CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
Recredenciado pela Portaria Ministerial nº 3.607, de 17/10/05, D.O.U. nº 202, de 20/10/2005
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Marcus Vinicius Germano de Abreu Pereira

DESENVOLVIMENTO DE UMA NOVA ARQUITETURA PARA A APLICAÇÃO FISIOKINECT

Palmas – TO
2016

Marcus Vinicius Germano de Abreu Pereira

DESENVOLVIMENTO DE UMA NOVA ARQUITETURA PARA A APLICAÇÃO FISIOKINECT

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fernando Luiz de Oliveira

Palmas – TO

2016

Marcus Vinicius Germano de Abreu Pereira

DESENVOLVIMENTO DE UMA NOVA ARQUITETURA PARA A APLICAÇÃO FISIOKINECT

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fernando Luiz de Oliveira

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. M.Sc. Fabiano Fagundes

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. M.Sc. Pierre Soares Brandão

Centro Universitário Luterano de Palmas - CEULP

Palmas - TO

2016

AGRADECIMENTOS

RESUMO

O presente trabalho teve como objetivo construir uma nova arquitetura para o FisioKinect baseada em padrões de projeto e arquitetura de software para solucionar os problemas de *layout* e implementação decorrentes da adição de novos softwares na aplicação. A questão central do estudo foi responder ao seguinte problema: como alterar a arquitetura da plataforma FisioKinect de modo a acomodar a implementação de novos softwares sem criar redundâncias de códigos e banco de dados, nem prejudicar seu *layout* por conta das implementações de novos componentes em sua estrutura? A definição de uma arquitetura mais adequada para o problema proposto se deu com base na fundamentação teórica, que foi adquirido através de pesquisas bibliográficas, livros, artigos científicos etc. A eficácia da arquitetura escolhida foi comprovada adicionando os softwares já existentes da arquitetura antiga na nova. A eficácia do *layout* da tela inicial foi confirmada acomodando os botões de acesso aos *softwares* em um *ComboBox* (Caixa de Combinação) e as telas de edição e visualização dos dados, onde continha várias abas, teve as abas transformadas em telas cada uma.

PALAVRAS-CHAVE: FisioKinect, Arquitetura de Software, Padrões de Projeto.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

1 SUMÁRIO

1 INTRODUÇÃO

O crescente uso da tecnologia na área da saúde tem estimulado o desenvolvimento de softwares destinados ao auxílio no tratamento de diversas enfermidades. Dentre as ferramentas tecnológicas utilizadas na recuperação de pacientes está o Kinect. O dispositivo Kinect criado pela Microsoft, divulgado em 2009, além de uma avançada combinação de hardware e software, é capaz de capturar movimentos de objetos a ele expostos.

Rocha, Defavari e Brandão (2012) estudaram as possibilidades de uso do Kinect com pacientes da área da Fisioterapia Neurológica, concluindo que o dispositivo pode ser viável em inúmeras possibilidades. Colaborando com esta conclusão, pesquisadores do Centro Universitário Luterano de Palmas – CEULP / ULBRA tem desenvolvido diversos trabalhos que resultaram em uma plataforma denominada FisioKinect.

O FisioKinect é uma plataforma que agrega softwares voltados para a área da saúde, os quais são desenvolvidos pelo GEPETS, que é um grupo de pesquisa que agrega pesquisadores dos seguintes laboratórios: Laboratório de Tecnologia em Saúde (LTS), Laboratório de Multimídia e Computação Gráfica (LABMIDIA) e Laboratório de Banco de Dados e Engenharia de Software (LBDES); todos localizados no Complexo de Informática do CEULP/ULBRA. Porém, com o desenvolvimento de novos softwares para a plataforma, surgem alguns problemas tanto na organização das implantações dos softwares quanto no *layout* para disponibilidade dos dados na plataforma.

Assim, o presente trabalho abordou o desenvolvimento de uma nova arquitetura para a aplicação FisioKinect, fundamentado em arquitetura de software e padrões de projeto para definir um novo esquema de organização dos arquivos referentes aos softwares. Esta proposta culminou com a reformulação do *layout* da tela inicial e tela de edição e visualização das informações dos pacientes, de modo que o *layout* não seja mais afetado com a implantação de novos softwares, bem como resultou em uma nova forma de distribuição dos arquivos que compõem cada um dos projetos que fazem parte do FisioKinect.

2 REFERENCIAL TEÓRICO

2.1 Kinect

Lançado em 04 de novembro de 2010, o Kinect é um dispositivo desenvolvido pela Microsoft em parceria com a empresa israelense PrimeSense (DOMINGOS, 2011). Com ele imagens podem ser capturadas em alta definição, o que motivou a criação de aplicações para resolver problemas em diversas áreas, sendo destaque a área da saúde.

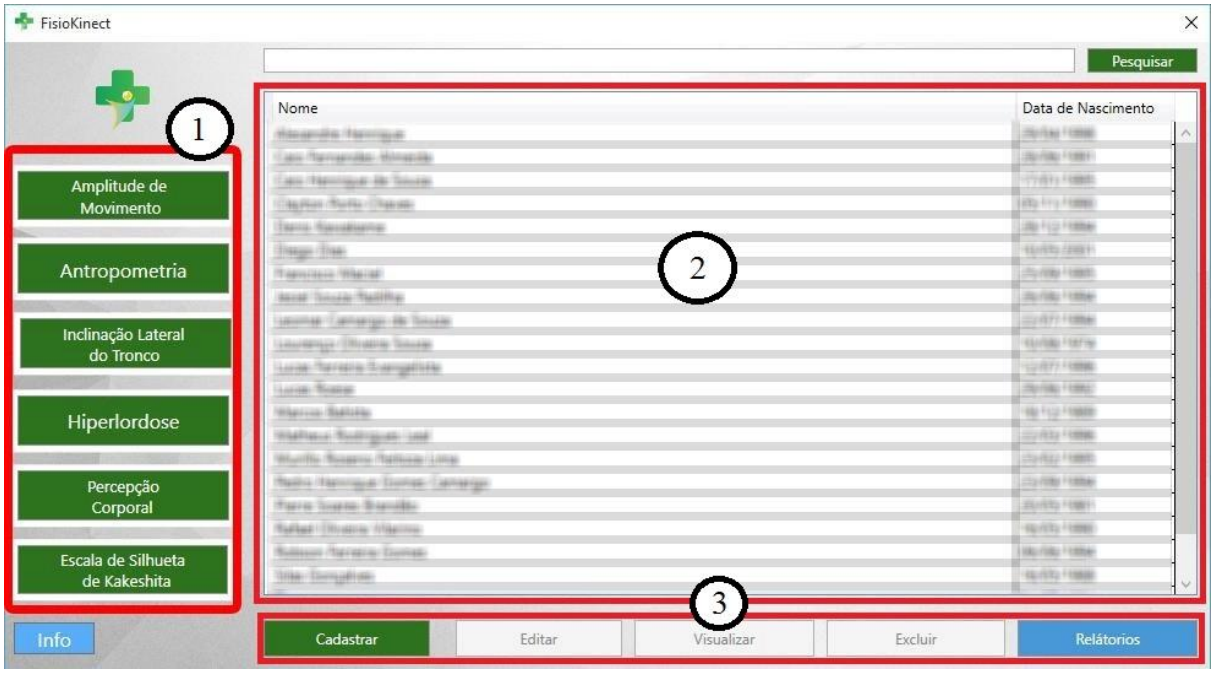
Dentro das utilizações do Kinect na área da saúde, alunos do Centro Universitário Luterano de Palmas desenvolveram uma plataforma chamada de FisioKinect apresentado a seguir.

2.1.1 FisioKinect

O FisioKinect é uma plataforma composta por softwares voltados para a área da saúde, desenvolvida por alunos dos cursos de Ciência da Computação e Sistemas de Informação do Centro Universitário Luterano de Palmas (CEULP) da cidade de Palmas – TO (LIMA, 2015).

Na plataforma é feita a integração entre os softwares, a qual possui uma base de dados única para o gerenciamento das pessoas avaliadas (LIMA, 2015). A seguir a Figura 1 apresenta a interface principal da plataforma.

Figura 1 - Interface Principal



Fonte: CAMARGO, 2015, p. 42

Como apresentado na figura 1, na interface o item 1 destaca o módulo que vincula os softwares desenvolvidos pelo grupo de pesquisa, cada botão referencia um software, neste caso o acesso para o software de Amplitude de Movimento, Hiperlordose, Antropometria, Escoliose e Percepção Corporal (CAMARGO, 2015). No modelo atual esses softwares ampliariam indefinidamente.

O item 2 é o campo que lista os pacientes já cadastrados no sistema. No item 3 estão os botões que permitem o cadastro, edição, exclusão e visualização dos pacientes, junto também está o botão para gerar os relatórios (CAMARGO, 2015).

2.2 Arquitetura de Software

Segundo Bass, Clements e Kazman (2003): "a arquitetura de software de um sistema computacional refere-se a sua estrutura, consistindo de elementos de software, propriedades externamente visíveis desses elementos e os relacionamentos entre eles". Uma arquitetura pode abranger mais de um tipo de estrutura, com tipos distintos de elementos e relacionamento entre eles. Assim sendo, esses elementos incluem informações de como eles se comunicam entre si.

Em 1992, Perry e Wolf descreveram a arquitetura de software a partir de um modelo composto por três elementos básicos (PETERS e PEDRYCZ, 2001):

- um *elemento de processamento* é uma estrutura de software que transforma suas entradas em saídas necessárias (PETERS e PEDRYCZ, 2001);
- um *elemento de dados* consiste nas informações necessárias para o processamento ou em informações a serem processadas por um elemento de processamento (PETERS e PEDRYCZ, 2001);
- os *elementos de conexão* são o "amálgama" que une as diferentes partes de uma arquitetura (PETERS e PEDRYCZ, 2001).

