# Uma Arquitetura de Referência Baseada em Plugins para Sistemas de Informação Mobile

**Enoque Joseneas**[1], **Sandro Andrade** [2]

[1]Análise e Desenvolvimento de Sistemas – Instituto Federal da Bahia (IFBA)
Caixa Postal S/N – 40301-015 – Salvador – BA – Brasil

[2]Departamento de Computação – Instituto Federal da Bahia (IFBA)
Caixa Postal S/N – 40301-015 – Salvador – BA – Brasil

{enoquejoseneas,sandroandrade}@ifba.edu.br

***Abstract.*** *The development of mobile applications has brought a number of challenges to Computer Science. With limitations on features such as battery, storage, and memory, software development for mobile devices imposes important non-functional requirements to be considered in application design. With the expansion of mobile networks, applications have become popular and designing them with reusable components and low coupling is not a trivial task. This work presents a reference architecture for the development of mobile applications oriented to plugins in the context of information systems.*

***Resumo.*** *O desenvolvimento de aplicativos móveis trouxe uma série de desafios para a Ciência da Computação. Com limitações de recursos como a bateria, armazenamento e memória, o desenvolvimento de software para dispositivos móveis impõe requisitos não-funcionais importantes a serem considerados no projeto de aplicativos. Com a expansão das redes móveis, os aplicativos tornaram-se populares e projetá-los com componentes reutilizáveis e baixo acoplamento não é uma tarefa trivial. Este trabalho apresenta uma arquitetura de referência para o desenvolvimento de aplicativos móveis orientado a plugins no contexto de sistemas de informação.*

## 1. Introdução

Os dispositivos móveis apresentam a cada dia novas oportunidades e desafios para as tecnologias da informação, tais como o acesso ubíquo, a portabilidade e a democratização do acesso a informação. Com a expansão da Internet e o grande volume de dados compartilhados nas redes sociais e aplicativos de troca de mensagens, surgiram novos paradigmas (ex: *Big Data*, *Cloud Computing*, *NoSQL*), novas tecnologias como o *Push Notification* e também novas oportunidades de negócio.

O número de downloads de aplicativos cresce a cada dia na *App Store* e *Google Play*, demonstrando uma certa disponibilidade dos usuários de passarem cada vez mais tempo utilizando os aplicativos do que os próprios navegadores de Internet [Berenice Gonçalves e Luiz Gomez 2013]. De acordo com uma pesquisa feita em 2017 [App 2018], os downloads de aplicativos ultrapassaram a marca de 175 bilhões, um crescimento de 60% com relação a 2015. Essa mesma pesquisa mostrou que o consumidor gastou, em média, 86 bilhões de dólares em 2017. Os números expressivos mostram que os usuários

possuem um engajamento forte com o uso de aplicativos o que indica um mercado a ser explorado.

Apesar de contar com diversas ferramentas tais como as IDEs (Android Studio e Eclipse) e frameworks (*Ionic* e *PhoneGap*), o desenvolvimento de aplicativos carece de soluções arquiteturais multiplataforma e componentes de *UI*[1] flexíveis e de alto nível que possibilite ao desenvolvedor criar interfaces com pouca escrita de código. No Android por exemplo, para construir uma interface gráfica é necessário modelar os componentes interativos através de xml e objetos java correspondentes a cada elemento da interface (botões, ícones, imagens etc.). Outra limitação encontrada, é a falta de suporte facilitado para comunicação *RESTful*, visto que os aplicativos utilizam na maioria dos casos algum *web service*.

Com o objetivo de oferecer uma solução para tais limitações, foi desenvolvido um framework baseado no Qt que dispõe do QML para a construção de componentes de UI. Através do QML, o desenvolvedor poderá criar interfaces para o aplicativo usando o paradigma declarativo através de código de alto nível. Este trabalho tem como objetivo o projeto, implementação e avaliação de uma arquitetura orientada a plugins e reutilizável para o desenvolvimento de sistemas de informação *mobile*. Através dos plugins, o desenvolvedor poderá implementar as funcionalidades do aplicativo com maior facilidade de extensão e baixo acoplamento entre os componentes.

