CÁSSIO BUENO VILELA CORRÊA

ESTÁGIO EM EMPRESA DE DESENVOLVIMENTO DE APLICAÇÕES PARA TERMINAIS POINT OF SALE

LAVRAS – MG

2014

CÁSSIO BUENO VILELA CORRÊA

ESTÁGIO EM EMPRESA DE DESENVOLVIMENTO DE APLICAÇÕES PARA TERMINAIS POINT OF SALE

Relatório de Estágio Supervisionado apresentado ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso para obtenção do título de Bacharel em Ciência da Computação

Orientadora
Dra. Marluce Rodrigues Pereira

LAVRAS – MG

2014

CÁSSIO BUENO VILELA CORRÊA

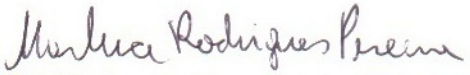
ESTÁGIO EM EMPRESA DE DESENVOLVIMENTO DE APLICAÇÕES PARA TERMINAIS POINT OF SALE

Trabalho de Conclusão de Curso de Graduação apresentado ao Colegiado do Curso de Bacharelado em Ciência da Computação, para obtenção do título de Bacharel.

APROVADA em 18 de novembro de 2014.

Dr. Wilian Soares Lacerda

Adalberto Mendes



Drª Marluce Rodrigues Pereira (Orientadora)

LAVRAS-MG
Novembro/2014

AGRADECIMENTOS

Primeiramente agradeço aos meus pais, José Luiz e Adozinda, pelo apoio incondicional em toda a minha vida.

Aos meus irmãos, Gisele e Flávio, que sempre se preocuparam com seu "irmãozinho".

Aos meus amigos que me deram apoio e se divertiram comigo tornando minha graduação inesquecível.

A minha namorada Marcelle, que sempre esteve ao meu lado me ajudando a superar todos os momentos difíceis.

Aos meus professores pelos desafios e conhecimentos transmitidos.

A minha orientadora Marluce Rodrigues pela a atenção e suporte no desenvolvimento deste trabalho.

Aos meus companheiros da Cloudwalk e Plano Bê que me receberam de braços abertos e confirmaram que eu estava certo ao escolher a curso de Ciência da Computação.

RESUMO

Terminais *point of sale* (POS) são sistemas embarcados que auxiliam no gerenciamento de ambientes comerciais. Um dos problemas encontrados no desenvolvimento de software para estes dispositivos, é a falta de portabilidade das aplicações desenvolvidas. Neste trabalho é apresentada a descrição de um período de estágio realizado na empresa Plano Bê Desenvolvimento de Sistemas, esta empresa possui como principal produto um framework, que permite a portabilidade de sistemas de software entre diferentes modelos de terminais POS. Foram realizadas as atividades de treinamento para utilização e desenvolvimento para terminais POS, foi acompanhado a aplicação do processo de desenvolvimento de software SCRUM, foi implementado na linguagem POSXML uma aplicação para integrar uma plataforma web de realização de pedidos e terminais POS, também foi realizado o atendimento a clientes. Além destas atividades neste trabalho é apresentado o estudo do padrão de segurança *Europay, Mastercard e Visa* (EMV) para transações monetárias em terminais POS, que utilizem cartões com chip de circuito integrado.

Palavras-Chave: Terminais Point of Sale; Framework; EMV; SCRUM.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AAC Application Authorization Cryptogram

AC Application Cryptogram

ADF Application Definition File

AFL Application File Locator

AID Application Identifier

AIP Application Interchange Profile

API Application Programming Interface

ARQC Application Request Cryptogram

ATC Application Transaction Counter

ATR Answer To Reset

CDA Combined Dynamic Authentication

CRM Card Risk Management

CVM Cardholder Verification Method

DDA Dynamic Data Authentication

DDL Dynamic Data Authentication Data Object List

EMV Europay, Mastercard and Visa

HTTP Hypertext Transfer Protocol

IAC Issuer Action Code

IAD Issuer Authorization Data

JSON JavaScript Object Notation

LCOL Lower Consecutive Offline Limit

POS Point Of Sale

POSXML Point Of Sale Extended Markup Language

PSE Payment System Environment

SDA Static Data Authentication

SP Story Points

TAC Terminal Action Code

TC Transaction Certificate

TEF Transferência Eletrônica de Fundos

TLV Tag Length Value

TRM Terminal Risk Management

TVR Terminal Verification Results

UCOL Upper Consecutive Offline Limit

1 INTRODUÇÃO

1.1 Contextualização

O desenvolvimento para sistemas embarcados vem se tornando uma importante área dentro da área de Ciência da Computação, isto se deve em parte a grande proporção que estes sistemas representam no mercado, chegando a abranger 99% do mercado de sistemas de software (GANSSLE, 1999). O desenvolvimento para estes sistemas se diferencia do desenvolvimento de aplicações de software para PCs (*Personal Computers*) (SANGIOVANNI-VINCENTELLI; MARTIN, 2001) devido ao fato de sofrer fortes restrições em questões como velocidade de processamento, capacidade de memória e consumo de energia.

Para se superar as restrições de desempenho que os sistemas embarcados sofrem, os programadores usam linguagens de baixo nível como *C* ou até mesmo *assembly* na implementação das aplicações para sistemas embarcados. Outro requisito das implementações para sistemas embarcados é a necessidade de suporte via hardware para *debug* e validação de desempenho.

Uma solução para o desenvolvimento para sistemas embarcados é o *design* baseado em plataforma (SANGIOVANNI-VINCENTELLI; MARTIN, 2001). Uma plataforma é uma camada de abstração que esconde os detalhes de implementação. É composta por uma biblioteca de elementos caracterizados por modelos que representam as funcionalidades dos elementos e oferecem estimativas de quantidades que são importantes para o projetista de aplicações para sistemas embarcados. Devido às características únicas de cada sistema embarcado a plataforma é uma abstração de uma familia de microarquiteturas. Os componentes de uma biblioteca de uma família de microarquiteturas podem ser usados por múltiplas aplicações, gerando diferentes sistemas embarcados.

Uma API (*Application Programming Interface*) é uma interface de alto nível utilizada para programar as aplicações de software embarcado, abstraindo

detalhes da arquitetura como os núcleos programáveis, o subsistema de memória, o subsistema de entrada e saída, e as conexões de rede.

Inúmeros sistemas embarcados são encontrados no mercado e cada um destes dispositivos se diferenciam em suas arquiteturas e aplicações, sendo que cada sistema embarcado tem como objetivo solucionar um número reduzido de problemas específicos (GANSSLE, 1999). A empresa Plano Bê Desenvolvimento de Sistemas, local de realização deste estágio, atua no desenvolvimento de soluções para terminais *point of sale* (POS), que são sistemas embarcados com o objetivo de substituir antigos dispositivos utilizados no gerenciamento de ambientes comerciais (WHATIS, 2014). Os terminais POS possuem a capacidade de processar transações e se conectar com outros dispositivos em uma rede. Vale lembrar que a capacidade de processamento de informações destes dispositivos é muito limitada se comparada a dos computadores pessoais.

Assim como o desenvolvimento para outros sistemas embarcados, o desenvolvimento para terminais POS é custoso. As aplicações desenvolvidas para um tipo de terminal POS não é portável para outro terminal POS. A empresa Plano Bê oferece uma solução para a portabilidade das aplicações para terminais *Point of Sale* (POS). São exemplos de usuários da solução desenvolvida pela empresa:

- Administradoras/Processadoras de Cartões
- Desenvolvedores de Softwares
- Redes de força de Vendas
- Operadores Logísticos
- Empresas de arrecadação e cobrança
- Entre outros diversos segmentos de mercado

O perído de estágio, descrito neste trabalho, teve a carga horária total de 300 horas. Sendo 30 horas semanais, as quais foram divididas em 6 horas diárias

no horário de 8 às 11 horas e das 13 às 16 horas como especificado no termo de compromisso de número 1292/2014 firmado entre a UFLA e a empresa.

1.2 Objetivos

A realização do período estágio tem como principal objetivo proporcionar ao aluno experiência profissional na área de desenvolvimento de software para terminais *POS*. Sendo objetivos específicos a realização das seguintes atividades durante o período de estágio:

- Desenvolver uma aplicação para um terminal *POS*;
- Acompanhar o processo de projeto e desenvolvimento de software;
- Interação com clientes;
- Aprendizado de conteúdos complementares.

1.3 Justificativa

O desenvolvimento de aplicações para sistemas embarcados exige muitos recursos devido as características destes sistemas, além disso cada aplicação necessita ser reestruturada para ser aplicável em novos dispositivos. Portanto é interessante desenvolver uma solução, a qual torne uma aplicação desenvolvida para um sistema embarcado portável para outros dispositivos, para assim se ter uma economia de esforços.

Este período de estágio proporciona a interação com uma solução para este problema desenvolvendo importantes capacidades para o aluno. Ao superar os desafios neste estágio supervisionado, o aluno aprimora suas habilidades de desenvolvimento de software para arquiteturas com limitações de hardware, obtém conhecimento de modelos de processos de desenvolvimento de software, vivencia

a interação com clientes e a necessidade constante de se obter novos conhecimentos para superar desafios.

1.4 Organização do texto

Este documento esta organizado em sete capítulos: introdução, descrição geral do local do estágio, descrição das atividades desenvolvidas, descrição dos processos técnicos e outras particulatidades, conclusão, referências bibliográficas e apêndice.

O capítulo 1 contextualiza e introduz os objetivos a serem alcançados durante o período de estágio. O capítulo 2 apresenta a descrição do local de estágio, contendo o histórico e a área de atuação da empresa. Além disso apresenta alguns clientes da empresa. O capítulo 3 descreve de forma breve as atividades realizadas no estágio, listando-as uma a uma, em ordem cronológica. No capítulo 4 é realizada a descrição detalhada das principais atividades executadas, apresentando a teoria que suporta estas atividades. O capítulo 5 expõe conclusões sobre o aproveitamento e dificuldades encontradas no período de estágio. Ao fim é apresentado as referências utilizadas neste trabalho e o apêndice, que descreve algumas das implementações realizadas durante o estágio.

2 DESCRIÇÃO GERAL DO LOCAL DE ESTÁGIO

2.1 Histórico

Fundada no final do ano de 2005, a empresa Plano Bê Desenvolvimento de Sistemas[1] tem como objetivo o desenvolvimento de aplicações para terminais *POS*. Com as diversas limitações e dificuldades encontradas no desenvolvimento para estes dispositivos foi identificada a necessidade de se desenvolver uma solução, que facilitasse o desenvolvimento de sistemas de software para terminais POS.

A solução deveria cumprir três premissas. A primeira seria a simplicidade de uso e praticidade, portanto ela deveria facilitar o desenvolvimento da aplicação com uma linguagem de fácil aprendizado e rápido desenvolvimento. A segunda premissa é que a aplicação desenvolvida deveria ser portável: uma mesma aplicação desenvolvida para um tipo de terminal POS deveria ser portável para outro sem modificar o código desenvolvido. A terceira e última premissa é que a aplicação deve ser possível de ser atualizada remotamente.