A arquitetura de software se bem documentada simplifica o entendimento da estrutura de um sistema, auxiliando na comunicação com os desenvolvedores e clientes. Além disto, a partir dela é possível tomar decisões em meio ao processo de desenvolvimento do software, o que influencia no sucesso do mesmo (JESUS, 2013).

2.2.1 Estilos de Arquitetura

Os estilos de arquitetura são utilizados como um modelo para desenvolver um software, funcionando como um guia que detalha como os módulos e componentes serão organizados.

Todo software é baseado em algum estilo de arquitetura e cada estilo tem sua forma de organizar os módulos e componentes do software. (JESUS, 2013).

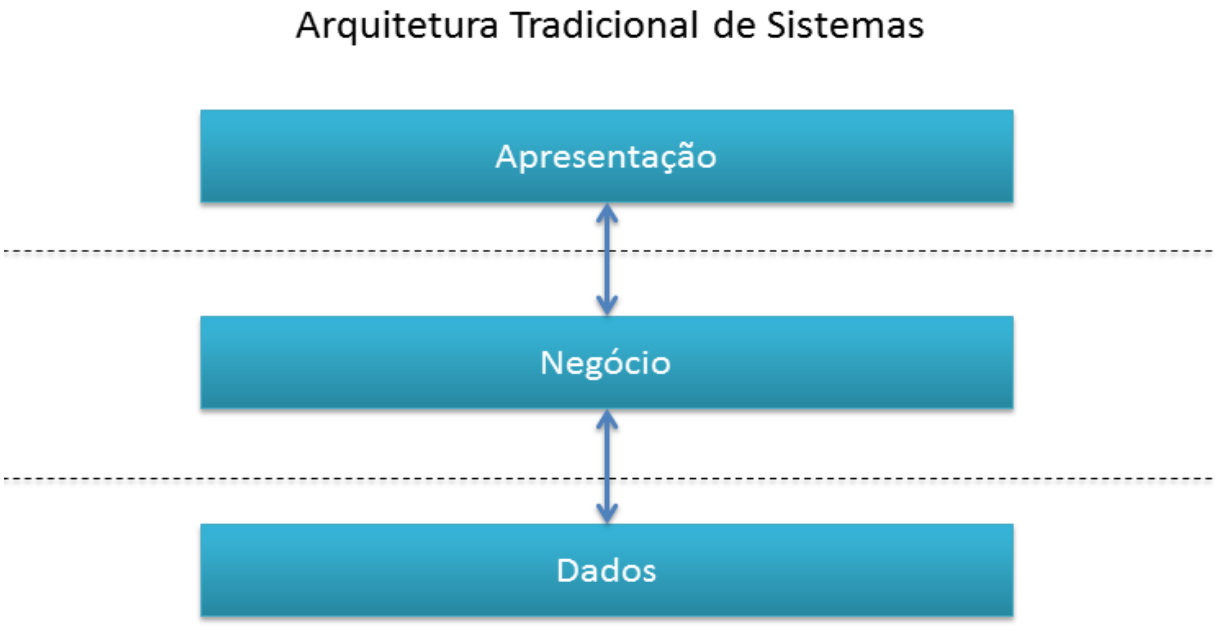
Com o aumento das dimensões e complexidade dos sistemas no decorrer do tempo, houve a necessidade de uma dedicação maior de como seria realizado a organização dos componentes nos sistemas computacionais (MENDES, 2002). Ao longo deste tempo engenheiros e projetistas fizeram uso de vários estilos de arquitetura, na maioria das vezes sem seguir algum princípio, deixando de lado a importância de ter conhecimento das principais características e como utilizá-las de maneira sistemática, buscando catalogá-las (MENDES, 2002).

Existem diversos estilos de arquitetura, por exemplo: arquitetura de repositório, arquitetura de camadas, arquitetura orientada a objetos, pipeline dentre outros. Na seção a seguir será descrito o estilo de arquitetura de camadas e repositório.

2.2.1.1 Arquitetura em Camadas

Neste estilo de arquitetura tem-se o sistema separado em camadas, sendo que cada camada possui uma funcionalidade bem definida, sem a dependência das outras e que se comunicam por meio de uma interface de comunicação bem definida. Basicamente uma camada presta serviços para a superior e faz uso dos serviços da camada inferior (GARLAN e SHAW, 1994). A seguir a Figura 2 apresenta uma arquitetura dividida em 3 camadas.

Figura 2 – Exemplo de arquitetura em 3 camadas



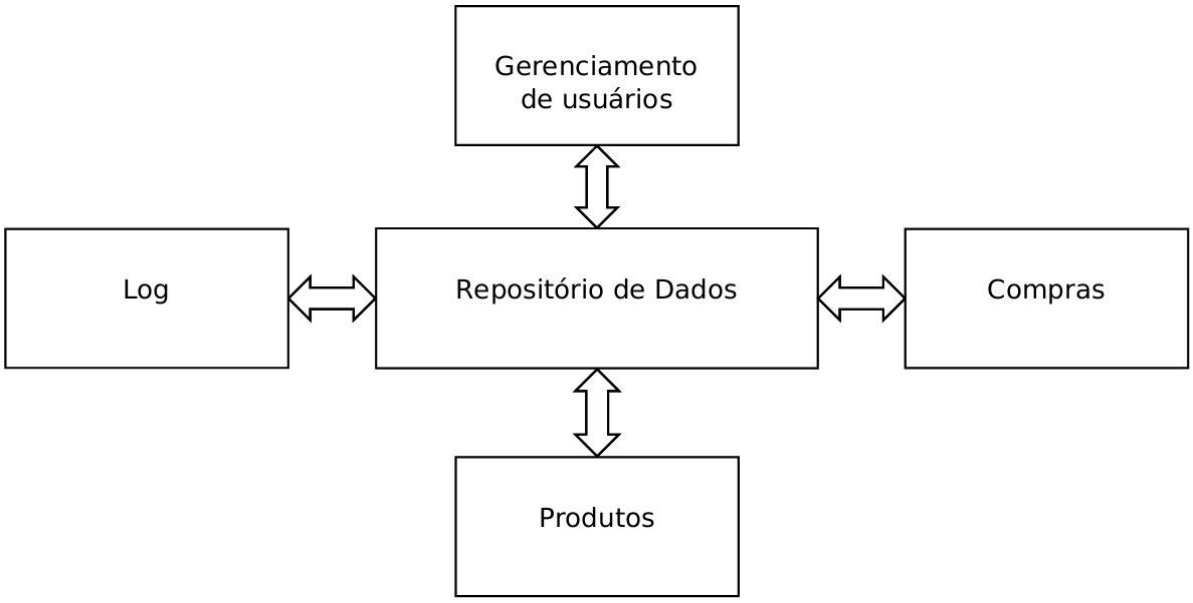
Fonte: Gehrke, 2012, p. 1

A arquitetura de camadas permite o desenvolvimento de um software multiplataforma, por ser simples de adicionar, substituir ou remover as camadas sem prejudicar as demais. Dependendo da quantidade de camadas o sistema perderá desempenho, pela necessidade de percorrer as camadas necessárias para responder as ações feitas no sistema (JESUS, 2013).

2.2.1.2 Arquitetura de Repositório

De acordo com Sommerville (2011), este estilo de arquitetura é ideal para sistemas que trabalham com um grande volume de dados, pelo fato destes tipos de sistemas serem divididos em componentes e os mesmos se comunicarem por meio do banco de dados compartilhado ou repositório.

Figura 3 - Estrutura de uma arquitetura de repositório.



Fonte: Rossi, 2013, p. 1

Segundo Preesman (2011), a centralização dos dados proporciona a integrabilidade, que é a possibilidade de fazer modificações na estrutura, seja para adicionar, remover ou alterar os componentes sem causar problemas na estrutura da aplicação, dada facilidade pelo fato dos componentes operarem independentemente.

No entanto, devido os componentes dependerem do repositório central para manterem comunicação entre si, caso ocorra alguma falha no repositório prejudicará todos os componentes (SOMMERVILLE, 2011).

2.3 Decomposição Modular

Conforme Sommerville (2007), após selecionada a forma que os componentes serão organizados, é preciso decidir que abordagem utilizar na decomposição dos subsistemas em módulos. Existem duas maneiras principais para se usar na decomposição dos subsistemas em módulos:

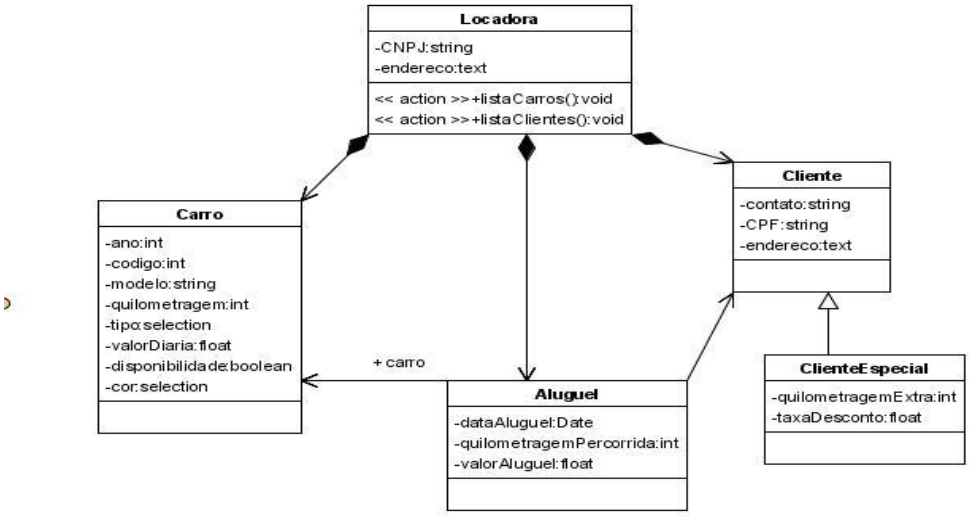
- *decomposição orientada a objetos*: no qual sistema é composto por um conjunto de objetos que se comunicam (SOMMERVILLE, 2007);
- *pipelining*: orientado a funções que são decompostas em módulos funcionais que aceitam dados de entrada e os converte em dados de saída (SOMMERVILLE, 2007).

