|  |
| --- |
| Universidade do Estado do Rio de Janeiro  Instituto de Matemática e Estatística  Departamento de Informática e Ciência da Computação |
| Ecurrencies API |
| Uma abordagem assíncrona na implementação de um serviço de automação de pagamentos |
|  |
| **Rafael Brandão de Andrade** |
|  |

|  |
| --- |
| Rio de Janeiro  fevereiro de 2013 |

**Ecurrencies API**

Rafael Brandão de Andrade

Monografia submetida ao corpo docente do Instituto de Matemática e Estatística da Universidade do Estado do Rio de Janeiro - UERJ, como parte dos requisitos necessários à obtenção do grau de Bacharel em Informática e Tecnologia da Informação.

Banca Examinadora:

Profo : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Nome Completo do Orientador - Orientador

Professor(a) IME/UERJ

Profa: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rosa Maria Costa

Professora IME/UERJ

Profa: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Maria Clicia Stelling de Castro

Professora IME/UERJ

Rio de Janeiro, 16 de fevereiro de 2013.

Resumo

Atualmente existem dezenas de serviços de pagamento realizados via Internet. Os mais conhecidos no Brasil são *PagSeguro[[1]](#endnote-1)*, o mais usado no país, e *PayPal[[2]](#endnote-2)*, o mais usado no mundo. Cada um destes serviços funciona como uma verdadeira conta bancária online, em que transferências de dinheiro ocorrem com a velocidade de um clique, sem as burocracias que existem nas tradicionais transferências bancárias. Muitos sites de e-commerce já aceitam pagamentos realizados através de algum serviço de pagamento digital.

Estes oferecem ferramentas de automação na forma de API’s, que permitem a realização de pagamentos em massa, consulta de saldo e extratos, que são úteis para se integrar a sistemas de pagamentos já existentes em uma empresa.

Um dos problemas enfrentados é o de não haver um padrão reconhecido internacionalmente para a construção dessas API’s, o que pode levar a dificuldades de integração com esse tipo de serviço.

O objetivo deste trabalho é apresentar uma arquitetura possa abstrair as implementações desses serviços de automação.

Sumário

[Introdução 1](#_Toc348760859)

[Conceitos Básicos 2](#_Toc348760860)

[Formas de interação com sites de pagamento digital 2](#_Toc348760863)

[Direta 2](#_Toc348760864)

[Shopping Cart Interface (SCI) 2](#_Toc348760865)

[Application Program Interface (API) 2](#_Toc348760866)

[Liberty Reserve API 3](#_Toc348760867)

[Operações 3](#_Toc348760868)

[Preparação 3](#_Toc348760869)

[Tecnologias suportadas 4](#_Toc348760870)

[Execução 5](#_Toc348760871)

[Arquitetura Proposta 6](#_Toc348760872)

[O Agente MOM 6](#_Toc348760873)

[O protocolo *AMQP* 7](#_Toc348760874)

[Estrutura De Uma Mensagem AMQP 7](#_Toc348760875)

[Roteamento de Mensagens 8](#_Toc348760876)

[Exchanges 9](#_Toc348760877)

[Filas 11](#_Toc348760878)

[Bindings 11](#_Toc348760879)

[Uso do Protocolo AMQP no Projeto 11](#_Toc348760880)

[Formato da mensagem 12](#_Toc348760881)

[Google Protocol Buffers 12](#_Toc348760882)

[Detalhes de Implementação 15](#_Toc348760883)

[Preparação inicial 15](#_Toc348760884)

[Scala 15](#_Toc348760885)

[Google Protocol Buffers 15](#_Toc348760886)

[Gradle 15](#_Toc348760887)

[O framework OSGI 16](#_Toc348760888)

[Arquitetura em camadas 17](#_Toc348760889)

[A estrutura do projeto 18](#_Toc348760890)

[Avaliação de Desempenho 19](#_Toc348760891)

[Trabalhos Relacionados 20](#_Toc348760892)

[Conclusões 21](#_Toc348760893)

[Bibliografia 21](#_Toc348760894)

Lista de figuras

[Figura 1 - Formulário de criação e edição de uma API 3](#_Toc348760895)

[Figura 2 - Diagrama de casos de uso do sistema 4](#_Toc348760896)

[Figura 3 - Diagrama de atividades que define a execução de uma operação na API 5](#_Toc348760897)

[Figura 4 - Diagrama de atividades adequado à arquitetura proposta 7](#_Toc348760898)

[Figura 5 – Estrutura de uma mensagem AMQP 8](#_Toc348760899)

[Figura 6 - Modelo de roteamento de mensagens do protocolo AMQP versão 0.9.1 9](#_Toc348760900)

[Figura 7 – Exemplo de uso de um *topic exchange* 10](#_Toc348760901)

[Figura 8 - Distribuição dos produtores e consumidores para o serviço de transferência 12](#_Toc348760902)

[Figura 9 - Parte do diagrama de classes que representa uma transferência 13](#_Toc348760903)

[Figura 10 – O modelo OSGI dividido em camadas bem definidas 17](#_Toc348760904)