Esta arquitetura visa atender quatro requisitos funcionais, disponibilizando para cada um deles, componentes de alto nível para os plugins. Os requisitos são: acesso a rede para comunicação com serviços *RESTful*, persistência de dados local via *SQLITE*, notificações do sistema via *push* e local (partindo do próprio aplicativo) e um mecanismo de comunicação entre objetos através de eventos.

Para avaliar a arquitetura proposta neste trabalho, foram desenvolvidas duas versões de um aplicativo móvel, sendo uma versão baseada nesta arquitetura e a outra versão sem utilizá-la. Após finalizar o desenvolvimento das duas versões, foram extraídas algumas métricas dos dois modelos com o objetivo de destacar os benefícios de utilizar a arquitetura proposta neste trabalho.

## 2. Referencial Teórico

Esta Seção, apresenta as principais referências que contextualizam este trabalho. A Subseção 2.1 apresenta o Qt e o QML. A Subseção 2.2 discute sobre Arquitetura de Software e a Subseção 2.3, descreve Arquitetura de Referência.

### 2.1. O Qt e o QML

O Qt é um *toolkit cross-platform* para desenvolvimento de aplicações com interface gráfica. O Qt é muito mais que um SDK, ele é uma estratégia de tecnologia que permite ao desenvolvedor, de forma rápida e econômica, projetar, desenvolver, implementar e manter uma aplicação multiplataforma oferecendo uma experiência de usuário agradável em todos os dispositivos [QtCompany 2018b].

O Qt possui um amplo apoio à internacionalização, suporte a um banco de dados *SQL*, *parsing* de JSON, gerenciamento de *threads* e suporte a rede [QtCompany 2018c]. O

[1] *User Interface* ou interface do usuário

Qt dispõe ainda de uma linguagem declarativa e interpretada para construir componentes de *UI*, o QML. O QML é uma especificação de interface de usuário e linguagem de programação que permite aos desenvolvedores criar aplicativos com boa performance, animações fluidas e visualmente atraentes. O QML oferece uma sintaxe JSON altamente legível e declarativa, com suporte para expressões imperativas JavaScript combinadas com ligações de propriedades dinâmicas [QtCompany 2018a].

## 2.2. Arquitetura de Software

De acordo com a definição clássica proposta por *Shaw e Garlan* [M. e D. 1996], arquitetura de software define o que é sistema em termos de componentes computacionais e os relacionamentos entre eles, os padrões que guiam suas composições e restrições. Arquitetura de software pode ser compreendida como uma especificação abstrata do funcionamento de um sistema e permite especificar, visualizar e documentar a estrutura e o funcionamento de um programa independente da linguagem de programação na qual ele será implementado [Leite 2017].

Os softwares estão em constante evolução e sofrem mudanças periodicamente, que ocorrem por necessidade de corrigir *bugs* ou de adicionar novas funcionalidades. As mudanças ocorridas no processo de evolução de um software podem torná-lo instável e predisposto a defeitos, além de causar atraso na entrega e custos acima do estimado. Porém, um software que é projetado orientado a arquitetura, possibilita os seguintes benefícios:

- Maior controle intelectual;
- Menor impacto causado pelas mudanças;
- Melhor atendimento aos requisitos não-funcionais;
- Padronização de comunicação entre os componentes e;
- Suporte a reuso de componentes e maior controle dos mesmos.

Neste trabalho, arquitetura de software pode ser compreendida nas decisões de implementação, nas restrições impostas pelo uso dos recursos disponibilizados e dos componentes reusáveis, além dos estilos arquiteturais provenientes das APIs utilizadas, dentre elas, o *Event-Based*, mecanismo de comunicação baseado em eventos provido pelo Qt. Outro aspecto arquitetural deste trabalho é um estilo de desenvolvimento orientado a plugins.