Na decomposição *orientada a objetos*, os módulos são objetos privados com funções definidas sobre eles. Na decomposição *Pipelining* os módulos são transformações funcionais. Nos dois casos podem ser implementados como componentes ou processos sequenciais (SOMMERVILLE, 2007).

2.3.1 Decomposição orientada a objetos

Neste modelo de decomposição o sistema é estruturado em um conjunto de objetos não firmemente unidos com interfaces bem definidas. Os objetos chamam as funções fornecidas por outros objetos. O relacionamento é feito por meio de classes de objetos, seus atributos e operações. A partir destas classes, os modelos são criados e por um certo modelo de controle é feito a coordenação dessas operações (SOMMERVILLE, 2007). A Figura 4 a seguir apresenta um modelo orientado a objetos.

Figura 4 - Modelo Orientado a Objetos.



Fonte: Silva, 2010, p.1

Devido os objetos basearem-se muito nas representações de entidades do mundo real, têm-se uma facilidade na interpretação da estrutura do sistema.

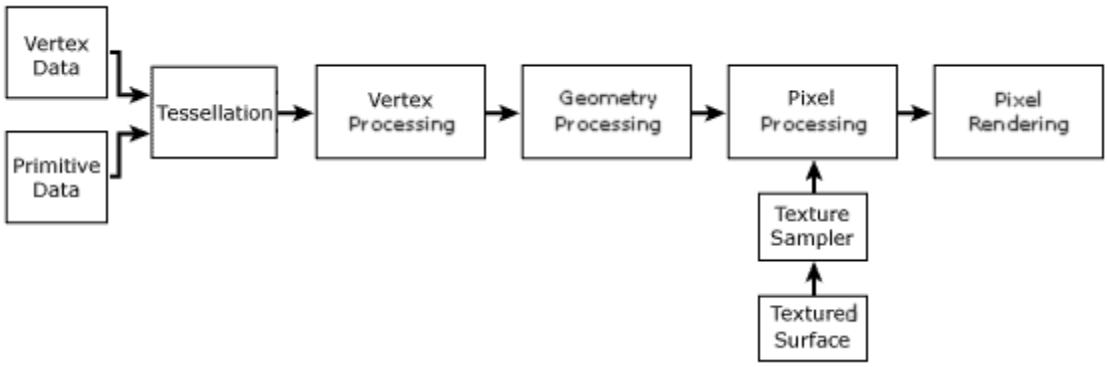
Por não haver uma ligação firme entre os objetos é possível fazer modificações na implementação dos objetos sem causar problemas para os demais objetos (SOMMERVILLE, 2007).

Porém, para que um objeto se comunique com outro é necessário a referência explícita do nome e interface do objeto em questão. Caso seja necessária alguma modificação na interface, essa mudança deve ser feita em todos os objetos que fazem uso da referência desta interface (SOMMERVILLE, 2007).

2.3.2 Pipelining orientado a funções

Nesta abordagem, também denominada de modelo de fluxo de dados, as modificações funcionais processam suas entradas e geram saídas. Os dados passam de uma função para outra e são modificados ao se moverem sequencialmente. Cada fase do processamento é implementada como uma modificação. Os dados de entrada passam por meio dessas modificações até serem transformados em saída. A Figura 5 a seguir apresenta um modelo de fluxo de dados (SOMMERVILLE, 2007).

Figura 5 - Modelo Pipelining



Fonte: Microsoft[1]

Uma desvantagem do uso deste modelo é a necessidade de manter um formato comum para a transferência dos dados para que seja reconhecido por todas as transformações. Esta abordagem é viável quando as transformações são reusáveis e independentes (SOMMERVILLE, 2007).

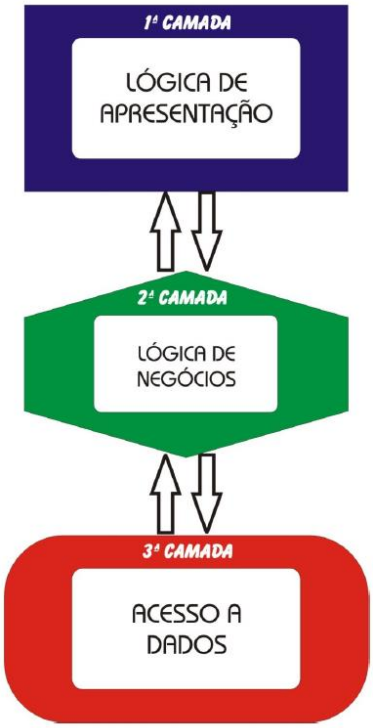
[1] https://msdn.microsoft.com/en-us/library/windows/desktop/bb219679(v=vs.85).aspx

2.4 Padrão de Arquitetura

Os padrões de arquitetura segundo Vasconcelos (BUSHMANN *et al*., 1996): “expressam um esquema de organização estrutural fundamental para sistemas de software, definindo subsistemas, especificando suas responsabilidades e incluindo regras e orientações para a organização dos relacionamentos entre os subsistemas”.

O padrão MVC divide o sistema em camadas com o objetivo de obter a eficiência, escalabilidade, reutilização e facilidade de manutenção da aplicação (VASCONCELOS, 2008). Na Figura 5 a seguir é apresentado o padrão MVC dividido em três camadas.

Figura 6 - Padrão MVC de três camadas.



Fonte: Santos, 2008, p. 24

De acordo com Santos (2008), o principal propósito deste padrão é separar a lógica de negócios (*Model*) da interface do usuário (*View*) e do fluxo da aplicação (*Controller*). Neste contexto, segundo Matos (2013),

- o *Model* é o objeto que representa os dados, manipula e controla as modificações
- a *View* faz a apresentação visual dos dados informados pelo *Model*.
- o *Controller* determina como os dados do *Model* serão modificados ou visualizado e em qual *View* será apresentado.

A *View* e o *Controller* trabalham juntos como parte da interface para aceitar a entrada do usuário e converter essa entrada para o formato correspondente do componente *Model*

(OTERO, 2012). O uso deste modelo mantém o código organizado, fazendo com que a manutenção seja simples e menos custosa (GORLA; FOSCHINI, 2013).

2.5 Padrões de Projetos

Segundo Gamma *et al*. (2000), os padrões facilitam a reutilização dos projetos e arquiteturas. Por serem técnicas com a eficiência já comprovada, tornam-se acessíveis para os novos desenvolvedores de sistemas, auxiliando na escolha de qual modelo de projeto utilizar para o desenvolvimento de um sistema reutilizável e a evitar escolhas que prejudiquem a reutilização.

Além disto, os padrões de projetos auxiliam na documentação e manutenção do sistema por fornecer detalhes do funcionamento das classes de objetos e objetivos subjacentes. Os padrões não se referem a projetos novos ou que ainda não foram testados, incluindo apenas projetos já aplicados várias vezes em diferentes sistemas, tornando-os mais confiáveis para se trabalhar (GAMMA et al, 2000, p. 19).

Os padrões de projeto descrevem o nome do padrão, o problema, a solução e as consequências do mesmo. O nome serve de referência para a solução de problemas e ajuda na criação de um vocabulário que facilite a comunicação com outros arquitetos e a documentação dos sistemas (BRIZENO, 2016).

De acordo com Alexander (GAMMA *et al*, 2000, p. 19) “cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca o fazer da mesma maneira”.

Existem padrões que se restringem a aplicações com problemas específicos de linguagem ou plataforma, como no caso do livro “Core J2EE Patterns”, que contêm soluções voltadas para aplicações da linguagem Java na plataforma Java 2 Enterprise Edition (BRIZENO, 2016).

2.5.1 Padrão Criacional

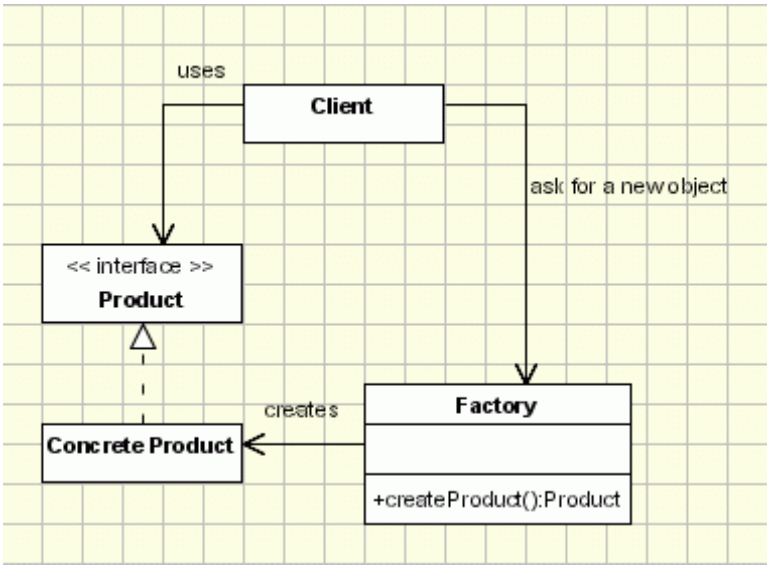
O padrão criacional abstrai o processo de instanciação, ajudando a tornar o sistema independente de como os objetos são criados, compostos e representados. Tendem a se tornar importantes a partir do momento que os sistemas evoluam na questão de depender mais do conjunto de objetos ao invés da herança de classes (GAMMA *et al.*, 2000).

