CENTRO EDUCACIONAL DA FUNDAÇÃO SALVADOR ARENA

FACULDADE DE TECNOLOGIA TERMOMECANICA

MATHEUS SUAREZ SILVA

RICARDO CARDOSO PETRÉRE

DESENVOLVIMENTO DE UMA APLICAÇÃO

MULTIPLATAFORMA UTILIZANDO QT

SÃO BERNARDO DO CAMPO

2014

MATHEUS SUAREZ SILVA

RICARDO CARDOSO PETRÉRE

DESENVOLVIMENTO DE UMA APLICAÇÃO

MULTIPLATAFORMA UTILIZANDO QT

Trabalho de Conclusão de Curso, realizado sob orientação do Prof. Ms. Eduardo Rosalém Marcelino, apresentado à Faculdade de Tecnologia Termomecanica como requisito para obtenção do título de Tecnólogo.

SÃO BERNARDO DO CAMPO

2014

MATHEUS SUAREZ SILVA

RICARDO CARDOSO PETRÉRE

DESENVOLVIMENTO DE UMA APLICAÇÃO

MULTIPLATAFORMA UTILIZANDO QT

Trabalho de Conclusão de Curso – Faculdade de Tecnologia Termomecanica

Comissão Julgadora

Professor Ms. Eduardo Rosalém Marcelino

Professor Ms. Ricardo S. Jacomini

SÃO BERNARDO DO CAMPO

2014

**AGRADECIMENTOS**

* Agradecemos
* Agradecemos

*“Citação*

*“*

**<Autor>**

**RESUMO**

Este trabalho tem como objetivo central o desenvolvimento de uma aplicação multiplataforma utilizando o framework Qt. <Continua>

***Palavras-chaves:*** Multiplataforma, Qt

**ABSTRACT**

This piece of work has the intent to

***Keywords:*** Cross-platform, Qt

**LISTA DE QUADROS**

**LISTA DE FIGURAS**

[Figura 1 – Logo do Qt 16](#_Toc400328401)

[Figura 2 – Programas que utilizam Qt 20](#_Toc400328402)

[Figura 3 – Wireshark 2 Preview em funcionamento 21](#_Toc400328403)

[Figura 4 – A comunidade KDE 22](#_Toc400328404)

[Figura 5 – Interface do Plasma 23](#_Toc400328405)

[Figura 6 – Tela inicial do Qt Creator 25](#_Toc400328406)

[Figura 7 - Exemplo de propriedades atreladas à outras (*property bindings*) 26](#_Toc400328407)

[Figura 8 – Exemplo de código em QML para criação de um retângulo 27](#_Toc400328408)

[Figura 9 – Alguns tipos utilizados no sistema de tipagem da linguagem QML. 27](#_Toc400328409)

[Figura 10 – Exemplo de tela com *widgets* aninhados 28](#_Toc400328410)

[Figura 11 – Exemplo de Estilos dos *Widgets*. 29](#_Toc400328411)

[Figura 12 - Exemplos de gerenciadores de *layouts* do módulo Qt Widgets 30](#_Toc400328412)

**LISTA DE SIGLAS**

**API –** Application Programming Interface

**CSS** **–** Cascading Style Sheets

**DOS –** Disk Operating System

**ENIAC –** Electronic Numerical Integrator Analyzer and Computer

**GTK+ –** GIMP Toolkit

**GUI –** Graphical User Interface

**IDE –** Integrated Development Environment

**KDE –** K Desktop Environment

**MVC -** Model-View-Controller

**PDA** **–** Personal Digital Assistant

**PDF –** Portable Document Format

**QML –** Qt Meta-Objects Language

**QPL –** Q Public License

**SO –** Sistema Operacional

**SVG –** Scalable Vector Graphics

**XML –** eXtended Markup Language

**WinRT –** Windows Runtime

**SUMÁRIO**

[1 INTRODUÇÃO 12](#_Toc400328413)

[1.1 OBJETIVO GERAL 13](#_Toc400328414)

[1.2 OBJETIVOS ESPECÍFICOS 13](#_Toc400328415)

[1.3 JUSTIFICATIVA 14](#_Toc400328416)

[1.4 METODOLOGIA 14](#_Toc400328417)

[1.5 ESTRUTURA DO TRABALHO 14](#_Toc400328418)

[2 TECNOLOGIA UTILIZADA 16](#_Toc400328419)

[2.1 INTRODUÇÃO 16](#_Toc400328420)

[2.2 QT 16](#_Toc400328421)

[2.2.1 HISTÓRIA DO QT 16](#_Toc400328422)

[2.3 EXEMPLOS DE APLICAÇÕES QUE UTILIZAM QT 20](#_Toc400328423)

[2.3.1 WIRESHARK 20](#_Toc400328424)

[2.3.2 KDE 22](#_Toc400328425)

[2.4 QT CREATOR 25](#_Toc400328426)

[2.4.1 QT QUICK 26](#_Toc400328427)

[2.4.1.1 QML 26](#_Toc400328428)

[2.4.2 QT WIDGETS 28](#_Toc400328429)

[3 ALTERNATIVAS AO QT 31](#_Toc400328430)

[4 ESTUDO DE CASO 32](#_Toc400328431)

[5 CONCLUSÃO 33](#_Toc400328432)

[6 TRABALHOS FUTUROS 34](#_Toc400328433)

[7 REFERÊNCIAS 35](#_Toc400328434)

1. INTRODUÇÃO

Vários anos se passaram desde a criação dos primeiros computadores, que basicamente faziam cálculos antes feitos manualmente por pessoas.

As máquinas que precederam os computadores da forma como são hoje eram chamadas de tabuladoras, e elas eram capazes de processar dados através da separação de cartões perfurados. O funcionamento desse sistema era bastante simples: a máquina atribuía o valor 0 (zero) para um espaço sem furo e o valor 1 (um) para furado.

A máquina tida como o primeiro computador digital-eletrônico, o *Electronic Numerical Integrator Analyzer and Computer* (ENIAC) era programável manualmente, através do uso de fios e chaves. Os dados a serem processados entravam via cartão perfurado, e seus programas costumavam demorar de uma hora a um dia inteiro para serem criados e executados (FONSECA FILHO, 2007).

Com o avanço da tecnologia, os computadores foram ficando cada vez menores, e o sistema de cartões perfurados foi substituído pela proposta do matemático húngaro John Von Neumann, que sugeriu o armazenamento das instruções antes passadas através de cartões perfurados na memória do computador, o que tornaria o acesso à elas mais rápido. Essa proposta deu certo, e os computadores de hoje seguem esse mesmo modelo.

Com essa evolução, começaram a surgir as primeiras linguagens de programação de alto nível por volta da década de 1950, e elas requeriam um compilador, que interpretava o código escrito e gerava um equivalente em linguagem de máquina.