Com o objetivo de atender a primeira premissa foi criada uma linguagem baseada no padrão *xml* e que possui comandos específicos para trabalhar com terminais *POS*. O *walk framework* foi a aplicação desenvolvida para que a segunda premissa fosse atendida. Para possuir a terceira premissa foi desenvolvida uma plataforma de atualização remota chamada *walk server* que é um *host* em que o terminal *POS* se conecta e baixa a aplicação de forma remota.

A Plano Bê ao longo dos anos desenvolveu diversas soluções para diversos ramos de mercado, sendo que em 2012 foi possível criar a empresa autônoma de desenvolvimento de soluções para o mercado de saúde e bem estar. Esta empresa hoje chamada de *CloudMed*[2].

[1] http://www.planobe.com.br
[2] http://cloudmed.io

Em 2013 foi desenvolvida a plataforma *CloudWalk*[3], que é a evolução da plataforma *walk* de desenvolvimento para terminais *POS* para um ambiente *Cloud Computing* onde as soluções já desenvolvidas foram disponibilizadas via serviços *WEB*. A solução *CloudWalk* teve uma boa aceitação e se tornou um dos principais produtos da Plano Bê.

2.2 Área de Atuação

A empresa Plano Bê desenvolvimento de sistemas atua no mercado de desenvolvimento de sistemas de software para sistemas embarcados. Sendo o foco principal da empresa os terminais POS. Os dois principais produtos desenvolvidos pela empresa são: o *framework* para desenvolvimento de aplicações portáveis para terminais POS chamado *Walk Framework* e a plataforma de desenvolvimento via *WEB* de aplicações para terminais POS chamada *CloudWalk*. Vale ressaltar que devido a função básica dos terminais *point of sale* ser realizar transações bancárias, a empresa tem grande preocupação com o requisito de segurança nos seus sistemas desenvolvidos.

2.2.1 Walk Framework

O *Walk Framework* implementa uma camada intermediária de comunicação entre o terminal POS e a aplicação desenvolvida na linguagem POSXML. Deste modo ele fornece portabilidade para as aplicações desenvolvidas, permitindo com que uma aplicação desenvolvida na linguagem POSXML possa ser executada em diversos modelos de terminais.

Este sistema de software tem como forte característica a necessidade de aprimoramento, pois com o surgimento de novos modelos de terminais POS o sistema deve se adaptar aos mesmos (LEHMAN *et al.*, 1997) para atender o mercado. Além disso, a constante implementação de novas funcionalidades no sistema

[3]https://www.cloudwalk.io

aumenta a deterioração e as chances de ocorrência de falhas no sistema (JERMAKOVICS; SCOTTO; SUCCI, 2007).

Na seção 4.2 é apresentada uma descrição detalhada da linguagem POSXML e juntamente é explicado o papel do *walk framework* na tradução dos comandos em POSXML para comandos na linguagem C específicos para cada modelo de terminal POS.

2.2.2 CloudWalk

A plataforma *CloudWalk* (CLOUDWALK, 2014) proporciona a seus usuários a possibilidade de desenvolver aplicações para terminais POS em um ambiente *Cloud Computing* através de um serviço web. Para auxiliar os usuários no desenvolvimento de suas aplicações diversas funcionalidade são oferecidas. Algumas das principais funcionalidades da plataforma *CloudWalk* são:

- Ambiente integrado de desenvolvimento ;
- Emulador de terminal POS;
- Atualização remota.

Entre os serviços oferecidos pela aplicação *Cloudwalk* está o ambiente integrado de desenvolvimento disponibilizado aos clientes desenvolvedores. Este serviço permite aos usuários desenvolverem suas aplicações para terminais POS em um navegador de internet sem a necessidade de nenhuma outra aplicação. Na figura 2.1 está apresentada a interface do ambiente de desenvolvimento para a linguagem POSXML.

Dentro do ambiente de desenvolvimento está presente a funcionalidade do emulador. O emulador é executado diretamente no ambiente de desenvolvimento da *CloudWalk*, ou seja, diretamente do browser web, dispensando a necessidade de qualquer tipo de instalação ou configuração, possibilitando assim testar o *workflow* da aplicação. Na figura 2.2 é ilustrada a interface do emulador de terminais POS.

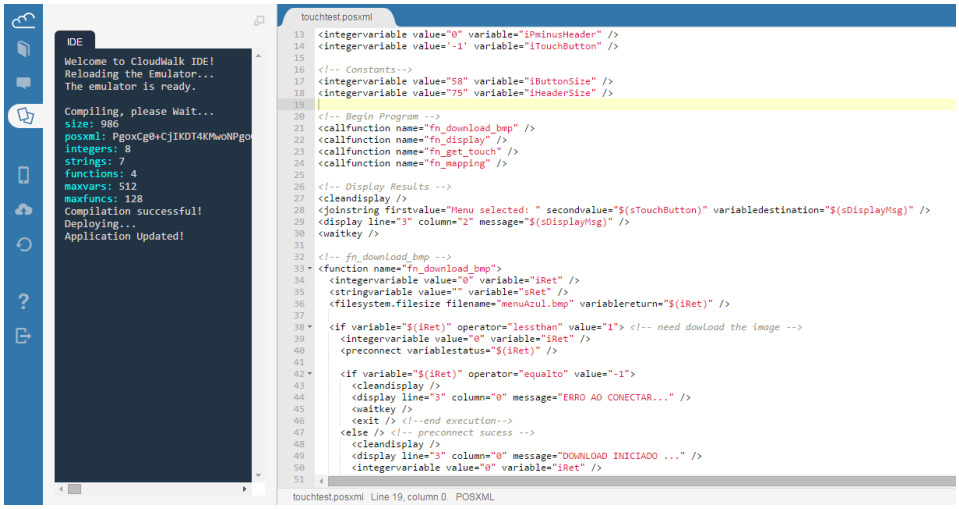


Figura 2.1: Ambiente de desenvolvimento *CloudWalk*

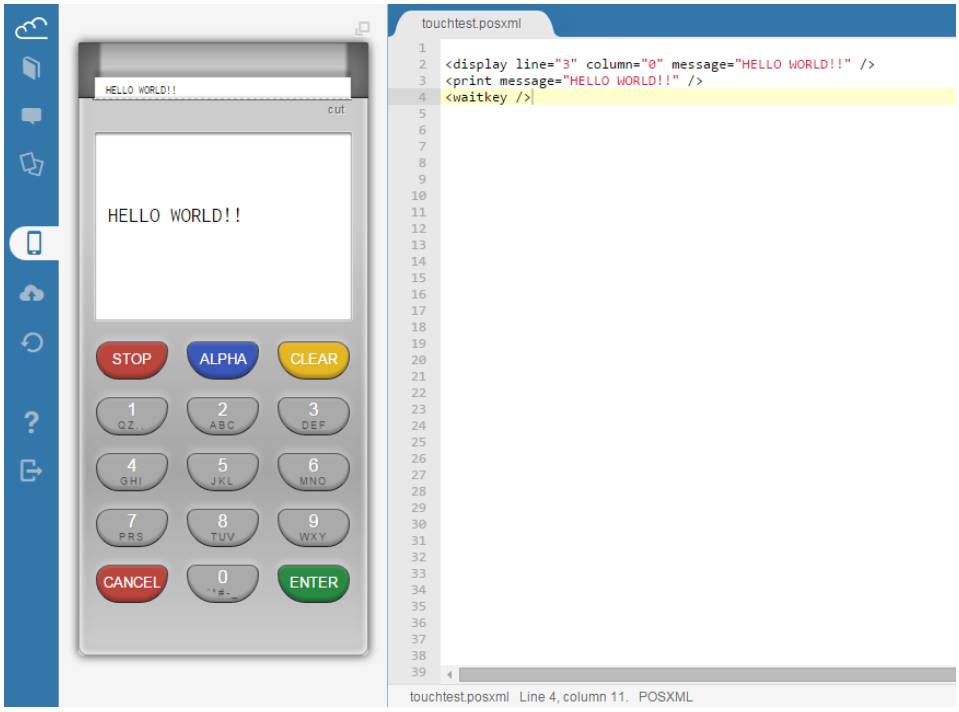


Figura 2.2: Emulador de terminal POS no ambiente *CloudWalk*

Outra importante funcionalidade é o serviço de atualização remota oferecido pela solução *CloudWalk*. Com este serviço uma aplicação desenvolvida pode ser disponibilizada a um grupo de terminais POS que possuem o sistema

CloudWalk instalado. Este serviço proporciona uma grande economia de mão de obra, já que não é necessário que um técnico vá a campo para realizar a instalação ou atualização de aplicações terminal por terminal. Ao desenvolver ou atualizar uma aplicação é possível que os terminais ao se conectarem ao servidor da *CloudWalk* realizem automaticamente a instalação e atualização de seu sistema e aplicações.

2.3 Clientes

Alguns dos clientes da Plano Bê são listados abaixo:

- RV Tecnologia[4];
- Stone[5];
- Sollus Cartões[6].

A RV Tecnologia é uma rede de transações eletrônicas e venda de serviços pré-pagos em nível nacional, sendo uma das líderes em vendas de crédito para celulares (RV, 2014). Esta oferece soluções via POS (Point of Sale), TEF (Transferência Eletrônica de Fundos) ou Internet, focadas na ampliação de disponibilidade de serviços pré-pagos e de aquisição, de acordo com o perfil e necessidade de cada um de seus parceiros: empresas de telefonia, grandes varejistas como redes de supermercados e também pequenos estabelecimentos comerciais.

Segundo Stone (2014) a empresa Stone é uma adquirente de meios de pagamento, autorizada pela Visa e Mastercard a credenciar lojistas, processar e autorizar transações de cartão de crédito com essas bandeiras e outras. Sendo licenciada para processar pagamentos eletrônicos em todo território nacional, no mundo físico e virtual. A Stone processa as transações realizadas com cartões de

[4]http://www.rvtecnologia.com.br
[5]http://stone.com.br
[6]http://www.solluscartoes.com.br

crédito e garante o pagamento ao lojista. Além disso, a mesma oferece diversos serviços para aumentar as vendas de seus clientes e melhorar o controle de seus recebimentos.

A Sollus Cartões tem como objetivo propor modelos de negócio inovadores, integrados, que otimizem os processos dos clientes e dos estabelecimentos credenciados, gerando valor agregado e ampliando a proposta de valor e a rentabilidade dos parceiros de negócios. Sendo reconhecida como uma das melhores administradoras de cartões de nicho do Brasil em 5 anos ela oferece diversas soluções como cartões híbridos, gestão de frotas e gestão de fornecedores (SOLLUS, 2014).

3 DESCRIÇÃO DAS ATIVIDADES DESENVOLVIDAS

Diversas foram as atividades realizadas pelo aluno no período de estágio, proporcionando o desenvolvimento de diversas habilidades.

Abaixo as atividades são listas em ordem cronológica de desenvolvimento, sendo que algumas ocorreram ao mesmo tempo.

- Treinamento para utilização e desenvolvimento para terminais POS;
- Acompanhamento do processo de desenvolvimento de software SCRUM;
- Integração de sistemas utilizando POSXML;
- Atendimento a clientes;
- Estudo de conteúdo auxiliar;
- Desenvolvimento do fluxo EMV.

Nas seções 3.1, 3.2, 3.3, 3.4, 3.5 e 3.6 estão apresentadas breves descrições destas atividades e suas cargas horárias.

