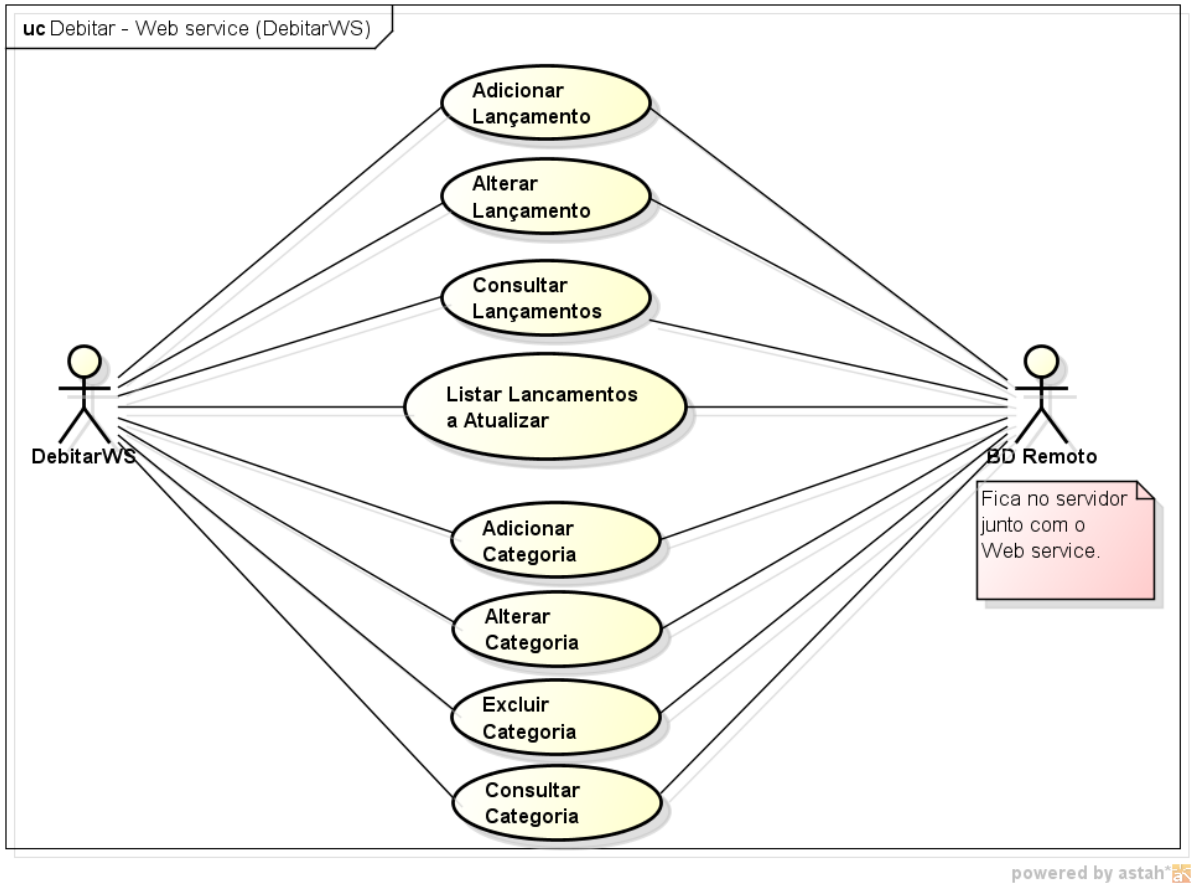
6.1 Casos de uso do módulo móvel



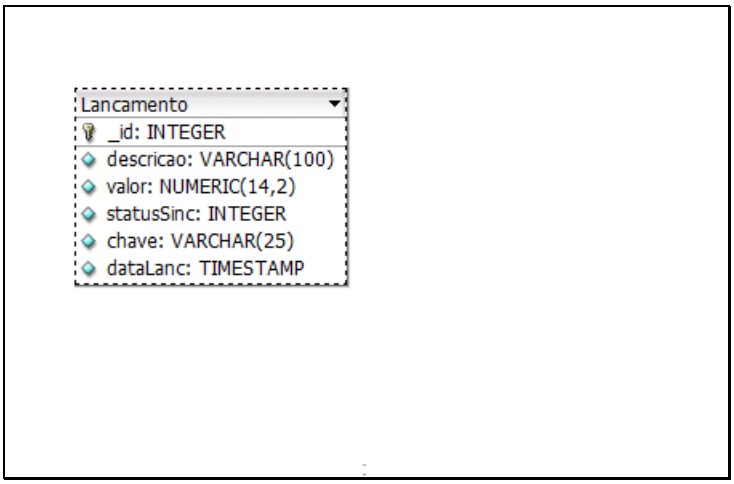
6.2 Casos de uso do módulo web



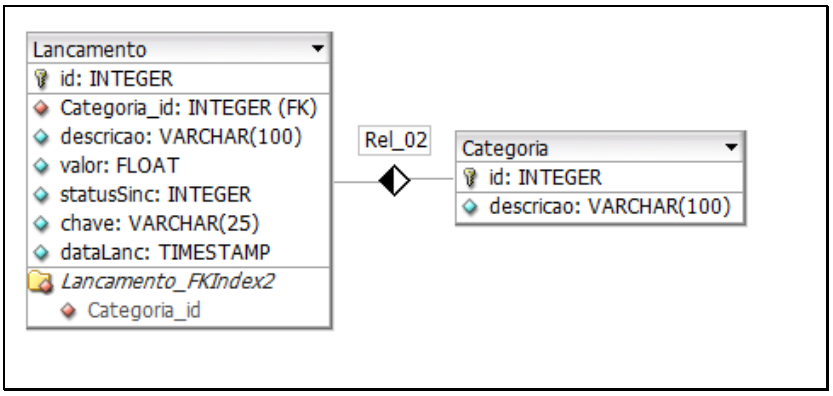
6.3 Casos de uso do Web service



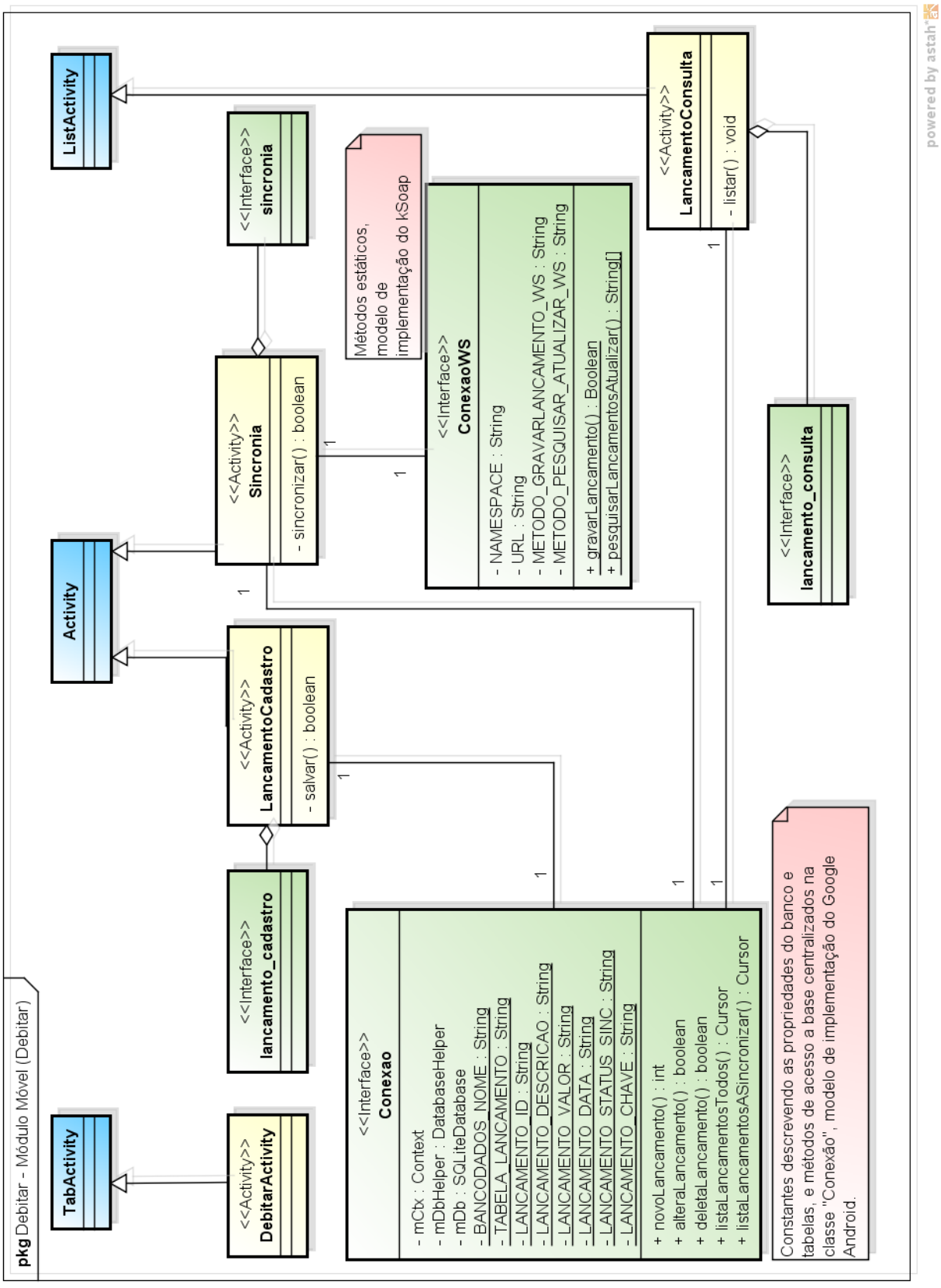
6.4 Diagrama Entidade-Relacionamento do módulo móvel