Os primeiros sistemas operacionais modernos surgiram entre a década de 1960 e 1970, sendo todos baseados em Unix, que foi o primeiro a ser escrito em linguagem C. Porém, ele se tratava de um sistema operacional para máquinas de grande porte, e com a popularização dos computadores pessoais, foi necessário o desenvolvimento de S.Os mais simples, sendo o primeiro deles o *Disk Operating System* (DOS), desenvolvido por Tim Paterson, e adquirido pelos fundadores da Microsoft, William Gates e Paul Allen. Este SO vendeu muitas cópias e foi considerado o sistema operacional padrão para computadores pessoais na década de 80 (FONSECA FILHO, 2007).

A partir disso, novos sistemas operacionais foram desenvolvidos, cada vez mais robustos, acompanhando o desenvolvimento dos microcomputadores, sendo os principais, o Windows, desenvolvido e distribuído pela Microsoft, o Machintosh, cujo nome foi posteriormente mudado para Mac OS e OS X, desenvolvido e distribuído pela Apple em seus computadores, e o que viria a ser o Linux, desenvolvido por Linus Torvalds, que é um sistema operacional livre (gratuito).

<Insira parágrafo sobre Qt e sistemas operacionais aqui>

O presente trabalho abordará a utilização da linguagem de programação de alto nível C++ aplicada para desenvolvimento multiplataforma, através do framework de aplicações Qt e a IDE Qt Creator, que utilizando compiladores específicos para cada plataforma, permite que o mesmo código-fonte funcione em vários sistemas operacionais sem a necessidade de adaptações, como em outras linguagens.

* 1. OBJETIVO GERAL

Estudo da tecnologia Qt no que tange o desenvolvimento de aplicações multiplataforma, analisando a responsividade e necessidade ou não de refatoração de código.

* 1. OBJETIVOS ESPECÍFICOS

Para atingir o objetivo principal, os seguintes passos serão seguidos:

Revisão bibliográfica sobre as tecnologias abordadas nesse trabalho de conclusão de curso, incluindo desenvolvimento e compilação de aplicações para sistemas operacionais distintos.

Verificar se uma aplicação Qt desenvolvida para uma determinada plataforma pode ser executada em outras plataformas sem a necessidade ou com um mínimo de adaptações no código fonte.

Averiguar o quão responsiva é a interface gráfica desenvolvida em uma aplicação Qt, testando sua adaptação em diferentes plataformas, como mobile e desktop.

Desenvolvimento de um protótipo de software de envio de mensagens instantâneas com o intuito de analisar o Qt, focando principalmente sua característica que permite o desenvolvimento de aplicações multiplataforma.

* 1. JUSTIFICATIVA

O desenvolvimento de aplicações e softwares pode parecer algo complicado, ainda mais quando o software desenvolvido tem como destino mais de um Sistema Operacional (SO), pois cada SO interpreta o código de uma maneira diferente. Para facilitar a vida dos desenvolvedores, algumas ferramentas e linguagens se propõem a anular ou minimizar as mudanças necessárias em código para que as aplicações funcionem perfeitamente em mais de uma plataforma. Dentre essas ferramentas, há o Qt, que trata-se de uma ferramenta que se propõe a desenvolver aplicações gráficas, multiplataformas e com todos os recursos oferecidos pela linguagem C++ **bla bla bla...**  **Se encontrarem dados sobre o uso e crescimento do Qt, aqui é um bom lugar para escrever e justificar seu uso e estudo.**

* 1. METODOLOGIA

Levantamento bibliográfico. Estudo de Caso, através do desenvolvimento de um protótipo de aplicação para trocas de mensagens de texto entre clientes rodando em diferentes plataformas (nota de rodapé?).

* 1. ESTRUTURA DO TRABALHO

Para contextualizar o trabalho, o documento foi dividido da seguinte maneira:

* No capítulo 2, intitulado “Tecnologia utilizada”, serão abordados diversos aspectos sobre a programação multiplataforma, focando-se no *framework* de aplicações Qt e a ferramenta que o utiliza, o Qt Creator;
* No capítulo 3, sob o título “Alternativas ao Qt”, serão apresentadas ferramentas e linguagens de programação com o mesmo intuito do Qt, e será feita uma breve análise comparativa entre as principais alternativas, assim como a análise de vantagens e desvantagens entre elas;
* No capítulo 4, sob o tema “Estudo de caso”, estarão descritas as funcionalidades do protótipo de aplicação para múltiplas plataformas, assim como seu processo de desenvolvimento;
* No capítulo 5, denominado “Conclusão” será apresentada a conclusão do trabalho, com destaque para as experiências adquiridas e as lições aprendidas;
* No capítulo 6, intitulado “Trabalhos futuros”, serão apresentadas sugestões para aqueles que desejarem continuar esse trabalho.
* Por fim, no capítulo 7, sob o título “Referências”, serão apresentadas as referências bibliográficas em que foram baseadas as pesquisas feitas visando a conclusão deste trabalho.

1. TECNOLOGIA UTILIZADA
   1. INTRODUÇÃO

Nesse capítulo serão abordados os principais aspectos sobre o *framework* de desenvolvimento multiplataforma Qt e a ferramenta que utiliza esse *framework* como base, o Qt Creator. Serão abordados temas como história, plataformas suportadas, tipos de aplicações e vantagens em sua utilização.

* 1. QT

Segundo Blanchette e Summerfield (2008), o Qt é um *framework* de desenvolvimento em C++ multiplataforma utilizando a filosofia “escreva uma vez, compile em qualquer lugar”. Seu intuito é de que programadores possam desenvolver aplicações utilizando apenas um código-fonte e compilando-o para as diversas plataformas nas quais seu programa será utilizado, sem alterações no código. A Figura 1 ilustra o logo do Qt.



Figura 1 – Logo do Qt

Fonte: Qt Digia, 2014a

* + 1. HISTÓRIA DO QT

De acordo com Blanchette e Summerfield (2008), o projeto que se tornaria no futuro o *framework* Qt foi idealizado em 1990 por Haavard Nord e Eirik Chambe-Eng, ambos mestres em ciência da computação pelo *Norwegian Institute of Technology* (Instituto Norueguês de Tecnologia). A ideia surgiu devido à necessidade deles de desenvolver uma aplicação em C++ que possuísse uma interface gráfica em Unix, Machintosh e Windows. Para resolver o problema que possuíam, iniciaram o desenvolvimento de um *framework* multiplataforma que suportava o paradigma de orientação a objetos.

Em 1991, Haavard começou a escrever as primeiras classes do *framework* com a ajuda de Eirik, que era responsável pelo *design*. Em 1993, eles haviam desenvolvido o primeiro kernel gráfico do *framework* Qt e puderam implementar suas primeiras aplicações.