3.1 Treinamento para utilização e desenvolvimento para terminais POS

Devido a área de atuação específica da empresa Plano Bê foi necessário a realização de um período de treinamento. O aluno acompanhou a documentação desenvolvida na empresa e recebeu o auxílio de outros funcionários para assim conhecer e desenvolver para os terminais POS utilizados pela empresa.

A empresa utiliza terminais POS dos fabricantes Verifone e Ingenico. Os terminais POS destas fabricantes se dividem em várias linhas. Os terminais da Verifone se dividem na linha Evo e Verix. Exemplos de terminais pertencentes a linha Evo são os vx520, vx675 e vx680. Já os terminais de modelo vx670 são

pertencentes a linha Verix. Os terminais da empresa Ingenico se dividem nas linhas Telium 1 e Telium 2.

É importante o conhecimento das características de cada linha e modelo das fabricantes, pois os sistemas operacionais e versões das aplicações utilizadas se diferenciam. Além disso, cada modelo possui hardware diferente o que faz com que os problemas encontrados sejam específicos de cada modelo. Um exemplo é que a linha Telium 1 da Ingenico possui um *buffer* de memória de menor tamanho que todos os outros modelos. Assim, ao carregar uma aplicação neste modelo de terminal POS deve-se ter uma atenção redobrada para o tamanho dos dados carregados para o *buffer* do aparelho.

Esse período de treinamento se estendeu por 1 semana, ou seja 30 horas. Ao fim do treinamento sobre os terminais POS utilizados pela empresa, foi sugerido que o aluno desenvolvesse uma aplicação básica em POSXML. O desenvolvimento desta aplicação propiciou o entendimento básico da linguagem e utilização do *walk framework*. Com o auxílio do *framework* a aplicação desenvolvida foi portável para todos os modelos de terminais homologados pela empresa. Foram utilizadas aproximadamente 5 horas para o desenvolvimento desta aplicação.

3.2 Acompanhamento do processo de desenvolvimento de software SCRUM

Durante o período de estágio, a empresa Plano Bê iniciou a implantação do processo de desenvolvimento de software chamado *SCRUM*. Mesmo não participando efetivamente da equipe de desenvolvimento, que adotou o modelo, o aluno acompanhou o processo de desenvolvimento para obter conhecimento sobre o modelo. O *SCRUM* foi aplicado na equipe de desenvolvimento de testes automatizados para a plataforma *walk framework*. A descrição detalhada do processo é apresentada na seção 4.1.

A atividade de acompanhamento iniciou-se na segunda semana de estágio e se estendeu por todo o período do mesmo. O acompanhamento foi realizado com a participação nas reuniões diárias e participação no dia de apresentação de resultados dos *Sprints*. No total foram realizadas aproximadamente 40 horas de atividades.

3.3 Integração de sistemas utilizando POSXML

Após o desenvolvimento do conhecimento sobre a linguagem POSXML foi realizada a atividade de integração de sistemas de software de clientes com o sistema da aplicação *Cloudwalk*. Esta integração possibilita que uma aplicação originalmente desenvolvida para outro tipo de sistema possa ser utilizada em um terminal POS.

A integração realizada foi a do sistema *Escolha Rápida*. Este sistema é um serviço web, que permite a lojistas receberem e gerenciarem pedidos a partir de uma aplicação web. A integração com terminais POS tem como objetivo permitir aos lojistas usufruírem da portabilidade oferecida pelos terminais POS.

Devido a necessidade de se revisar o documento de requisitos, o estagiário realizou várias conversas com o cliente para assim definir corretamente os requisitos desejados pelo cliente. Devido a esta dificuldade foram necessárias 2 semanas de desenvolvimento para que a integração fosse concluída, assim foram totalizadas aproximadamente 60 horas de trabalho. Na seção 4.3 estão apresentadas mais informações desta atividade.

3.4 Atendimento a clientes

Para propiciar uma maior experiência de interação do estagiário com clientes foi solicitado que o estagiário realizasse o suporte aos clientes. A empresa Plano

Bê utiliza a ferramenta *Zendesk* para atender seus clientes. Informações desta ferramenta podem ser encontradas em Zendesk (2014).

Durante a realização desta atividade, duas solicitações de clientes se destacaram. A primeira foi um erro na obtenção de resposta pelos terminais POS da linha Evo da fabricante Verifone. O segundo foi a solicitação de clientes para o desenvolvimento de um modelo de aplicação para se utilizar a funcionalidade *touchscreen* no terminal de modelo vx680 da fabricante Verifone.

A realização desta atividade se estendeu por 1 semana e meia. Totalizando aproximadamente 45 horas de atividade.

3.5 Estudo de conteúdo auxiliar

A empresa Plano Bê adicionou à sua aplicação a possibilidade de se realizar transações utilizando o padrão de segurança *Europay, MasterCard and Visa* (EMV). Foi solicitado que o estagiário realizasse um estudo aprofundado sobre o padrão.

Devido ao grande volume de informação necessária para o entendimento completo do padrão de segurança EMV a atividade de estudo referente ao padrão de segurança totalizou aproximadamente 30 horas de atividade.

3.6 Desenvolvimento do fluxo EMV

Após a leitura e entendimento do padrão de segurança EMV para transações com cartões com chip de circuito integrado, foi solicitado que o estagiário implementasse utilizando a linguagem POSXML o fluxo de uma transação que utilize o protocolo EMV. Esta atividade compõe o treinamento do estagiário para que, posteriormente o mesmo tenha capacidade de desenvolver em baixo nível o fluxo da transação EMV para o terminal PAX D200.

Esta atividade foi realizada nas últimas semanas do período de estágio, sendo que a mesma totalizou aproximadamente 90 horas de trabalho. A imple-

mentação realizada não pode ser apresentada neste trabalho, pois é de propriedade da empresa Plano Bê.

4 DESCRIÇÃO DOS PROCESSOS TÉCNICOS E OUTRAS PARTICULARIDADES

4.1 Processo de desenvolvimento de software Scrum

O processo de desenvolvimento de softwares *Scrum* se baseia na idéia de desenvolvimento em ciclos (KNIBERG, 2011) e entrega de um produto palpável a cada iteração. O modelo *Scrum* possui como pontos principais a criação dos *product bracklogs* e reuniões *sprint*. Nas seções 4.1.1 e 4.1.2 estes conceitos são apresentados.

4.1.1 Product Backlogs

O *product backlog* é basicamente uma lista de itens que normalmente são chamados de *estórias*.

As *estórias* possuem os seguintes componentes (KNIBERG, 2011):

- ID - Identificação única que facilita o controle quando é alterado os nomes das *estórias*;
- Nome - Um nome curto e descritivo para a estória;
- Importância - Uma pontuação de importância da estória para o *product owner*. Não há uma regra de qual deve ser a escala de importância das estórias sendo responsabilidade da equipe definir esta escala;
- Estimativa Inicial - Este campo é a estimativa inicial de quanto tempo é necessário para implementar esta estória. O mais importante não é definir o tempo exato de desenvolvimento e sim definir um valor se o tempo para desenvolvimento desta estória for comparado com o de outras. Normalmente é utilizado a escala de pontos baseado na relação "homens/dias de trabalho";

- Como demonstrar - Uma descrição de como será demonstrado o resultado da estória. Uma especificação simples como: "Faça isso, então faça aquilo e então isso deverá ocorrer".

Na seção 4.1.3 será apresentado exemplo prático de *product backlog* desenvolvido na empresa Plano Bê.

4.1.2 Sprint

Os *sprints* são fases curtas de desenvolvimento sendo que cada *sprint* normalmente abrange de uma a quatro semanas. Ao se concluir o período de um *sprint* espera-se que seja desenvolvida uma funcionalidade importante (RISING; JANOFF, 2000).

Antes do início de um período de *sprint* é realizada uma reunião de planejamento do *sprint*. É essencial que antes da reunião de planejamento se tenha o *product backlog* concluído (KNIBERG, 2011), pois na reunião são definidas quais *estórias* do *product backlog* serão desenvolvidas no período definido para o *sprint* e este período de tempo nunca deve ser alterado. Caso a equipe de desenvolvimento não entregue todas as funcionalidades originalmente propostas para o *sprint*, a mesma deve negociar a entrega de um conjunto reduzido de funcionalidades (RISING; JANOFF, 2000). Além da reunião de planejamento do *sprint* são realizadas reuniões diárias curtas para comunicação da equipe, onde cada integrante da equipe deve responder as perguntas (RISING; JANOFF, 2000):

- O que relacionado ao backlog você completou desde a última reunião?
- Quais obstáculos estão atrapalhando você a completar seu trabalho?
- O que relacionada ao backlog você planeja completar até a próxima reunião?

4.1.3 Aplicação do modelo SCRUM na empresa Plano Bê

Para o desenvolvimento de uma infraestrutura de testes automatizada para o sistema *Walk Framework* foi utilizado o modelo de processo de desenvolvimento SCRUM.

Inicialmente foi desenvolvido o *product backlog* apresentado na figura 4.1, com o objetivo de se definir todas as atividades, as quais são chamadas de *estórias*, para o desenvolvimento desta solução.

ID	Name	Importance	Estimate (story points)
1	Update rake command for EVO [setup_test]	140	2
2	[EMV] - Test emv.open with all possible returns	135	0.5
3	[EMV] - Test emv.settimeout with all possible returns	130	0.5
4	[EMV] - Test emv.loadtables with all possible returns	129	0.5
5	[EMV] - Test emv.cleanStructures	128	0.5
6	[EMV] - Test emv.adddata (init) with all possible returns	127	2
7	[EMV] - Test emv.inittransaction with all possible returns	126	0.5
8	[EMV] - Test emv.getinfo (init) with all possible returns	125	1
9	[EMV] - Test emv.adddata (process) with all possible returns	124	2
10	[EMV] - Test emv.processtransaction with all possible returns	123	1
11	[EMV] - Test emv.getinfo (process) with all possible returns	122	1
12	[Iso_8583] - Test iso8583.initfieldtable with all possible returns	120	0.5
13	[Iso_8583] - Test iso8583.initmessage with all possible returns	120	1.5
14	[Iso_8583] - Test iso8583.putfield with all possible returns	120	1
15	[Iso_8583] - Test iso8583.endmessage with all possible returns	120	1
16	[EMV] server	119	??
17	[Iso8583] server	119	??
18	[EMV] - Test emv.adddata(finish) with all possible returns	119	1
19	[Iso_8583] - Test iso8583.transactmessage with all possible returns	118	2
20	[Iso_8583] - Test iso8583.analysemessage with all possible returns	118	1
21	[Iso_8583] - Test iso8583.getfield with all possible returns	118	2
22	[EMV] - Test emv.finishtransaction with all possible returns	118	1
23	[EMV] - Test emv.getinfo (finish) with all possible returns	117	0.5
24	[EMV] - Test emv.removecard with all possible returns	116	0.5
25	[EMV] - Test emv.readcard with all possible returns	116	2
26	[EMV] - Test emv.inputtransaction with all possible returns	115	3
27	[Iso_8583] Test iso8583.transactmessagesubcampo with all possible returns	100	??
28	[Connectivity] - Test predial with all possible returns	100	1
29	[Connectivity] - Host service	95	??
30	[Connectivity] - Test preconnect with all possible returns	90	2
31	[Connectivity] - Test shutdownmodem with all possible returns	90	0.5
32	[Connectivity] - Test network.checkgprssignal with all possible returns	90	1
33	[Connectivity] - Test network.hostdisconnect with all possible returns	90	0.5
34	[Connectivity] - Test network.ping with all possible returns	90	0.5
35	[Connectivity] - Test network.send with all possible returns	90	1
36	Test Dashboard	80	10

Figura 4.1: *Product Backlog* do desenvolvimento de uma infraestrutura de teste automatizado para o produto *Walk Framework*

Foram definidas no total 36 *estórias*, cada uma com um grau de importância que segue uma escala de 0 (menor nível de importância) a 150 (maior nível

de importância). Foi utilizada como estimativa de tempo a idéia de *story points* (SP) sendo que 1 SP é equivalente a um dia completo de trabalho (6 horas) de uma pessoa.