Lista de Tabelas

Lista de tabelas

Lista de Siglas e Símbolos

AMQP *Advanced Message Queuing Protocol*  
API *Application Program Interface*EAR *Enterprise Archive*  
HTML *HyperText Markup Language*HTTP *HyperText Transport Protocol*HTTPS *HyperText Transport Protocol Secure*IBAN *International Bank Account Number*IP *Internet Protocol*ISO *International Organization for Standardization*JAR *Java Archive*JMS *Java Messaging Service*  
JSON *JavaScript Object Notation*MOM *Message Oriented Middleware*NAPT *Network Address and Port Translation*NAT *Network Address Translation*NVP *Name Value Pair*OSI *Open Standards Interconnection*RPC *Remote Procedure Call*SASL *Simple Authentication and Security Layer*  
SCI *Shopping Cart Interface*SOA *Service Oriented Arquitecture*SOAP *Simple Object Access Protocol*SWIFT *Society for Worldwide Interbank Financial Telecommunication*TLS *Transport Layer Security*WAR *Web Archive*  
XML *Extensible Markup Language*

Capítulo 1

Introdução

Ao longo dos anos 90 a Internet possibilitou uma rápida expansão para empresas de e-commerce, removendo fronteiras, permitindo qualquer um a realizar uma compra diretamente do conforto de sua casa. Bastava apenas um cartão de crédito na hora de pagar e um endereço para a entrega de algum produto. Porém, um dos problemas enfrentados nessa época era com a segurança de dados críticos na hora de se realizar esse tipo de transação, houve diversos casos de fraudes e roubos realizados por pessoas mal intencionadas, justamente porque não havia controles de segurança adequados, que permitissem uma integração segura entre estes sites de e-commerce às empresas de cartão de crédito.

Isso permitiu o surgimento de um novo tipo de serviço de pagamento especializado para que não ocorram fraudes, em que as transações não podem ser rastreadas, dando mais confiança a quem realiza pagamentos pela Internet. Cada cliente de um desses serviços recebe um número de uma conta, que pode ser operada online, através do seu respectivo site ou pode manipulada por meio de API’s.

Uma das cinco maiores empresas em número de clientes que fornecem esses serviços é a *Liberty Reserve[[3]](#endnote-3)*. Este trabalho irá implementar um cliente para a *API[[4]](#endnote-4)* que ela disponibiliza, de uma forma que componentes de software que já tenham sido criados ou utilizados nessa implementação possam ser reutilizados nas implementações de outros serviços semelhantes (por exemplo, ao se implementar a API do serviço PayPal).

Ele será organizado da seguinte forma. O capítulo 2 abordará os conceitos básicos sobre cada serviço oferecido por esta API. O capítulo 3 discutirá sobre a arquitetura proposta, bem como a organização de alguns componentes e porque ela é capaz de abstrair a implementação desse tipo de serviço de automação em um projeto que permitirá a adição de outros serviços similares, prestados por outras empresas. O capítulo 4 mostrará os detalhes de implementação do sistema que será o cliente, como as tecnologias utilizadas na sua construção. O capítulo 5 irá comparar esta implementação com as implementações já oferecidas em diversas tecnologias pela própria empresa *Liberty Reserve*. O capítulo 6 finalizará com as conclusões tiradas ao longo da elaboração deste trabalho.

Capítulo 2

Conceitos Básicos



Formas de interação com sites de pagamento digital

Direta

Acontece através do próprio site, como um serviço de internet banking tradicional, em que o cliente deve fazer um login seguro para poder realizar operações em sua conta.

Shopping Cart Interface (SCI)

Esta é a forma que sites de e-commerce usam para poder oferecer a seus clientes um meio seguro de pagamento através de serviços de pagamento digital. Estes serviços oferecem botões e formulários HTML personalizados para que os sites de e-commerce possam redirecionar seus clientes para pagarem por uma compra efetuada.

Esta também é uma forma de interação direta, em que o cliente deve passar por controles de segurança adequados para provar sua identidade.

Application Program Interface (API)

API’s fornecem um meio de automação de acesso e operação de uma determinada conta, permitindo que o cliente não seja necessariamente uma pessoa física em frente a uma tela de computador.

As API’s normalmente são utilizadas na integração com um sistema de pagamentos já existente. Elas oferecem no mínimo três serviços:

* Serviço de transferência
* Serviço verificação de extrato (histórico de transações)
* Serviço de verificação se saldo

Liberty Reserve API

Operações

Além das operações fundamentais que uma API desse tipo de serviço deve realizar (transferência, verificação de extrato e verificação de saldo), a API da empresa Liberty Reserve possui mais dois:

* Serviço de verificação de nome de conta
* Serviço de verificação sobre transação efetuada

Preparação

De acordo com o guia que o site dispõe para consulta, há uma série de passos que devem ser seguidos para fazer um determinado sistema interagir com uma determinada conta:

* Criar uma API e habilitá-la. Por questões de segurança, o acesso realizado por API é desabilitado por padrão.
* Criar uma chave de segurança para esta API.
* Definir quais operações a API criada pode executar.