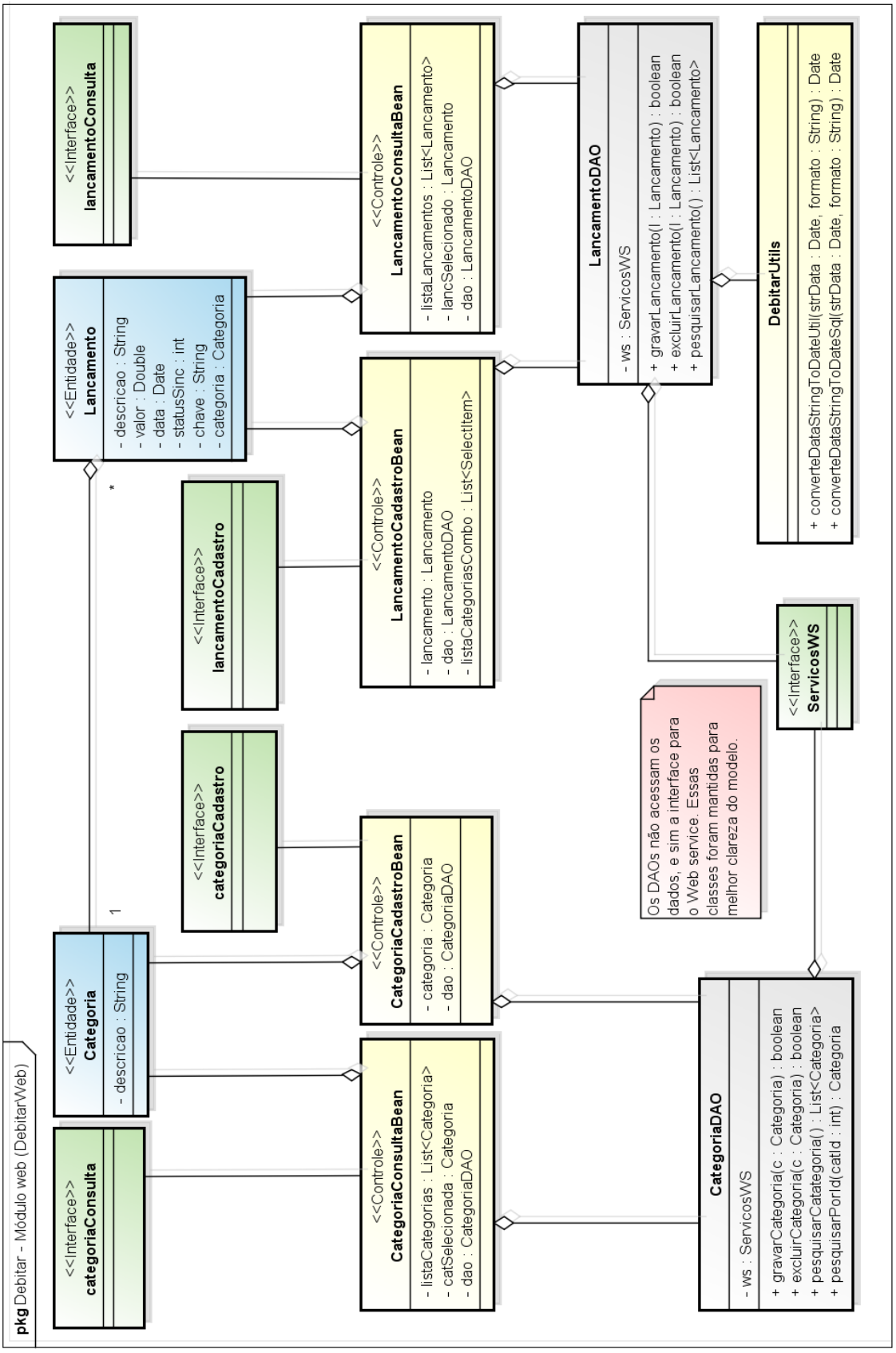
6.5 Diagrama Entidade-Relacionamento do módulo web