Após a criação do *product backlog* foi realizada a primeira reunião de *sprint*. O período definido para este *sprint* foi de duas semanas (dez dias úteis). Para este *sprint* foram alocados dois desenvolvedores, portando para este *sprint* foram selecionadas as *estórias* de maior importância até se atingir uma estimativa total de tempo de 15 *story points*. As *estórias* selecionadas para este *sprint* estão apresentadas na figura 4.2

Plano Bê (Team Lucas), Sprint 1	
sprint goal	
	Demo WALKFramework testing infrastructure Phase 1
sprint backlog (estimates in parenthesis)	
	Update rake command for EVO [setup_test] (2)
	[EMV] - Test emv.open with all possible returns (0.5)
	[EMV] - Test emv.settimeout with all returns (0.5)
	[EMV] - Test emv.loadtables with all possible returns (0.5)
	[EMV] - Test emv.clenStructures (0.5)
	[EMV] - Test emv.adddata (init) with all possible returns (2)
	[EMV] - Test emv.inittransaction with all possible returns (0.5)
	[EMV] - Test emv.getinfo (init) with all possible returns (1)
	[EMV] - Test emv.adddata (process) with all possible returns (2)
	[EMV] - Test emv.processtransaction with all possible returns (1)
	[EMV] - Test emv.getinfo (process) with possible returns (1)
	[Iso_8583] - Test iso8583.initfieldtable with all possible returns(sucess and error) (0.5)
	[Iso_8583] - Test iso8583.initmessage with all possible returns(sucess and error) (1.5)
	[Iso_8583] - Test iso8583.putfield with all possible returns(sucess and error) (1)
	[Iso_8583] - Test iso8583.endmessage with all possible returns(sucess and error) (1
	estimated velocity (15.5) focus factor (78%)
Schedule	
	Sprint period: 05-09-2014 to 18-09-2014
	Daily scrum: 09:00 - 09:15 in the Plano Bê office
	Sprint demo: 19-09-2014 in the meeting (green room) @ Plano Bê office - ALL WELCOME!!
Team	
	Lucas Castejon
	Lucas Pires
	David Kelleher (Scrum Master)

Figura 4.2: Primeiro *Sprint* do *Product Backlog* do desenvolvimento de uma infraestrutura de teste automatizado para o produto *Walk Framework*

Ao fim do período da *sprint* é alocado um dia de apresentação das atividades desenvolvidas. Portanto os desenvolvedores devem apresentar para o *product owner* o resultado de cada atividade desenvolvida. Para que a equipe de desenvolvimento pudesse acompanhar o seu ritmo de desenvolvimento durante a *sprint* foi utilizado um gráfico da relação *story points* completados e dias passados. O gráfico foi desenhado em um mural para fácil visualização da equipe. O resultado após o período de duas semanas da *sprint* é apresentado na figura 4.3.

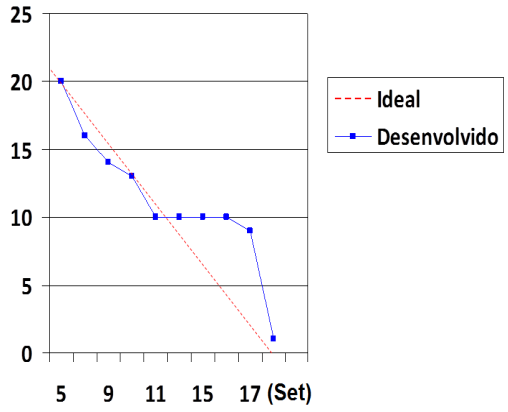


Figura 4.3: Gráfico do primeiro *sprint*, qual relaciona *story points* desenvolvidos e tempo.

Pode-se vizualizar que a equipe de desenvolvimento conseguiu realizar todos as *estórias* planejadas para o *sprint*, mas a atribuição de tempo de desenvolvimento para cada *estórias* não foi bem realizada. Algumas *estórias* foram realizadas em menos tempo que o previsto e outras levaram mais tempo que o planejado. Este era um resultado já previsto pela equipe, pois por se tratar do primeiro *product backlog* desenvolvido é esperado o erro no planejamento do tempo de desenvolvimento para cada *estória*.

Após este *sprint* o estagiário pode acompanhar outros dois períodos de *sprint* durante o estágio. Os períodos de duração dos *sprints* foram os mesmos que o do primeiro. Ao fim destes *sprints* todas as *estórias* para eles também foram

concluídas, mas houve uma melhora na atribuição dos valores de *story points* para cada *estórias*. O gráfico resultante do segundo *sprint* é apresentado na figura 4.4.

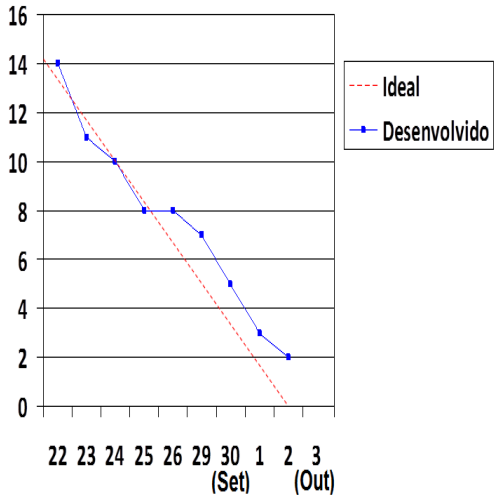


Figura 4.4: Gráfico do segundo *sprint*, qual relaciona *story points* desenvolvidos e tempo.

4.2 Linguagem POSXML

POSXML (acrônimo para *Point Of Sale Extended Markup Language*) é uma linguagem de programação usada para criar aplicações para um terminal POS (*Point Of Sale*). A seguir estão apresentados alguns pontos sobre a mesma, de acordo com Planobe (2009).

Descrito em seu nome a linguagem POSXML é uma expansão da linguagem XML, sendo organizada em estruturas de *TAGs*, apresentando níveis e subníveis de comandos e instruções para assim formar a lógica de uma aplicação. Vale ressaltar que o programador tem a limitação de que o tamanho da aplicação não ultrapasse 32 kbytes de código compilado.

Assim como outras linguagens de programação, POSXML é uma linguagem compilada, de modo que após compilada torna-se um conjunto de bytecodes que são interpretados através de um *framework* (máquina virtual) e assim resul-

tando na execução do programa no terminal POS. O código escrito em POSXML ao ser compilado torna-se um bytecode único identificado na biblioteca de comandos do *framework*.

Uma aplicação em POSXML compilada e instalada em um terminal POS fica localizada na 4a. camada de implementação encontrada no terminal sendo as camadas representadas na figura 4.5.

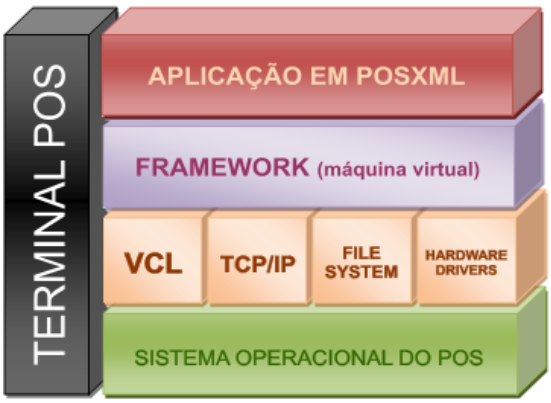


Figura 4.5: Camadas de implementações encontrados nos terminais POS

Ao se ter *frameworks* que interpretam bytecodes que seguem um modelo de compilação, se permite que um programa escrito em linguagem POSXML possa ser interpretado para diferentes terminais POS. Esta portabilidade permite que se tenha um esforço mínimo, ou nenhum para a migração de aplicações entre terminais POS. Na figura 4.6 ilustra-se a portabilidade de uma aplicação POSXML, que pode ser executada em dispositivos de diferentes fabricantes.

O *Walk Framework* apresentado na seção 2.2.1 é a maquina virtual desenvolvida pela empresa Plano Bê para interpretar os comandos de uma aplicação desenvolvida em POSXML para os comandos na linguagem C para assim ser convertido em código objeto e ser utilizado pelo sistema operacional do terminal POS.

Para possibilitar a solução *hardware agnostic* para terminais POS oferecida pelo *Walk Framework* foi necessário o desenvolvimento de um *framework* específico para cada tipo de dispositivo, como apresentado na figura 4.6.

Figura 4.6: Portabilidade de uma aplicação em POSXML para vários terminais

Os modelos homologados para utilização do *Walk Framework* até o momento são apresentados na tabela 4.1.

Tabela 4.1: Modelos de terminais POS homologados para utilizarem o *Walk Framework*

VeriFone Verix	**Verifone Evo**	**Ingenico Telium** 1	**Ingenico Telium** 2
Vx510	Vx520	EFT930G	ICT220
Vx610	Vx675	EFT930S	ICT250
Vx670	Vx680		IWL220
	Vx820		IWL250
	Vx805		IWL280

4.3 Integração do serviço Escolha Rápida

4.3.1 Descrição

A primeira atividade prática de desenvolvimento de software designada para o estagiário no período de estágio foi o desenvolvimento da integração de uma aplicação de um cliente com o sistema *CloudWalk*. O objetivo desta atividade é permitir

que uma aplicação desenvolvida pelo cliente se integre com terminais *Point of Sale*.

A aplicação do cliente consiste em uma plataforma web para usuários realizarem pedidos chamada *Escolha Rápida* (ESCOLHARAPIDA, 2014). Esta aplicação provê um sistema em que clientes realizam pedidos em diversas lojas com produtos de gênero alimentício. Além deste sistema é oferecida uma plataforma de gerenciamento para os lojistas, possibilitando ao lojista atender os pedidos e gerenciá-los entre os estados: pendente, atendido, em preparo, em entrega, concluído e cancelado.

Foi especificado que a integração possibilite o gerenciamento dos pedidos realizados utilizando apenas um terminal POS. Assim o funcionário do estabelecimento, que contrata o serviço, não precisa acompanhar os pedidos através de um computador. Portanto o proprietário do estabelecimento necessita apenas possuir um terminal POS em sua loja para utilizar o serviço de acompanhamento de pedidos.

Esta solução também permite uma maior integração da equipe de trabalho do estabelecimento, pois um pedido pode ter suas etapas acompanhadas e atualizadas por outros funcionários e não apenas um responsável pela atualização do estado dos pedidos. Um exemplo é a presença de um terminal POS na cozinha, o cozinheiro pode ser responsável em indicar se o pedido atendido está em preparo ou não. Além disso devido a portabilidade dos terminais POS o estado dos pedidos pode até mesmo ser acompanhado pelos entregadores, pois os terminais podem se conectar por rede GPRS e informar se há algum pedido para ser entregue.

