UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JORDANO RIBEIRO CELESTRINI

NUIMOD: UM AMBIENTE PARA MODELAGEM DE POSES E GESTOS

VITÓRIA
2014

JORDANO RIBEIRO CELESTRINI

NUIMOD: UM AMBIENTE PARA MODELAGEM DE POSES E GESTOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.
Orientador: Prof. Dr. Crediné Silva de Menezes.

VITÓRIA
2014

Jordano Ribeiro Celestrini

NuiMod - Um ambiente para modelagem de poses e gestos/ Jordano Ribeiro Celestrini. – Brasil, 2014-

109 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Crediné Silva de Menezes

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2014.

1. Interação Humano-Computador. 2. Interfaces Naturais de Usuário. 3. Sensores de Movimento. 4. Kinect. I. Crediné Silva de Menezes. II. Universidade Federal do Espírito Santo. III. Centro Tecnológico. IV. NuiMod - Um ambiente para modelagem de poses e gestos

CDU 00:000:000.0

JORDANO RIBEIRO CELESTRINI

NUIMOD: UM AMBIENTE PARA MODELAGEM DE POSES E GESTOS

Dissertação submetida ao programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do Grau de Mestre em Informática.

Aprovada em _____ de _____ de 2014.

COMISSÃO EXAMINADORA

Prof. Dr. Crediné Silva de Menezes
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Orivaldo de Lira Tavares
Universidade Federal do Espírito Santo

Prof. Dr. Celso Alberto Saibel Santos
Universidade Federal do Espírito Santo

Prof. Dr. Alberto Nogueira de Castro Júnior
Universidade Federal do Amazonas

Este trabalho é dedicado à minha família.

Agradecimentos

Ao professor Crediné, que acreditou no potencial desta pesquisa. Sem o seu apoio, este trabalho não seria possível.

À minha esposa Alessandra, pelo carinho e compreensão nos momentos mais difíceis desta jornada.

Aos meus pais, Luiz e Néia, por sempre fomentarem a busca por novos conhecimentos. Obrigado pelo amor incondicional.

Aos professores Marcelo Schimidt e Rodrigo Varejão pelas orientações e pelo apoio para iniciar esta etapa de minha formação.

Aos colegas de laboratório, pelo convívio e aprendizado. Ao amigo Roberto Morati, pelas discussões e colaboração nos testes realizados.

À banca examinadora desta pesquisa, por estar presente e contribuir para a minha formação e desenvolvimento de um grande projeto.

"A persistência é o caminho do êxito." (Chaplin)

Resumo

A utilização de sensores de movimento para aplicações diversas é uma prática crescente, potencializada pelo surgimento de novos dispositivos como o Kinect, inicialmente criado para a interação entre jogadores e jogos digitais sem a necessidade de controles atrelados ao corpo.

Atualmente, aplicações voltadas para educação, fisioterapia e medicina fazem uso de sua tecnologia para capturar gestos e poses dos usuários. Contudo, no contexto de desenvolvimento de soluções, o suporte à modelagem e utilização de Interfaces Naturais do Usuário baseadas em gestos ainda é uma lacuna a ser preenchida.

Este dissertação apresenta o NuiMod, um ambiente para modelagem de poses e gestos cujo objetivo é permitir a criação e utilização de modelos gestuais, bem como proporcionar o uso de Interfaces Naturais de Usuário baseadas em gestos no ambiente web.

Palavras-chaves: interação humano-computador. interfaces naturais de usuário. sensores de movimento. kinect.

Abstract

The use of motion sensors for various applications is a growing practice, boosted by the emergence of new devices like Kinect, originally created for interaction between players and digital games without a control.

Nowdays, applications related to education, physiotherapy and medicine make use of its technology. However, in the context of developing solutions, support for modeling using natural gestures and interfaces in a web environment is still a gap to be filled.

This work presents NuiMod, an environment for modeling poses and gestures whose goal is to facilitate the creation and use of sign models, as well providing the use of Natural User Interfaces in a web environment.

Keywords: human–computer interaction. natural user interface. motion sensor. kinect.