2.5.1.1 Factory Pattern

Segundo Yener e Theedon (2015) “como um dos padrões de criação, o objetivo da *Factory* é criar objetos. A lógica criacional é encapsulada dentro da *Factory* e também fornece um método que retorna um objeto recém-criado (método *Factory Pattern*) ou delegados a criação do objeto para uma subclasse (padrão *Factory Abstract*) ”.

É comum utilizar o *pattern factory* quando se deseja fazer um sistema independente de como os objetos serão criados, compostos e representados (DATHAN; RAMNATH, 2015). Logo em seguida, a Figura 6 demonstra um exemplo de utilização.

Figura 7 - Exemplo de implementação do padrão Factory.



Disponível em: < http://www.oodesign.com/factory-pattern.html> Acesso em abr. 2016.

No exemplo da Figura 6, o *client* precisa de um produto, mas em vez de criá-lo diretamente utilizando o operador *new*, pede ao objeto *factory* um novo produto, fornecendo as informações sobre o tipo de objeto que ele precisa.

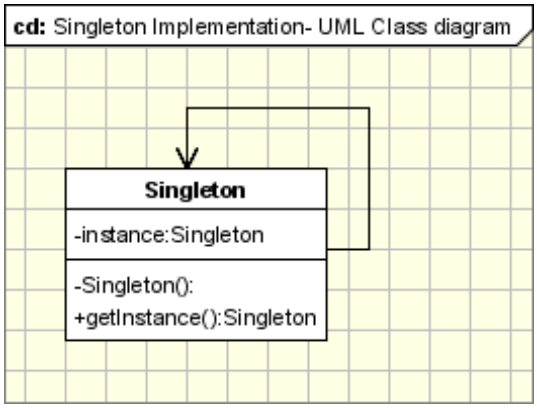
A *factory* instancia um novo produto concreto e, em seguida, retorna para o cliente o produto recém-criado. Então, o cliente utiliza os produtos como abstratos sem estar consciente sobre a sua implementação concreta.

2.5.1.2 Singleton Pattern

Segundo Santos (2008) “o padrão Singleton Pattern garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela ”. Por exemplo, a instanciação de um objeto de conexão de bancos de dados seria o suficiente para a manipulação de dados.

De acordo com Kuchana (2004), este padrão é útil por assegurar que haverá apenas uma instância de um objeto em particular, o mesmo propõe que os objetos do cliente devam ser capazes de acessar esta instância de modo consistente. Logo em seguida, a Figura 7 demonstra um exemplo de utilização.

Figura 8 - Exemplo de implementação do padrão Singleton.



Disponível em: < http://www.oodesign.com/factory-pattern.html> Acesso em abr. 2016.

A implementação envolve um membro estático na classe *Singleton*, um construtor privado e um método público estático que retorna uma referência para o membro estático. O padrão Singleton define uma operação *getInstance()* que expõe a instância única, que é acessado pelos clientes. *getInstance()* é responsável pela criação sua classe única instância no caso de ele ainda não está criado e retornar essa instância.

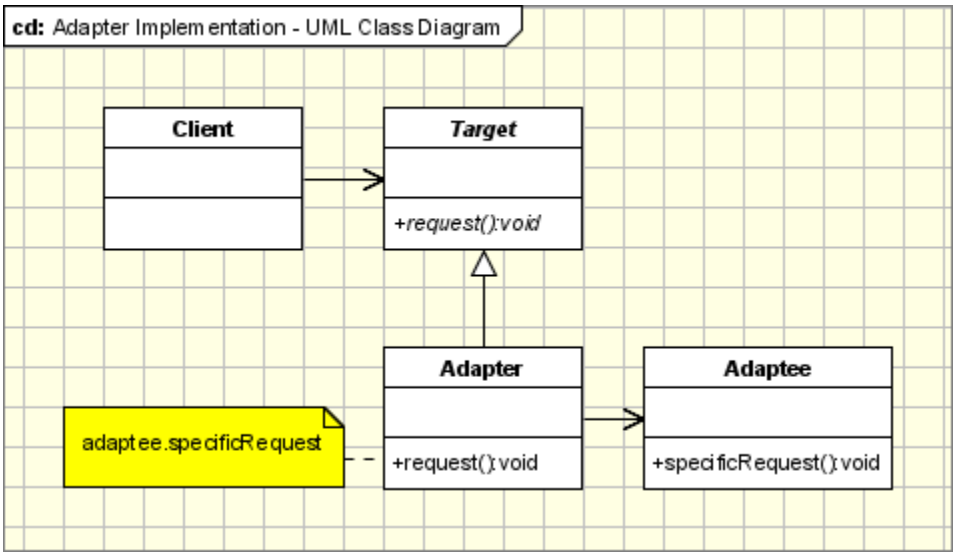
2.5.2 Padrão Estrutural

O padrão estrutural se importa como é a forma da composição das classes e objetos para se tornarem estruturas maiores (GAMMA et al., 2000).

2.5.2.1 Adapter

Converte a interface de uma classe para uma interface esperada pelo cliente. Permite que as interfaces incompatíveis funcionem em conjunto, sendo que de outra forma seria impossível (GAMMA *et al*., 2000). Logo em seguida, a Figura 8 demonstra um exemplo de utilização.

Figura 9 - Exemplo de implementação do padrão Adapter.



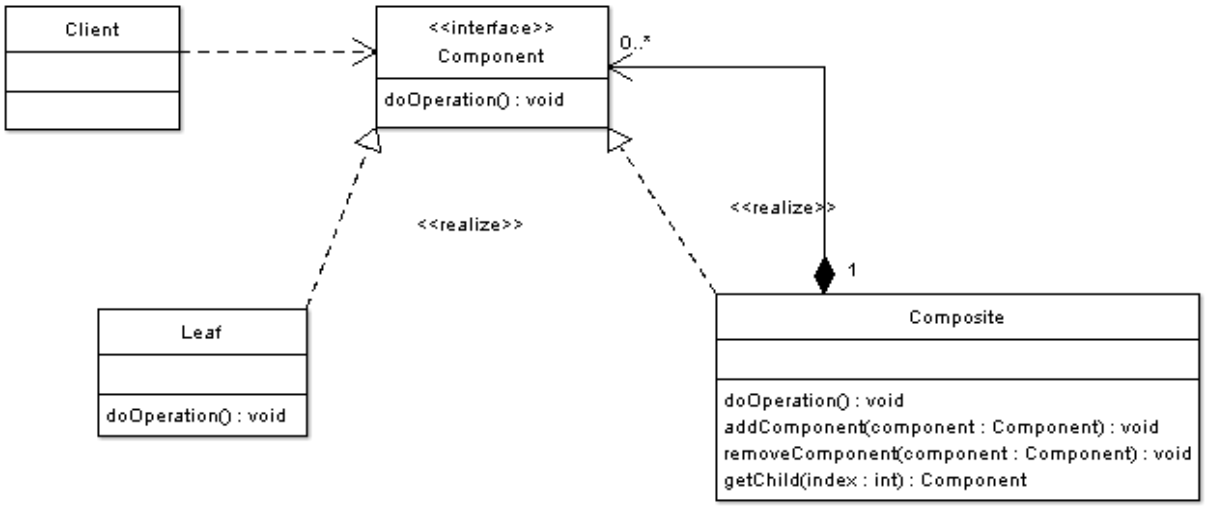
Disponível em: < http://www.oodesign.com/factory-pattern.html> Acesso em abr. 2016.

A classe (Target) chama os métodos definidos em uma interface e tem uma outra classe (Adapter) que não implementa a interface, mas implementa as operações que devem ser invocadas a partir da primeira classe através da interface. O adaptador irá implementar a interface e será a ponte entre as 2 classes.

2.5.2.2 Composite

Estrutura os objetos no formato de árvore para apresentar hierarquias partes-todo. O padrão Composite permite aos clientes tratarem de modo uniforme os objetos individuais e as composições de objetos (GAMMA *et al*., 2000). Logo em seguida, a Figura 9 demonstra um exemplo de utilização.

Figura 10 - Exemplo de implementação do padrão Composite.



Disponível em: < http://www.oodesign.com/factory-pattern.html> Acesso em abr. 2016.

Um exemplo com base em editores de ilustrações, é a simples forma de uma linha, que em uma forma complexa é um retângulo que é feito de quatro *objectos* de linha. O padrão Composite pode ser usado para permitir que o programa lide com todas as formas de maneira uniforme.