Os requisitos foram:

- Receber pedidos pendentes automaticamente;
- Atualizar o status de um pedido;
- Consultar pedido por status;

- Listar empregados cadastrados no estabelecimento.

4.3.2 Desenvolvimento da Solução

Para se atender todos os requisitos solicitados pelo cliente foi necessário se definir como o terminal POS interage com o sistema *Escolha Rápida* e lógica de negócio do sistema. Nas seções 4.3.2.1 e 4.3.2.2 são apresentados descrição destes requisitos. Na seção 4.3.2.3 é apresentado como foi implementado as funcionalidades exigidas.

4.3.2.1 Interação do terminal POS com sistema do cliente

A interação entre o terminal POS e o sistema desenvolvido pelo cliente é realizada através de requisições *HTTP* (Hypertext Transfer Protocol). O terminal para obter informações realiza uma requisição *GET* e para passar informações realiza uma requisição *POST* ao sistema do cliente.

Utilizando a url: *http://www.domain.com/cloudwalk_api/order_list?* é possível realizar uma requisição *GET* com o objetivo de listar os pedidos, sendo necessário passar obrigatoriamente os atributos *token* e *status* sendo que:

- Token: identificador do lojista, que é definido na configuração na aplicação do lojista;
- Status: estado dos pedidos, que serão filtrados.

Na tabela 4.2 é apresentado o valor do atributo *Status* em relação o estado dos pedidos para as requisições *GET*.

Utilizando a url: *http://www.domain.com/cloudwalk_api/order_list?* é possível realizar uma requisição *POST* com o objetivo de atualizar o estado de um pedido, sendo necessário passar obrigatoriamente os atributos *token*, *order* e *status*. Ao realizar a atualização de um pedido para o estado "em entrega", além

Tabela 4.2: Relação do valor do atributo com o estado do pedido para requisições *GET*

Valor	Estado do pedido
1	Pendente
2	Atendido
3	Em preparo
4	Em entrega
5	Finalizado
6	Cancelado
7	Em espera

dos atributos já apresentados é necessário informar o atributo *employee*. Sendo o significado de cada atributo:

- Token: identificador do lojista, que é definido na configuração na aplicação do lojista;
- Order: número de identificação do pedido que será atualizado;
- Status: estado para o qual o pedido será atualizado;
- Employee: número de identificação do entregador, que realizará a entrega.

Na tabela 4.3 é apresentado o valor do atributo *Status* em relação ao estado dos pedidos para as requisições *POST*.

Tabela 4.3: Relação do valor do atributo com o estado do pedido para requisições *POST*

Valor	Estado do pedido
attended	Atendido
cook	Em Preparo
delivery	Em entrega
close	Concluído
cancel	Cancelado
wait	Em espera

4.3.2.2 Lógica de negócio

No desenvolvimento da integração uma das necessidade foi seguir a lógica de negócio do lojista. O lojista definiu que um pedido ao ser realizado sempre se inicia com o estado "pendente"e que este pedido deve automaticamente ser apresentado no terminal POS. O funcionário que está operando o terminal POS deve então obrigatoriamente aceitar ou rejeitar o pedido. O funcionário não consegue consultar outro pedido na fila de pedidos até que tenha tomado a decisão em relação ao primeiro pedido da fila. Após um pedido ser aceito o estado do mesmo é alterado automaticamente para o estado "atendido"sendo que a partir deste ponto o estado do pedido deve ser atualizado por um funcionário. Não há restrições para qual estado o pedido pode ser alterado. Assim é de responsabilidade dos funcionários o controle dos estados dos pedidos.

Vale lembrar que os dados referentes aos pedidos são compartilhados entre o terminal POS e o sistema de gerenciamento do cliente, portanto ao alterar as informações referentes aos pedidos as informações são atualizadas para ambos os sistemas. Outro ponto é que algumas funcionalidades só podem ser gerenciadas através da aplicação do cliente, por exemplo, o cadastro de entregadores e alteração do valor do token de identificação do estabelecimento.

4.3.2.3 Implementação da solução

A interação entre o terminal POS e o sistema do cliente é realizada através da troca de mensagens. Onde o terminal envia uma requisição *http* para o sistema do cliente e este responde uma mensagem no formato *JavaScript Object Notation* (JSON). Sendo que a função *BuildOrderMsg* contida no apêndice realiza a construção da mensagem para a requisição *GET* para obter os pedidos com estado "pendente". O construção das outras mensagens de requisição *http* são realizadas de forma semelhante mudando apenas alguns campos, como apresentado na seção 4.3.2.1.

A linguagem POSXML possui a instrução $< string.getvaluebykey/ >$. Esta instrução devolve o valor de atributos contidos em uma string no formato: *attribute_key* = *value*. O código fonte do *parser* para as mensagens *JSON* enviadas pela aplicação do cliente é apresentada no apêndice.

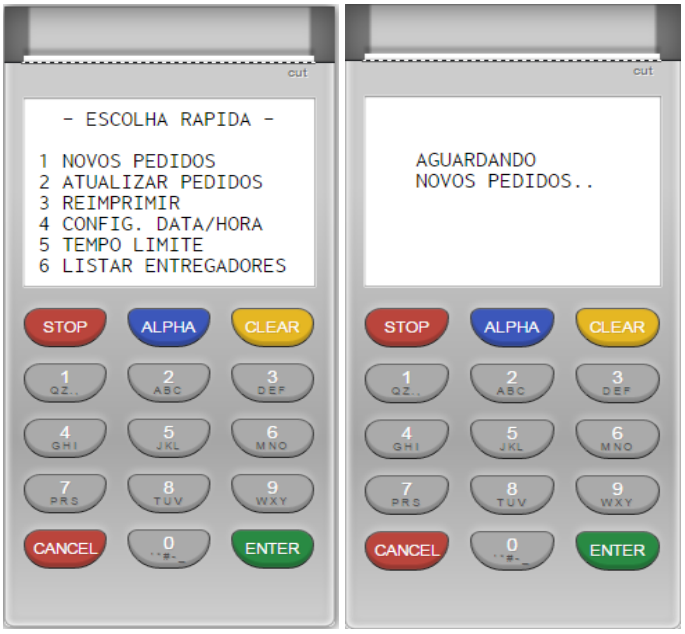
Tendo desenvolvido a montagem e *parse* das mensagens trocadas entre os terminais POS e o sistema do cliente, foi necessário desenvolver a lógica de negócio solicitada pelo lojista em POSXML. O início do fluxo de execução da aplicação se dá com a seleção da funcionalidade pelo cliente. As funcionalidades apresentadas ao usuário são:

- 1 - Novos pedidos: Terminal realiza requisição no servidor do cliente para obter os novos pedidos pendentes;
- 2 - Atualizar pedidos: Opção para atualização do estado de um pedido;
- 3 - Reimprimir: Reimpressão do último pedido recebido;
- 4 - Configurar Data/Hora: Configuração da data e hora do terminal. Esta função é relacionada a opção *Tempo Limite*;
- 5 - Tempo Limite: Nesta opção é configurado o horário de funcionamento do estabelecimento, de modo que fora deste horário de funcionamento o terminal não realiza consultas ao servidor do cliente;
- 6 - Listar Entregadores: Imprime lista com nome e número de identificação dos entregadores cadastrados no estabelecimento.

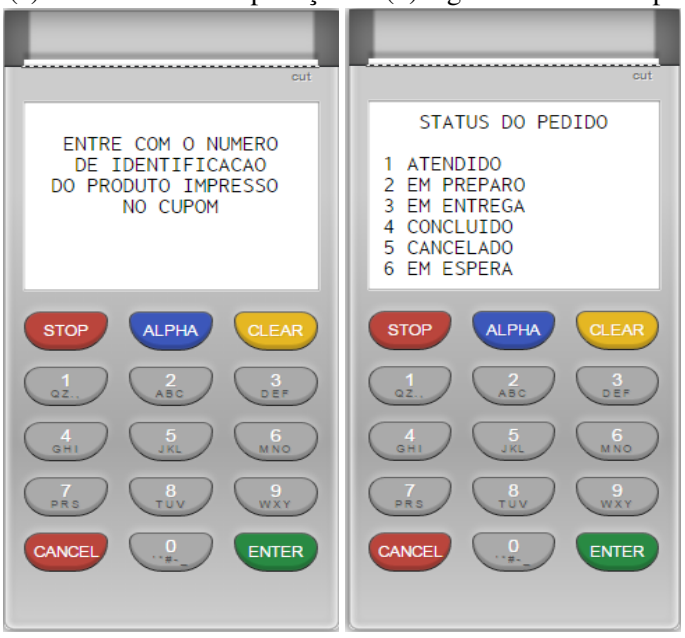
Assim é possível que o usuário ao utilizar o teclado numérico presente no terminal POS selecione facilmente a funcionalidade desejada.

Nas figuras 4.7 (a), (b), (c) e (d) são apresentados alguns resultados de interação do usuário com a solução. Toda a implementação da solução desenvolvida não pode ser apresentada, pois a mesma é de propriedade particular da empresa

concedente do estágio e para preservar informações de seus clientes a mesma não permitiu apresentar toda a solução.



(a) Menu inicial da aplicação (b) Aguardando novos pedidos



(c) Atualizar pedido etapa 1 (d) Atualizar pedido etapa 2

Figura 4.7: Telas de interação do usuário com a interface da integração *Escolha Rápida*.

4.4 Padrão de segurança EMV

O padrão EMV de segurança descreve como são as regras para que uma transação utilizando um cartão que possui um chip de circuito integrado seja realizada. O padrão EMV se difere pelo fato que a aplicação de pagamento está no chip embarcado no cartão de pagamento feito de plástico. Este chip possui três características chaves: ele pode armazenar informações, realizar processamento, realizar criptografia e por se tratar de um elemento seguro pode armazenar informações secretas de segurança (EMVCO, 2011d).

Existem diferenças fundamentais entre a leitura de uma tarja magnética e a transação por um chip que utiliza EMV (EMVCO, 2011d). Para uma tarja magnética o cartão é simplesmente um armazenamento de dados lido pelo terminal e depois desta leitura o cartão não é mais utilizado. O terminal realiza todo o processamento e aplica as regras de pagamento. Durante uma transação EMV, o chip é capaz de realizar o processamento das informações e define várias regras para o pagamento. O terminal ajuda a aplicar o conjunto de regras definidas pelo emissor contidas no chip. É importante destacar que a decisão final de que uma transação será aprovada ou não é tomada pelo chip.

Para realizar uma transação diversas etapas de processamento e tomada de decisões são realizadas pelos terminais POS e cartões de circuito integrado (EMVCO, 2011e). As etapas são:

- Tratamento do *answer to reset* (ATR);
- Seleção da aplicação;
- Inicialização da aplicação;
- Recuperação dos dados;
- Autenticação;
- Processamento de restrições;
- Verificação do portador;
- Gerenciamento de risco do terminal;
- Análise de ação do terminal;
- Análise de ação do cartão;
- Processamento online;
- Autenticação do emissor;

- Processamento de *scripts*;
- Finalização.

Na figura 4.8 retirada de EMVCo (2011b) é apresentado o fluxograma.