## 2.3. Arquitetura de Referência

Uma arquitetura de referência consiste em uma forma de apresentar um padrão genérico para um projeto [Zambiasi 2012]. Com base nessa arquitetura, o desenvolvedor projeta, desenvolve e configura uma aplicação prototipando-a por meio de componentes reutilizáveis [Zambiasi 2012]. Para compor uma arquitetura de referência é necessário apresentar os tipos dos elementos envolvidos, como eles interagem e o mapeamento das funcionalidades para estes elementos [Hofmeister, Nord e Soni 2000]. De maneira geral, uma arquitetura de referência deve abordar os requisitos para o desenvolvimento de soluções, guiado pelo modelo de referência e por um estilo arquitetural de forma a atender as necessidades do projeto [MacKenzie et al. 2006].

Arquitetura de referência pode ser entendida neste trabalho como uma forma de disponibilizar um padrão genérico para o desenvolvimento de novos aplicativos no contexto de sistemas de informação.

## 3. Trabalhos Relacionados

Nesta seção, serão apresentados os trabalhos relacionados com este projeto. Para cada trabalho relacionado, será descrito uma análise resumida do trabalho.

O primeiro trabalho correlato analisado, consistiu de uma arquitetura de referência denominada *CReAMA – Component-Based Reference Architecture for Collaborative Mobile Applications* [Melotti 2014] e teve como principal objetivo orientar o desenvolvimento de aplicativos móveis baseados em componentes para a plataforma Android. Sistemas desenvolvidos de acordo com essa arquitetura, devem dar suporte ao desenvolvimento de componentes e à criação de aplicações colaborativas por meio da composição desses componentes. O trabalho proposto por Maison Melotti se relaciona com este trabalho pelo fato de terem objetivos semelhantes, que é propor uma arquitetura para facilitar o desenvolvimento de aplicativos móveis, permitindo o reuso facilitado de componentes. Apesar de estarem focados em domínios diferentes, os trabalhos se relacionam no atendimento de três requisitos funcionais: o cache de dados (persistência local); notificações do aplicativo (local e *push notification*) e acesso a rede.

O segundo trabalho correlato, teve como objetivo principal, realizar análise e estudo sobre as tecnologias de desenvolvimento de aplicativos móveis multiplataforma, utilizando a junção dos frameworks *PhoneGap* e *Sencha Touch* [Júnior 2014]. O estudo propôs uma modelagem facilitada de integração entre um aplicativo móvel e outro sistema através de uma aplicação *RESTFul* utilizando *Java EE*. Com a análise das ferramentas e tecnologias levantadas, pode-se concluir que o desenvolvimento de aplicativos utilizando os frameworks *PhoneGap* e *Sencha Touch* tem muitas vantagens. Uma delas é a facilidade de portar o aplicativo para qualquer plataforma móvel. O trabalho realizado por Jauri da Cruz Junior se relaciona com esta arquitetura como um estudo comparativo dos recursos provido pelo Qt com o *PhoneGap*. Identificou-se que o PhoneGap possui APIs para o build do aplicativo nas plataformas mobile e dispõe de uma API de alto nível em Javascript para o desenvolvedor utilizar os recursos do dispositivo independente da plataforma. Os recursos podem ser desde a câmera do aparelho até notificações do sistema. No entanto, o PhoneGap não possui componentes de interface prontos para serem adicionadas na aplicação, por isso no trabalho de Jauri foi utilizado outro framework para composição das telas.