O nome Qt foi criado da seguinte forma: a letra Q foi escolhida como prefixo por sua aparência no editor de texto Emacs de Haavard, enquanto que a letra t vem de *toolkit* (kit de ferramentas), inspirado pelo Xt, uma ferramenta para o desenvolvimento de aplicações *Graphical User Interface* (GUI) para o sistema X11, comum em algumas distribuições Unix (BLANCHETTE; SUMMERFIELD, 2008).

Em 1995, o Qt 0.90 foi liberado ao público pela primeira vez. O novo *framework* podia ser utilizado para o desenvolvimento tanto em Windows quanto em Unix, oferecendo a mesma *Application Programming Interface* (API)para ambas as plataformas. O Qt estava disponível em dois tipos de licenças: uma comercial, voltada para o desenvolvimento de aplicações para fins comerciais, e uma gratuita, para o desenvolvimento de softwares livres.

Em março de 1996, Haavard e Eirik contrataram mais um desenvolvedor, e no mesmo ano, em setembro, foi lançada a versão 1.0 do Qt. Ainda em 1996, iniciou-se o projeto de desenvolvimento do *K Development Environment* (KDE, que será mais detalhado no tópico 2.3.2), liderado por Matthias Ettrich.

No início de 1997 foi lançado o Qt 1.2, e foi decidido que o KDE seria desenvolvido com o uso desse *framework*, o que consolidou o desenvolvimento GUI em C++ para Linux (BLANCHETTE; SUMMERFIELD, 2008).

Em 1998, a última atualização do Qt 1, a versão 1.40 foi desenvolvida, trazendo melhorias de performance.

Em junho de 1999, a versão 2.0 foi lançada, trazendo consigo um novo tipo de licença de código aberto, a *Q Public License* (QPL), que cumpria a definição de código aberto (*Open Source Definition*). Ainda em 1999, o *framework* Qt ganhou o prêmio LinuxWorld como melhor biblioteca/ferramenta.

De acordo com Blanchette e Summerfield (2008), no ano 2000 foi lançado o Qtopia Core (na época chamada de Qt/Embedded), criado para funcionar em sistemas Linux embarcados, e providenciava seu próprio sistema de janelas. No fim do mesmo ano foi lançada a primeira versão do Qtopia, para telefones móveis e *Personal Digital Assistant* (PDA’s). O Qtopia Core recebeu o prêmio de melhor solução para Linux embarcado por 2 anos seguidos, 2001 e 2002.

Em 2001 foi lançado o Qt 3.0, que estava disponível para Windows, Mac, Unix e Linux (*desktop* e embarcado). Nessa versão, foram implementadas 42 novas classes, e seu código ultrapassou 500.000 linhas.

* + - 1. QT 4

Em 2005, a versão 4.0 do Qt foi lançada, trazendo 5 novas tecnologias ao *framework*, sendo elas um conjunto de classes de contêiners, arquitetura *Model-View-Controller* (MVC), um *framework* de design gráfico 2D, um novo renderizador de texto Unicode e um novo tipo de janelas baseadas em ações (QT PROJECT, 2014l).

Ainda em 2005, a versão Qt 4.1 foi lançada, trazendo como novidades suporte à *Scalable Vector Graphics* (SVG) e um módulo back-end *Portable Document Format* (PDF) para o sistema de impressões do Qt.

Segundo Qt Project (2014l), em 2006 a versão 4.2 trouxe o suporte ao Windows Vista e suporte a *Cascading Style Sheets* (CSS) nativo, além do *framework* QgraphicsView para renderização eficiente de objetos 2D.

Em 2007, com a versão 4.3, veio uma melhoria no suporte ao Windows Vista, além de um suporte experimental para renderização de gráficos 3D através do OpenGL.

Em 2008, as maiores inovações na versão 4.4 do Qt foram a melhoria no suporte multimídia, o suporte a *eXtended Markup Language* (XML) e a compatibilidade com o Windows CE, sistema operacional da Microsoft para *tablets* e dispositivos móveis na época.

Na versão 4.5, lançada em 2009, novamente houveram melhorias nos motores gráficos e o Qt para Mac OS X teve várias partes reescritas para oferecer suporte à API Cocoa da Apple, permitindo que as aplicações desenvolvidas para esse sistema operacional pudessem funcionar em hardware Machintosh de 64-bits (QT PROJECT, 2014l).

Ainda em 2009, a versão 4.6 do *framework* foi lançada, trazendo suporte a gestos e *multi-touch* em dispositivos móveis, possibilitando a criação de interfaces intuitivas para os usuários. Ainda nessa versão foram introduzidos suporte ao sistema operacional mobile Symbian e suporte ao Windows 7.

No fim do ano de 2010, a versão 4.7 do Qt trouxe aos desenvolvedores o *Qt Meta-objects Language* (QML), uma linguagem de programação declarativa baseada em JavaScript. Com ela também foi introduzido o Qt Quick, módulo de desenvolvimento para aplicações com interfaces de usuário, que utiliza tanto o QML quanto o C++ como linguagens de desenvolvimento.

Por fim, em 2011 foi lançada a última atualização da versão 4 do Qt, a 4.8, onde basicamente foram trazidas melhorias de performance (QT PROJECT, 2014l).

* + - 1. QT 5

No final de 2012 foi lançada a mais recente versão do *framework*, o Qt 5. Com ele se tornou possível a criação de interfaces mais intuitivas para desenvolvimento em várias plataformas e suporte total à tecnologia de *touch screen*. Segundo Qt Project (2014m), essa nova versão também traz maior flexibilidade ao desenvolvedor, graças à melhoria na integração entre JavaScript, QML e C++.

Em julho de 2013, foi lançada a versão 5.1 do Qt, que trouxe suporte experimental para Android e iOS. No fim do mesmo ano, com o lançamento da versão 5.2 do *framework*, foi anunciado oficialmente o suporte para ambas plataformas.

Em maio de 2014 foi lançada a mais recente versão do Qt, a 5.3 que está focada em desempenho, estabilidade e usabilidade, além do suporte experimental para o Windows Runtime e a plataforma Windows Phone (QT PROJECT, 2014m).

Até o fim de 2014 pretende-se lançar a versão 5.4 do *framework*, e as principais novidades serão o suporte total à interface Windows RT, melhoria de motores gráficos e o suporte para que os componentes tenham a aparência nativa do Android.

O Qt tem aumentado cada vez mais sua popularidade, sendo a plataforma utilizada na criação de diversos softwares de renome, como Amazon Kindle, Google Earth, Guitar Pro, KDE, EA Origin, Oracle VirtualBox e o futuro Wireshark 2 (no momento em estágio *Preview*) (WIRESHARK, 2014a), como exemplificado na Figura 2:



Figura 2 – Programas que utilizam Qt

Fonte: Própria

Atualmente, o Qt é capaz de compilar aplicações para sistemas *desktop*, como Windows, Mac OS X e distribuições Linux, assim como algumas plataformas *mobile*, como Android, iOS, Windows CE e BlackBerry (QTPROJECT, 2014b). Segundo o Qt Project (2014a), está em processo de desenvolvimento o suporte completo para Windows Runtime (WinRT), permitindo a compilação para Windows Phone e utilização da interface MetroUI das versões Windows 8 e Windows 8.1.