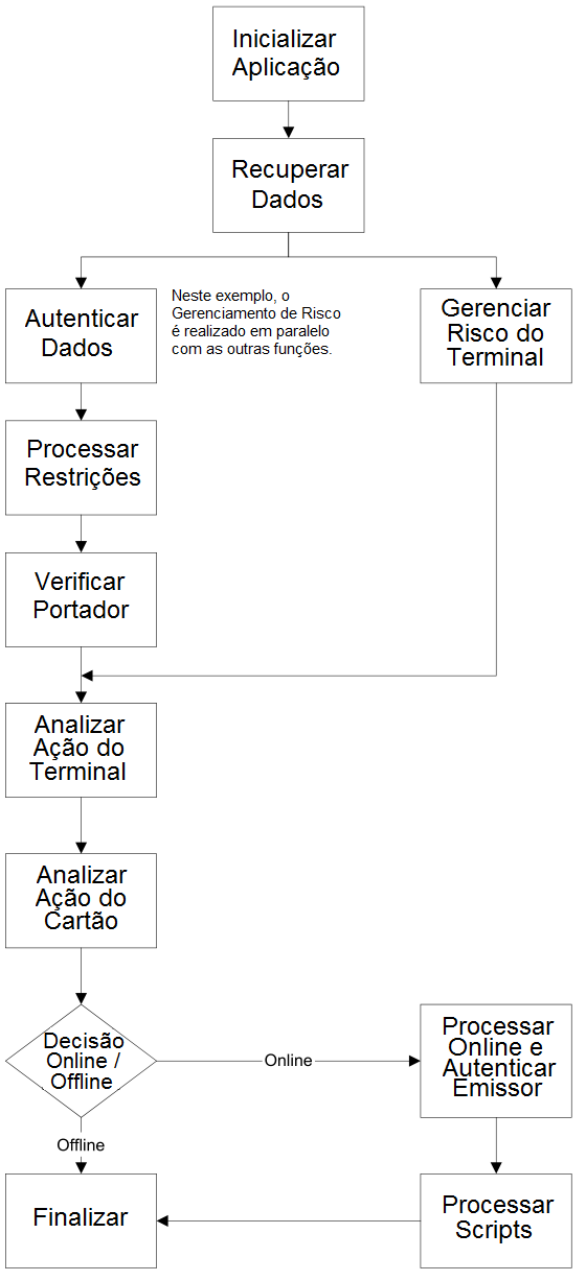


Figura 4.8: Exemplo do fluxo de execução de uma transação EMV

4.4.1 Tratamento do ATR

Nesta primeira etapa de processamento de uma transação é realizada uma análise do campo ATR, que é retornado pelo cartão de circuito integrado após o mesmo ser reiniciado pelo terminal (EMVCO, 2011a) . O ATR é uma string de bytes, que possui informações básicas que definem como é a comunicação entre o terminal e o cartão de circuito integrado. Além disso permite o terminal tomar decisão sobre o tipo de sessão que será iniciada.

4.4.2 Seleção da aplicação

Os cartões de circuito integrado podem suportar diferentes tipos de aplicações. Por exemplo, existem cartões que suportam apenas aplicações de crédito da Master Card, enquanto outros podem possuir aplicações de crédito e débito da Visa. Portanto, após início do fluxo da transação EMV o terminal deve identificar as aplicações suportadas pelo cartão e selecionar a aplicação. Existem duas maneiras de selecionar uma aplicação EMV (EMVCO, 2011b): seleção por *Payment System Environment* (PSE) ou seleção usando uma lista de *Application Identifiers* (AIDs).

O PSE é um arquivo especial, definido pela ISO7816, que contém uma lista de entrada de Directory Definition File (DDF) e Application Definition File (ADF). Portando ele armazena diversas informações sobre as aplicações suportadas pelo cartão como: nível de prioridade da aplicação, opções do titular, entre outras. Este arquivo possui a vantagem de que o terminal pode acessar toda a lista de aplicações diretamente e selecionar a aplicação desejada entre elas.

Diferente na seleção da aplicação utilizando um arquivo PSE na seleção utilizando AIDs o terminal deve montar uma lista de aplicações candidatas. O terminal tenta selecionar todos os AIDs de aplicações que ele suporta uma a uma e o cartão responde o File Control Information (FCI) da aplicação se ele suportá-la. Com a lista de FCIs o terminal pode decidir qual aplicação selecionar. O FCI

contém o AID da aplicação, o indicador de prioridade o *Processing Data Object List* e os dados proprietários do emissor desta aplicação.

A aplicação pode ser selecionada pelo terminal, que no caso sempre seleciona a aplicação de maior prioridade ou a aplicação pode ser selecionada de forma iterativa pelo usuário.

4.4.3 Inicialização da aplicação

Após selecionar a aplicação o terminal necessita de informações extras para a execução da mesma. Assim o terminal solicita informações sobre as opções de processamento para o cartão EMV. As informações retornadas são o *Application Interchange Profile* (AIP) e o *Application File Locator* (AFL) (EMVCO, 2011b). O AIP contém diversas informações do cartão sobre o tipo de autenticação de dados suportada, a verificação do portador, o gerenciamento de risco do terminal e autenticação do orgão emissor.

4.4.4 Recuperação dos dados

Para codificar os dados trocados entre os terminais e o cartões é utilizado o formato *Tag Length Value* (TLV) (EMVCO, 2011c). Este formato define que cada dado deve possuir os campos etiqueta (T), tamanho (L) e valor (V). A etiqueta informa como o conteúdo do dados (valor) deve ser tratado, o campo tamanho e valor armazenam as informações sobre o tamanho do dados e conteúdo do dado respectivamente. Devido a heterogeneidade dos dados trocados entre os terminais e os cartões a codificação TLV é flexível em relação ao tamanho de seus campos. Portando, os campos etiqueta e tamanho possuem tamanhos variáveis.

O primeiro *byte* que compõe o campo etiqueta possui informações sobre o escopo da etiqueta, formato e valor. Os *Bytes* subsequentes da etiqueta possuem em seu primeiro *bit* informação se este é o ultimo *byte* que compõe a etiqueta. Os demais *bits* são em relação ao valor da mesma.

Os *bytes* que compõem o campo tamanho são identificados em seu primeiro *bit* se o mesmo contém informação sobre o tamanho do dado em si ou se informa o número de *bytes* subsequentes que conterão o campo tamanho.

O campo valor possui seu formato definido de acordo com a etiqueta definida para o dados. Sendo que o campo valor pode conter inteiros, caracteres, registros binários, imagens, entre outros. Vale lembrar que os valores das etiquetas são padronizados, portanto uma etiqueta de um valor sempre define o mesmo formato para o dado.

Então para cada AFL recebido pelo terminal ele armazena uma lista de dados TLVs para que assim de acordo com a aplicação executada o terminal possui os dados necessários para a execução da mesma.

4.4.5 Autenticação

A autenticação dos dados em uma transação EMV é realizada utilizando-se de chaves públicas e chaves privadas, onde as chaves públicas são chaves disponibilizadas ao público instaladas nos terminais, que são utilizadas para autenticar o emissor. As chaves privadas são chaves de propriedade única dos órgãos emissores que são utilizadas para assinar informações contidas no cartão.

Durante uma transação EMV a autenticação pode ser realizada via autenticação estática de dados (SDA), autenticação dinâmica de dados ou autenticação dinâmica de dados combinada (CDA) (EMVCO, 2011e).

4.4.5.1 Autenticação estática de dados

Ao realizar a SDA é considerado um cenário onde o órgão emissor do cartão utilizando-se de sua chave privada realiza a criptografia de parte dos dados contidos no cartão. Estes dados criptografados são utilizados para gerar a assinatura SDA, que é instalada no cartão. Portanto, esta assinatura SDA é única por cartão. Além da assinatura SDA o órgão emissor instala no cartão o certificado da

sua chave pública. A instalação das chaves públicas dos esquemas nos terminais é realizada pelos órgãos adquirentes. As chaves públicas são únicas para cada órgão emissor dos cartões.

Durante a transação o terminal lê os dados do cartão juntamente com a assinatura SDA e o certificado do emissor. O terminal autentica o emissor com a chave do esquema. Umas das informações contidas no certificado do emissor é a chave pública do emissor, ao obter a chave pública do emissor e a assinatura SDA o terminal consegue autenticar a identidade do cartão. O resultado da autenticação é armazenado no *terminal verification results* (TVR) que é uma *string* de bytes que indica se a autenticação foi bem sucedida.

4.4.5.2 Autenticação dinâmica de dados

Na DDA se tem um cenário em que no cartão estão armazenadas a chave privada do cartão, a chave pública do cartão certificada pelo emissor e a chave pública do emissor certificada pelo esquema. As certificações de chaves públicas são realizadas com a utilização das chaves privadas dos órgãos responsáveis. O adquirente é responsável pela instalação da chave pública do esquema nos terminais.

Durante a transação ao realizar a DDA o terminal recupera o certificado do cartão e o certificado do emissor. Após a obtenção destes dados o terminal autentica o emissor utilizando a chave pública do esquema e obtém a chave pública do emissor a partir do certificado do emissor. Utilizando a chave pública do emissor o terminal consegue obter a chave pública do cartão a partir do certificado do cartão. Com estas informações o terminal controla os dados *Dynamic Data Authentication Data Object List* (DDOL) e envia um desafio para o cartão. O cartão ao receber o DDOL calcula uma assinatura e retorna a mesma para o terminal. Ao fim o terminal utilizando a chave pública do cartão verifica esta assinatura gerada e assim o mesmo consegue autenticar o cartão.

4.4.5.3 Autenticação dinâmica de dados combinada

A CDA utiliza as mesmas chaves e mecanismos da assinatura DDA e também utiliza informações do criptograma da transação para gerar a assinatura DDA. Assim, utilizado o *application cryptogram* (AC) ou o *transaction Certificate* (TC) e um *unpredictable number* gerado pelo terminal, é gerada a assinatura DDA para a autenticação do cartão.

4.4.6 Processamento de restrições

O processamento de restrições verifica o grau de compatibilidade do terminal com as aplicações presentes no cartão. Sendo verificado o número de versão da aplicação e o controle do uso da aplicação. O controle de uso apresenta as restrições geográficas e tipo de transação suportado em uma aplicação. Além disso, o controle de uso de uma aplicação também fornece informações sobre a data de vigência e expiração de uma aplicação.

O teste de restrições é facultativo para o cartão mas é obrigatória a realização do mesmo para o terminal (EMVCO, 2011b). O resultado do processamento de restrições é armazenado no segundo *byte* do TVR, sendo armazenadas informações referentes a compatibilidade das versões da aplicação e terminal, validade da aplicação, se a operação foi autorizada pelo cartão, entre outras.

4.4.7 Verificação do Portador

A verificação do portador tem como objetivo garantir que a pessoa em poder do cartão é o verdadeiro titular. Para isso o cartão armazena uma lista de regras de verificação do titular. Esta lista contém dois campos de quatro *bytes* para indicar a quantidade de condições apresentadas e uma lista com os códigos dos *cardholder verification methods* (CVMs) solicitados pelo cartão.