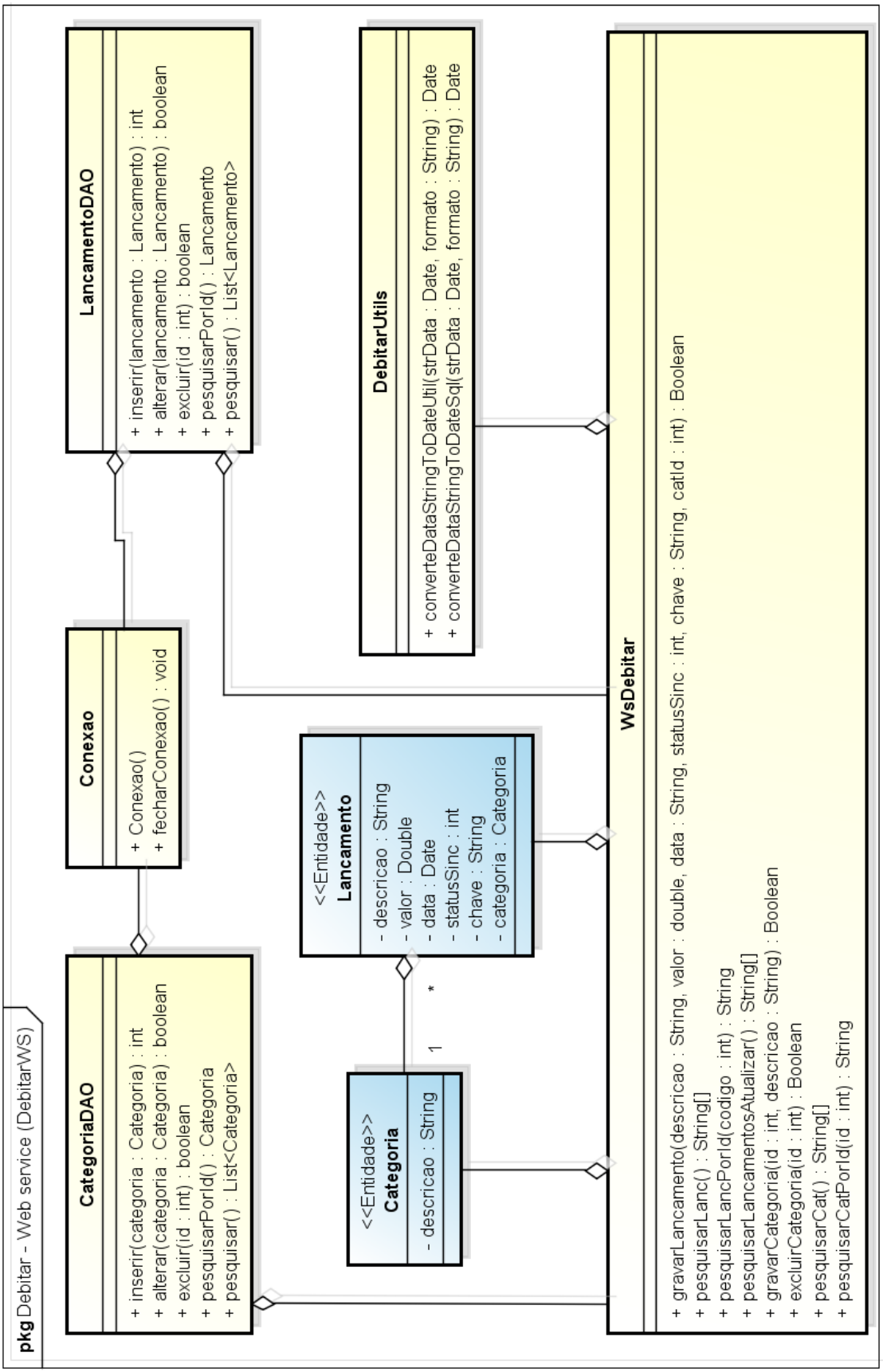
6.6 Diagrama de classe do módulo móvel



6.7 Diagrama de classe do módulo web

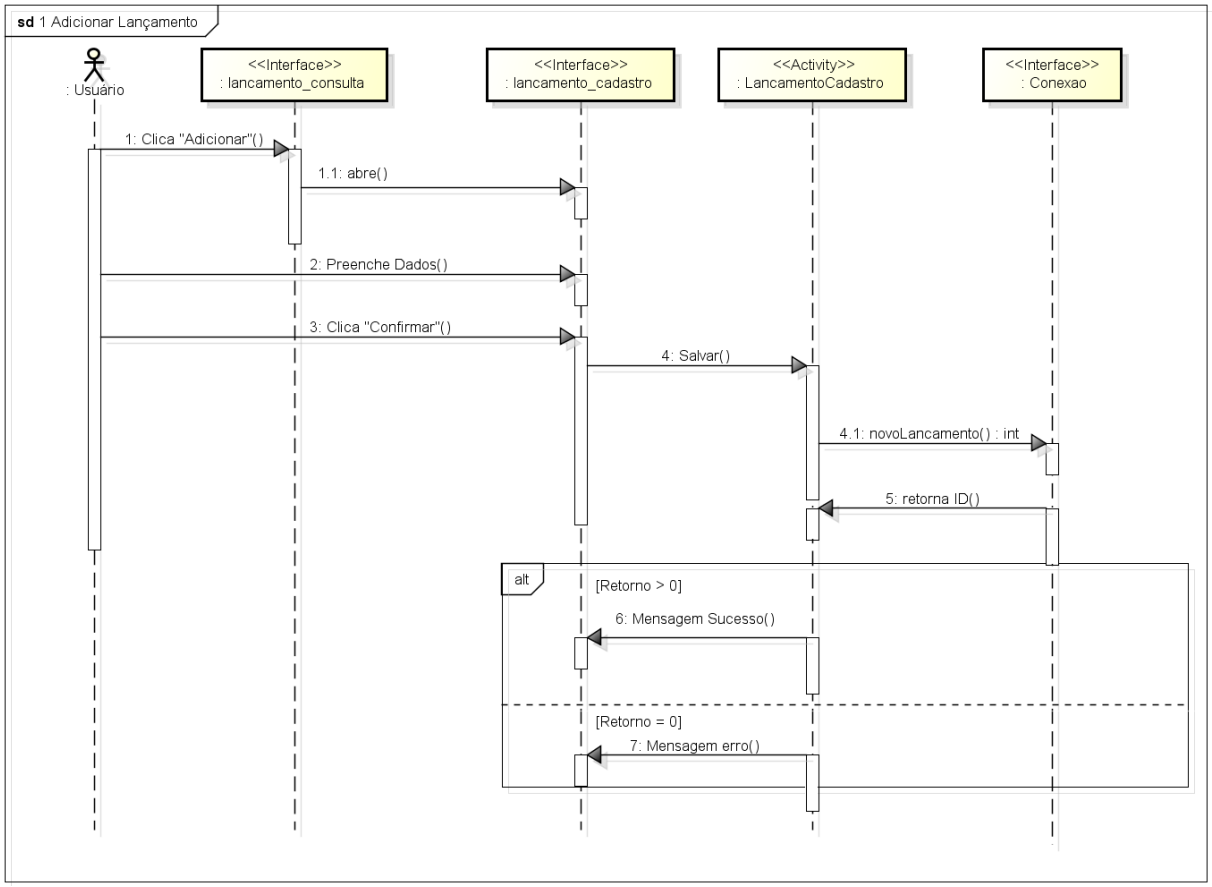


6.8 Diagrama de classe do Web service

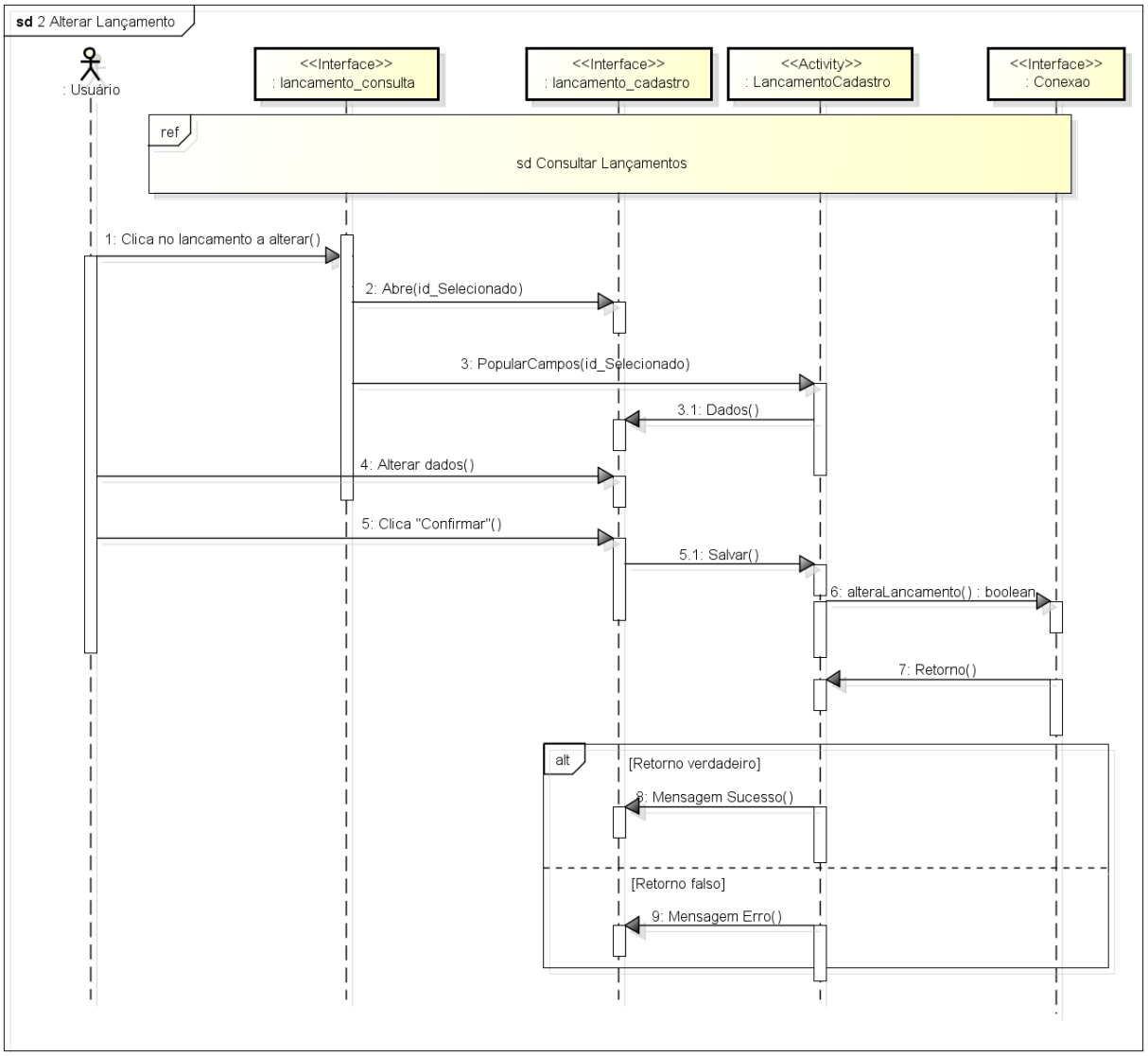


6.9 Diagramas de sequência do módulo móvel