* + 1. SINAIS E SLOTS (*SIGNALS AND SLOTS*)

Colocar texto de Qt Project (2014o) [**http://qt-project.org/doc/qt-4.8/signalsandslots.html**](http://qt-project.org/doc/qt-4.8/signalsandslots.html)

* 1. EXEMPLOS DE APLICAÇÕES QUE UTILIZAM QT

**<Insira introdução aqui>**

* + 1. WIRESHARK

O Wireshark é um analisador de protocolos de rede, sendo considerado o principal *software* do ramo. Ele foi inicialmente criado em 1997 por Gerald Combs, formado em ciência da computação pela Universidade do Missouri-Kansas City, sob o nome de Ethereal, nome que foi mudado para Wireshark em 2006 (WIRESHARK, 2014b).

De acordo com o Wireshark (2014b), o desenvolvimento da ferramenta começou quando Combs estava precisando rastrear um problema de rede na empresa onde trabalhava, e não havia encontrado uma ferramenta que lhe servisse, então decidiu criar seu próprio *software*.

Seu primeiro lançamento foi na versão 0.2.0 em julho de 1998. Em questão de meses, começaram a surgir mais e mais pessoas dispostas a contribuir com o projeto, seja com contribuições realizadas no código-fonte ou inserindo novos decodificadores de protocolos (também chamados de *dissectors* pelo Wireshark). Segundo Wireshark (2014c), atualmente, a lista de colaboradores já passa de 800 pessoas.

Até o presente momento, o Wireshark é desenvolvido utilizando a ferramenta de desenvolvimento multiplataforma GTK+ (*GIMP Toolkit*), mas, a partir da versão 1.11, foi iniciada a migração do código-fonte para o *framework* Qt, de modo a unificar a interface gráfica entre as plataformas (Gerald Combs alega que a versão do Wireshark para o Mac OS X não “aparenta ou age de forma alguma como uma aplicação para Mac OS X”) (WIRESHARK, 2014d).

Desde então, ao instalar a versão mais atual do Wireshark, são instaladas as duas versões (Wireshark original e Wireshark 2 *Preview*), de modo que os usuários possam se acostumar à nova interface. A Figura 3 indica o estado atual da nova interface do Wireshark em funcionamento:



Figura 3 – Wireshark 2 Preview em funcionamento

Fonte: Própria

* + 1. KDE

O KDE é um ambiente de trabalho criado inicialmente para os sistemas operacionais baseados no Unix, e que atualmente faz parte da comunidade KDE, uma comunidade voltada para o desenvolvimento de aplicações de código livre e *Open Source* (KDE, 2014c). A comunidade KDE é estruturada conforme a Figura 4:



Figura 4 – A comunidade KDE

Fonte: KDE, 2014c

Segundo KDE (2014a), o KDE foi projetado em 1996 por Matthias Ettrich, que se encontrava insatisfeito com os ambientes de trabalho existentes na época para os sistemas Unix. Ele alegava que era necessária a existência de uma GUI que oferecesse uma aparência e sensação comum para todas as aplicações. Portanto, iniciou o projeto KDE em busca de apoio para que tal ambiente fosse criado.

Desde o princípio do projeto, Ettrich especificou que seria utilizado o Qt para o desenvolvimento de sua ferramenta, devido à sua facilidade na utilização de bibliotecas C++ e portabilidade. Sua preferência pelo Qt alavancou ainda mais a popularidade do *framework*, além de firmar o Qt como a principal ferramenta para o desenvolvimento de aplicações GUI em C++ (BLANCHETTE; SUMMERFIELD, 2008).

Devido ao fato de utilizar o Qt como base, o KDE permite que as aplicações criadas nele possam ser compiladas para diversos sistemas operacionais. Existem versões da ferramenta (e de suas aplicações) para sistemas Unix, Mac OS X e Windows. (KDE, 2014c).

Segundo KDE (2014b), o ambiente de trabalho do KDE (e seu principal produto) se chama Plasma, e é composto pela interface gráfica exemplificada na Figura 5:



Figura 5 – Interface do Plasma

Fonte: KDE, 2014b

Um dos conceitos principais do Plasma é o dos *widgets*. Segundo KDE (2014b), *widgets* são unidades visuais individuais e interdependentes que podem ser posicionadas no ambiente de trabalho. Podem ser adicionados, removidos, redimensionados e interagidos de diversas formas, e podem ter as mais variadas utilidades, como: previsão do tempo, calculadora, compartilhamento de arquivos, entre outros. O Plasma suporta a inserção e utilização de *widgets* feitos especificamente para ele (chamados Plasmóides), assim como provenientes de terceiros, como Google Gadgets e Dashboard, do Mac OS X.

Um ambiente de trabalho padrão do Plasma possui os seguintes componentes:

* Painel: contém *widgets* como o relógio, a área de notificação e a barra de tarefas, além de permitir o posicionamento de *widgets* adicionais nas bordas da tela.
* Caixa de ferramentas do Plasma: localizado no topo direito da tela e à direita do painel, permite alterar as configurações da ferramenta.
* Exibição de Pasta: uma janela que possui visão configurável para qualquer pasta do sistema (o padrão é a área de trabalho do usuário), que é onde se pode manipular os arquivos da pasta em questão. É permitido possuir mais de uma exibição de pasta na área de trabalho, assim como posicioná-las nos paineis.
* Área de trabalho: a tela como um todo, onde os *widgets* e ícones estão presentes.

Segundo KDE (2014d), em 1998 foi fundada a *KDE Free Qt Foundation*, uma organização criada pela representação legal do KDE, KDE e. V., e a Trolltech (criadora do *framework* Qt). Esta fundação tem o propósito de garantir que a ferramenta Qt continue disponível pelas licenças LGPL e GPL, garantindo assim sua utilização para a criação de softwares livres, mais especificamente a ferramenta KDE. O acordo firmado prevê que, caso a detentora dos direitos do Qt cesse tal disponibilidade, a fundação tem todo o direito de liberar a utilização do Qt sob uma licença *open source*.

* 1. QT CREATOR



Figura 6 – Tela inicial do Qt Creator

Fonte: Própria

Qt Creator é um ambiente de desenvolvimento integrado (IDE) que provê ao usuário ferramentas para modelar e desenvolver aplicações com o framework de aplicação Qt (QT PROJECT, 2014c).