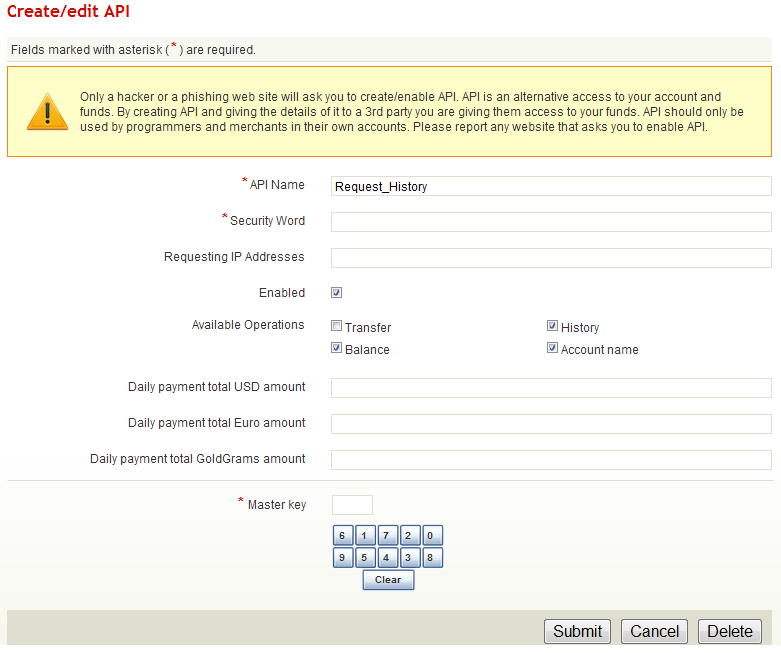


Figura - Formulário de criação e edição de uma API

De acordo com o site, quando o sistema estiver em um ambiente de produção é recomendado usar um servidor com IP fixo e fazer com que somente esse IP possa acessar a API criada, preenchendo o campo *“Requesting IP Adresses”* do formulário de criação de API’s. Mas através da técnica de *NAT / NAPT* será possível colocar vários servidores com endereços IP privados e utilizar somente um endereço IP público e fixo.

Das operações suportadas derivamos o seguinte diagrama de casos de uso:

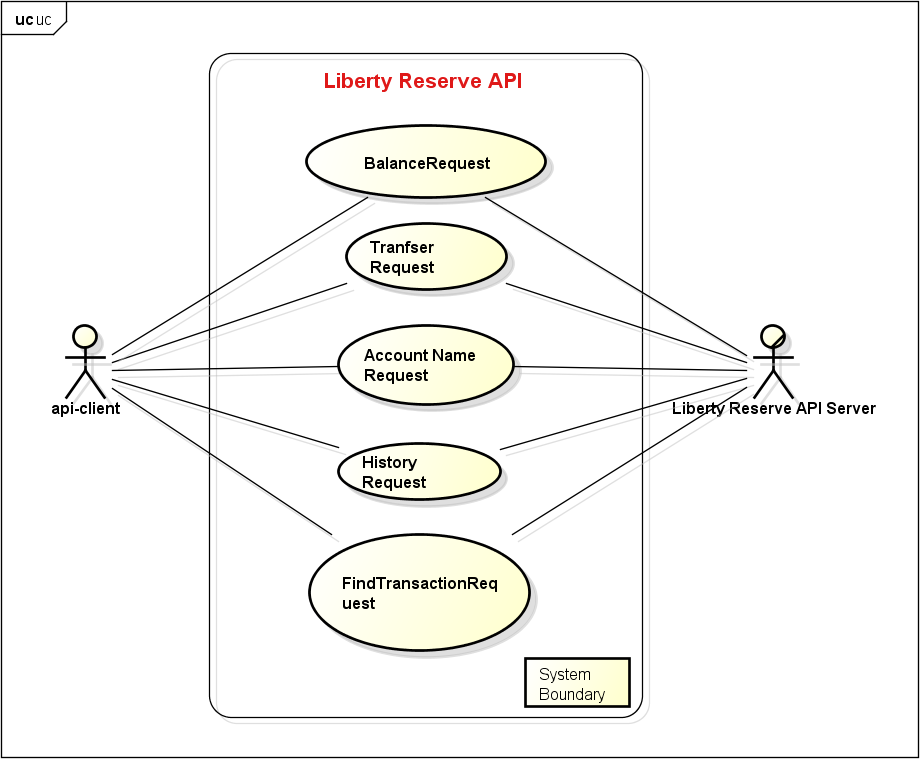


Figura - Diagrama de casos de uso do sistema

Tecnologias suportadas

Para cada uma das cinco operações disponibilizadas pela API, é necessária a elaboração de uma mensagem em um dos formatos suportados. A API oferece grande flexibilidade na integração, pois são suportados os seguintes formatos:

* XML
* JSON
* NVP
* SOAP

O manual disponível online no site da empresa mostra como deve ser feita a elaboração das mensagens para cada um dos formatos listados.

Execução

Para se acessar uma API devem-se seguir os seguintes passos:

* Elaborar uma mensagem no formato escolhido.
* Fazer uma requisição HTTPS (GET ou POST) ao servidor.
* Tratar a resposta do servidor.