## 4. Projeto da Arquitetura

A arquitetura proposta neste trabalho utiliza os estilos arquiteturais *Client-Server* e *Event-Based* [Taylor, Medvidovic e Dashofy 2009]. *Client-Server* pelo fato de permitir ao aplicativo consumir algum serviço *RESTful* através de uma API de alto nível para acesso a rede, tonando o aplicativo um cliente, enquanto que o serviço *RESTful* representa o servidor. Já o *Event-Based* é um estilo arquitetural que caracteriza a comunicação entre objetos através de eventos [Taylor, Medvidovic e Dashofy 2009], que neste trabalho, é um recurso disposto através de componentes reusáveis. A arquitetura utiliza componentes escritos em C++ que fazem uso de classes do Qt e está restritamente dependente do ambiente de desenvolvimento Qt, que inclue o *Qt Creator* e os módulos do Qt, além ainda do android SDK e NDK.

O núcleo da arquitetura é independente dos plugins e os plugins estão desacoplados do framework. Os plugins não precisam estar mapeados em arquivos de configuração

*qrc*[2], basta que estejam em um diretório dentro de plugins seguido de um arquivo json de configuração e os arquivos necessários para o seu funcionamento (arquivos QML, javascript e imagens). Os plugins serão empacotados no APK (android) e no IPA (ios) e carregados dinamicamente através de uma classe c++ instanciada pelo núcleo da aplicação.

### 4.1. Requisitos Funcionais Suportados

O projeto desta arquitetura visa atender quatro requisitos funcionais entendidos como básicos em todo sistema de informação. Para atender aos requisitos, foi implementado APIs usando classes C++ nativas do Qt. Apesar de o QML dispôr de funcionalidades que poderiam atender a estes requisitos, foi decidido implementar em C++ por questões de desempenho devido menos código interpretado, melhor gerenciamento de memória e para simplificar a implementação de código nos plugins. Os requisitos listados a seguir, foram disponibilizados na arquitetura através de APIs de alto nível que serão detalhadas posteriormente, estes quatro requisitos são:

1. Acesso a rede para comunicação com serviços *RESTful*;
2. Persistência de dados local via *SQLITE*;
3. Notificações do aplicativo via *push* e local;
4. Comunicação entre objetos facilitado.

### 4.2. Infraestrutura de Plugins

As funcionalidades de um aplicativo baseado nesta arquitetura devem ser implementadas através de plugins, atendendo aos requisitos do aplicativo a ser desenvolvido utilizando apenas QML. Os plugins são independentes entre sí e podem incluir arquivos QML, TXT, HTML e imagens em seu diretório. Qualquer componente de um plugin pode reutilizar os componentes públicos usando a diretiva *import "qrc:/publicComponents/"*. Ao todo, dez componentes reutilizáveis estão disponíveis para os plugins.

Os plugins serão conhecidos em tempo de execução e basta adicionar um novo plugin no diretório *plugins* e ele será carregado no próximo *build* do aplicativo. Para que um plugin seja identificado e carregado na aplicação, é necessário obedecer as seguintes restrições:

1ª estar em um sub-diretório dentro de *plugins*;
2ª conter um arquivo *config.json* neste sub-diretório;
3ª conter pelo menos um arquivo QML.

O arquivo *config.json* de um plugin deve ser um objeto json contendo as seguintes propriedades:

1. *listeners* (array): uma lista de strings que identifica os arquivos que serão instanciados como observadores de eventos.
2. *pages* (array): uma lista de objetos que identifica as páginas do plugin que serão acessadas a partir do menu do aplicativo.

Cada objeto em *pages* poderá conter as seguintes propriedades:

[2]qrc – *Qt resource collection* é um arquivo xml que mapeia os arquivos que serão empacotados no aplicativo.

- *qml* (string): O nome do arquivo correspondente à página. Se essa propriedade não for definida, a página não será carregada;
- *title* (string): O título correspondente à página a ser exibido no menu. Esse valor também é requerido, se não for definido, a página não será adicionada ao menu;
- *awesomeIcon* (string) (opcional): O nome de um ícone do *Awesome Icons*[3] que será exibido no menu, ao lado do título;
- *order* (int): Um valor numérico que define a ordem em que a página será exibida na lista de itens no menu. O desenvolvedor deverá definir um valor acima de zero e quanto maior o valor, maior a prioridade da página na lista de itens dos menus.