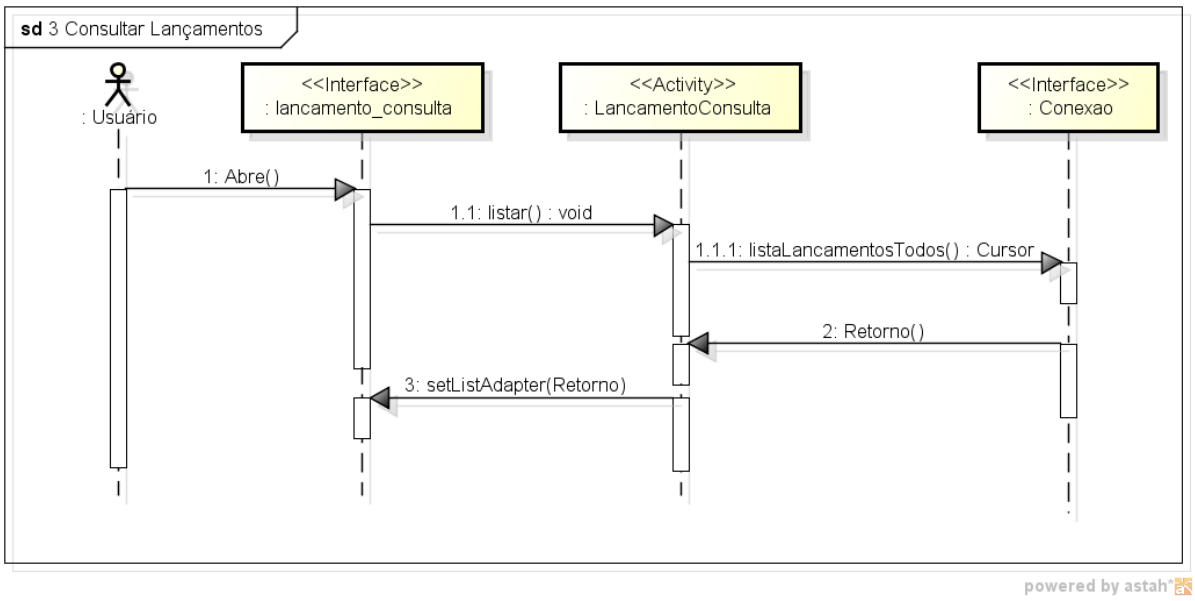
Adicionar Lançamento



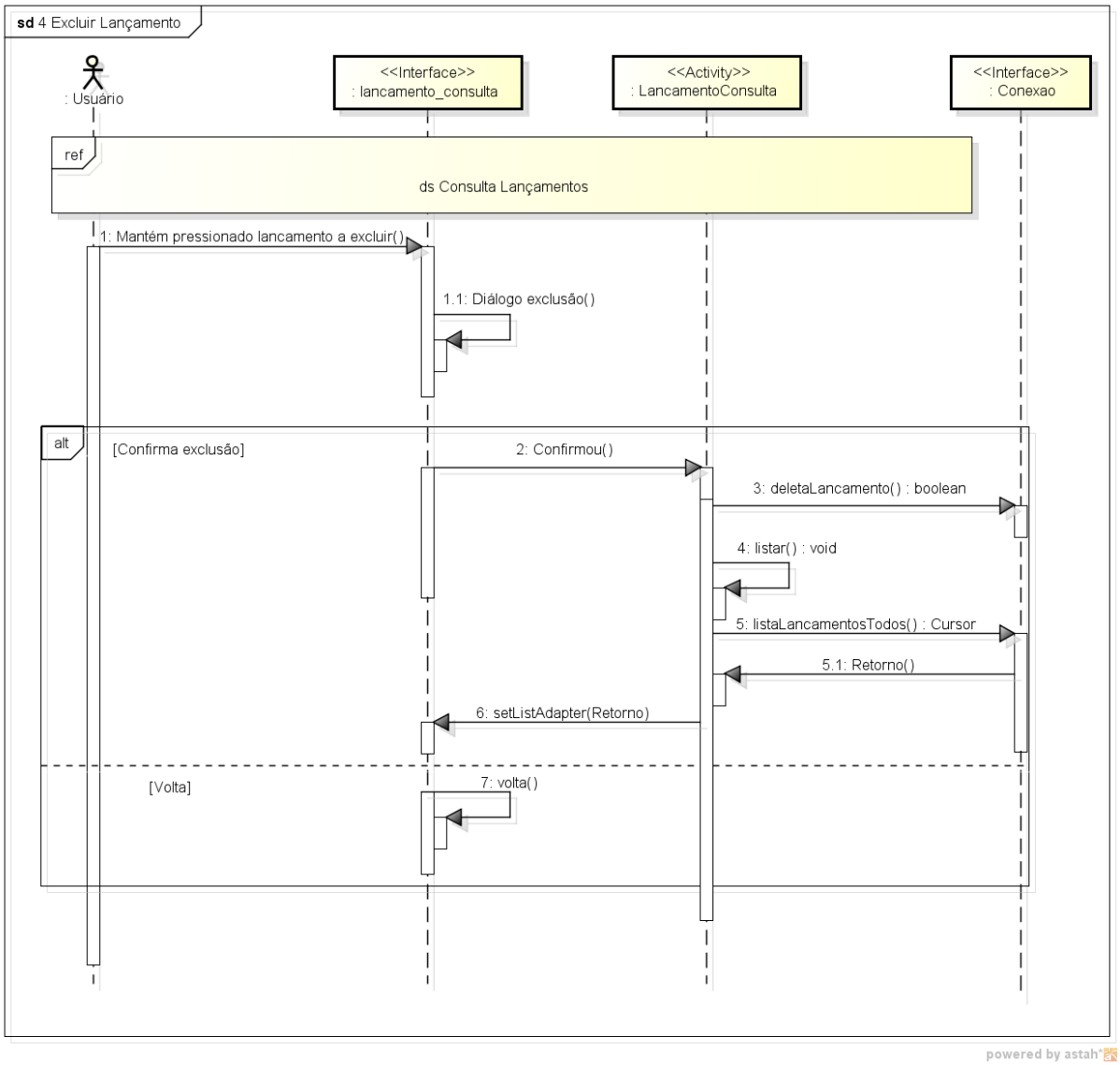
Alterar Lançamento



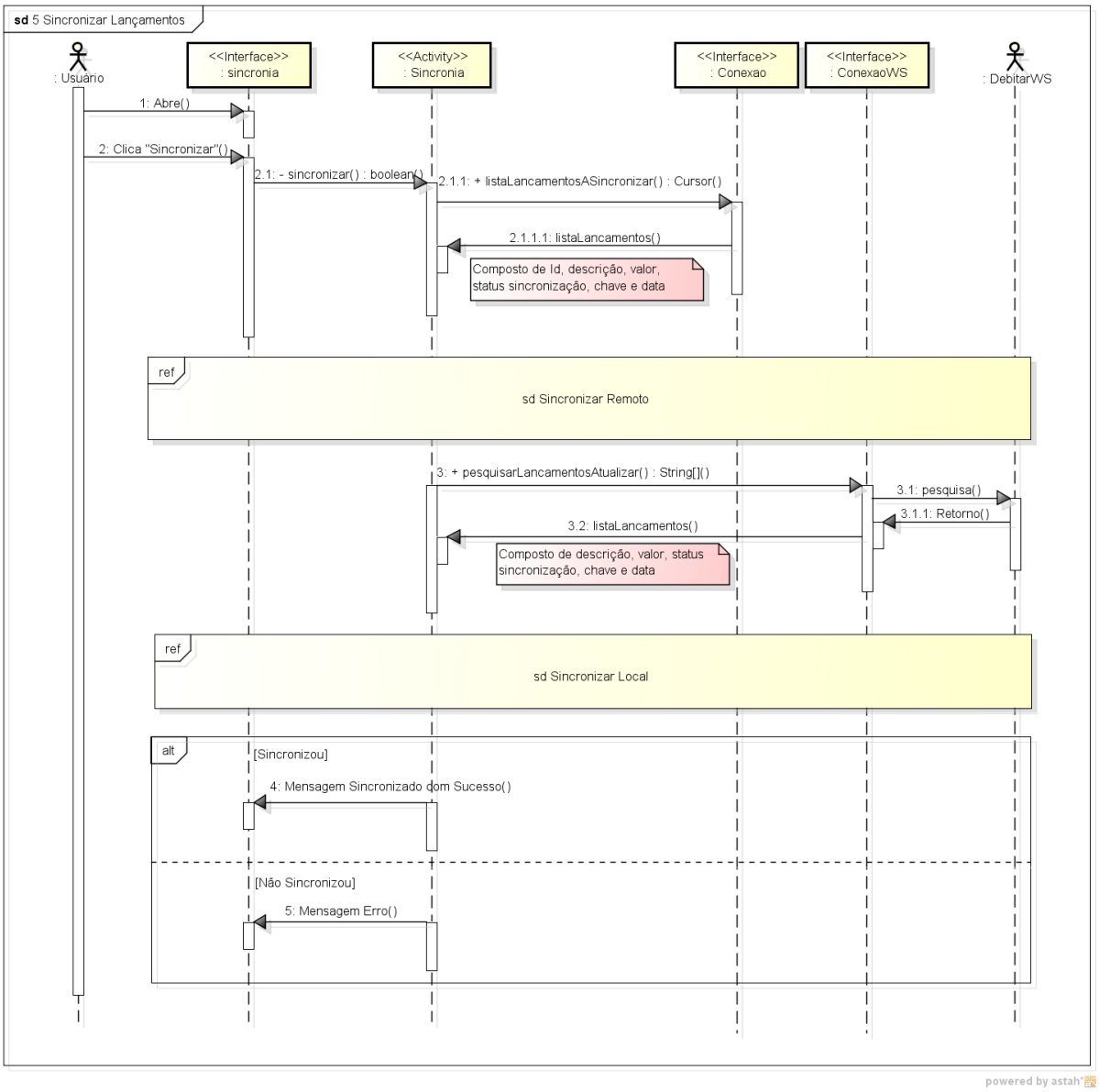
Consultar Lançamentos



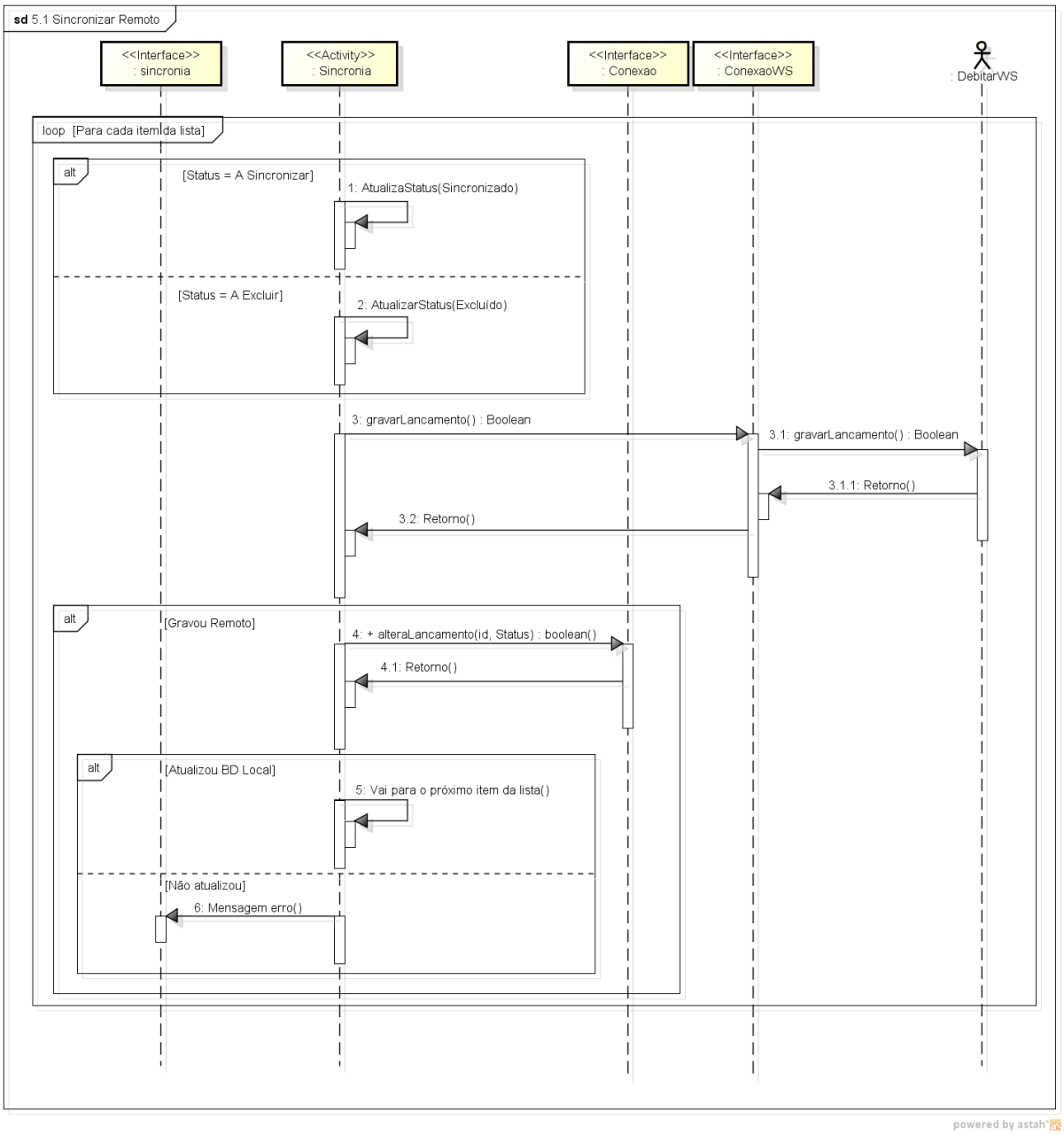
Excluir Lançamento