Os CVMs podem especificar diferentes requisitos para a verificação do portador como requisitar o PIN acima ou abaixo de um valor *X*, tipo do PIN requi-

sitado, entre outras possibilidades. Os resultados da verificação do portador são armazenados no terceiro *byte* TVR, onde são armazenadas informações referentes ao sucesso da verificação do portador, se o limite de tentativas de obter o PIN foi excedida, PIN online capturado, entre outras.

4.4.8 Gerenciamento de risco do terminal

O gerenciamento de risco tem como objetivo principal ser um mecanismo de proteção para o Adquirente e o Emissor contra possíveis fraudes. O gerenciamento de risco segue os parâmetros de gerenciamento de risco do terminal (TRM), que são definidos pelo Adquirente e pelo Emissor. Também auxiliam o gerenciamento de risco os parâmetros de gerenciamento de risco do cartão (CRM), que são definidos pelo órgão Emissor. Assim há uma distribuição do risco de modo que a decisão é tomada tanto pelo terminal quanto pelo cartão.

São três os principais parâmetros considerados no gerenciamento de risco: o *floor limit checking*, o *random transaction selection* e o *velocity checking* (EMVCO, 2011e).

O *floor limit checking* tem como objetivo impedir que um cartão realize operações de valores superiores ao limite definido pelo Adquirente de maneira off-line ou realize muitas vendas de maneira off-line. Para isso o cartão armazena um acumulador de valores das transações realizadas off-line. Assim, quando o limite é ultrapassado a operação é realizada obrigatoriamente de maneira on-line.

Para inserir um elemento de aleatoriedade no gerenciamento de risco é utilizado o parâmetro *random transaction selection*. O terminal armazena uma etiqueta com a porcentagem da seleção aleatória, este valor é definido pelo Adquirente. Assim o terminal força aleatoriamente algumas transações que seriam aprovadas off-line sejam completadas on-line.

Para evitar que um cartão realize muitas transações off-line consecutivas é utilizado o parâmetro *velocity checking*. Utilizando este parâmetro como referên-

cia, é verificada a última transação registrada no *application transaction counter* (ATC) e compara o número da transação com os valores *lower consecutive offline limit* (LCOL) e *upper consecutive offline limit* (UCOL), que são definidos pelo orgão Emissor. Assim se a transação estiver fora dos limites de transações *offline* consecutivas permitidas é tomada a decisão para obrigar a transação a ser realizada de maneira *online*.

As informações resultantes do gerencimento de risco são adicionadas aos *bytes* TVR, onde são armazenadas informações de se o LCOL ou o UCOL foi excedido, se a transação excedeu o teto limite, se a transação foi selecionada aleatoriamente para processamento *online*, entre outras.

4.4.9 Análise de ação do terminal

Durante uma transação o terminal pode tomar a decisão de recusar a transação, tentar realizar a transação de maneira online ou tentar realizar a transação de maneira offline. De acordo com a decisão tomada o terminal gera sinal de comando que ele envia para o cartão. Além da sua decisão neste comando estão contidos os dados da transação.

O padrão EMV define três tipos (EMVCO, 2011e) de pedido de autorização realizado por um terminal:

- *Transaction Cryptogram* (TC): Este sinal é enviado quando o terminal toma a decisão de aprovar a transação;
- *Application Request Cryptogram* (ARQC): Enviado quando o terminal toma a decisão de requisitar a aprovação online da transação;
- *Application Authorization Cryptogram* (AAC): Sinal gerado quando o terminal recusa a transação.

Para decidir qual o sinal de comando será enviado pelo terminal são definidos os códigos de ação. Estes códigos de ação são definidos pelo órgão emissor

e pelo terminal sendo eles o *issuer action code* (IAC) e o *terminal action code* (TAC). Os dois códigos de ação podem ter condição de recusar a transação, tentar aprovar online ou recusar a transação após uma falha de comunicação.

O processamento dos IACs e TACs é realizado com a comparação com o TVR. É verificado se algum *bit* setado no TVR está presente no IAC ou no TAC. Após esta análise é tomada a decisão de emitir um sinal TC, ARQC ou AAC.

4.4.10 Análise de ação do cartão

Ao receber a sinal de comando do terminal o cartão realiza seu próprio gerenciamento de risco e toma sua decisão. Vale lembrar que a última decisão é tomada pelo cartão. Então uma transação aprovada pelo terminal não quer dizer necessariamente que será aprovada pelo cartão. Após realizar seu próprio gerenciamento de risco (CRM) o cartão retorna o criptograma de resposta adequado (AAC, ARQC ou TC).

4.4.11 Processamento online e autenticação do emissor

Ao realizar o processamento de uma transação online se deseja garantir que o órgão emissor participe da autorização de uma transação. O órgão emissor EMV ao receber do terminal, o sinal ARQC e dados da transação, verifica os mesmos e toma a decisão de autorizar ou negar a transação. Após tomar a decisão o emissor gera os dados de autenticação e os retorna para o terminal. Estes dados são posteriormente usados pelo terminal e cartão para decidirem aprovar ou negar uma transação.

Os dados da resposta do emissor chamados de *issuer authorization data* (IAD) são enviados ao cartão através do comando *External Authenticate* ou utilizando um segundo comando *Generate AC*. Portando, a autenticação do emissor pode ser realizada em duas sequencias de ações. Na primeira, o terminal envia os IADs para o cartão e espera a resposta do cartão em relação aos dados. Se for

respondido o sucesso na análise os IADs o terminal gera um segundo AC, obrigatoriamente um TC ou AAC já que a autenticação online foi realizada, e espera a resposta com sinal TC ou AAC do cartão para que assim o terminal aceite ou recuse uma transação. No segundo caso o terminal em posse dos IADs utiliza o comando *external authenticate* e envia diretamente o segundo AC com os dados de autenticação do emissor. Assim é utilizada uma rodada de troca de mensagens entre o cartão e o terminal a menos.

4.4.12 Processamento de *scripts*

O processamento de *scripts* permite com que o emissor envie comandos de forma protegida para o cartão (EMVCO, 2011c). O emissor envia um lote de comandos (*scripts*) encapsulados em estruturas TLV. Este processo é transparente para o terminal e o mesmo só desempacota e envia estes comandos para o cartão. É definido pelo padrão EMV que o terminal deve negar a transação se o cartão não retornar sucesso para todos os comandos.

4.4.13 Finalização

Como apresentado anteriormente quem toma a decisão final de aceitação ou recusa de uma transação é o cartão. Portanto o processamento de uma transação só é finalizado quando o cartão envia um sinal de TC para aceitar a transação ou um sinal AAC para a recusa da transação. A finalização de uma transação pode ser realizada depois do primeiro ou do segundo comando *Generate AC*.

5 CONCLUSÃO

Este relatório de estágio refere-se ao período de estágio realizado na empresa Plano Bê Desenvolvimento de Sistemas, que possui como área de atuação o desenvolvimento de aplicações para terminais *Point of Sale* (POS). A realização do estágio supervisionado proporcionou o aprendizado, num ambiente prático, sobre a organização de uma empresa de desenvolvimento de sistemas de software.

Desenvolver sistemas para terminais POS requisitou o conhecimento de áreas específicas no desenvolvimento de software, esta necessidade levou o estagiário a estudar áreas complementares a sua formação. Sendo exemplos o estudo sobre o padrão EMV de segurança, estudo do modelo de processo de desenvolvimento *SCRUM* e aprendizado da linguagem *POSXML*. Além disso o desenvolvimento para a arquitetura limitada dos terminais POS se mostrou um desafio, forçando o estagiário aplicar os conhecimentos desenvolvidos em sua graduação.

A atuação na empresa propiciou a vivência de cenários práticos no desenvolvimento de software como: a interação com clientes, trabalho em equipe e necessidade de implementar as aplicações de software com limitações rígidas de prazos. Portanto, a realização do estágio supervisionado se mostrou importante como complemento a formação acadêmica de Ciência da Computação.

REFERÊNCIAS BIBLIOGRÁFICAS

CLOUDWALK. *CloudWalk Docs*. 2014. Disponível em: <https://docs.cloudwalk.io/pt-BR/introduction>. Acesso em: 30-10-2014.

EMVCO. *Application Independent ICC to Terminal Interface Requirements*. 2011. 175 p. Disponível em: <http://www.emvco.com/specifications.aspx?id=223>. Acesso em: 22-10-2014.

EMVCO. *Application Specification*. 2011. 214 p. Disponível em: <http://www.emvco.com/specifications.aspx?id=223>. Acesso em: 22-10-2014.

EMVCO. *Cardholder, Attendant, and Acquirer Interface Requirements*. 2011. 140 p. Disponível em: <http://www.emvco.com/specifications.aspx?id=223>. Acesso em: 22-10-2014.

EMVCO. *A Guide to EMV*. 2011. <http://www.emvco.com/best_practices.aspx?id=217>. Acesso em: 25-05-2014.

EMVCO. *Security and Key Management*. 2011. 162 p. Disponível em: <http://www.emvco.com/specifications.aspx?id=223>. Acesso em: 22-10-2014.

ESCOLHARAPIDA. *Escolha Rápida*. 2014. Disponível em: <http://www.escolharapida.com.br/>. Acesso em: 20-10-2014.

GANSSLE, J. Embedded systems programming. Embedded Y2K, p. 97–99, Fev 1999.

JERMAKOVICS, A.; SCOTTO, M.; SUCCI, G. Visual identification of software evolution patterns. In: ACM. *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*. [S.l.], 2007. p. 27–30.

KNIBERG, H. Scrum e xp direto das trincheiras. *InfoQ Disponível em:< http://www. infoq. com/br/minibooks/Scrum-xp-from-the-trenches*, 2011. Acesso em: 15-09-2014.

LEHMAN, M. M.; RAMIL, J. F.; WERNICK, P. D.; PERRY, D. E.; TURSKI, W. M. Metrics and laws of software evolution-the nineties view. In: IEEE. *Software Metrics Symposium, 1997. Proceedings., Fourth International*. [S.l.], 1997. p. 20–32.

PLANOBE. *POSXML*. 2009. Disponível em: <http://pt.wikipedia.org/wiki/POSXML>. Acesso em: 25-05-2014.

RISING, L.; JANOFF, N. S. The scrum software development process for small teams. *IEEE software*, IEEE Computer Society, v. 17, n. 4, p. 26–32, 2000.

RV. *RV Tecnologia - Rede de transações eletrônicas | Venda de serviços pré-pagos*. 2014. Disponível em: <http://www.rvtecnologia.com.br/a-empresa/nossos-valores/>. Acesso em: 20-10-2014.

SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. *Design Test of Computers, IEEE*, v. 18, n. 6, p. 23–33, Nov 2001.

SOLLUS. *Sobre nós - Sollus Administradora de Cartões*. 2014. Disponível em: <http://www.solluscartoes.com.br/home/historico/>. Acesso em: 20-10-2014.

STONE. *Stone - Soluções em pagamentos*. 2014. Disponível em: <http://stone.com.br/sobre.html>. Acesso em: 20-10-2014.

WHATIS. *What is point-of-sale terminal (POS terminal)?* 2014. Disponível em: <http://whatis.techtarget.com/definition/point-of-sale-terminal-POS-terminal>. Acesso em: 22-10-2014.

ZENDESK. *Customer Service Software | Support Ticket System*. 2014. Disponível em: <http://www.zendesk.com/>. Acesso em: 20-10-2014.