Segundo o Qt Project (2014c), o Qt Creator provê dois editores visuais integrados, Qt Quick Designer e Qt Designer, cada um responsável por gerenciar os dois módulos de interface gráfica do Qt: Qt Quick e Qt Widgets

O Qt Creator possui duas versões, uma delas totalmente grátis, que permite a criação de aplicações para plataformas *desktop* (Windows, Linux e Mac) e *mobile* (Android, iOS e mais recentemente Windows Phone 8). A outra versão, denominada Qt Creator Enterprise, oferece uma gama ainda maior de plataformas, incluindo sistemas embarcados, como computadores de bordo de carros **(procurar foto)**, painéis digitais e até mesmo possibilita a criação de aplicações de missão crítica, que funcionam em sistemas de tempo real, onde o tempo de resposta deve ser constante e pré-definido. (QT DIGIA, 2014a).

* + 1. QT QUICK

O Qt Quick (*Qt Quick User Interface Creation Kit*) é um dos submódulos do *framework* Qt. Ele teve seu surgimento na versão 4.7 do *framework* e foi criado pensando em otimizar a criação de interfaces de usuário para plataformas mais responsivas, principalmente os sistemas operacionais *mobile* (QT PROJECT, 2014d).

Uma das particularidades do Qt Quick é relacionada ao modo de programação utilizando tal modo. O código-fonte de uma aplicação do Qt Quick não precisa se basear apenas no C++, pois este módulo introduziu também a linguagem QML, que será tratada com mais detalhes no próximo tópico.

* + - 1. QML

QML é uma linguagem declarativa que faz parte do *framework* Qt. QML é utilizada no desenvolvimento de aplicativos *cross-platform* (explicar) e busca facilitar o projeto e a implementação de UIs para dispositivos móveis através da rapidez na codificação e na prototipagem. (ROSA et al., 2011). Ela possibilita a criação de interfaces fluidas e animadas, além de integração com bibliotecas em C++.

Para Rosa et al. (2011) e Qt Project (2014e), o estilo de programação da linguagem QML é baseado nas linguagens CSS e JavaScript, tornando-se de aprendizado rápido e fácil para programadores C, Qt/C++, Java e principalmente desenvolvedores web. O QML permite o uso de funções em JavaScript para sua lógica, e propriedades entrelaçadas (quando o valor de uma propriedade é relativo ao valor de outra propriedade), como ilustrado na Figura 7:



Figura 7 - Exemplo de propriedades atreladas à outras (*property bindings*)

Fonte: Rosa et al., 2011

Os objetos QML são especificados por meio de seus elementos e cada elemento possui um conjunto de propriedades. Essas propriedades são formadas por pares nome-valor (por exemplo, color:“blue”) e assumem uma variedade de tipos de dados que podem ser referências para outros objetos, strings, números, etc., como exemplificado na Figura 8. Em QML, as propriedades são fortemente tipadas, ou seja, se uma propriedade possui um tipo específico então um valor de tipo diferente não pode ser atribuído à ela. (ROSA et al., 2011)



Figura 8 – Exemplo de código em QML para criação de um retângulo

Fonte: Rosa et al., 2011

Uma aplicação QML é executada através da máquina de execução QML, também chamada de QML runtime. Existem duas maneiras de se iniciar essa máquina de execução: (1) a partir de uma aplicação Qt/C++ (utilizando a classe QDeclarativeView) ou (2) através da ferramenta Qt QML Viewer. (ROSA et al., 2011)

A Figura 9 ilustra alguns dos tipos de dados utilizados em uma aplicação QML:



Figura 9 – Alguns tipos utilizados no sistema de tipagem da linguagem QML.

Fonte: Rosa et al., 2011

A linguagem QML e sua *engine* de infraestrutura são disponibilizadas através do módulo Qt QML, um *framework* para o desenvolvimento de aplicações e bibliotecas utilizando a linguagem QML, além de prover uma API para estender a linguagem com tipos customizados e integrar um código em QML com JavaScript e C++ (QT PROJECT, 2014h).

Entretanto, Qt Project (2014h) também cita que, enquanto o módulo Qt QML provê a linguagem e a infraestrutura para aplicações em QML, o módulo Qt Quick oferece vários componentes visuais, suporte à arquitetura Modelo-View (*Model-View Architecture*), *framework* de animação, e muitas outras funcionalidades para gerar interfaces com usuário.

* + 1. QT WIDGETS

Segundo Qt Project (2014f)*, widgets* são os componentes fundamentais para a criação de interfaces de usuário (UIs) no Qt. Eles provêm *layout* e comportamento no estilo padrão dos componentes de aplicações *desktop*, e podem servir para exibir dados de saída, receber dados de entrada do usuário, ou como contêiners para outros *widgets* se aninharem. A Figura 10 mostra um exemplo de agrupamento de *widgets* em uma janela, e suas respectivas classes:



Figura 10 – Exemplo de tela com *widgets* aninhados

Fonte: Qt Project, 2014f

O módulo Qt Widgets está presente no Qt desde as primeiras versões, sendo o módulo mais maduro para a criação de interfaces gráficas. Segundo Qt Project (2014j), seu intuito, ao contrário do módulo Qt Quick, é de oferecer interfaces de usuário com a aparência nativa da plataforma, por isso ele é mais recomendado para ser utilizado em aplicações voltadas para as plataformas *desktop*, como o Windows, Mac OS X e as distribuições Linux. Isso se deve ao uso de Estilos (*Styles*) na interface de usuário. A Figura 11 exemplifica alguns estilos de interface de usuário do Qt (podendo ser configurados através da classe QStyle). Vale lembrar que os estilos para Windows e Mac OS X só são disponíveis em suas respectivas plataformas (BLANCHETTE; SUMMERFIELD, 2008; QT PROJECT, 2014f).



Figura 11 – Exemplo de Estilos dos *Widgets*.

Fonte: Blanchette; Summerfield, 2008

Outra funcionalidade importante do módulo Qt Widgets é o conceito de *Layouts* e Gerenciadores de *Layouts* (*Layout Managers*). Para Blanchette e Summerfield (2008), o gerenciador de *layouts* é “um objeto que define o tamanho e a posição dos *widgets* que estão sob sua responsabilidade”. O Qt possui diversos gerenciadores de *layout* (vide Figura 12), dentre eles podem-se citar:

* QHBoxLayout: Alinha os *widgets* horizontalmente
* QVBoxLayout: Alinha os *widgets* verticalmente
* QGridLayout: Alinha os *widgets* em células dentro de um *grid*. É possível configurar de modo que um *widget* possa ocupar mais de uma célula.
* QFormLayout: Alinha os *widgets* no formato padrão de dois objetos por linha, visando a utilização do padrão “descrição – campo”.



Figura 12 - Exemplos de gerenciadores de *layouts* do módulo Qt Widgets

Fonte: Qt Project, 2014n

Os gerenciadores de *layout* ajustam automaticamente os objetos-filho assinalando valores de posicionamento e dimensões, de modo que a interface como um todo mantenha sua proporção e usabilidade (BLANCHETTE;SUMMERFIELD, 2008; QT PROJECT, 2014n).