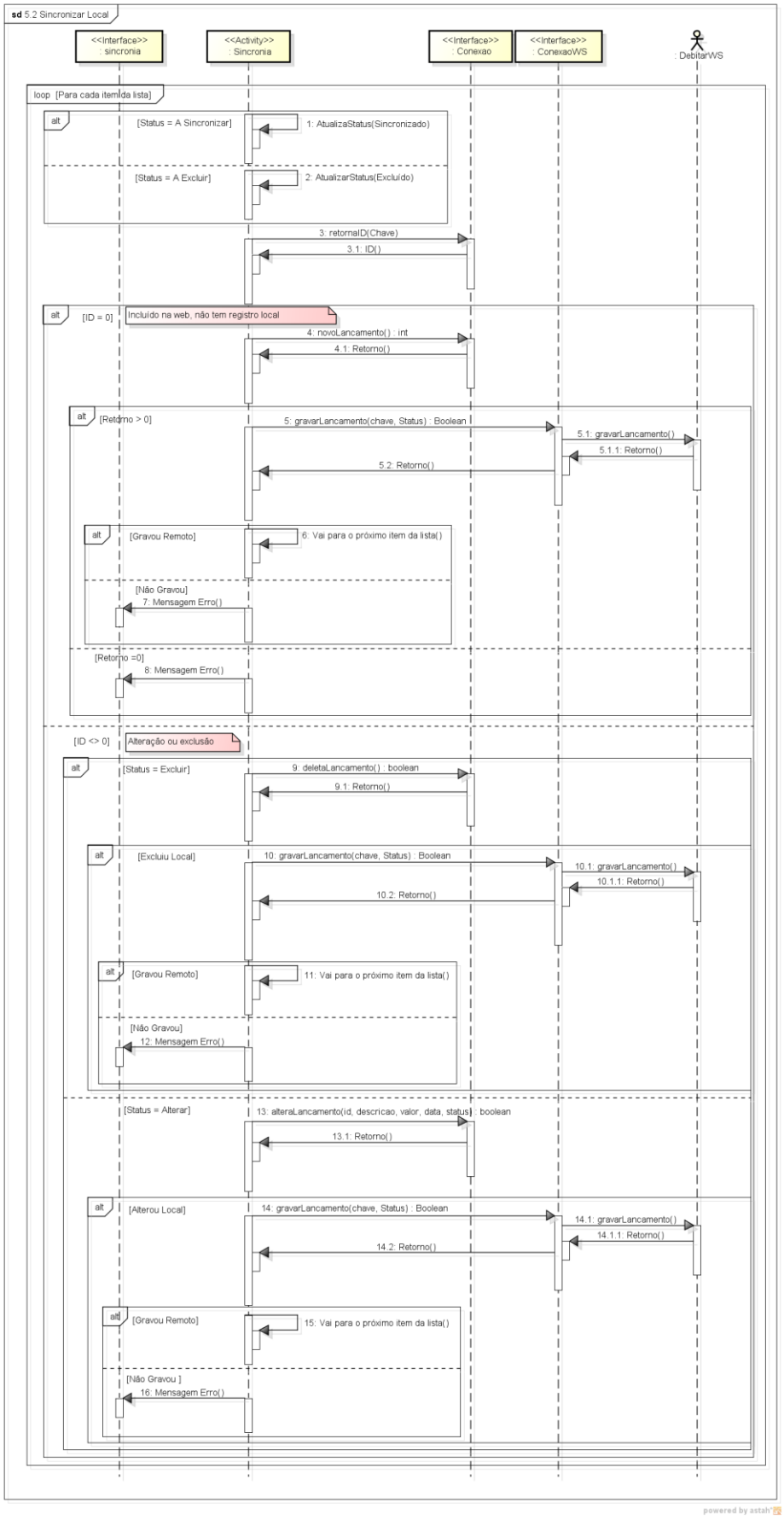
Sincronizar Lançamentos



Sincronizar Remoto

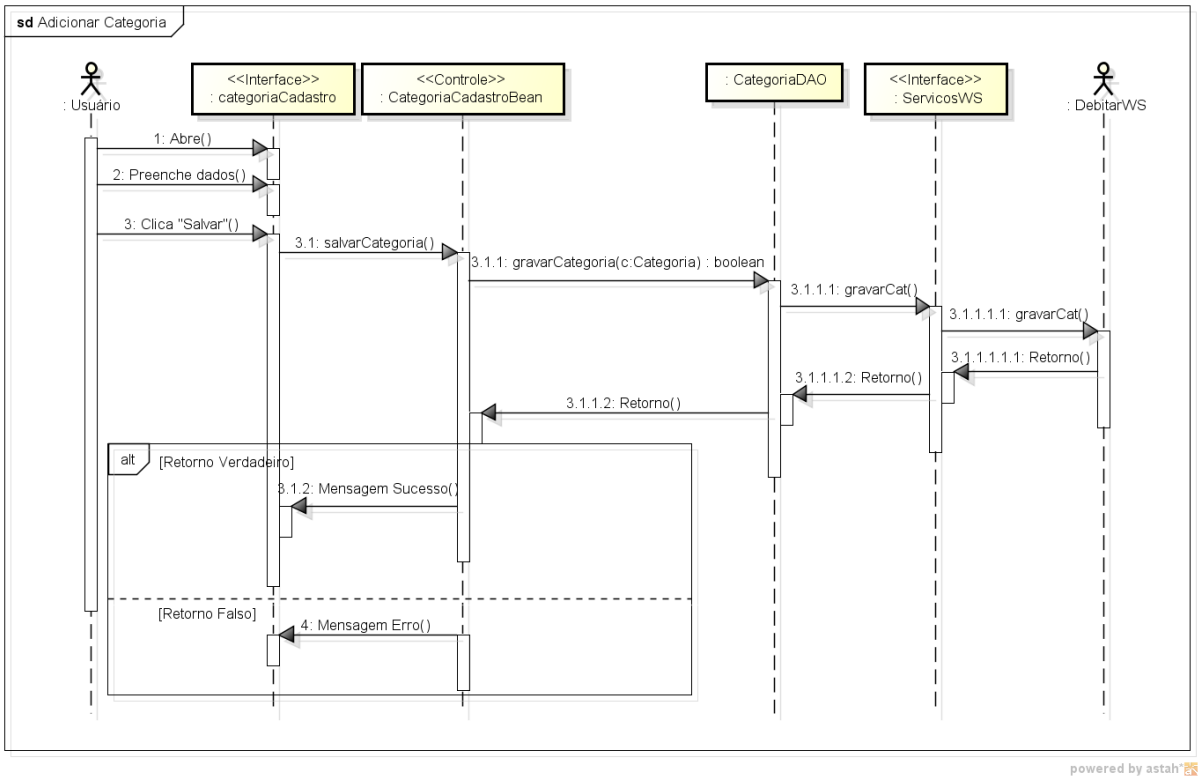


Sincronizar Local

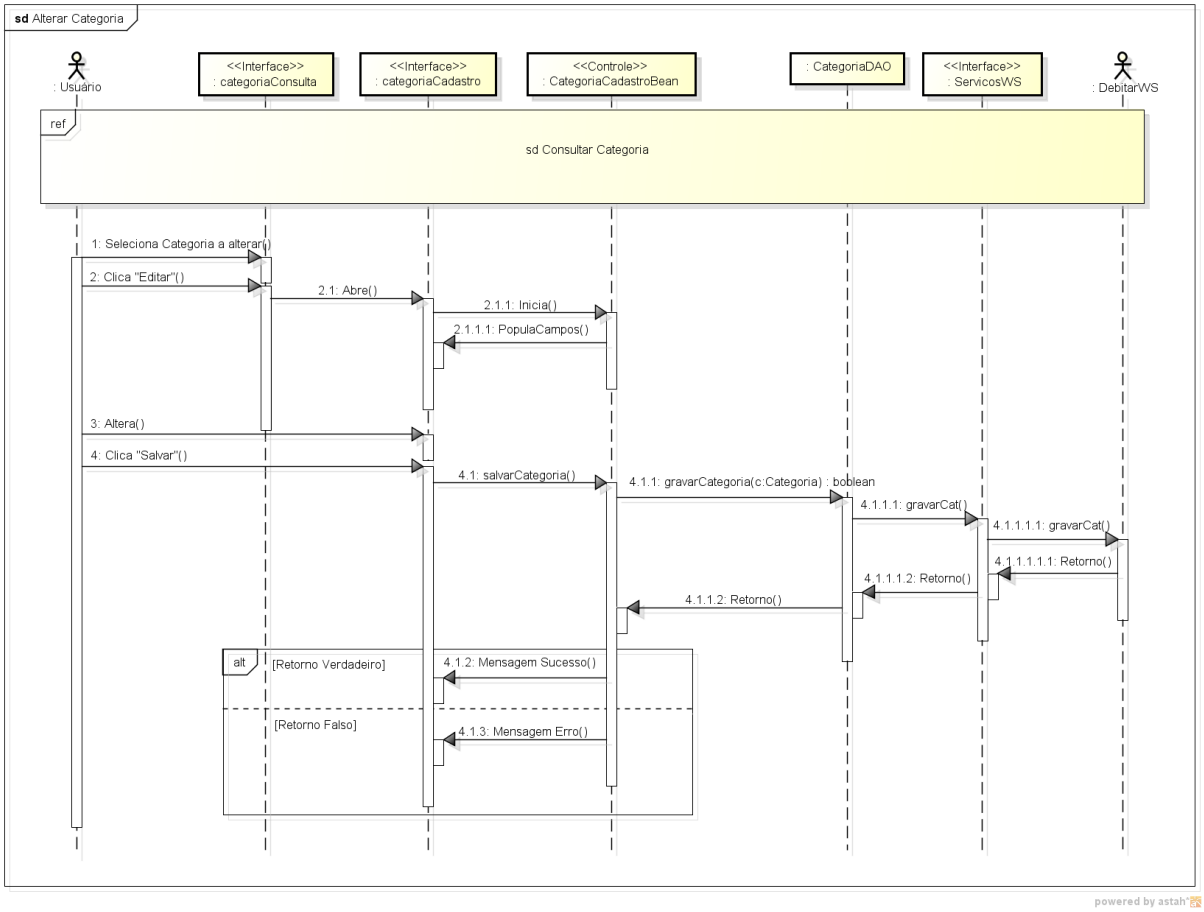


6.10 Diagramas de sequência do módulo web

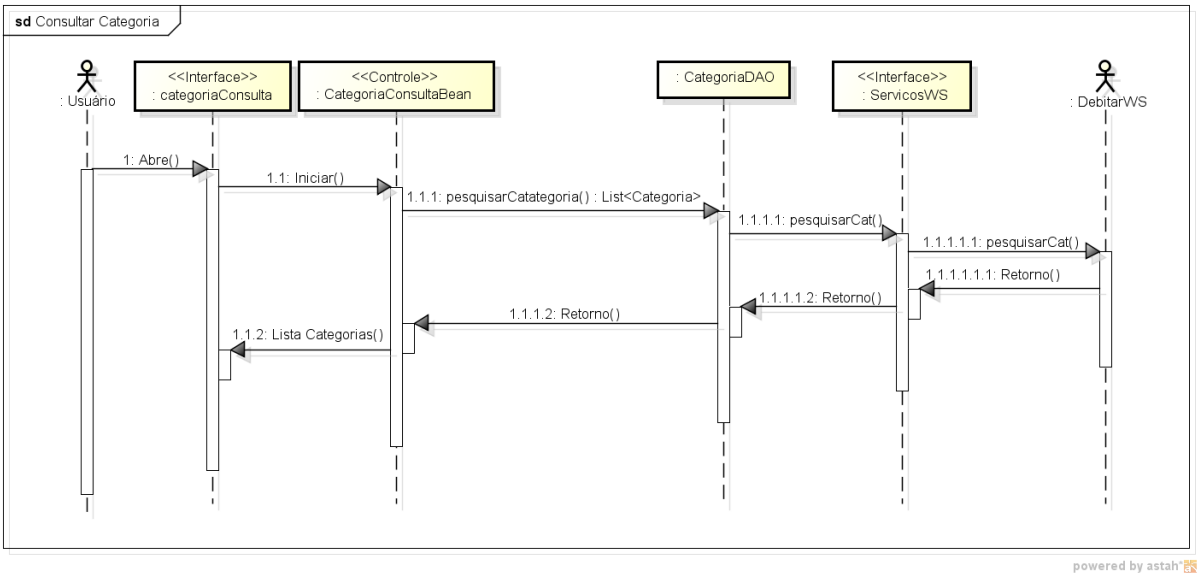