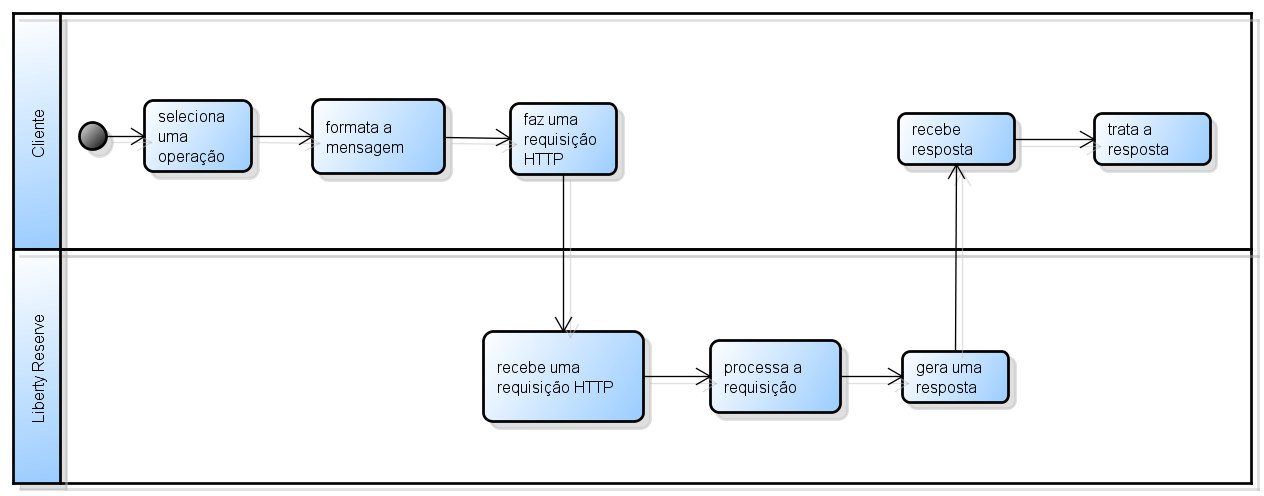


Figura - Diagrama de atividades que define a execução de uma operação na API

Capítulo 3

Arquitetura Proposta

A intenção deste projeto é criar um sistema que sirva de cliente da API criada pela empresa Liberty Reserve a fim de automatizar serviços fornecidos por ela, porém, de uma forma que permita a adição de novas *e-currencies* [[5]](#endnote-5), sem aumentar a complexidade do projeto.

A chave para isso está na criação de um sistema modular, orientado a mensagens. O cliente passa a ser o produtor de mensagens específicas ao domínio da *e-currency*. Com isso ele só precisa se preocupar na formatação das mensagens e no envio a um sistema de filas (agente *MOM*).

O Agente MOM

O agente orientado a mensagens é um sistema de filas. Como parte da arquitetura, optou-se pelo protocolo *AMQP*[[6]](#endnote-6) (protocolo avançado de orientação a mensagens). *AMQP* é baseado na API do *JMS*, mas em nível de protocolo, ou seja, em tese qualquer cliente *AMQP* pode se comunicar com qualquer broker *AMQP* (o sistema de filas), independente da plataforma ou linguagem usada.

O sistema de filas roteará as mensagens a consumidores específicos, e estes serão os responsáveis pela transformação dessas mensagens em um formato reconhecido pela API e também pela comunicação com esta API.

Isso denota um fraco acoplamento entre o produtor e o consumidor, permitindo o uso de tecnologias distintas no consumidor e no produtor, tornando mais fácil a integração do cliente com sistemas legados, visto que há apenas a necessidade de se preocupar com o formato da mensagem e com a comunicação com um sistema de filas.

No caso de adição de novas *e-currencies* ao sistema, para cada *e-currency* adicionada corresponderá um novo formato de mensagem, ou seja, um novo domínio.