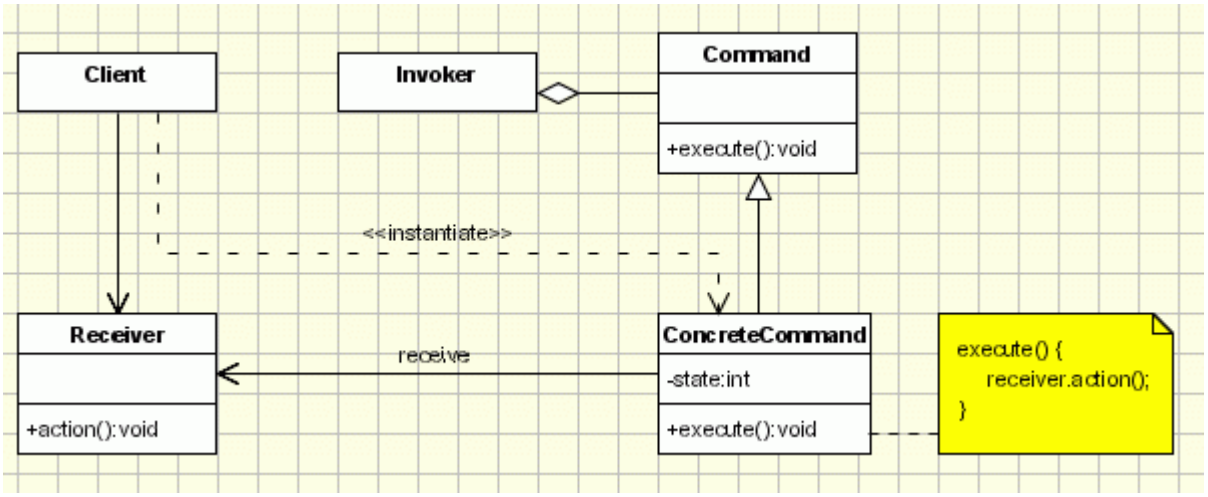
2.5.3 Padrão Comportamental

O padrão comportamental se importa com os algoritmos e a atribuição de comprometimento entre os objetos. Não apenas descreve padrões de objetos ou classes, mas também padrões de comunicação entre eles. Os padrões comportamentais de classes fazem uso da herança para dividir o comportamento entre classes. Já os padrões de objetos fazem uso da formação dos objetos em vez de herança (GAMMA *et al.*, 2000).

2.5.3.1 Command

Encapsula uma solicitação como um objeto, sendo assim, permite parametrizar clientes com distintas requisições, alinhar ou fazer o registro (*log*) das requisições e tolerar operações que possam ser desfeitas (GAMMA *et al.*, 2000). Logo em seguida, a Figura 10 demonstra um exemplo de utilização.

Figura 11 - Exemplo de implementação do padrão *Command*.



Disponível em: < http://www.oodesign.com/factory-pattern.html> Acesso em abr. 2016.

As classes que participam no padrão são:

- *Command*: declara uma interface para a execução de uma operação;
- *ConcreteCommand*: estende a interface de comandos, implementando o método de execução, invocando as operações correspondentes no receptor. Ele define uma ligação entre o receptor e a ação.
- *Client*: cria um objeto *ConcreteCommand* e define seu receptor;

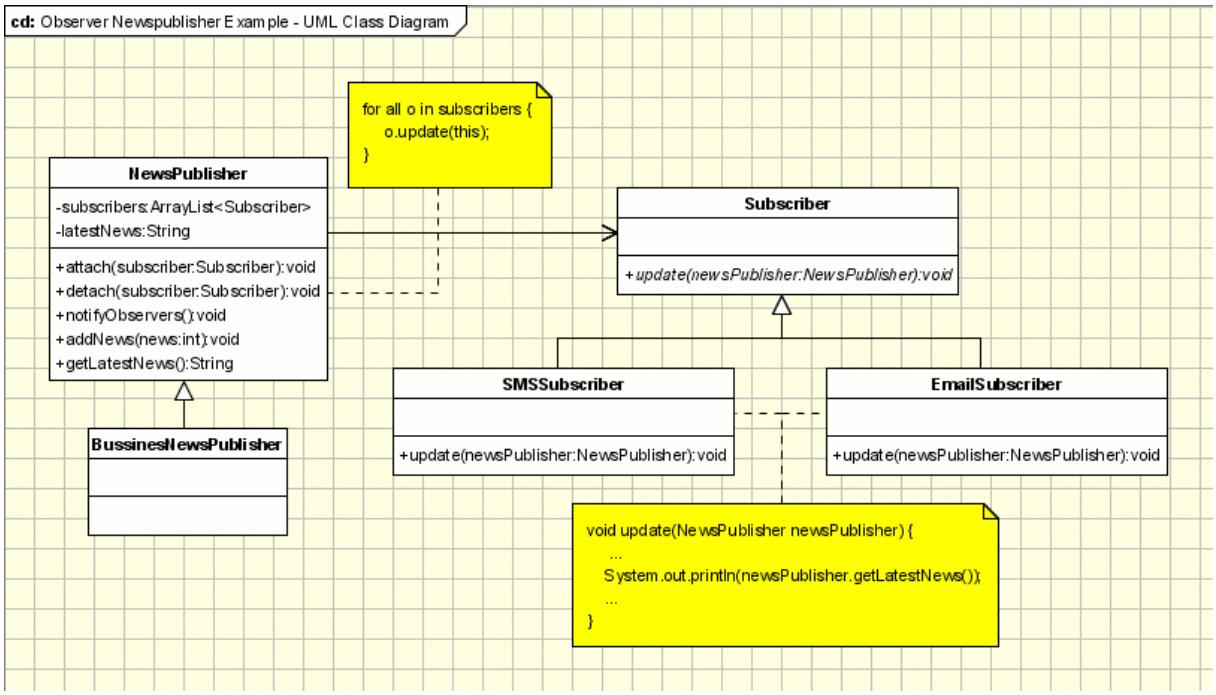
- *Invoker*: pede o comando para realizar o pedido;
- *Receiver*: sabe como executar as operações;

O cliente solicita que um *command* seja executado. O *Invoker* toma o comando, encapsula-o e coloca-o em uma fila, no caso de haver alguma outra coisa para fazer em primeiro lugar, o *ConcreteCommand* que está a cargo do *command* solicitado, envia seu resultado para o *receiver*.

2.5.3.2 Observer

Define uma dependência de um-para-muitos entre os objetos, de modo que quando um objeto altera seu estado os seus dependentes são informados e atualizados automaticamente (GAMA *et al*., 2000).

Figura 12 - Exemplo de implementação do padrão Observer.



Disponível em: < http://www.oodesign.com/factory-pattern.html> Acesso em abr. 2016.

A lógica do *Observer* é implementada em *NewsPublisher*. Ela mantém uma lista de todos os assinantes de TI e informa-los sobre as últimas notícias. Os assinantes são representados por alguns observadores (*SMSSubscriber*, *EmailSubscriber*). Ambos os observadores mencionados acima são herdados do Assinante. O assinante é a classe abstrata que é conhecido para a editora. A editora não sabe sobre observadores concretos, ela só sabe sobre a sua abstração.

Na próxima seção são apresentados os procedimentos metodológicos a serem utilizados no decorrer da pesquisa.

3 METODOLOGIA

3.1 Desenho de Estudo

O objetivo do trabalho foi definir uma nova arquitetura para o FisioKinect e um *layout* que seja adaptável aos softwares desenvolvidos pelo Grupo de Estudos e Pesquisa em Tecnologia, Saúde e Qualidade de Vida (GEPETS). A definição de qual arquitetura seria a mais adequada para o problema proposto se deu por meio de uma fundamentação teórica, adquirida através de pesquisas bibliográficas, livros, artigos científicos etc. Para validar a arquitetura escolhida foi feito a reorganização dos softwares já existentes.

3.2 Local

O trabalho foi desenvolvido no Laboratório de Tecnologia em Saúde (LTS) com auxílio do Laboratório de Multimídia e Computação Gráfica (LABMIDIA) e do Laboratório de Banco de Dados e Engenharia de Software (LBDES), todos localizados no Complexo de Informática do Centro Universitário Luterano de Palmas.

3.3 Hardware

Em termos de *hardware* foram utilizados um computador de uso pessoal, um Kinect acompanhando de um adaptador - responsável pela conexão entre Kinect e computador - disponibilizado pelo GEPETS.

3.4 Linguagem de Programação e Software

Abaixo são listados a linguagem de programação e os *softwares* a serem utilizados no desenvolvimento da aplicação que verificará a eficácia da nova arquitetura:

- *C# (CSharp)* - é a linguagem de programação desenvolvida pela Microsoft para a plataforma .NET, sendo a principal linguagem da mesma. Possui tipagem dinâmica e estática, além de ser orientada a objetos;
- *Microsoft Windows 10*: atualmente a última versão do sistema operacional da Microsoft;
- *Microsoft SDK 1.8*: é o kit de desenvolvimento da Microsoft para aplicações do Kinect;
- *Visual Studio 2013*: é a IDE oficial da Microsoft para desenvolvimento de softwares na plataforma .NET. Também é usada no desenvolvimento de aplicações na plataforma ASP.NET.

3.5 Desenvolvimento da arquitetura

Inicialmente, a arquitetura da aplicação atual foi analisada a fim de decidir se este trabalho iria utilizar a estrutura atual ou seria desenvolvida uma nova para organizar os arquivos. Como resultado da análise, foi concluído a necessidade de uma nova arquitetura, visto que a maior parte dos arquivos, mesmo com objetivos distintos, estavam misturados e sem uma organização que facilitasse a sua localização.

Assim, foi criado o modelo da nova arquitetura baseado no conceito de camadas, para organizar os arquivos com funções similares. A aplicação teve os arquivos distribuídos em quatro camadas, nomeadas de *solution* FisioKinect, *solution* Data, *solution* Manager e *solution* Util; sendo estas as responsáveis por manter a organização dos arquivos.

Logo depois, houve a escolha de uma abordagem para decompor os subsistemas em módulos. Em razão do modelo da arquitetura desenvolvido, foi selecionada a abordagem de decomposição de objetos, por motivo de que a *solution* FisioKinect faria uso das classes contidas na *solution* Manager por meio de objetos para a manipulação dos dados.

Em seguida, foi definido o padrão de projetos mais adequado para usar na nova arquitetura. Sabendo que a *solution* FisioKinect é dependente da *solution* Manager para fazer uso das classes por meio de objetos e que a aplicação tende a conter mais *softwares*, decidiu-se aplicar o padrão de projeto criacional. Esse padrão de projeto deve ajudar a tornar a arquitetura independente do modo em que os objetos são criados, gerados, compostos e representados na plataforma FisioKinect.