O Exemplo 1, apresenta um código json contendo uma página a ser exibida no menu do aplicativo.

```
{
    pages:[
        {
            qml:"Page1.qml",
            title:"Pagina 1",
            awesomeIcon:"commenting",
            order:3,
            roles:["student"],
            showInDrawer:true,
            showInTabBar:true
        }
    ]
}
```

**Exemplo 1. JSON de configuração de um plugin**

### 4.3. Utilizando as APIs do framework

Os exemplos a seguir demonstrarão trechos de código das principais APIs disponibilizadas para os plugins. É importante destacar que a escrita de código dos plugins está estritamente ligado a escrita declarativa do QML, logo é importante que o desenvolvedor tenha familiaridade com essa tecnologia.

```
import QtQuick 2.8
import RequestHttp 1.0

RequestHttp{
    id:requestHttp
    baseUrl:"https://static.api.foo.com"
    authorizationUser:"foo"
    authorizationPass:"bar.2018"
}
...
Item{
    Component.onCompleted:{
        var queryString = {
```

[3]https://fontawesome.com/icons

```
            foo:"bar",
        }
        requestHttp.get("bar",queryString)
    }
}
```

**Exemplo 2. Utilizando o RequestHttp para carregar dados de uma url**

O exemplo 3 demonstra como persistir dados usando o mecanismo chave-valor de uma das APIs de persistência de dados do framework, através da classe *Settings* que será instanciada na inicialização do aplicativo e disponibilizada como objeto global. Essa classe possui três métodos principais: *read* para leitura de dados, *save* para persistir dados e *remove* para deletar um dado do banco local do aplicativo.

```
import QtQuick 2.8

Item{
    Component.onCompleted:{
        var foo = Settings.read("foo")
        if (bar != foo)
            Settings.save("foo",bar)
        Settings.remove("bar")
    }
}
...
Item{
    ...
    key:"bar"
    count:Settings.read(key,Settings.TypeInt)
    ...
    Component.onCompleted:{
        Settings.save(key,++count)
    }
}
```

**Exemplo 3. Lendo e persistindo dados através do componente Settings**

## 5. Avaliação Experimental

Esta seção apresentará o estudo conduzido para avaliar a arquitetura proposta neste trabalho. A avaliação consistiu no desenvolvimento de duas versões de um aplicativo móvel e na coleta de métricas relacionadas ao desenvolvimento de cada versão. As métricas foram utilizadas para validar a eficácia e os benefícios da arquitetura proposta neste trabalho. As subseções a seguir, apresentarão os detalhes do estudo conduzido na avaliação.

### 5.1. Planejamento

O objeto de estudo utilizado na avaliação foi o aplicativo *Emile*. O *Emile* consiste de um sistema para facilitar a comunicação acadêmica, permitindo aos professores enviar mensagens aos alunos de suas turmas de eventos diversos que possam ocorrer durante o semestre letivo [GSORT. Grupo de Pesquisa em Sistemas Distribuídos s.d.].

O processo da avaliação consistiu na implementação de três *features* do aplicativo em duas versões separadas, sendo uma com o framework proposto neste trabalho e outra versão escrita sem utilizar o framework. A versão sem utilizar o framework também foi escrita em Qt/QML. As *features* escolhidas estão descritas a seguir:

- Login do usuário. Esse recurso inclui o logout para permitir que o usuário possa encerrar uma seção.
- Gerenciamento de mensagens. Esse recurso inclui o envio de mensagens usando um perfil de professor e a visualização das mensagens enviadas pelo professor, além de visualização das mensagens recebidas por um aluno, ou seja, o aluno também poderá logar no aplicativo e visualizar as mensagens enviadas para a turma a qual ele está matriculado.
- Gerenciamento de perfil do usuário. Esse recurso inclui a visualização e a edição dos dados do usuário. No entanto, apenas a edição dos campos email e senha foram suportados.