Adicionar Categoria



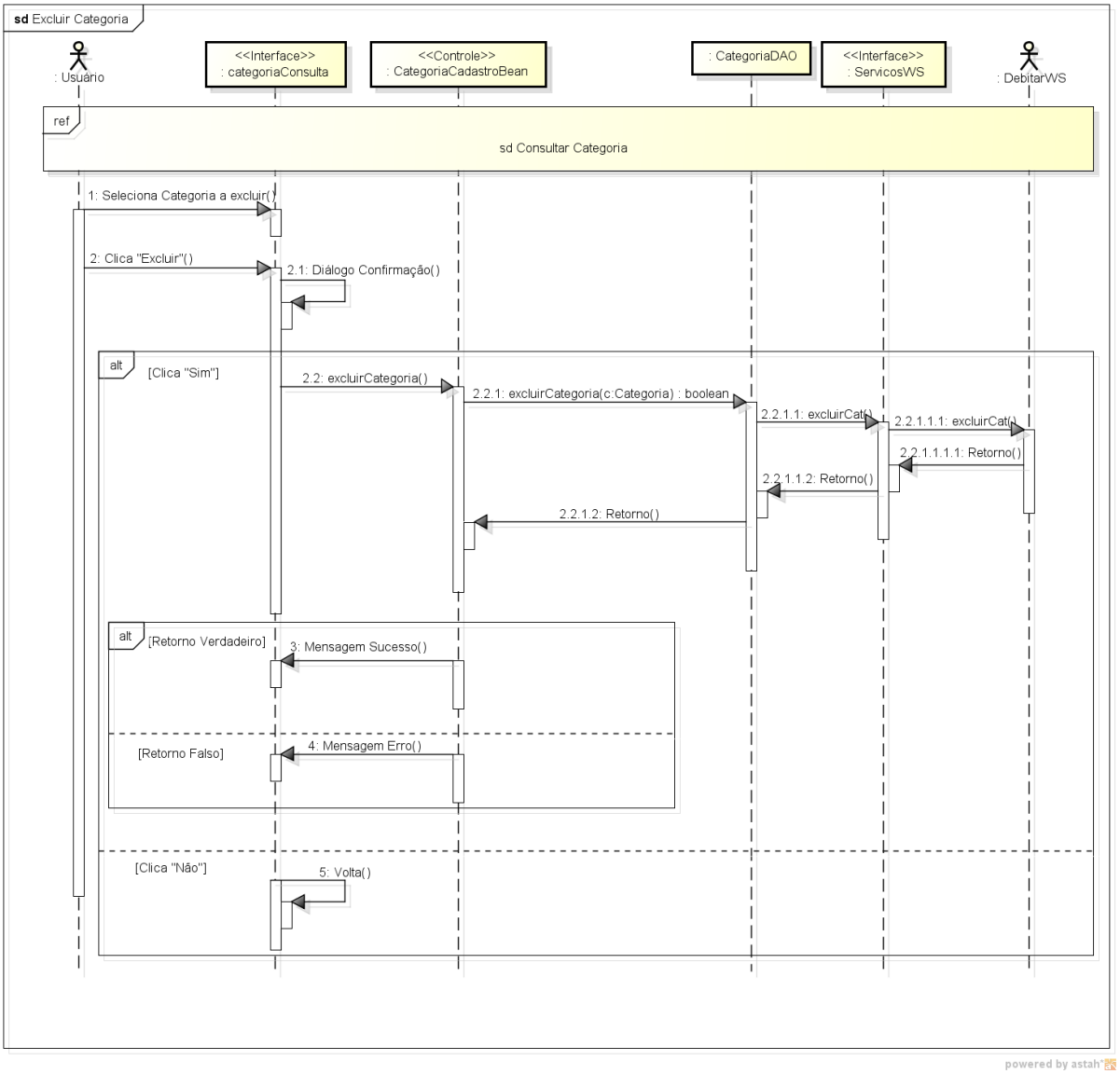
Alterar Categoria



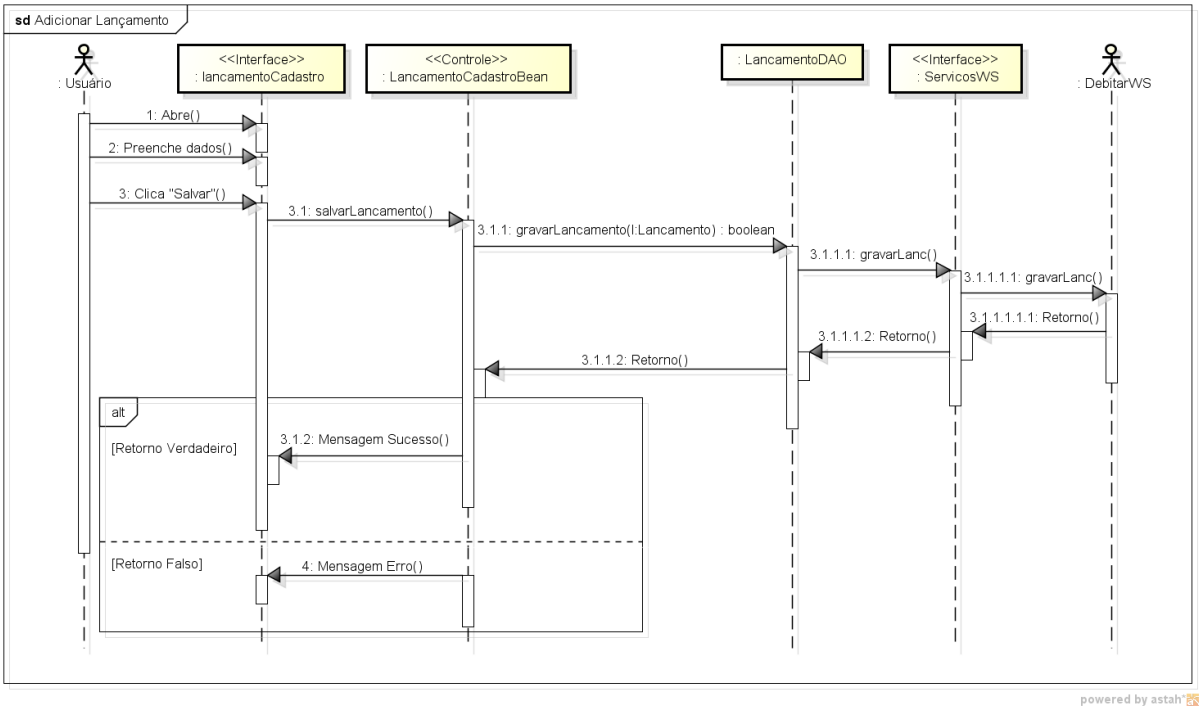
Consultar Categoria



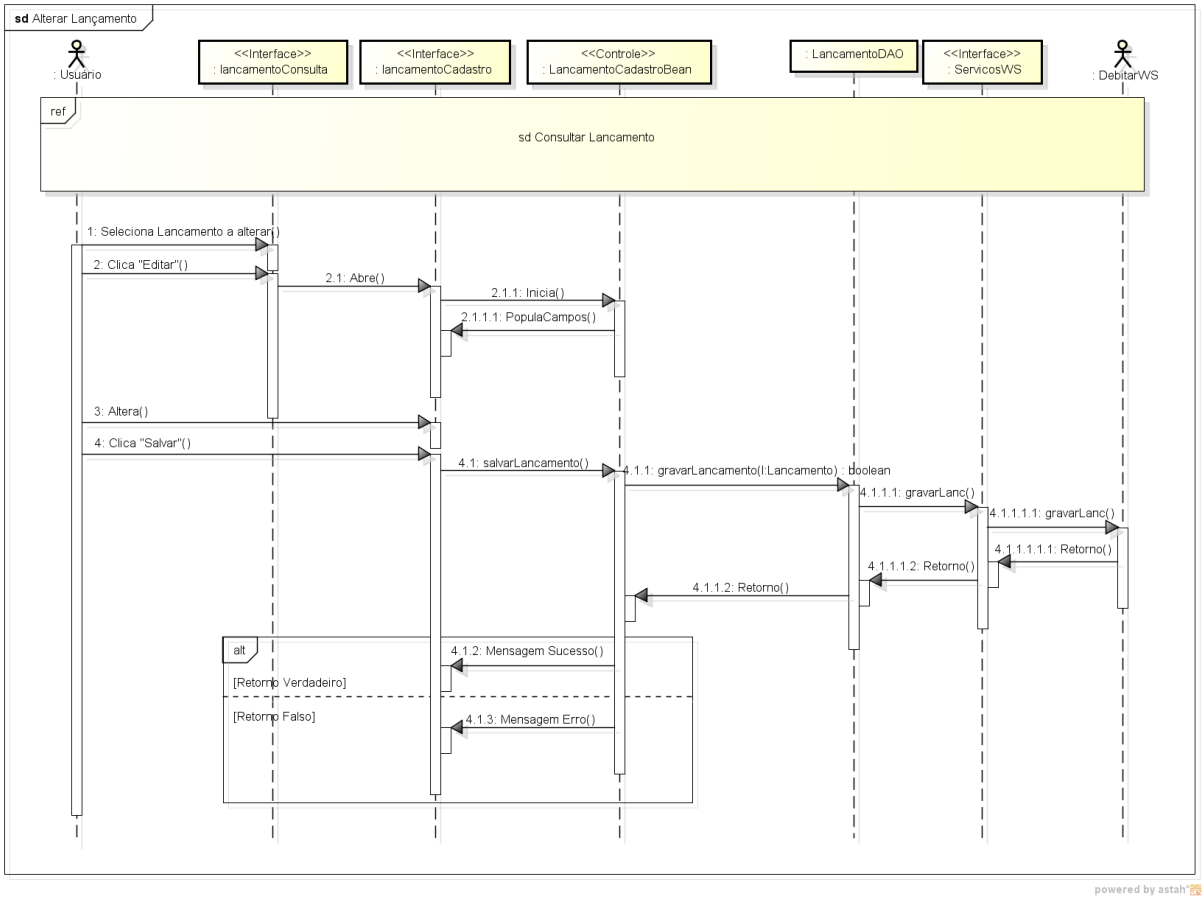
Excluir Categoria



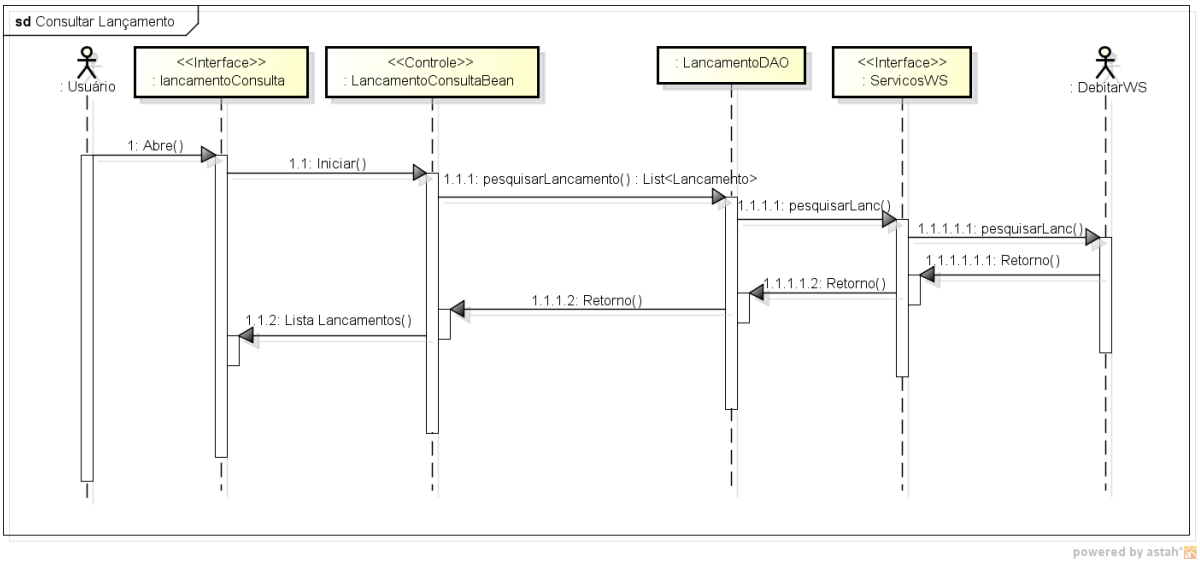
Adicionar Lançamento