6 APÊNDICE

A função *BuildOrderMsg* realiza a construção das mensagens para a requisição GET com o objetivo de se obter os pedidos com o estado "pendente". Apresentado na tabela 4.2 o valor 1 é o referente ao estado pendente. A mensagem é montada com a concatenação das substrings que possuem as informações necessárias, sendo elas: url da requisição, token do usuário, estado do pedido.

A função *ParserBufRecv* é responsável por alterar a *string* recebida e encontrar os atributos relacionados aos pedidos. As substring relacionadas aos atributos são:

- order_list: Identificação para inicio dos dados relacionados a um pedido;
- status: Estado do pedido;
- payment: Informações relacionadas ao pagamento como troco, método de pagamento e valor;
- delivery: Custo do envio;
- client: Informações do cliente como endereço e telefone.
- list_items: Informações do pedido como nome do produto, tamanho e opcionais;
- employees: Identificador que apresenta informações dos entregadores;
- idemployee: número de identificação do entregador;
- name: Nome do entregador.

A função *JoinBufRecv* tem como objetivo realizar mais algumas alterações na *string* com dos dados e adicionar em uma variável específica o valor do atributo encontrado. É escolhida a variável de acordo com o nome do atributo encontrado

na função *ParserBufRecv*. As alterações na string são realizadas com o auxílio da função *CheckBufRecv*.

A seguir são apresentadas as implementações na linguagem *POSXML* dessas funções:

```
<function name="BuildOrderMsg">
  <stringvariable value="" variable="sBuf" />
  <stringvariable value="1" variable="sStatus" />
  <stringvariable value="GET /" variable="sVerb" />
  <stringvariable value="cloudwalk_api/order_list?"
  variable="sPath" />
  <joinstring firstvalue="$(sVerb)" secondvalue="$(sPath)"
  variabledestination="$(sBuf)" />
  <joinstring firstvalue="$(sBuf)" secondvalue="token="
  variabledestination="$(sBuf)" />
  <joinstring firstvalue="$(sBuf)" secondvalue="$(sToken)"
  variabledestination="$(sBuf)" />
  <joinstring firstvalue="$(sBuf)" secondvalue="$(sAmp)"
  variabledestination="$(sBuf)" />
  <joinstring firstvalue="$(sBuf)" secondvalue="status="
  variabledestination="$(sBuf)" />
  <joinstring firstvalue="$(sBuf)" secondvalue="$(sStatus)"
  variabledestination="$(sBuf)" />
</function>

<function name="ParserBufRecv">
  <integervariable value="0" variable="iRet" />
  <integervariable value="0" variable="iRetEmploy" />
  <integervariable value="0" variable="iLen" />
  <integervariable value="0" variable="iElements" />
  <integervariable value="0" variable="iElementIndex" />
  <string.replace original_string="$(sBufRecv)"
  old_substring="[{" new_substring="},"
  variablereturn="$(sBufRecv)" />
  <string.elements string="$(sBufRecv)" delimiter="},"
```

```
variablereturn="$(iElements)" />
<while variable="$(iElementIndex)" operator="lessthan"
value="$(iElements)">
  <if variable="$(iOrder)" operator="greaterthan" value="1">
    <break />
  </if>
  <stringvariable value="" variable="sObj" />
  <string.elementat string="$(sBufRecv)"
  element_index="$(iElementIndex)" delimiter="},"
  variablereturn="$(sObj)" />
  <string.length value="$(sObj)" variablereturn="$(iLen)" />
  <if variable="$(iLen)" operator="greaterthan" value="5">
    <!-- order_list -->
    <string.find string="$(sObj)" substring="order_list" start="
        0" variablereturn="$(iRet)" />
    <if variable="$(iRet)" operator="notequalto" value="-1">
      <stringvariable value="order_list" variable="sLastParam" /
          >
      <callfunction name="JoinBufRecv" />
   <else />
      <!-- status -->
      <string.find string="$(sObj)" substring="status" start="0"
           variablereturn="$(iRet)" />
      <if variable="$(iRet)" operator="notequalto" value="-1">
        <stringvariable value="status" variable="sLastParam" />
        <callfunction name="JoinBufRecv" />
      <else />
        <!-- payment -->
        <string.find string="$(sObj)" substring="payment" start=
            "0" variablereturn="$(iRet)" />
        <if variable="$(iRet)" operator="notequalto" value="-1">
          <stringvariable value="payment" variable="sLastParam"
              />
          <callfunction name="JoinBufRecv" />
        <else />
```

```
<!-- delivery -->
<string.find string="$(sObj)" substring="delivery"
    start="0" variablereturn="$(iRet)" />
<if variable="$(iRet)" operator="notequalto" value="-1
    ">
  <stringvariable value="delivery" variable="
      sLastParam" />
  <callfunction name="JoinBufRecv" />
<else />
  <!-- client -->
  <string.find string="$(sObj)" substring="client"
      start="0" variablereturn="$(iRet)" />
  <if variable="$(iRet)" operator="notequalto" value="
      -1">
    <stringvariable value="client" variable="
        sLastParam" />
    <callfunction name="JoinBufRecv" />
  <else />
    <!-- list_items -->
    <string.find string="$(sObj)" substring="
        list_items" start="0" variablereturn="$(iRet)"
         />
    <if variable="$(iRet)" operator="notequalto" value
        ="-1">
      <stringvariable value="list_items" variable="
          sLastParam" />
      <callfunction name="JoinBufRecv" />
    <else />
      <!-- Employess -->
      <string.find string="$(sObj)" substring="
          employees" start="0" variablereturn="$(iRet)
          " />
        <if variable="$(iRet)" operator="notequalto"
            value="-1">
```

```
                    <stringvariable value="employees" variable="
                        sLastParam" />
                      <callfunction name="JoinBufRecv" />
                  <else />
                    <!-- IdEmployee -->
                    <string.find string="$(sObj)" substring="
                        idemployee" start="0" variablereturn="$(
                        iRet)" />
                    <if variable="$(iRet)" operator="notequalto"
                         value="-1">
                      <stringvariable value="idemployee"
                          variable="sLastParam" />
                      <callfunction name="JoinBufRecv" />
                    <else/>
                      <!-- Employee Name -->
                      <string.find string="$(sObj)" substring="
                          name" start="0" variablereturn="$(iRet
                          )" />
                      <if variable="$(iRet)" operator="
                          notequalto" value="-1">
                        <stringvariable value="name" variable="
                            sLastParam" />
                        <callfunction name="JoinBufRecv" />
                      </if>
                    </if>
                  </if>
                </if>
              </if>
            </if>
          </if>
        </if>
      </if>
    </if>
<integeroperator operator="++" variablesource="$(iElementIndex
    )" />
```

```
  </while>
</function>

<function name="CheckBufRecv">
  <stringvariable value="" variable="sAux" />
  <!-- Replace the special chars to use getvaluebykey -->
  <if variable="$(sBufTemp)" operator="notequalto" value="">
    <string.replace original_string="$(sBufTemp)" old_substring='"
        :"' new_substring='=' variablereturn="$(sBufTemp)" />
    <string.replace original_string="$(sBufTemp)" old_substring='"
        : ' new_substring='=' variablereturn="$(sBufTemp)" />
    <string.replace original_string="$(sBufTemp)" old_substring=',
         "' new_substring='", "' variablereturn="$(sBufTemp)" />
    <string.replace original_string="$(sBufTemp)" old_substring=':
         "' new_substring='=' variablereturn="$(sBufTemp)" />
    <string.replace original_string="$(sBufTemp)" old_substring='
        {"coments"' new_substring='{"coment"' variablereturn="$(
        sBufTemp)" />
    <string.replace original_string="$(sBufTemp)" old_substring='
        ""' new_substring='"' variablereturn="$(sBufTemp)" />
  </if>
</function>

<function name="JoinBufRecv">
  <integervariable value="0" variable="iRet" />
  <!-- CLIENT -->
  <if variable="$(sLastParam)" operator="equalto" value="client">
    <stringvariable value="$(sBufClient)" variable="sBufTemp" />
    <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
        variabledestination="$(sBufTemp)" />
    <callfunction name="CheckBufRecv" />
    <stringvariable value="$(sBufTemp)" variable="sBufClient" />
  </if>
  <!-- STATUS -->
  <if variable="$(sLastParam)" operator="equalto" value="status">
    <stringvariable value="$(sBufStatus)" variable="sBufTemp" />
```

```
  <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
      variabledestination="$(sBufTemp)" />
  <callfunction name="CheckBufRecv" />
  <stringvariable value="$(sBufTemp)" variable="sBufStatus" />
</if>
<!-- DELIVERY -->
<if variable="$(sLastParam)" operator="equalto" value="delivery"
    >
  <stringvariable value="$(sBufDelivery)" variable="sBufTemp" />
  <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
      variabledestination="$(sBufTemp)" />
  <callfunction name="CheckBufRecv" />
  <stringvariable value="$(sBufTemp)" variable="sBufDelivery" />
</if>
<!-- PAYMENT -->
<if variable="$(sLastParam)" operator="equalto" value="payment">
  <stringvariable value="$(sBufPayment)" variable="sBufTemp" />
  <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
      variabledestination="$(sBufTemp)" />
  <callfunction name="CheckBufRecv" />
  <stringvariable value="$(sBufTemp)" variable="sBufPayment" />
</if>
<!-- LIST ITENS -->
<if variable="$(sLastParam)" operator="equalto" value="
    list_items">
  <stringvariable value="$(sBufCategory)" variable="sBufTemp" />
  <string.length value="$(sBufTemp)" variablereturn="$(iLength)"
      />
  <!-- insert a delimiter between the itens -->
  <if variable="$(iLength)" operator="greaterthan" value="0">
    <string.find string="$(sObj)" substring="list_items" start="
        0" variablereturn="$(iRet)" />
    <if variable="$(iRet)" operator="notequalto" value="-1">
      <joinstring firstvalue="$(sBufTemp)" secondvalue='"'
          variabledestination="$(sBufTemp)" />
```

```
        <string.replace original_string="$(sBufTemp)"
            old_substring='"' new_substring='' variablereturn="
            $(sBufTemp)" />

        <joinstring firstvalue="$(sBufTemp)" secondvalue="|"
            variabledestination="$(sBufTemp)" />
      </if>
    </if>
    <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
        variabledestination="$(sBufTemp)" />
    <callfunction name="CheckBufRecv" />
    <stringvariable value="$(sBufTemp)" variable="sBufCategory" />
  </if>
  <!-- ID EMPLOYEE -->
  <if variable="$(sLastParam)" operator="equalto" value="
      idemployee">
    <stringvariable value="$(sBufIdEmployee)" variable="sBufTemp"
        />
    <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
        variabledestination="$(sBufTemp)" />
    <callfunction name="CheckBufRecv" />
    <stringvariable value="$(sBufTemp)" variable="sBufIdEmployee"
        />
  </if>
  <!-- EMPLOY NAME -->
  <if variable="$(sLastParam)" operator="equalto" value="name">
    <stringvariable value="$(sBufName)" variable="sBufTemp" />
    <joinstring firstvalue="$(sBufTemp)" secondvalue="$(sObj)"
        variabledestination="$(sBufTemp)" />
    <callfunction name="CheckBufRecv" />
    <stringvariable value="$(sBufTemp)" variable="sBufName" />
  </if>
</function>
```