O objetivo do estudo era extrair métricas que pudessem destacar as vantagens e os benefícios de utilizar esta arquitetura. As métricas escolhidas foram:

- Número de linhas de código implementado em cada versão.
- Densidade de bugs encontrados em cada versão.

### 5.2. Execução e Coleta de Dados

A avaliação foi realizada no decorrer de quinze dias e primeiramente foi implementado a versão sem o framework. Uma sequência de passos deu início a configuração do aplicativo e as *features* nesta versão foram implementadas em três plugins. É importante destacar, que foi utilizado um serviço REST como servidor do aplicativo. O serviço é necessário para o login e obtenção dos dados do usuário, além de listagem das turmas para envio de mensagem pelo professor.

### 5.3. Resultados e Discussão

Após finalizar o desenvolvimento das versões *Emile1* e *Emile2*, observou-se que a versão sem o framework resultou em um arquivo final (APK) um pouco menor do que a versão com o framework, além do *build* do projeto ocorrer em menor tempo.

Em relação ao número ao linhas, na versão sem o framework foi considerado apenas os plugins, ou seja, foi efetuado a contagem somente dos arquivos presentes no diretório *plugins*. A versão sem o framework foi efetuado a contagem de todos os arquivos. Para utilizar a contagem, foi utilizado o comando *shell*: *wc -l 'find <folder_path> -type f'*. Os resultados obtidos foram:

- Emile1 (versão com o framework): **1116** total de linhas.
- Emile2 (versão sem o framework): **5206** total de linhas.

Em relação ao bugs identificados em cada versão, destaca-se maior complexidade dos bugs na versão com o framework, pois era difícil de depurá-los, pois foram erros internos que exige do usuário do framework conhecimento do código fonte. Na versão sem o framework, foi indentificado os seguintes bugs:

1. Quando o dispositivo não estava conectado a Internet, as requisições HTTP ficavam intermináveis.

2. Erros de requisição HTTP não capturados quando ocorre exceção no servidor e o mesmo não responde um json válido, retornando um HTML.
3. O click no botão "voltar"do android não retorna para a página anterior ou minimiza o aplicativo quando apenas uma página foi acessada pelo usuário. O que ocorre é o fechamento inesperado do aplicativo.
4. Não foi possível permitir que o usuário altere a imagem de perfil, selecionando uma imagem da galeria de arquivos do sistema, pois o QML não dispõe de um componente para acesso a galeria de imagens do dispositivo no android, sendo necessário implementar uma API em java ou C++ com JNI. Na versão com o framework esse recurso já está implementado e pode ser utilizado via *procedure call*.

A lista a seguir, descreve os bugs encontrados na versão com o framework:

1. Após finalizar a implementação, em alguns momentos o aplicativo finalizou inesperadamente.
2. Após um determinado tempo utilizando o aplicativo, após logar, visualizar as mensagens e navegar em algumas páginas, algumas requisições não são iniciadas ou finalizadas. Por exemplo, após fazer o logout e tentar logar novamente o aplicativo não responde.
3. A precisão de cliks no *ToolBar* para abrir o *Drawer* Menu não é muito boa, em algums momentos é preciso clicak até três vezes para abrir o menu.
4. Para utilizar bem os recursos da arquitetura, é preciso conhecer os detalhes do framework, pois ao implementar a persistência de mensagens foi preciso olhar o código fonte da classe *DatabaseComponent* para saber o nome do parâmetro do sinal *itemLoaded* emitido após realizar uma consulta.
5. O build do projeto é muito mais lento do que a versão sem o framework e o tamanho do arquivo final (apk) é em torno de 20% maior.