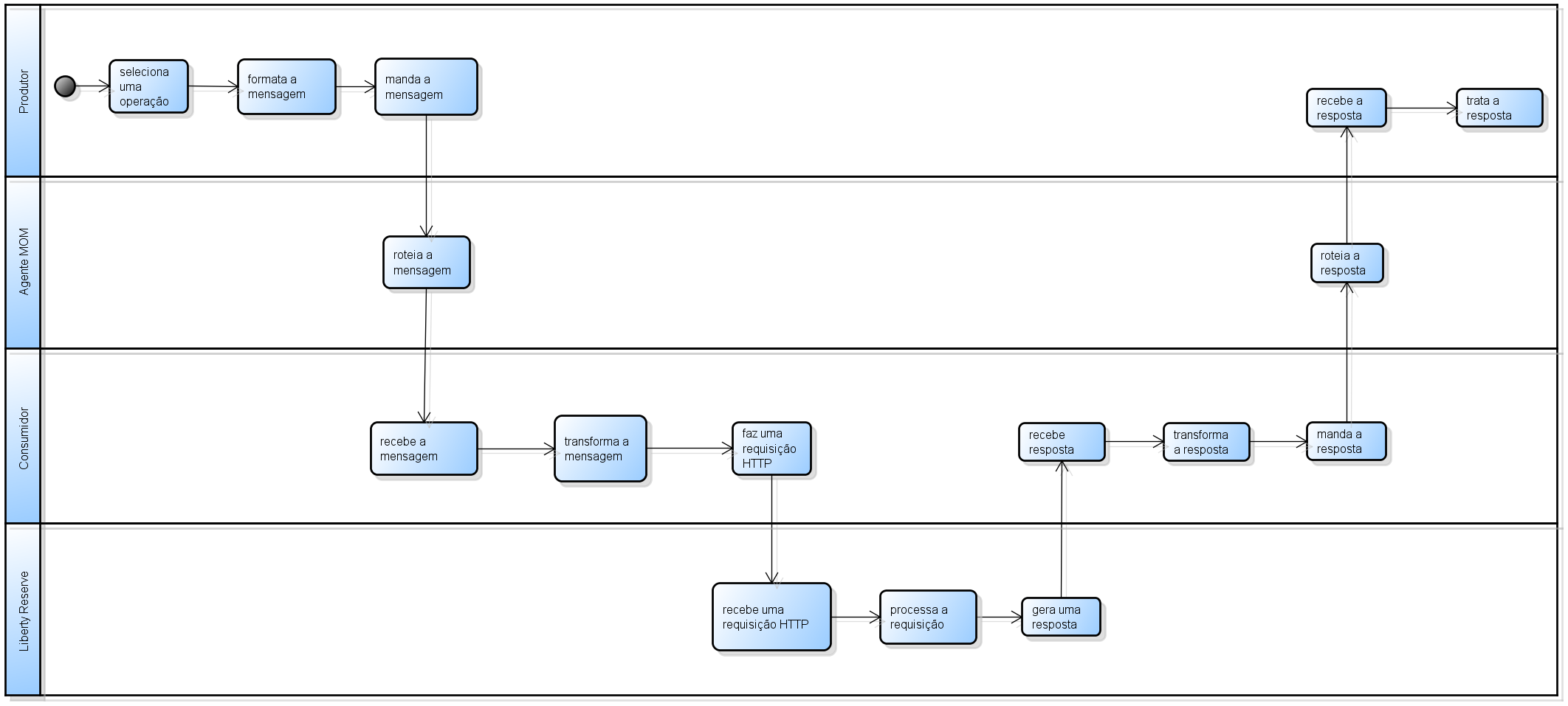


Figura - Diagrama de atividades adequado à arquitetura proposta

O protocolo *AMQP*

O protocolo *AMQP* define o formato da mensagem que deve ser codificada em um *stream* de bytes para ser mandado pela rede. Então qualquer cliente ou *broker* que possa criar ou interpretar mensagens que se adequem a esse formato pode se comunicar com qualquer outro cliente ou *broker* *AMQP*, independente da linguagem e/ou sistema operacional usado em sua implementação.

Ele é um protocolo binário, que atua na camada de aplicação do modelo *OSI*, elaborado para suportar de forma eficiente uma gama de aplicações orientadas a mensagens. Ele provê controle de fluxo, comunicação orientada a mensagens e garantias de entrega, como “*at least once”*, “*at most once”* e *“exactly once”*. Também provê mecanismos de autenticação e encriptação utilizando *SASL* e *TLS*.

Estrutura De Uma Mensagem *AMQP*

Mensagens no modelo *AMQP* possuem atributos (propriedades e cabeçalho) e *payload* (a mensagem em si, em um array de bytes). Propriedades são metadados já previstos pela especificação 0.9.1, bem como:

* Conteúdo da mensagem
* Codificação
* *Routing Key*
* Modo de Entrega
* Prioridade
* Momento da entrega (*timestamp*)
* Período de expiração da mensagem
* Identificação do produtor

O cabeçalho oferece ao programador é qualquer atributo criado pelo programador que pode ser mais uma metainformação da mensagem.



Figura – Estrutura de uma mensagem AMQP

Roteamento de Mensagens

Em sua versão 0.9.1, a imagem abaixo ilustra o protocolo em seu mais alto nível de abstração:

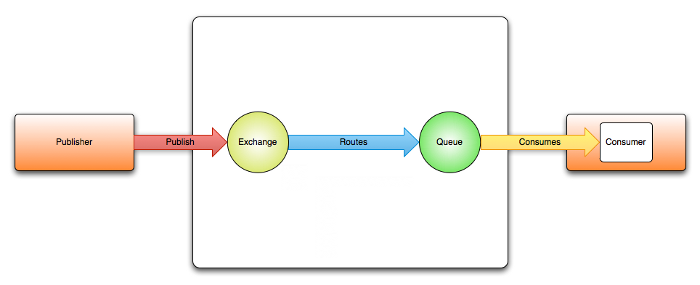


Figura - Modelo de roteamento de mensagens do protocolo AMQP versão 0.9.1

O agente responsável pelo sistema de filas é o *broker* *AMQP*. Ele oferece três entidades que devem ser manipuladas pelos clientes, tanto produtores quanto consumidores, para que estes possam enviar ou receber mensagens. São as filas, os *exchanges* e as *binding keys*.

Exchanges

São as entidades para as quais os produtores devem enviar as mensagens. Elas roteiam as mensagens para zero ou mais filas. O algoritmo de roteamento varia de acordo com o tipo de *exchange* criado e com um parâmetro também passado pelo produtor, chamado de *routing key*. Os tipos são:

1. *Direto* – Redireciona as mensagens baseadas na *routing key* utilizada para a publicação.
   1. Uma fila se vincula a um *exchange* direto com uma *binding key* **K** ;
   2. Quando uma nova mensagem com uma *routing key* **R** chega a um *exchange* direto, ele entrega a mensagem à esta fila se **K = R**.
2. *Fanout* – Redireciona uma cópia de qualquer mensagem que chega a todas às filas vinculadas a este tipo de *exchange*, independente da *routing key* utilizada na publicação da mensagem.
3. *Topic* – Redireciona mensagens a uma ou mais filas baseado na *routing key* utilizada na publicação e no padrão utilizado para vincular a fila ao *exchange*. As *routing keys* utilizadas na publicação não podem ser arbitrárias, devem seguir um padrão de palavras separadas por pontos, como por exemplo *"stock.usd.nyse"*. Vejamos um exemplo:

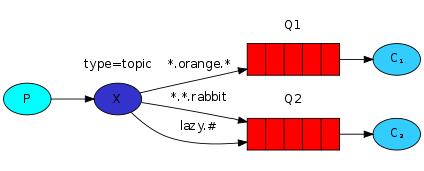


Figura – Exemplo de uso de um *topic exchange*

Vamos supor que os produtores usem como *routing keys* que consistem em três palavras seguidas por pontos: *velocidade*.*cor*.*animal*. A fila Q1 está vinculada ao *exchange* **X** pela *binding key “\*.orange,\*”*. A fila Q2 está vinculada ao exchange **X** com duas *binding keys*: *“\*.\*.rabbit”* e *“lazy.#”*.