Lista de ilustrações

Figura 1 – Processo de interação humano-computador. 18
Figura 2 – Medição de profundidade por triangulação e luz estruturada. 25
Figura 3 – Medição do tempo de vôo. 25
Figura 4 – Estrutura básica do sensor Kinect 27
Figura 5 – Imagens dos dispositivos Carmine e Capri da Primesense. 29
Figura 6 – Visualização de um esquema real (a) e visualização esquemática (b) do *Leap Motion*. 31
Figura 7 – *Structure Sensor*, desenvolvido pela Occipital. 31
Figura 8 – Diagrama esquemático e foto do dispositivo *DepthSense 325*. 33
Figura 9 – API do OpenNI 35
Figura 10 – Arquitetura do Kinect SDK for Windows. 36
Figura 11 – Tela principal do Kinoogle 41
Figura 12 – Interface do jogo XDigit 42
Figura 13 – Poses para executar ações no XDigit 43
Figura 14 – Arquitetura FAAST. 45
Figura 15 – Regra criada com o FAAST 46
Figura 16 – Arquitetura do Xkin e detecção do contorno das mãos. 49
Figura 17 – Imagem do KinectSocketServer. 50
Figura 18 – Demo do Kinect.js. 51
Figura 19 – Arquitetura do Kinected Browser. 51
Figura 20 – Arquitetura do XDKinect. 53
Figura 21 – Arquitetura do NuiMod 56
Figura 22 – Representação do modelo NMD 57
Figura 23 – Modelo de esqueleto com 20 articulações adotado pelo NuiMod 57
Figura 24 – Tipos de comandos e respostas do NuiMod Connector 58
Figura 25 – Arquitetura que representa os elementos do *NuiMod Engine* 59
Figura 26 – Problema da distância euclidiana em sujeitos de tamanhos diferentes . 62

Figura 27 – Seguimentos de reta do esqueleto para cálculo do ângulo 62
Figura 28 – Triplas utilizadas para cálculo de similaridade entre esqueletos 63
Figura 29 – Exemplo de gesto no NuiMod. 64
Figura 30 – Exemplo parcial de projeto no formato NMD 65
Figura 31 – Tela principal e tela de configuração do *NuiMod Connector* 66
Figura 32 – Exemplo de mensagem para envio de comando ao NuiMod Connector . 66
Figura 33 – Inteface para comparação de poses 68
Figura 34 – Arquitetura do *NuiMod Editor* 69
Figura 35 – Tela de login e registro do *NuiMod Editor* 70
Figura 36 – Tela de edição de projetos no *NuiMod Editor* 72
Figura 37 – Painel de controle do *NuiMod Connector* 73
Figura 38 – Exemplo de projetos exportados pela API do *NuiMod Editor* 74
Figura 39 – Fluxo de autenticação com OAuth 2.0 75
Figura 40 – Etapa de teste realizada durante os experimentos com o NuiMod . . . 79
Figura 41 – Página inicial do *impress.js Editor* 81
Figura 42 – Tela de login do *impress.js Editor* 82
Figura 43 – Tela de edição de apresentações do *impress.js Editor* 82
Figura 44 – Metadata do plugin NuiMod no Greasymonkey 83
Figura 45 – Execução de apresentação no Google Drive Presentation utilizando o NuiMod 84
Figura 46 – Visão de futuro do projeto NuiMod 89
Figura 47 – Cabeçalho definido para o *namespace* CommonTypes 98
Figura 48 – Sintaxe para especificação de articulações 99
Figura 49 – Sintaxe para especificação de esqueletos, poses e gestos 100
Figura 50 – Cabeçalho definido para o *namespace* NuiModSchema 101
Figura 51 – Sintaxe definida pra representar projetos no NuiMod 102
Figura 52 – Sintaxe definida pra representar componentes no NuiMod 103
Figura 53 – Dados de comparação para os Projetos 1, 2 e 3 105
Figura 54 – Dados de comparação para os Projetos 4 e 5 106
Figura 55 – Cronograma de desenvolvimento da dissertação. 108

Figura 56 – Controle de mudanças no cronograma. 109

Lista de tabelas

Tabela 1 – Características da metodologia científica aplicada ao trabalho. 21
Tabela 2 – Especificações do Kinect. 28
Tabela 3 – Especificações do Xtion. 29
Tabela 4 – Especificações do Carmine. 30
Tabela 5 – Especificações do *Structure Sensor*. 32
Tabela 6 – Especificações do *DepthSense 325*. 34
Tabela 7 – Tabela de comparação entre projetos para validação da *Engine* 79
Tabela 8 – Semântica de articulações 98
Tabela 9 – Semântica de esqueletos, poses e gestos 101
Tabela 10 – Semântica de projetos no NuiMod 102