1. ALTERNATIVAS AO QT
   1. INTRODUÇÃO

Nesse capítulo são apresentados alguns exemplos de ferramentas e linguagens de programação que possuem o mesmo propósito do Qt, facilitação do desenvolvimento multiplataforma, e serão analisadas algumas vantagens e desvantagens de cada um.

* 1. XAMARIN

Xamarin é uma plataforma de desenvolvimento multiplataforma voltada principalmente para dispositivos móveis, embora também ofereça suporte para alguns sistemas operacionais desktop. Com Xamarin, os desenvolvedores utilizam a linguagem de programação C# (lê-se C Sharp) para desenvolver seus softwares.

A plataforma Xamarin foi criada e é suportada pela empresa que leva o mesmo nome, que possui como principal sede a de São Francisco, Califórnia. A empresa possui, segundo dados próprios (XAMARIN, 2014a), 170 funcionários que se encontram em 14 países diferentes, aproximadamente 15000 clientes, em 120 países e mais de 700000 desenvolvedores utilizando sua plataforma de desenvolvimento. A Figura 13 apresenta o logo da empresa.



Figura 13 - Logo da empresa Xamarin

Fonte: Xamarin, 2014a

As plataformas suportadas pela Xamarin são: iOS, Android, Windows Phone, Mac e Windows (desktop). Segundo dados da própria empresa (XAMARIN, 2014b), utilizando-se a mesma linguagem de programação, no caso C#, as mesmas APIs e as mesmas estruturas de dados, é possível reaproveitar aproximadamente 75% do código fonte ao portar-se o software desenvolvido de uma plataforma para outra, sendo necessário apenas a reconstrução das interfaces de usuário de cada plataforma desejada. Com a utilização da API Xamarin.Forms(adicionar número do tópico), esse reaproveitamento se aproxima dos 100%.

* + 1. XAMARIN.FORMS

A Xamarin.Forms é uma API feita para que os desenvolvedores não precisem se preocupar com a criação de diversas interfaces, uma para cada sistema operacional em que deseje que seu aplicativo rode. A Xamarin.Forms é voltada somente para desenvolvimento de aplicativos para dispositivos móveis, sendo as plataformas suportadas: Android, iOS e Windows Phone. A API utiliza um conceito onde a interface é desenvolvida utilizando-se *layouts* e controles comuns, como botões, *labels*, listas, entre outros, que posteriormente são combinados com códigos da própria API, formando componentes nativos para as plataformas.

A figura Figura 14 ilustra um trecho de código para a criação de uma interface básica de login, e a figura Figura 15 mostra como essa tela é apresentada por cada um dos sistemas operacionais.



Figura 14 - Código para a criação de uma tela de login comum

Fonte: Xamarin, 2014c



Figura 15 - Código mostrado na Figura 14 rodando, respectivamente, em iOS, Android e Windows Phone

Fonte: Xamarin, 2014c

Existem ainda APIs voltadas completamente para o Android (Xamarin.Android) e para o iOS (Xamarin.iOS), e, segundo a empresa (XAMARIN, 2014c), códigos desenvolvidos utilizando-se a Xamarin.Forms podem ser combinados com códigos das APIs específicas para cada sistema operacional, potencializando o desenvolvimento dos aplicativos.

* + 1. IDEs

A plataforma Xamarin pode ser utilizada em duas IDEs: uma delas é o Visual Studio, da Microsoft, e a outra é uma IDE própria, chamada Xamarin Studio.

O Xamarin para Visual Studio permite a criação de aplicativos com interface gráfica nativa para as plataformas Apple iOS, Google Android e para o próprio Windows. Utilizando a tecnologia denominada IntelliSense que o Visual Studio possui, que mostra sugestões para que o código se auto-complete, trazendo descrições como parâmetros de métodos, sua descrição, sobrecargas, entre outras informações, é possível tornar o desenvolvimento mais rápido e descobrir funcionalidades das APIs para Android e iOS de maneira mais simples.

Para que se possa utilizar o Xamarin no Visual Studio, é necessário fazer o download e instalação do plug-in do Xamarin para a IDE, que pode ser encontrado no próprio site da empresa. Existem alguns requisitos que devem ser atendidos para que o Xamarin possa ser utilizado, sendo eles a necessidade de utilizar Windows 7 ou alguma versão posterior, Visual Studio 2010 *Professional* ou posterior (versões *Express* do Visual Studio não são compatíveis, pois não aceitam plug-ins) e o plug-in do Xamarin. (Xamarin, 2014d)

No entanto, não é possível compilar um aplicativo para iOS sem o compilador da Apple, assim como não é possível fazer o *deploy* de uma aplicação sem os certificados e assinaturas digitais da mesma empresa. O Windows não possui suporte à tais atividades, portanto o Visual Studio precisa que a máquina Windows onde está instalado possua conexão via rede à alguma máquina Mac que possua o sistema operacional Mac OS X na versão 10.8, ou mais recente, que também possua o Xamarin instalado, para que essa segunda máquina possa executar os procedimentos necessários, visto que ela possui os recursos adicionais necessários.

O Xamarin Studio oferece suporte ao desenvolvimento de aplicações para iOS, Android e Mac, e está diponível tanto para Windows quanto para Mac OS X. Na versão para Windows porém, só é possível o desenvolvimento de aplicações para Android e para o próprio Windows. Para desenvolver-se algo para iOS ou Mac OS é necessário utilizar o Xamarin Studio instalado em uma máquina que possua o sistema operacional da Apple instalado e utilizando a API Xamarin.iOS. Como citado anteriormente, o plug-in do Xamarin para a IDE Visual Studio da Microsoft fornece suporte ao desenvolvimento de tais aplicações, desde que possa se conectar à uma máquina com o Mac OS X instalado via rede.

O Xamarin Studio apresenta funcionalidades como código que se auto-completa durante a digitação, ferramentas para a publicação de aplicativos diretamente na App Store da Apple ou na Google Play Store, além de um debugger completo. O Xamarin Studio ainda possui módulos de design de interfaces para Android e iOS, e através das APIs Xamarin.Android e Xamarin.iOS, além da Xamarin.Forms, que permitem a criação de aplicações com interfaces e desempenho de aplicações nativas. (Xamarin, 2014e)

A Figura 16 representa a criação de uma interface gráfica para iOS no Visual Studio da Microsoft, enquanto a imagem Figura 17 representa a interface do Xamarin Studio.

Figura 16 - Criação de interface para iOS utilizando-se o Xamarin para Visual Studio

Fonte: Xamarin, 2014b



Figura 17 - Criação de interface para iOS utilizando-se o Xamarin Studio

Fonte:Xamarin, 2014e

No próprio site da Xamarin é possível fazer o download dos pacotes básicos de desenvolvimento, tanto para Mac OS X quanto para Windows, sendo que em ambos estão inclusos o Xamarin Studio e as APIs Xamarin.iOS e Xamarin.Android, sendo que para Windows o pacote é completado com o plug-in de integração do Xamarin para Visual Studio, e para o Mac OS X o item final é a API Xamarin.Mac, que permite o desenvolvimento de aplicações para a plataforma.