Com relação à tela inicial, o acesso aos softwares passou a ser feito por meio de um ComboBox (Caixa de combinação). Dessa forma, a implantação de novos softwares não afetaria o *layout*, já que não seria mais necessário a adição de botões. Na tela de edição e visualização dos dados dos pacientes, as abas referentes às avaliações foram removidas, o acesso às avaliações segue a mesma ideia utilizada na tela inicial.

3.6 Testes da Arquitetura e *Layout*

Uma vez concluído o processo de desenvolvimento da arquitetura e *layout*, foram realizados os testes para verificar o comportamento dos mesmos. Nesses testes realizou-se a implantação dos *softwares* já existentes, a fim de verificar o funcionamento de modo que qualquer anormalidade identificada fosse revista.

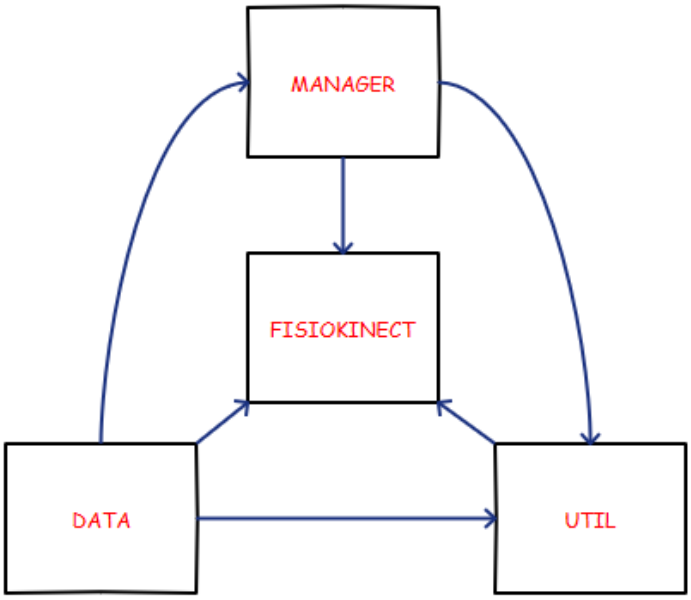
4 RESULTADOS E DISCUSSÃO

A presente seção descreve as conclusões do desenvolvimento do presente trabalho. Será apresentado a antiga e nova arquitetura e *layout*, bem como o funcionamento dos mesmos.

4.1 Modelo da Arquitetura

Para compreender a divisão dos projetos e banco de dados, logo abaixo, na Figura 13, é apresentado a arquitetura desenvolvida.

Figura 13 – Arquitetura desenvolvida



A arquitetura desenvolvida, figura 13, foi baseada no conceito de camadas apresentado por Garlan e Shaw (1994). Com base nesse conceito a aplicação é descentralizada em *solutions* (camadas) para melhor organizar os arquivos. Abaixo é explicado detalhadamente o papel de cada *solution*:

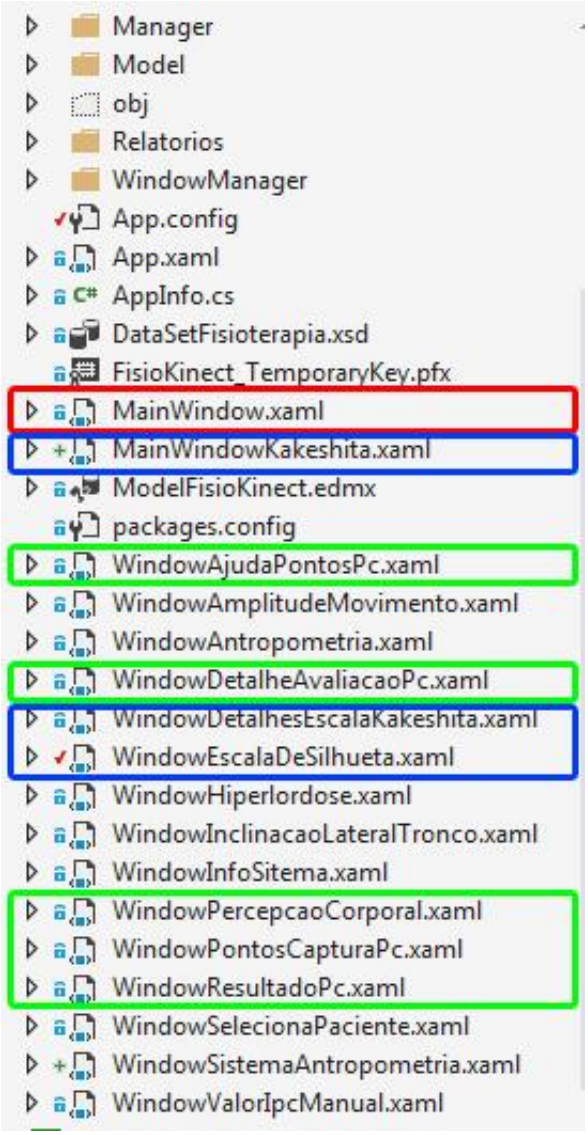
- **FisioKinect**: é a *solution* principal, responsável apenas por gerenciar os pacientes e ser a porta de entrada para os demais softwares. Nenhuma outra responsabilidade é atribuída a ele.
- **Data**: responsável por armazenar todo o esquema do banco de dados, ou seja, a parte de persistência dos dados;
- **Manager**: é o repositório responsável por manipular os dados. Uma outra alternativa seria utilizar um manager para cada *software* integrado na sua respectiva *solution*, evitando assim qualquer alteração desnecessária nas classes de outros sistemas;
- **Util**: Repositório de imagens.

As *solutions* apresentadas são indispensáveis para o funcionamento da plataforma, pois dependem umas das outras. Esses *solutions* são responsáveis pela organização dos componentes na plataforma.

4.2 A arquitetura

Para validar a arquitetura desenvolvida, foram adicionados e organizados os *softwares* atuais, banco de dados e demais conteúdos nas *solutions* correspondentes. Contudo, inicialmente será apresentado, na figura 14, como era feita a organização destes arquivos.

Figura 14 - Antiga Solution da plataforma FisioKinect

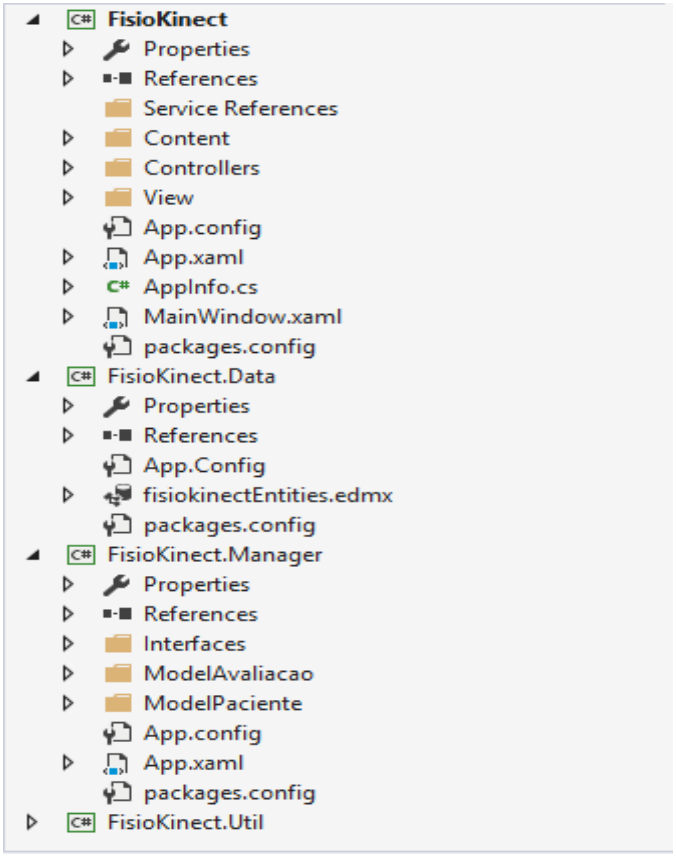


Fonte: Sousa, 2016

Na Figura 14 é representado a parte referente aos módulos dos *softwares* na *solution*. Cada cor de borda representa um módulo de um *software*. Sendo assim, no decorrer do tempo os desenvolvedores teriam problemas para implantação de novos *softwares* devido a forma

em que os atuais estão organizados e consequentemente haveria o risco de afetar algum *software* já implantado. Logo em seguida, na Figura 15, é apresentado a nova forma de organizá-los, já com os arquivos dos diversos projetos distribuídos nas camadas da arquitetura proposta.

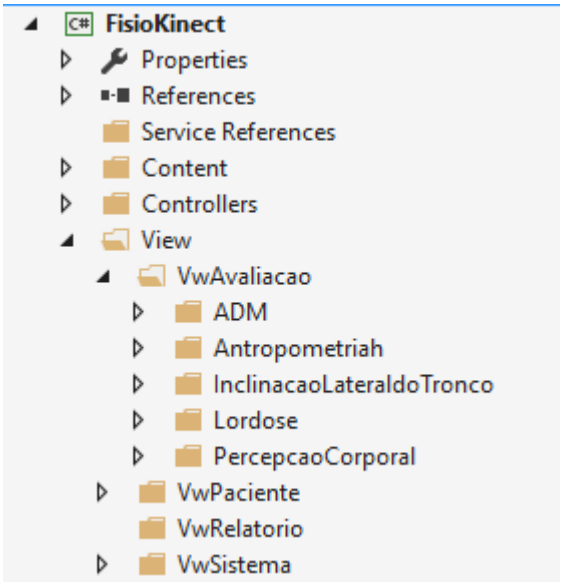
Figura 15 - Nova Solution da plataforma FisioKinect



A divisão dos arquivos em *solutions* facilitou novas implantações, visto que cada módulo é alocado em uma pasta específica, correspondente a função do mesmo. Assim reduz o risco de os desenvolvedores modificarem outros arquivos por engano e consigam manter os mesmos organizados.