Lista de abreviaturas e siglas

API	*Application Programming Interface* (Interface de Programação de Aplicações)
AVC	Acidente Vascular Cerebral
CCD	*Charge-Coupled Device* (Dispositivo de Carga Acoplada)
CLI	*Command Line Interface* (Interface por Linha de Comando)
CMOS	*Complementary Metal-Oxide-Semiconductor* (Semicondutor Metal-óxido Complementar)
CSS	*Cascading Style Sheets* (Folhas de Estilo Encadeadas)
FPS	*Frames per second* (Quadros por segundo)
GUI	*Graphical User Interface* (Interface Gráfica do Usuário)
HD	*High Definition* (Alta Definição)
HMM	*Hidden Markov Model* (Modelo Oculto de Markov)
IHM	Interação Humano-Computador
JSON	*JavaScript Object Notation* (Notação de Objetos JavaScript)
NUI	*Natural User Interface* (Interface Natural do Usuário)
RA	Realidade Aumentada
RGB	*Red, Green, Blue* (Vermelho, Verde e Azul)
SDK	*Software Development Kit* (Kit de de Desenvolvimento de Aplicativos)
SGBD	Sistema Gerenciador de Banco de Dados
ToF	*Time of flight* (Tempo de Vôo)
USB	*Universal Serial Bus*
VRPN	*Virtual-Reality Peripheral Network* (Rede Periférica de Realidade Virtual)
WSGI	*Web Server Gateway Interface*

Sumário

1 INTRODUÇÃO 18

1.1 Motivação 19

1.2 Objetivos 20

1.3 Questões norteadoras 21

1.4 Metodologia de Pesquisa 21

1.5 Estrutura do trabalho 22

2 FUNDAMENTOS TEÓRICOS, METODOLÓGICOS E TECNOLÓGICOS 23

2.1 Interação humano-computador 23

2.2 Interação natural 24

2.3 Reconstrução tridimensional 24

2.4 Dispositivos de Interação Natural 26

2.4.1 Kinect 26

2.4.2 ASUS Xtion PRO Live 27

2.4.3 Carmine 28

2.4.4 Leap Motion 30

2.4.5 Structure Sensor 31

2.4.6 DepthSense 325 (DS325) 33

2.5 Frameworks para dispositivos de Interação Natural 34

2.5.1 OpenNI 34

2.5.2 OpenKinect 35

2.5.3 Kinect for Windows SDK 36

2.6 Considerações finais 37

3 CONTEXTO DO PROBLEMA E TRABALHOS CORRELATOS . . 39

3.1 Navegação em mapas 39

3.2 Jogos digitais . . . 41
3.3 Reabilitação de pacientes . . . 43
3.4 Trabalhos Correlatos . . . 44
3.4.1 Flexible Action and Articulated Skeleton Toolkit . . . 45
3.4.2 XKIN . . . 48
3.4.3 KinectJS . . . 49
3.4.4 Kinected Browser . . . 50
3.4.5 XDKinect . . . 52
3.5 Considerações finais . . . 53

4 PROPOSTA DE SOLUÇÃO . . . 55
4.1 Modelagem de componentes . . . 55
4.2 Captura dos dados . . . 57
4.3 Processamento dos dados . . . 58
4.3.1 Detecção de estabilidade . . . 60
4.3.2 Reconhecimento de poses . . . 61
4.3.3 Reconhecimento de gestos . . . 63
4.4 Solução tecnológica para o NuiMod . . . 64
4.4.1 Modelo NMD . . . 64
4.4.2 NuiMod Connector . . . 65
4.4.3 NuiMod Engine . . . 67
4.4.4 NuiMod Editor . . . 68
4.4.4.1 Tecnologias empregadas . . . 68
4.4.4.2 Arquitetura do *NuiMod Editor* . . . 69
4.4.4.3 Acesso ao sistema . . . 70
4.4.4.4 Edição de projetos . . . 71
4.4.4.5 Serviços . . . 73
4.5 Considerações finais . . . 76

5 EXPERIMENTAÇÕES COM O NUIMOD . . . 78
5.1 Experimento 1 - *Engine* . . . 78

5.2 Experimento 2 - Editor de apresentações 79
5.3 Experimento 3 - Script para Greasemonkey 83
5.4 Considerações finais 85