* + 1. CLIENTES DA XAMARIN

A Xamarin, como já mencionado (XAMARIN, 2014a), possui aproximadamente 15000 clientes ao redor do mundo, e esse tópico apresenta exemplos de dois dos principais clientes: Rdio e Kimberly-Clark.

Rdio é um serviço de *streaming* de músicas que possui um grande catálogo, de aproximadamente 25 milhões de títulos. Quando foi lançado, ele possuía 3 códigos fontes diferentes, um para iOS, um para Android e outro para Windows Phone, o que dificultava atualizações e manutenções nos códigos. Em dezembro de 2012 eles migraram o aplicativo para uma nova versão desenvolvida com Xamarin.

Após a atualização houve um ganho de desempenho pois os aplicativos são nativos de cada plataforma, e houve uma facilitação nas questões de atualizações e manutenções, pois mais de 50000 linhas de código eram comuns para todas as plataformas (XAMARIN, 2014F). A Figura 18 representa o Rdio em um iPad rodando iOS e um *smartphone* rodando Android.



Figura 18 - Software Rdio rodando em iOS (tablet) e Android (smartphone)

Fonte: Xamarin, 2014f

A Kimberly-Clark por sua vez, desejava aumentar a força de vendas de seus produtos, voltados para higiene e bem-estar, a partir do uso de iPads para criar propostas customizadas para os clientes. A empresa já havia tentado a utilização de um aplicativo desenvolvido em HTML5 para isso, porém sem sucesso.

A Kimberly Clark então trocou para o Xamarin, que permitiu que eles pudessem reaproveitar grande parte do código já existente em C# além dos serviços web, o que proporcionou muitas horas economizadas em desenvolvimento.

Com o novo aplicativo, desenvolvido com Xamarin, a Kimberly-Clark conseguiu um aumento significativo em sua receita e uma redução no tempo de vendas, o que proporcionou uma maior satisfação para os clientes.

*“Os resultados obtidos a partir de nosso novo aplicativo de vendas de campo são fenomenais – nossa equipe de vendas ama o aplicativo e são capazes de conquistar os clientes e fechar negócios de maneira mais eficiente. A chave para o sucesso do aplicativo é a bela e rápida experiência do usuário possibilitada pelo Xamarin.” (XAMARIN, 2014f).*

A Figura 19 mostra a interface do aplicativo desenvolvido com Xamarin que a Kimberly-Clark utiliza na área de vendas de sua empresa.



Figura 19 - Aplicativo da empresa Kimberly-Clark desenvolvido com utilização do Xamarin

Fonte: Xamarin, 2014f

* + 1. XAMARIN TEST CLOUD

O Xamarin Test Cloud é um serviço oferecido pela empresa baseado no modelo de computação em nuvem para que os desenvolvedores possam criar rotinas de testes automatizados, principalmente para realizar os testes de aceitação de interface gráfica, que são conhecidos por serem custosos (Xamarin, 2014g). Esse teste consiste em instalar o aplicativo desenvolvido em vários aparelhos diferentes que rodem a mesma plataforma, e analisar o comportamento do software em cada um dos equipamentos. O teste de aceitação é importante para que se possam corrigir erros que eventualmente apareçam para um aparelho em específico, antes da liberação do aplicativo para uso público, maximizando a satisfação do usuário, independentemente de qual aparelho ele possua.

O serviço disponibiliza aos desenvolvedores centenas de plataformas diferentes para que os aplicativos desenvolvidos possam ser testados, tanto para iOS quanto para Android, desde aparelhos mais antigos, com versões obsoletas dos sistemas operacionais, até os aparelhos mais recentes com versões atuais das plataformas.

Os testes automatizados podem ser desenvolvidos utilizando-se dois frameworks oferecidos pelo Xamarin, o Xamarin.UITest, que permite a criação de rotinas de teste em C# e é mais voltado para desenvolvedores experientes na criação de testes automatizados, e o Calabash, que permite a escrita das rotinas de teste na linguagem Ruby utilizando o Cucumber, que é um software voltado para testes automatizados.

Após escritos os testes, os desenvolvedores devem submeter seu aplicativo aos servidores da Xamarin, onde ele sera submetido aos testes nos aparelhos desejados pelo desenvolvedor, e depois de feitos os testes, o serviço gera relatórios com resultados obtidos nos testes, *screenshots* e métricas de desempenho, para que o desenvolvedor possa analisá-los e fazer as correções que achar necessárias (Xamarin, 2014h).

A Figura 20 representa a interface do serviço Test-Cloud, para testes automatizados em nuvem.



Figura 20 - Interface do Xamarin Test Cloud

Fonte: Xamarin, 2014h

**Falar sobre Xamarin, Mono, PhoneGap, etc. Atentar sobre Xamarin e Mono, parece que são interligadas, não sei o quanto. Talvez se tornasse redundante falar das duas, talvez não**

1. ESTUDO DE CASO
   1. INTRODUÇÃO

Nesse capítulo, serão apresentados os dados obtidos através do desenvolvimento do protótipo, assim como informações sobre o mesmo, incluindo trechos de código-fonte, diagramas UML e detalhes técnicos sobre a tecnologia escolhida e métodos de desenvolvimento.

* 1. APLICAÇÃO CLIENTE (QTCC)
  2. APLICAÇÃO SERVIDOR (QTCC\_Server)

**Aqui se fala sobre todo o processo de desenvolvimento da aplicação. Quem for escrever o código-fonte provavelmente vai ser quem vai mexer nesse capítulo (até onde eu entendi)**

1. CONCLUSÃO

**A que resposta chegamos em relação à justificativa de pesquisa? Quais são as considerações finais sobre o desenvolvimento do protótipo, e, consequentemente, sobre o tema de pesquisa**

1. TRABALHOS FUTUROS

**Quais são as possíveis implementações que poderiam/deveriam (não falar que não deu por tempo) ter sido implementadas, e para que a pesquisa atual serviria de base em uma pesquisa futura (tese, talvez?)**

1. REFERÊNCIAS

BLANCHETTE, J.; SUMMERFIELD, M.; *C++ GUI Programming with Qt 4, Second Edition.* 2008.Disponível em <<http://www.bogotobogo.com/cplusplus/files/c-gui-programming-with-qt-4-2ndedition.pdf>>. Acesso em 31 agosto 2014.

FONSECA FILHO, C. *História da Computação: O caminho do pensamento e da tecnologia.* 2007. Disponível em: <<http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf>>. Acesso em 29 setembro 2014.

KDE. *KDE Project Announced*. 2014a. Disponível em: <<http://www.kde.org/announcements/announcement.php>>. Acesso em 28 setembro 2014.