A seguir, é demonstrado como os módulos dos softwares são organizados no momento da implantação. Na figura 16 é apresentado a *solution* FisioKinect, onde estão localizados os módulos referentes às telas dos *softwares* na pasta "*View*", as classes utilizadas pelo Kinect na pasta "*Controllers*" e a pasta "*Content*", foi mantida com as imagens utilizadas pelos atuais *softwares*.

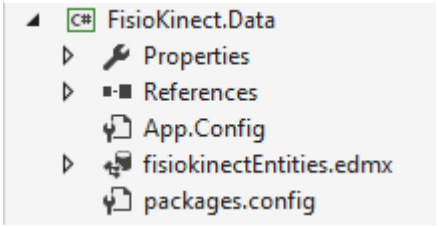
Figura 16 - Solution FisioKinect



Note que a pasta "*View*" é composta por 4 subpastas, sendo estas: a subpasta VwAvaliacao constituída por outros subpastas, as quais são tituladas com o nome do *software* e a mesmas armazenam os módulos referentes às telas deste *software*; A VwPaciente contém os módulos referentes ás telas dos pacientes; O VwRelatorio possui os módulos referentes às telas dos relatórios; O VwSistema tem os módulos referentes a tela de informações e a tela inicial da aplicação.

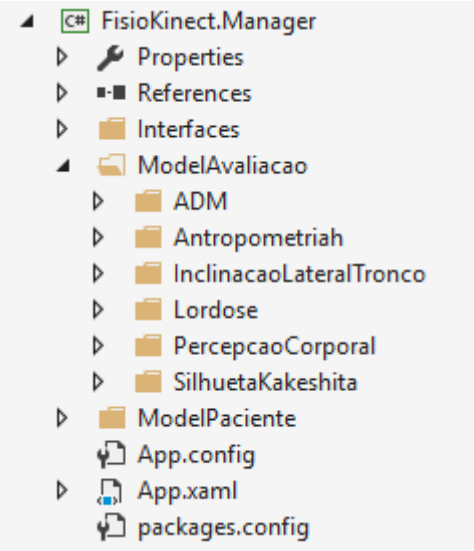
Em seguida, na figura 17, a *solution* Data contém apenas o banco de dados da aplicação denominado de *fisiokinectEntities*.

Figura 17 - Solution Data



Nesta *solution* nada mais deve ser adicionado, apenas o arquivo do banco de dados deve estar contido nela. Já na figura 18, a *solution* Manager é composta pelas classes responsáveis pela manipulação dos dados, seja para obtê-los ou salvá-los no banco de dados.

Figura 18 - Solution Manager

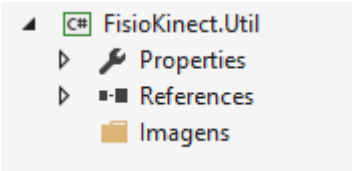


As classes estão incluídas em pastas tituladas com o nome do *software*, precedidas do prefixo “Model”. Dessa forma, os arquivos pertencentes a pasta cujo o prefixo seja “Model”, serão apenas as classes contidas na *solution* Manager.

Para fazer a comunicação da *solution* FisioKinect com as *solutions* Manager e Data é utilizado a ideia da decomposição de objetos conforme Sommerville (2007), criando um objeto das classes e do banco de dados para a manipulação das informações. Utilizando o conceito do padrão criacional *singleton pattern* segundo Santos (2008), estes objetos são criados globalmente para não ser preciso repetir a criação dos mesmos.

E mais adiante, na figura 19, tem-se a *solution* Util, local onde se armazena os arquivos de imagens.

Figura 19 - Solution Util



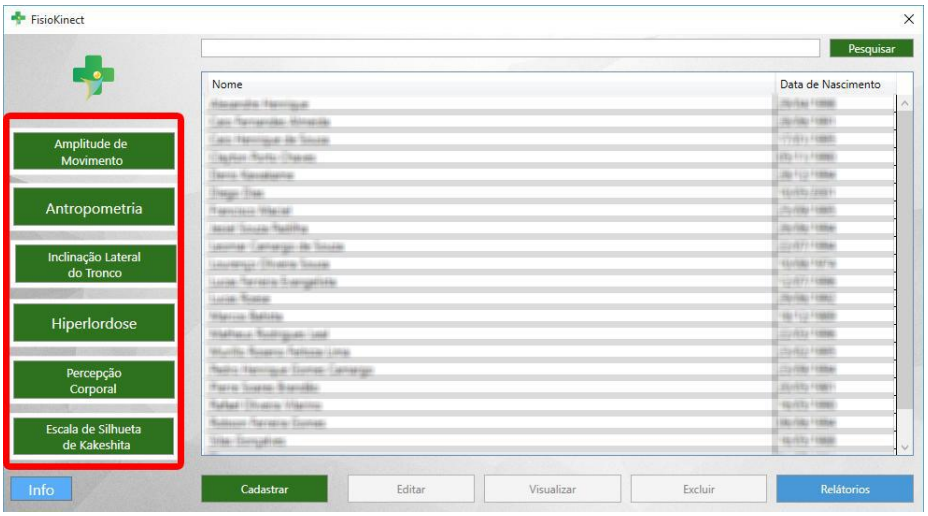
Até o momento não há arquivos guardados nesta *solution*. Possivelmente, a partir da implantação de novos *softwares* é que ela começará a ser preenchida. Na próxima seção serão descritas as alterações realizadas no *layout* do FisioKinect.

4.3 *Layout*

Em relação ao *layout*, o maior problema estava na tela inicial, onde, a cada inserção de um novo *software* era necessário mais espaço no *menu* para adicionar os botões de acesso aos *softwares*.

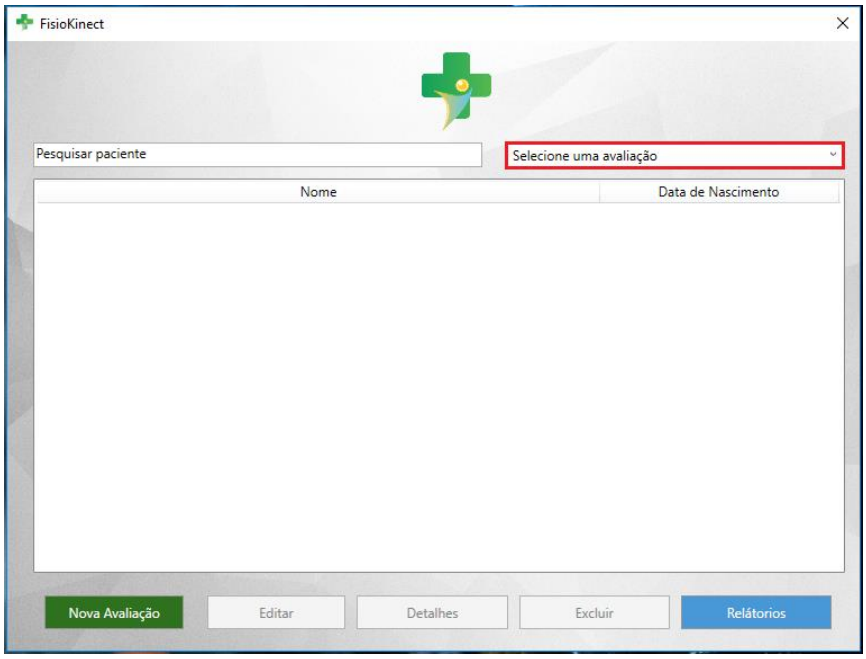
Na Figura 20, é apresentado a tela inicial do FisioKinect atual, onde o acesso a cada *software* é feito pelos botões à esquerda.

Figura 20 – Antiga tela inicial do FisioKinect



Como nota-se na figura acima (Figura 20), à medida que novos softwares eram implantados, mais difícil se tornava a inserção de outro botão de acesso sem que o *layout* fosse prejudicado. Na figura seguinte é mostrada a nova tela inicial da plataforma.

Figura 21 - Nova tela inicial do FisioKinect



Na tela inicial, foi alterado o modo de proceder com o cadastro das avaliações por meio do Kinect e manualmente. Para realizar o cadastro manualmente de uma avaliação, primeiro é preciso selecionar um paciente, em seguida, clicar no botão denominado "Nova Avaliação", o mesmo utilizado para cadastrar um paciente. Já o cadastro por meio do Kinect, após selecionar um paciente, clicar na caixa denominada de "Selecione uma avaliação" e selecionar uma avaliação.

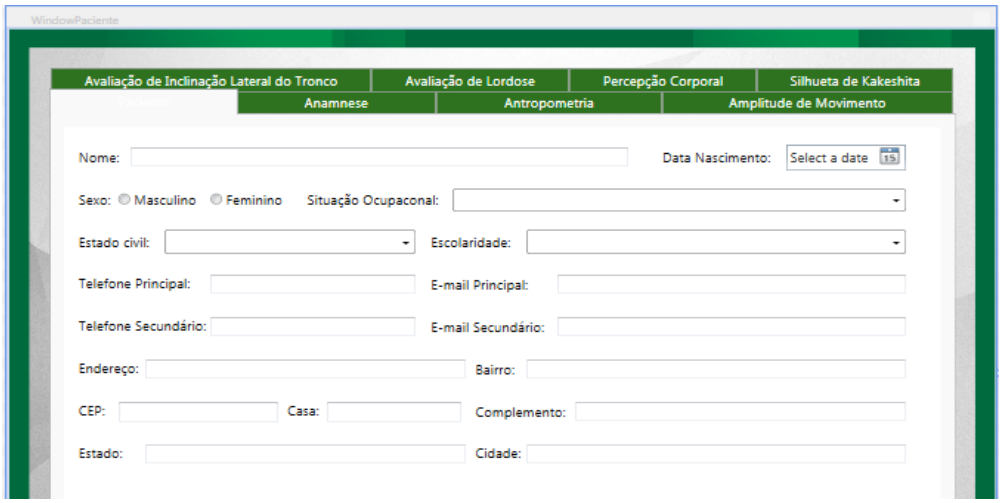
Para retornar à função padrão do botão de cadastro do paciente, basta clicar com o lado direito do mouse em qualquer local da tela da aplicação.