O curinga *\** pode ser substituído por uma palavra e o curinga *#* pode ser substituído por zero ou mais palavras. A partir daí pode-se concluir que os consumidores da fila Q1 estão interessados em qualquer animal laranja e que os consumidores da fila Q2 estão interessados em qualquer animal que seja um coelho ou qualquer animal que seja preguiçoso.

1. *Cabeçalho* – As mensagens são roteadas a filas específicas de acordo com o conteúdo dos seus cabeçalhos. Este *exchange* é mais indicado para rotear mensagens que expressam melhor seus atributos em seus próprios cabeçalhos do que o uso de uma *routing key*.
2. *Padrão* – O *exchange* padrão é um *exchange* direto, representado por uma *string* vazia, previamente criado pelo *broker*. Qualquer fila declarada por algum cliente sempre estará vinculada ao *exchange* padrão pelo seu próprio nome como *binding key*. Então quando uma mensagem chega a este *exchange* com uma *routing key* **R**, ele entregará a mensagem à fila declarada com o nome **F** se **R = F**.

Filas

Assim como em qualquer outro sistema orientado a mensagens, as filas no modelo *AMQP* devem ser o ponto de armazenamento das mensagens para os consumidores da aplicação. Se mais de um consumidor estiver vinculado a uma fila, o *broker AMQP* automaticamente distribuirá as mensagens em um esquema *round robin*.

Bindings

*Bindings* são as regras que os *exchanges* usarão para rotear as mensagens às filas. Qualquer fila declarada já estará vinculada automaticamente ao *exchange* padrão pelo seu nome. Como já visto na seção sobre *exchanges*, as filas devem ser vinculadas por meio de *binding* keys a um ou mais *exchanges* para que possam receber mensagens.

Uso do Protocolo AMQP no Projeto

A parte que implementará este projeto será o consumidor de uma fila do *broker AMQP*. A figura abaixo mostra o esquema utilizado neste projeto:

Todas as mensagens de clientes do sistema devem ser enviadas ao *exchange* “*ecurrencies*” com uma *routing key* que denote a identificação do provedor da *e-currency* e a identificação do serviço requisitado: *“<ecurrency-id >.<service-id>”*.

Como a fila estará vinculada a este *exchange* com a *binding key* *“\*.\*”* ela receberá qualquer mensagem enviada com uma *routing key* que siga este padrão, como por exemplo *“liberty-reserve.transfer”*. Ao receber uma mensagem o consumidor redirecionará a mensagem à implementação do serviço que corresponde ao requisitado pela *routing key*, **permitindo que sejam disponibilizadas implementações de serviços de diversas *e-currencies***.

No caso da API da empresa Liberty Reserve, todas as mensagens que se destinam ao processamento pela sua API deverão ser publicadas com uma *routing key* *“liberty-reserve.<servisse-id>”*, como por exemplo *“liberty-reserve.transfer”*.

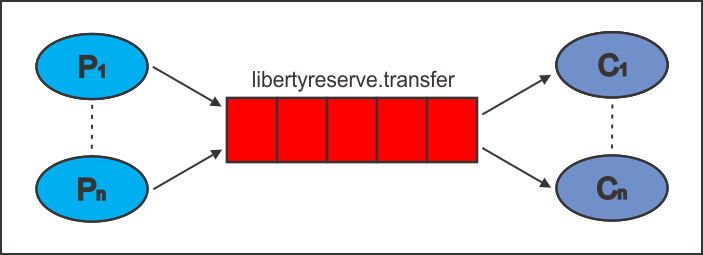


Figura - Distribuição dos produtores e consumidores para o serviço de transferência

Formato da mensagem

Como o sistema possui flexibilidade para integrar diferentes plataformas e linguagens, é necessário adotar um formato serializável de mensagem neutro à linguagem e à plataforma para a representação do domínio da *e-currency*.

Google Protocol Buffers

Ao invés de escolher formatos já conhecidos no mercado, como *XML* ou *JSON*, optou-se pela utilização de *Protocol Buffers* [[7]](#endnote-7). Trata-se de um formato de dados estruturados, desenvolvido pelo Google para uso interno como protocolo *RPC*. Há algum tempo a empresa o disponibilizou como código aberto para a comunidade.

O diferencial é que esse formato gera automaticamente as classes que representam esses dados, assim como os métodos que serializam e desserializam os dados para um formato binário, mais enxuto que *XML* e *JSON*. Além disso, a serialização de formatos de texto como *XML* e *JSON*, exigida pelo protocolo *AMQP*, a acopla à linguagem utilizada no processo, fazendo com que produtor e consumidor tenham que implementar a mesma linguagem de programação.

Com *protocol buffers,* define-se como os dados devem ser estruturados em um arquivo de extensão *proto*, compila-se esse arquivo e automaticamente são geradas as classes e métodos de serialização das classes referentes ao domínio. A partir deste momento o termo *protocol buffers* será abreviado para *protobuffers*.

Oficialmente são suportados os compiladores para as linguagens Java, C++ e Python. Porém existem diversos projetos não oficiais que suportam compiladores para outras dezenas de linguagens.