## 6. Conclusão

Este trabalho apresentou o projeto de uma arquitetura de software baseado em plugins para o desenvolvimento de sistemas de informação mobile. Destacou-se os recursos de baixo acoplamento entre componentes através dos plugins. A característica de plugins desacopla os recursos do aplicativo do núcleo da aplicação, o que permite ao desenvolvedor adicionar, modificar ou remover funcionalidades com maior facilidade. Outra característica, é que o desenvolvedor irá escrever muito menos código, pois o framework já implementa funcionalidades que os plugins podem utilizar através de APIs, evitando ter que escrever código Java, C++ ou Objective C. Também foi realizado uma avaliação da arquitetura através do desenvolvimento de um aplicativo em duas versões, uma com e outra sem a arquitetura com o objetivo de analisar as vantagens e os benefícios de utilizar o framework proposto neste trabalho. Através da avaliação, concluiu-se que a arquitetura carece de depuração facilitada e melhorias na API de acesso a rede, além de uma documentação para cada uma das *features* disponibilizadas.

## Referências

App, Rank My (2018). *App Statistics: retrospectiva de 2017 e projeções para 2018*. URL: https://www.rankmyapp.com/pt-br/mercado/app-statistics-retrospectiva-de-2017-e-projecoes-para-2018. Acessado em: 19/06/2018.

Berenice Gonçalves e Luiz Gomez, Valéria Feijó e (2013). "Heurística para avaliação de usabilidade em interfaces de aplicativos smartphones: utilidade, produtividade e imersão". Em: *Design e Tecnologia* 3.06, pp. 33–42. ISSN: 2178-1974. DOI: 10.23972/det2013iss06pp33-42. URL: https://www.ufrgs.br/det/index.php/det/article/view/141.

GSORT. Grupo de Pesquisa em Sistemas Distribuídos Otimização, Redes e Tempo-Real. IFBA Campus Salvador (s.d.). *Um sistema open-source para comunicação acadêmica*. URL: http://emile.ifba.edu.br. Acessado em 18/06/2018.

Hofmeister, Christine, Robert Nord e Dilip Soni (2000). *Applied Software Architecture*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., p. 432. ISBN: 0-201-32571-3.

Júnior, Jauri Da Cruz (2014). *Solução Multiplataforma Para Smartphone Utilizando Os Frameworks Sencha Touch E Phonegap Integrado À Tecnologia Web Service Java*.

Leite, Jair Cavalcanti (2017). *Design da arquitetura de componentes*. URL: https://www.dimap.ufrn.br/~jair/ES/c7.html. Acessado em: 15/12/2017.

M., Shaw e Garlan D. (1996). *Software Architecture. Perspectives on an Emerging Discipline*. Prentice Hall, p. 242.

MacKenzie, C. Matthew et al. (2006). "Reference Model for Service Oriented Architecture". Em: Acessado em: 18/06/2018. Disponível em: https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html.

Melotti, M. (2014). *CReAMA: Uma Arquitetura de Referência para o Desenvolvimento de Sistemas Colaborativos Móveis Baseados em Componentes.* URL: http://repositorio.ufes.br/handle/10/4270.

QtCompany (2018a). *QML Applications | Qt 5.11*. URL: http://doc.qt.io/qt-5/qmlapplications.html. Acessado em: 15/06/2018.

— (2018b). *Qt - Cross-platform software development for embedded and desktop*. URL: https://www.qt.io. Acessado em: 15/06/2018.

— (2018c). *Qt Toolkit*. URL: http://www.linuxjournal.com/article/201. Acessado em: 15/06/2018.

Taylor, R. N., N. Medvidovic e E. M. Dashofy (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing. ISBN: 0470167742, 9780470167748. URL: https://dl.acm.org/citation.cfm?id=1538494.

Zambiasi, Saulo Popov (2012). "Uma arquitetura de referência para softwares assistentes pessoais baseada na arquitetura orientada a serviços". Em: *Tese (doutorado) - UFSC, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas* 77, p. 331. URL: https://repositorio.ufsc.br/handle/123456789/99348.