Outra alteração realizada foi quanto à disposição dos botões de acesso aos *softwares* que antes se encontravam à esquerda, na nova tela inicial. Esses botões foram removidos e o acesso passou a ser feito a partir de um *ComboBox* (Caixa de Combinação). Assim evita o acumulo de botões na tela.

A alteração do acesso aos *softwares* evita a possibilidade de o *layout* ser prejudicado com a implantação de outro *software*, além da tela inicial ter ficado com um visual mais simples, sem o acúmulo de botões. Os demais botões foram mantidos por não haver a necessidade de muda-los de local.

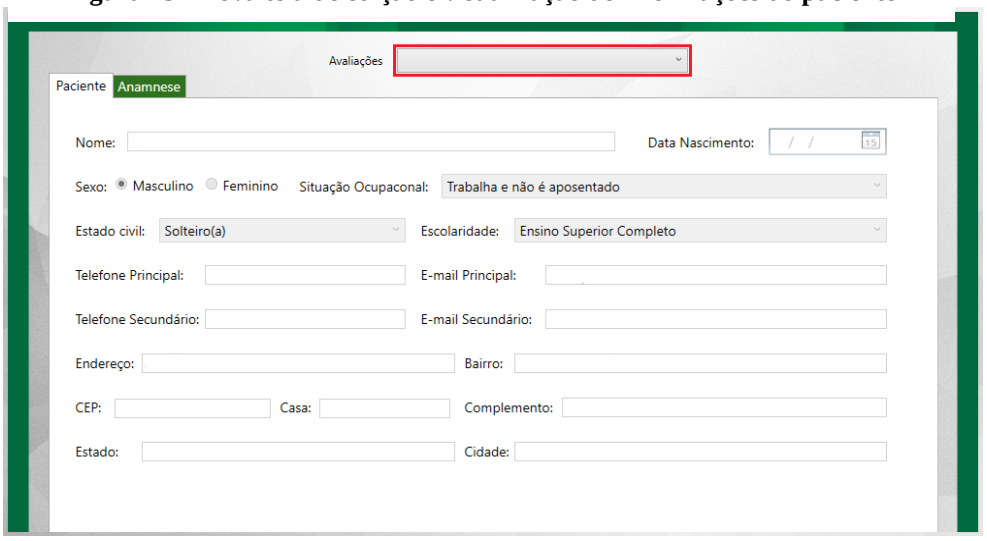
Além do problema mencionado na Figura 20, outro seria na tela de visualização e edição das informações do paciente, onde, a cada software implantado, era necessário a inserção de uma nova aba para carregar as avaliações daquele paciente. A Figura 22 mostra essa tela atualmente.

Figura 22 - Antiga tela de edição e visualização de informações do paciente



Até o momento esta tela satisfazia por não haver muitas abas, mas com a inserção de novos projetos o layout seria prejudicado e consequentemente o usuário teria dificuldades no manuseio. Em seguida, na Figura 23, a nova tela para visualizar e editar as informações dos pacientes.

Figura 23 - Nova tela de edição e visualização de informações do paciente



Como feito na Figura 21, a nova tela para a edição e visualização teve as abas de avaliações removidas e o acesso passou a ser feito por meio de um *ComboBox* (Caixa de Combinação).

Na seção seguinte será apresentado a forma em que os módulos devem organizados na nova arquitetura.

5 COMO ORGANIZAR OS MÓDULOS DOS SOFTWARES

No momento da implantação dos *softwares* na nova arquitetura, os módulos devem ser organizados conforme a nova orientação. A seguir, é explicado os procedimentos adequados para cada módulo e de como deve ser feito o armazenamento dos arquivos.

- Módulo de Telas: na *solution* FisioKinect, as telas dos *softwares* devem ser armazenadas na pasta View, dentro da subpasta correspondente ao que a tela se refere. Por exemplo: telas referentes as avaliações são armazenadas na pasta VwAvaliacao.

- Módulo do Banco de Dados: a base de dados deve ser armazenada dentro da *solution* Data.

- Módulo de Classes: as classes devem ser armazenadas nas pastas correspondentes da *solution* Manager. Por exemplo: classes que manipulam informações referentes as avaliações são armazenadas em subpastas com o nome da avaliação, dentro da pasta ModelAvaliacao.

- Módulo de Imagens: as imagens utilizadas para compor as telas dos *softwares* devem ser armazenadas dentro da *solution* Util.

Seguindo estes procedimentos a nova arquitetura deve funcionar conforme o novo padrão.

6 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo o desenvolvimento de uma arquitetura para a organização dos módulos dos *softwares* e a criação de um *layout* para acomodar novas implementações. Para verificar a viabilidade da arquitetura, os projetos existentes foram adicionados e organizados nas *solutions* conforme os módulos. O *layout* da tela inicial teve alterações na disposição do acesso aos *softwares* utilizados pelo Kinect, agora o acesso é feito por uma *ComboBox* (Caixa de Combinação) denominada “Selecione uma avaliação”. Já o *layout* das telas de edição e visualização de dados, onde comportava várias abas, cada aba se tornou uma tela.

A arquitetura desenvolvida ajuda a manter os módulos organizados, de modo que, os arquivos de outros projetos dificilmente serão modificados por engano no momento da implantação de um software, pois cada arquivo está armazenado em pastas com nomenclaturas
que distingue suas funções. Com relação aos *layouts*, os novos modelos não correm o risco de sofrer alterações por conta do excesso de componentes em sua estrutura.

Dessa forma, conclui-se que com a nova arquitetura os desenvolvedores terão mais facilidade para implementarem ou modificarem os *softwares*. Sobre o *layout*, independentemente da quantidade de novos *softwares* que venham a ser implementados, a sua estrutura não será atingida.

6 REFERÊNCIAS

BRIZENO, Marcos. **Primeiros passos com padrões de projeto.** [s. I.]: Leanpub, 2016.

BASS, L., CLEMENTS, P., KAZMAN, R., **Software Architecture in Practice, Second edition**, Addison Wesley, 2003.

BRANDÃO, Pierre S.; ROCHA, Pollyeverlin R.; DEFAVARI, Alex H. **Estudo da viabilidade da utilização do Kinect como ferramenta no atendimento fisioterapêutico de pacientes neurológicos**. 2012. Disponível em: <http://sbgames.org/sbgames2012/proceedings/papers/gamesforchange/g4c-04.pdf>. Acesso em: 11 abr. 2016.

CAMARGO, Thayso Henrique Capuchinho. **Avaliação da amplitude de movimento (ADM) em plano coronal anterior utilizando o sensor Kinect:** articulações do ombro e do quadril. 2015. 80 f. TCC (Graduação) - Curso de Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas, 2015. Cap. 42.

DATHAN, Brahma; RAMNATH, Sarnath. **Object-Oriented Analysis, Design and Implementation.** 2. ed. [s. I.]: Springer, 2015.

DOMINGOS, Marlon David. **Avaliação postural computadorizada utilizando o Kinect.** 2014. 73 f. TCC (Graduação) - Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas, 2014. Cap. 38.

GAMMA, Erich et al. **Padrões de Projeto**: Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

GARLAN, D.; SHAW, M. **An Introduction to Software Architecture**. Pittsburgh, PA, USA, 1994.

GORLA, João Paulo Ferro; FOSCHINI, Ivan João. **Arquitetura para Desenvolvimento Web Baseado em JSF 2.0 utilizando Padrões de Projeto**. São Carlos. v. 2, n. 3, p. 230-241, set/dez 2013.

JESUS, Vinicius Fernandes de. **Arquitetura de Software**: Uma proposta para a primeira aplicação. 2013. 46 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Estadual de Londrina, Londrina, 2013. Cap. 15.

LIMA, Stephanie Tamarozzi. **A confiabilidade e consistencia de medidas antropométricas através do sensor Microsoft Kinect Palmas – TO.** 2015. 54 f. TCC (Graduação) - Curso de Serviço Social, Centro Universitário Luterano de Palmas, Palmas, 2015. Cap. 16.

MATOS, Fernando Ferreira. **Desenvolvimento de um sistema web para enviar, armazenar e receber documentos oficiais.** 2013. 81 f. TCC (Graduação) - Curso de Sistemas de Informação, Faculdade de Balsas, Balsas, 2013. Cap. 25.

MENDES, Antônio. **Arquitetura de Software**: desenvolvimento orientado para arquitetura. Rio de Janeiro: Campus, 2002.

PETERS, James F.; PEDRYCZ, Witold. **Engenharia de Software:** Teoria e Prática. Rio de Janeiro: Campus, 2001.

PRESSMAN, R. S. **Engenharia de Software**: uma abordagem profissional. 7. ed. Porto Alegre: Bookman, 2011.

SANTOS, Gustavo P. **APLICAÇÃO DO PADRÃO DE PROJETO MVC COM JSF.** 2008. 56 f. Monografia (Especialização) - Curso de Ciência da Computação, Faculdade de Jaguariúna, Jaguariúna, 2008. Cap. 25.

SOMMERVILLE, Ian. **Engenharia de Software.** 8. ed. São Paulo: Pearson Addison Wesley, 2007.

SOMMERVILLE, Ian. **Software Engineering.** 9. ed. [s. I.]: Pearson, 2011.

YENER, Murat; THEEDOM, Alex. **Java EE Design Patterns.** Indianapolis: Wrox, 2015.