Um diagrama de classes foi feito em cima de todo o domínio da API da *e-currency* Liberty Reserve. Ele foi refinado para se adaptar ao uso de *protobuffers*.

Como o diagrama de classes é razoavelmente grande para ser exibido como uma imagem neste documento, destacou-se apenas um trecho dele para se mostrar como foi feito parte do arquivo *proto* que gerou as classes do domínio na plataforma escolhida para a implementação, neste caso o Java.

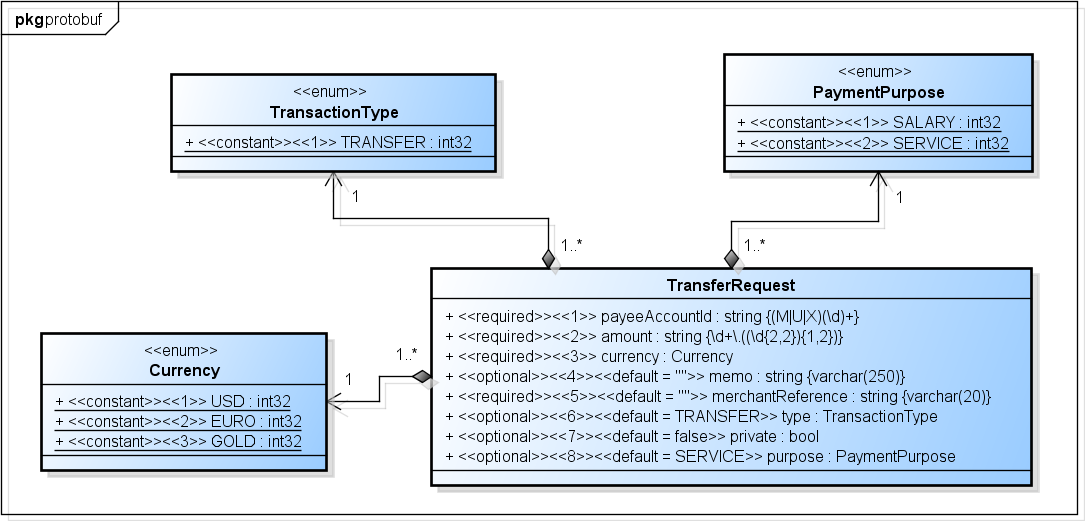


Figura - Parte do diagrama de classes que representa uma transferência

Do diagrama de classes, já refinado para o uso de *protobuffers,* foi criado o seguinte trecho do arquivo *proto*:

|  |
| --- |
| **enum** Currency {  **USD** = 1;  **EURO** = 2;  **GOLD** = 3;  }  **enum** PaymentPurpose {  **SALARY** = 1;  **SERVICE** = 2;  }  **enum** TransactionType {  **TRANSFER** = 1;  }  **message** TransferRequest {  **required string** payeeAccountId = 1;  **required string** amount = 2;  **required** Currency currency = 3;  **required string** memo = 4;  **required string** merchantReference = 5;  **optional** TransactionType type = 6 [**default** = TRANSFER];  **optional** **bool** private = 7 [**default** = false];  **optional** PaymentPurpose purpose = 8 [**default** = SERVICE];  } |

Capítulo 4

Detalhes de Implementação

Este capítulo irá expor os detalhes pertinentes às tecnologias escolhidas para a implementação do sistema de filas, o formato de mensagem adotado, assim como a disposição dos produtores e consumidores do sistema.

O repositório *git* que mantém todo o código fonte deste projeto e este documento encontra-se em <https://github.com/rafael-st-rj/ecurrencies>.

Preparação inicial

O ambiente de desenvolvimento deve ser preparado para a correta compilação e distribuição do projeto.

Scala

O projeto é todo elaborado na linguagem de programação *Scala*, que roda sobre a plataforma Java, então é necessário instalar e configurar corretamente o *Java Development Kit 6* ou superior, disponibilizado pela Oracle.

Google Protocol Buffers

O compilador *protobuffer* deve ser baixado do site do projeto e copiado para uma pasta na máquina. No caso de um sistema operacional \*Unix os seus fontes devem ser compilados, no caso de Windows ele já é disponibilizado como um arquivo executável.

Deve-se adicionar a pasta onde se encontra o arquivo executável do compilador à variável *path* do sistema operacional.

Gradle

*Gradle* [[8]](#endnote-8)é uma ferramenta escrita em *Groovy* que gerencia builds automatizados de software e a busca de artefatos e bibliotecas. Seus plug-ins oficiais dão a ele uma gama de funções como compilação, realização testes de unidade, realização testes de integração, empacotamento e distribuição dos artefatos criados, conectar-se a repositórios de bibliotecas por diversos protocolos, relatórios de build e de testes detalhados, gerenciamento de releases, integração com ferramentas já conhecidas como *Ant*[[9]](#endnote-9) e *Maven*[[10]](#endnote-10), dentre muitas outras. Tudo isso é realizado em um pipeline de fases bem definidas. Fora isso, ainda existe a possibilidade de criação do seu próprio plug-in para se adequar a qualquer requisito de gerência de software.

Todo o projeto usa o *Gradle* para compilação, execução de testes, gerenciar dependências e release.

A sua instalação é simples. Além da plataforma Java JDK instalada, deve-se baixar a ferramenta do link <http://www.gradle.org/downloads>, descompactar para uma pasta e adicionar o endereço da pasta *bin* (dentro da pasta aonde a ferramenta foi descompactada) à variável *path* do sistema operacional.