\_\_\_\_\_\_*Plasma*. 2014b. Disponível em: <<https://userbase.kde.org/Plasma>>. Acesso em 28 setembro 2014.

\_\_\_\_\_\_*What is KDE*. 2014c. Disponível em: <<https://userbase.kde.org/What_is_KDE>>. Acesso em 28 setembro 2014.

\_\_\_\_\_\_*KDE Free Qt Foundation*. 2014d. Disponível em: <<http://www.kde.org/community/whatiskde/kdefreeqtfoundation.php>>. Acesso em 28 setembro 2014.

MOLKENTIN, D. *Book of Qt4.* 2006. Disponível em <<http://www-cs.ccny.cuny.edu/~wolberg/cs221/qt/books/BookOfQt4.pdf>>. Acesso em 14 setembro 2014.

QT DIGIA. *Qt Creator Enterprise*. 2014a. Disponível em: <<http://qt.digia.com/Product>>. Acesso em 05 setembro 2014.

\_\_\_\_\_\_*Qt 5.4 Alpha Available.* 2014b. Disponível em: <<http://blog.qt.digia.com/blog/2014/09/08/qt-5-4-alpha-available/>>. Acesso em 01 outubro 2014.

QT PROJECT. *Qt for WinRT*. 2014a. Disponível em: <<http://qt-project.org/wiki/WinRT>>. Acesso em 28 agosto 2014.

\_\_\_\_\_\_*Supported Platforms*. 2014b. Disponível em: <<http://qt-project.org/doc/qt-5/supported-platforms.html>>. Acesso em 31 agosto 2014.

\_\_\_\_\_\_*IDE Overview.* 2014c. Disponível em: <<http://qt-project.org/doc/qtcreator-3.2/creator-overview.html>>. Acesso em 31 agosto 2014.

\_\_\_\_\_\_*Qt Quick*. 2014d. Disponível em: <<http://qt-project.org/doc/qt-5/qtquick-index.html>>. Acesso em 01 setembro 2014.

\_\_\_\_\_\_*QML Applications*. 2014e Disponível em: <<http://qt-project.org/doc/qt-5/qmlapplications.html>>. Acesso em 01 setembro 2014.

\_\_\_\_\_\_*Qt Widgets.* 2014f. Disponível em: <<http://qt-project.org/doc/qt-5/qtwidgets-index.html>>. Acesso em 01 setembro 2014.

\_\_\_\_\_\_*New Features in Qt 5.3*. 2014g. Disponível em: <<http://qt-project.org/wiki/New-Features-in-Qt-5.3>>. Acesso em 05 setembro 2014.

\_\_\_\_\_\_*Qt QML*. 2014h. Disponível em: <<http://qt-project.org/doc/qt-5/qtqml-index.html>>. Acesso em 19 setembro 2014.

\_\_\_\_\_\_*Qt Widgets C++ Classes.* 2014i. Disponível em: <<http://qt-project.org/doc/qt-5/qtwidgets-module.html>>. Acesso em 30 setembro 2014.

\_\_\_\_\_\_*User Interfaces.* 2014j. Disponível em: <<http://qt-project.org/doc/qt-5/topics-ui.html>>. Acesso em 30 setembro 2014.

\_\_\_\_\_\_*Qt WebKit.* 2014k. Disponível em: <<http://qt-project.org/doc/qt-5/qtwebkit-index.html>>. Acesso em 30 setembro 2014.

\_\_\_\_\_\_*Whats’s New in Qt 4.* 2014l. Disponível em: <<http://qt-project.org/doc/qt-4.8/qt4-intro.html>>. Acesso em 30 setembro 2014.

\_\_\_\_\_\_*What’s New in Qt 5.* 2014m. Disponível em: <<http://qt-project.org/doc/qt-5/qt5-intro.html>>. Acesso em 30 setembro 2014.

\_\_\_\_\_\_*Layout Management*. 2014n. Disponível em: <<http://qt-project.org/doc/qt-5/layout.html>>. Acesso em 06 outubro 2014.

\_\_\_\_\_\_*Signals & Slots.* 2014o. Disponível em: <<http://qt-project.org/doc/qt-4.8/signalsandslots.html>>. Acesso em 06 outubro 2014.

ROSA, R. E. V. S.; GIL, A. M.; MENDONÇA, P. R. B.; COSTA FILHO, C. F. F.; LUCENA JR., V. F.; *Desenvolvimento Rápido de Aplicações Móveis Utilizando a Linguagem Declarativa QML.* 2011. Disponível em: <<http://www.die.ufpi.br/ercemapi2011/minicursos/MC10.pdf>>. Acesso em 15 setembro 2014.

WIRESHARK. *QtShark*. 2014a. Disponível em: <<http://wiki.wireshark.org/Development/QtShark>>. Acesso em 15 setembro 2014.

\_\_\_\_\_\_*User’s Guide.* 2014b. Disponível em: <<https://www.wireshark.org/download/docs/user-guide-us.pdf>>. Acesso em 28 setembro 2014.

\_\_\_\_\_\_*About.* 2014c. Disponível em: <<https://www.wireshark.org/about.html>>. Acesso em 28 setembro 2014.

\_\_\_\_\_\_*We’re switching to Qt.* 2014d. Disponível em: <<https://blog.wireshark.org/2013/10/switching-to-qt/>>. Acesso em 28 setembro 2014.

XAMARIN. *About Xamarin.* 2014a. Disponível em: <<http://xamarin.com/about>>. Acesso em 19 de outubro 2014.

\_\_\_\_\_\_*Platform.* 2014b. Disponível em: <<http://xamarin.com/platform>>. Acesso em: 19 de outubro 2014.

\_\_\_\_\_\_*Xamarin.Forms.* 2014c. Disponível em: <<http://xamarin.com/forms>>. Acesso em 19 de outubro 2014.

\_\_\_\_\_\_*Installing Xamarin.iOS on Windows.* 2014d. Disponível em: <<http://developer.xamarin.com/guides/ios/getting_started/installation/windows/>>. Acesso em 19 de outubro 2014.

\_\_\_\_\_\_*Introduction do Xamarin Studio.* 2014e. Disponível em: <<http://xamarin.com/studio>>. Acesso em 19 de outubro 2014.

\_\_\_\_\_\_*Xamarin’s Customers.* 2014f. Disponível em: <<http://xamarin.com/customers>>. Acesso em 20 de outubro 2014.

\_\_\_\_\_\_*Introduction to Xamarin Test Cloud.* 2014g. Disponível em: <<http://developer.xamarin.com/guides/testcloud/introduction-to-test-cloud/>>. Acesso em 20 de outubro 2014.

\_\_\_\_\_\_*Xamarin Test Cloud.* 2014h. Diposnível em: <<http://xamarin.com/test-cloud>>. Acesso em 20 de outubro 2014