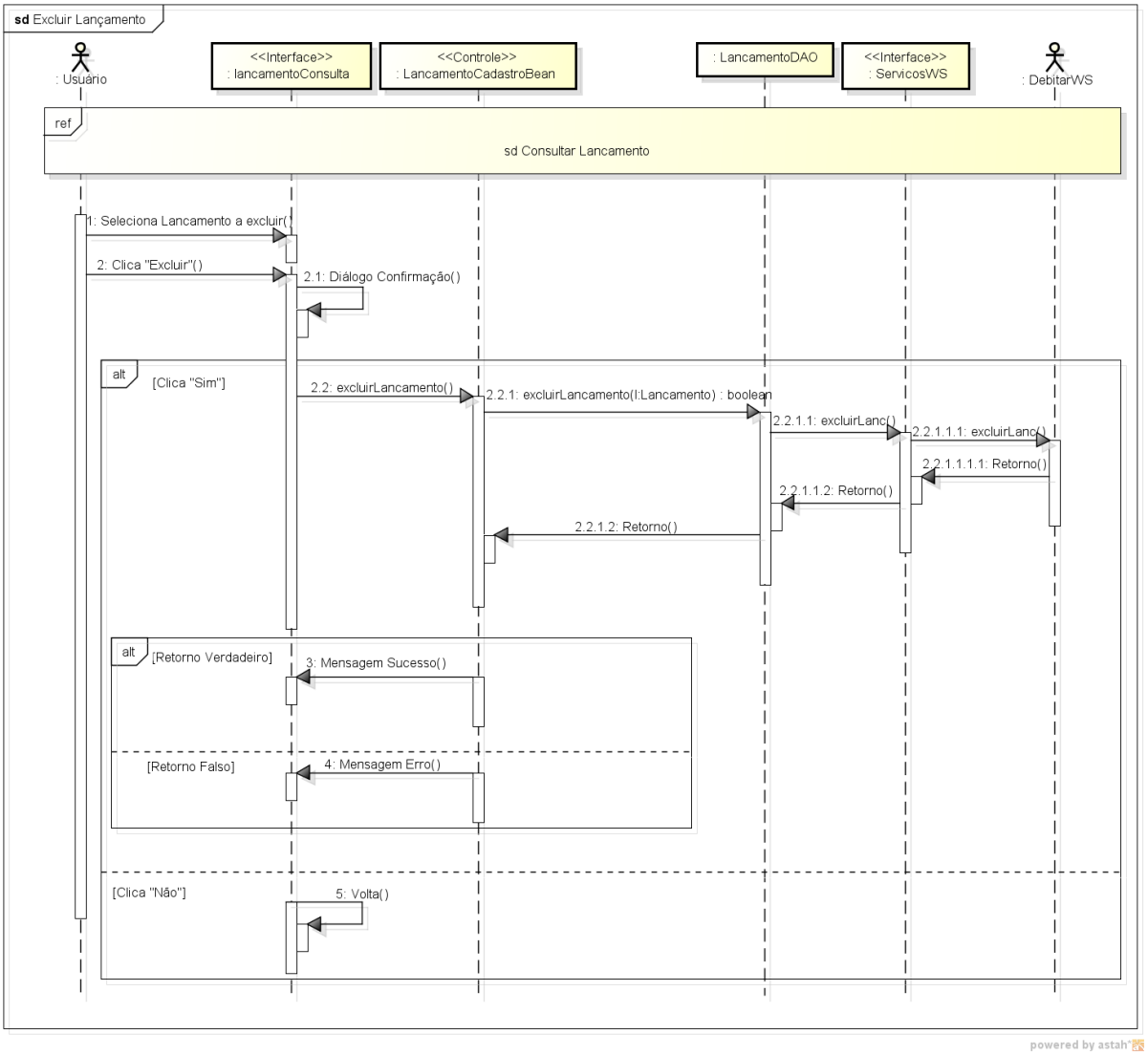
Alterar Lançamento



Consultar Lançamento

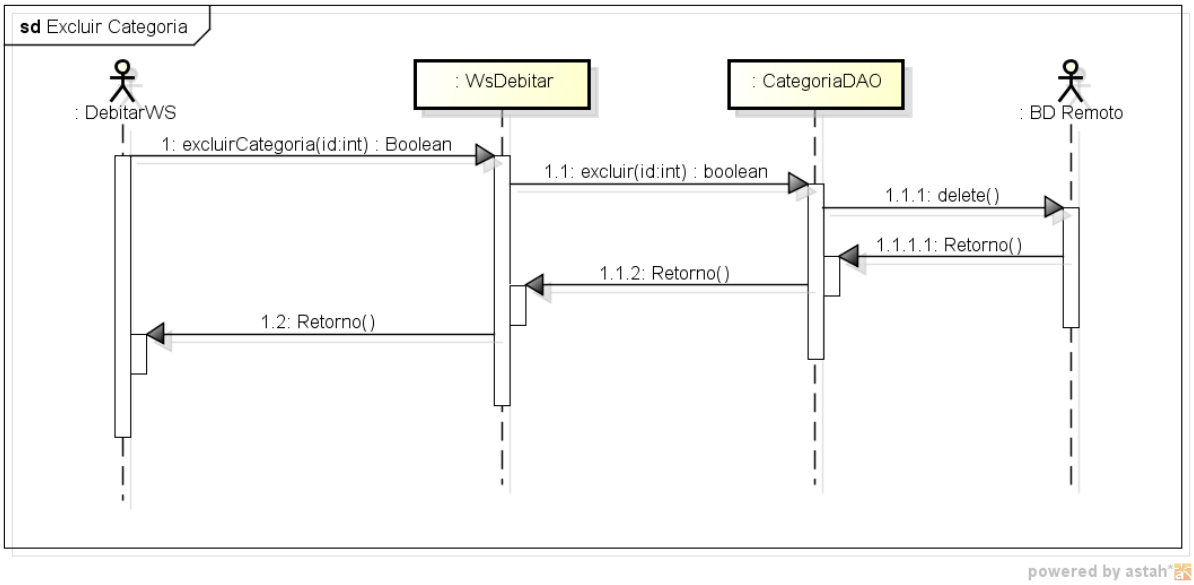


Excluir Lançamento

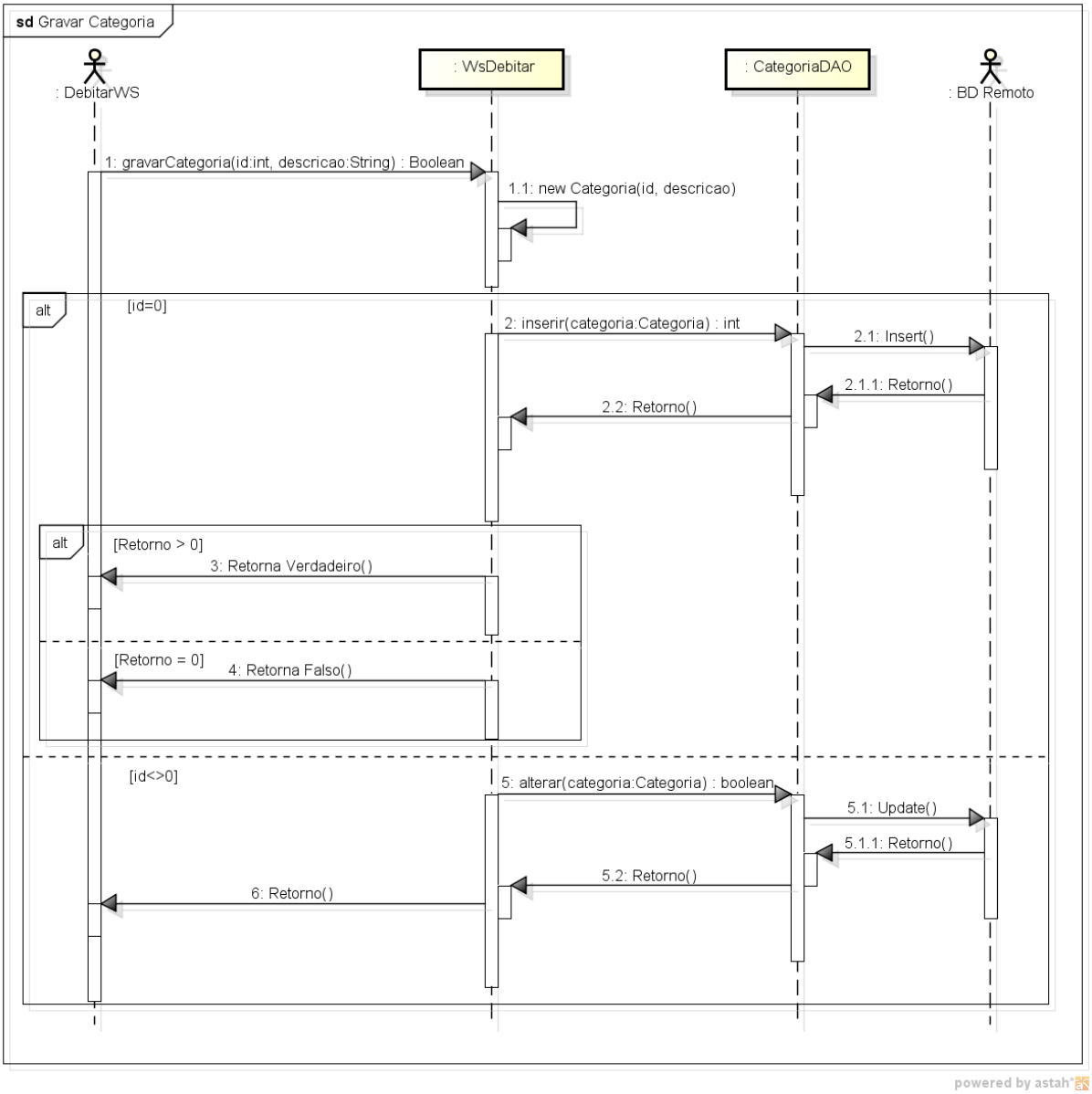


6.11 Diagramas de sequência do Web service

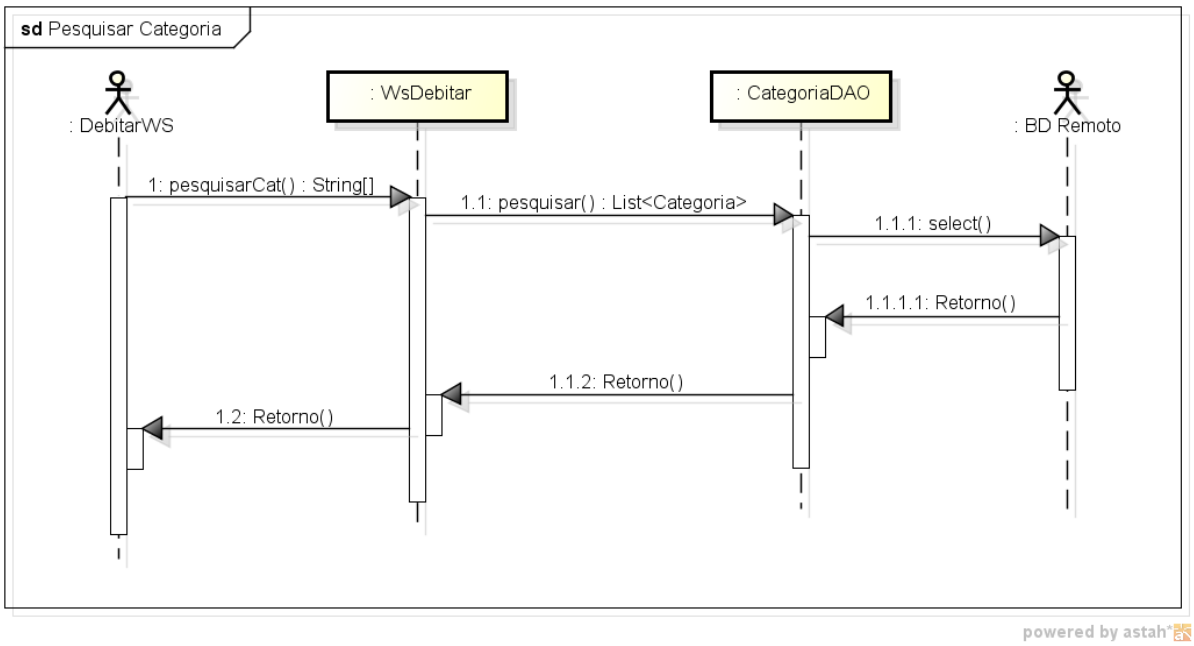
Excluir Categoria