O Toolkit Akka

Um dos requisitos do projeto é desenvolver um sistema modular orientado a componentes, em que cada módulo possui uma função ou propósito bem definido.

O toolkit *Akka*[[11]](#endnote-11) oferece um ambiente para o desenvolvimento de aplicações concorrentes, orientadas a eventos, altamente escaláveis e distribuídas. Suporta as linguagens Java e Scala. Foi uma ótima escolha para a implementação da API Liberty Reserve e para posterior implementação de qualquer outra API semelhante.

O modelo tradicional de distribuição de bibliotecas (normalmente arquivos *JAR*) e execução de aplicações (arquivos *WAR* ou *EAR*) do Java dificulta a adição ou modificação dos componentes de um sistema em tempo de execução. Na verdade não é possível atualizar ou adicionar um componente em tempo de execução, ou seja, para atualizar um componente é necessário reiniciar todo o sistema (ou o contêiner que roda o sistema, no caso de aplicações corporativas), causando alguma interrupção no serviço prestado.

Normalmente todas as dependências e bibliotecas de um sistema rodam sobre um mesmo *classpath* e não são permitidas versões diferentes de uma mesma biblioteca sendo executadas ao mesmo tempo.

A tecnologia *OSGI* [[12]](#endnote-12) foi criada para criar um ambiente colaborativo de desenvolvimento de software. Ela define uma série de especificações que tornam a plataforma Java mais dinâmica, permitindo a programação orientada a componentes. Proporciona uma forma de se implementar *SOA* em Java, em que componentes se comunicam através de serviços com escopos bem definidos.

*OSGI* permite a criação de módulos altamente coesos e fracamente acoplados. Este é um pilar para o desenvolvimento de aplicações robustas e fáceis de testar. *OSGI* facilita o reuso dos componentes de software em diferentes aplicações ou sistemas. Módulos podem ser desenvolvidos, testados, atualizados e gerenciados de uma forma individual, sem causar grandes impactos a outros módulos que compõem um sistema.

Arquitetura em camadas

A seguinte imagem define bem como a arquitetura do framework *OSGI* é dividida em camadas:

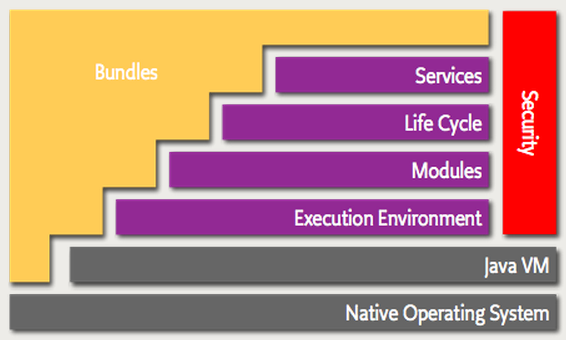


Figura – O modelo OSGI dividido em camadas bem definidas

A seguir segue uma sucinta descrição das camadas do framework:

* **Bundles** – um *bundle* é um arquivo JAR que possuí o arquivo META-INF/MANIFEST.MF com algumas entradas adicionais, como o nome que o identifica e o seu número de versão.
* **Services** – a camada de serviços conecta os *bundles* de uma forma dinâmica, oferecendo um modelo de procura, publicação e ligação, fazendo com que qualquer objeto Java possa ser publicado como um serviço**.**
* **Life-Cycle** – esta camada oferece uma API para a instalação, inicialização, parada, atualização e desinstalação de *bundles*.
* **Modules** – define como um *bundle* importar ou exportar código de/para outros bundles.
* **Security** – é a camada que lida com os aspectos de segurança.
* **Execution Environment** – é a plataforma que implementa o framework *OSGI*, oferecendo um contêiner que gerencia todos os bundles. As três principais são: Eclipse Equinox, Apache Felix e Knopflerfish.

A estrutura do projeto

capítulo 5

Avaliação de Desempenho

Capítulo 6

Trabalhos Relacionados

capítulo 7

Conclusões

# Bibliografia

**Não há fontes bibliográficas no documento atual.**

1. PagSeguro [Online] em <https://pagseguro.uol.com.br/> [↑](#endnote-ref-1)
2. PayPal [Online] em <https://www.paypal.com/> [↑](#endnote-ref-2)
3. Liberty Reserve [Online] em <https://www.libertyreserve.com/> [↑](#endnote-ref-3)
4. Liberty Reserve API [Online] em <https://www.libertyreserve.com/en/help/apiguide/> [↑](#endnote-ref-4)
5. E-Currency [Online] em <http://en.wikipedia.org/wiki/Electronic_money> [↑](#endnote-ref-5)
6. AMQ [Online] em <http://www.amqp.org/> [↑](#endnote-ref-6)
7. Protocol Buffers [Online] em <https://developers.google.com/protocol-buffers/docs/overview> [↑](#endnote-ref-7)
8. Gradle [Online] em <http://www.gradle.org/> [↑](#endnote-ref-8)
9. Apache Ant [Online] em <http://ant.apache.org/> [↑](#endnote-ref-9)
10. Apache Maven [Online] em <http://maven.apache.org/> [↑](#endnote-ref-10)
11. Akka Toolkit [Online] em <http://akka.io/> [↑](#endnote-ref-11)
12. OSGI [Online] em [http://www.osgi.org](http://www.osgi.org/)/ [↑](#endnote-ref-12)