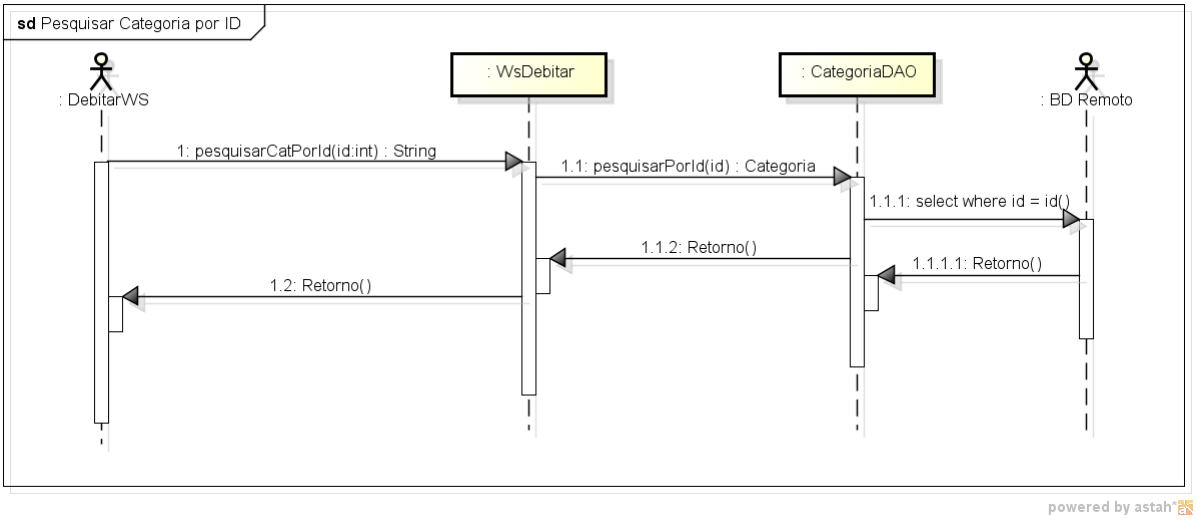
Gravar Categoria



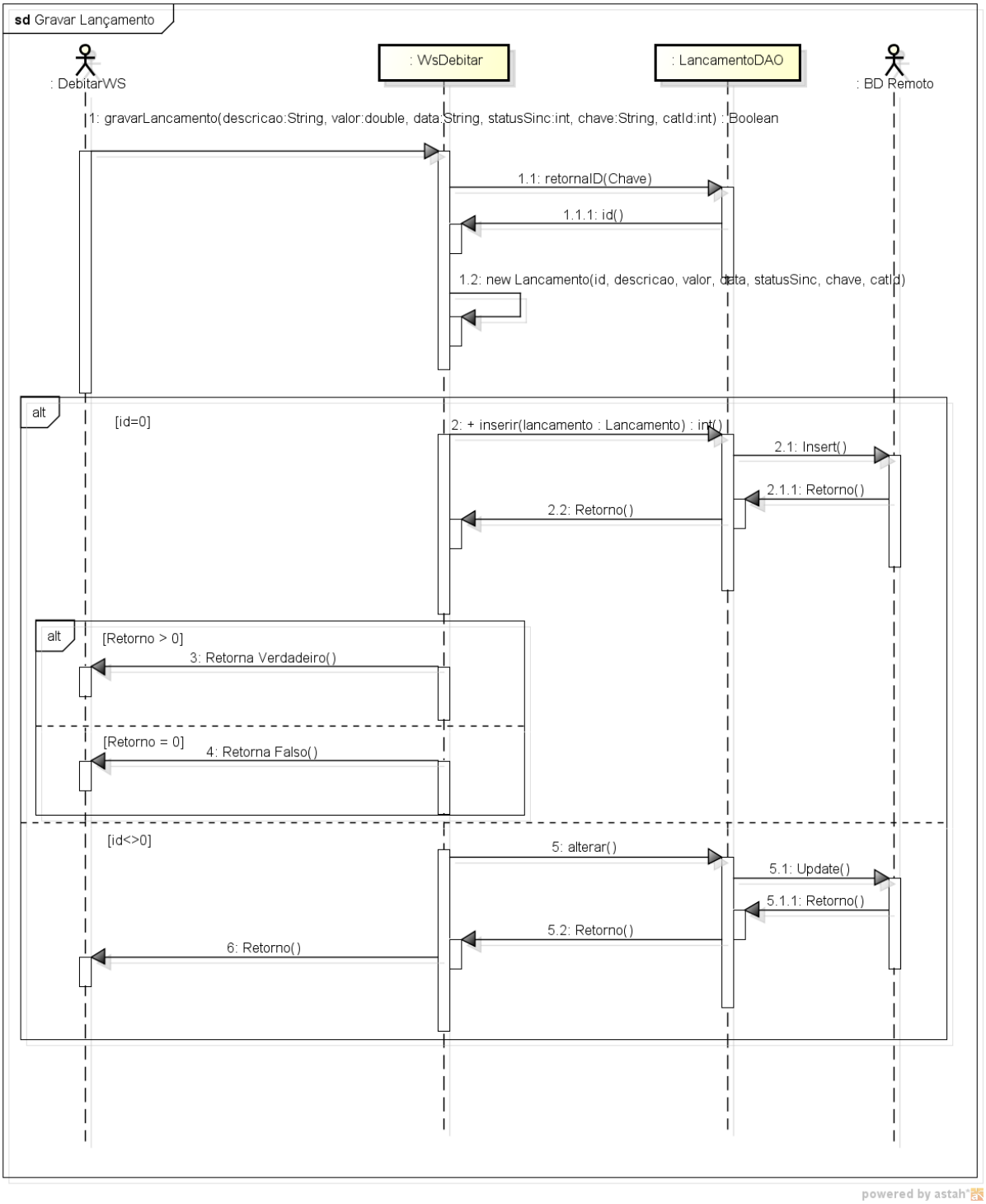
Pesquisar Categoria



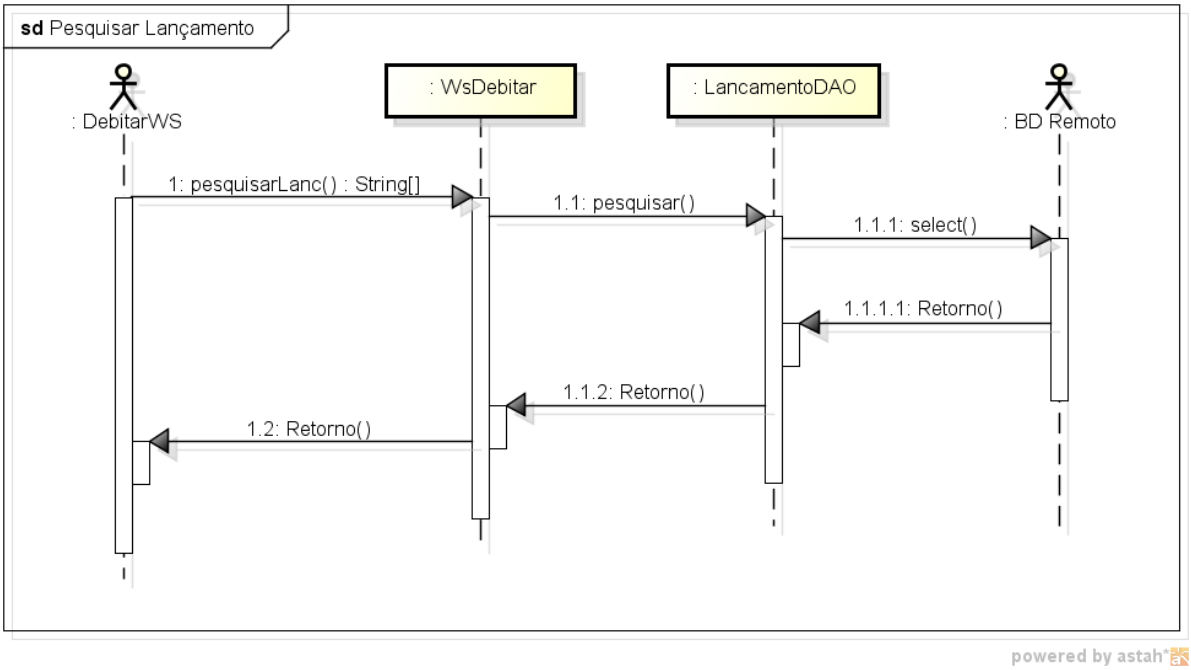
Pesquisar Categoria por ID



Gravar Lançamento



Pesquisar Lançamento



Pesquisar Lançamentos a Atualizar