6 CONCLUSÕES E PERSPECTIVAS 86
6.1 Contribuições 86
6.2 Limites da proposta 87
6.3 Lições aprendidas 87
6.4 Trabalhos futuros 88
6.5 Conclusão 89

Referências 92

APÊNDICES 96

APÊNDICE A – NUIMOD XML SCHEMA 97
A.1 Parte I - Dados Comuns 97
A.1.1 Articulações 98
A.1.2 Esqueletos, Poses e Gestos 100
A.2 Parte II - NuiMod 101

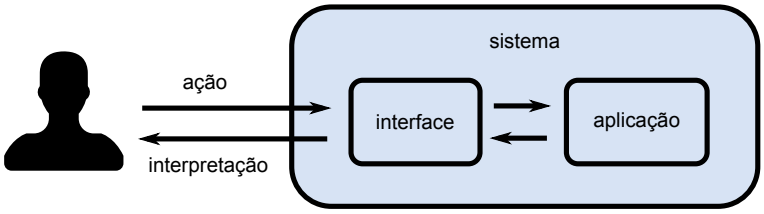
APÊNDICE B – DADOS DE PRECISÃO DA *ENGINE* 104

APÊNDICE C – CRONOGRAMA DA DISSERTAÇÃO 107

1 Introdução

Com a evolução dos computadores, surgiram também novas formas de interação humano-computador. A interação é um processo que engloba as ações do usuário sobre a interface de um sistema, e suas interpretações sobre as respostas reveladas por esta interface (SOUZA et al., 1999), conforme ilustra a Figura 1:

Figura 1 – Processo de interação humano-computador.



Fonte: (SOUZA et al., 1999)

O termo IHC (Interface Humano-Computador) emergiu na segunda metade dos anos 80, como forma de descrever um novo campo de pesquisa preocupado não apenas com o design da interface de sistemas computacionais, mas também com o foco de interesse e demandas do público (GUEDES, 2008).

A história da IHC pode ser dividida em 4 gerações. Na primeira geração encontra-se a origem da comunicação homem-computador, por meio de dispositivos para leitura de cartões perfurados. O ENIAC, primeiro computador digital eletrônico de grande escala, utilizava este recurso para inclusão de dados e comandos no sistema.

A segunda geração é marcada pela interação por meio de um teclado e um monitor monocromático. Nesta época, a interação ocorria por comandos executados a partir de uma interface via terminal, conhecida como Interface de Linha de Comando (CLI - *Command Line Interface*).

O advento das Interfaces Gráficas do Usuário (GUI - *Graphical User Interface*) marca o início da terceira geração. Neste tipo de interface, a interação ocorre por meio da utilização de mouse e teclado através de elementos gráficos como ícones e outros indicadores visuais.

As Interfaces Naturais do Usuário (NUI - *Natural User Interface*) referem-se à quarta geração de interfaces de usuário e representam uma quebra de paradigma em relação às gerações anteriores, caracterizando-se pela inexistência de contato com o dispositivo de interação, permitindo ao usuário controlar o computador por meio de movimentos corporais.

A maior parte das interfaces humano-computador requer algum tipo de dispositivo para que a interação ocorra, seja um cartão perfurado, teclado, mouse, controle remoto ou joystick. As NUI revolucionam a maneira de interagir, pois descartam o uso de dispositivos extras e tornam a interação invisível, uma vez que o próprio corpo torna-se o instrumento de interação.

Uma das grandes vantagens nesse tipo de interface é o aprendizado rápido, tendo em vista que o sujeito não precisa aprender a controlar um dispositivo de interação. Segundo (BLAKE, 2011), uma interface natural é desenvolvida para reutilizar habilidades existentes e interagir de maneira apropriada com o conteúdo. A interação com interfaces naturais podem ocorrer de diferentes maneiras, seja por meio de toque, rastreamento de movimentos ou voz.

No contexto dos jogos digitais, a evolução das NUI tem sido percebida de maneira mais acentuada, com o lançamento de novos dispositivos que melhoram a interação homem-computador. Um dos primeiros dispositivos a ganhar notoriedade neste campo foi o Kinect, sensor de movimentos desenvolvido inicialmente para Xbox 360, console de *games* da Microsoft. Após o seu lançamento, surgem as primeiras iniciativas para utilização deste tipo de sensor extrapolando o universo dos consoles de jogos digitais como, por exemplo, para navegação em mapas, proposto por (KAMEL BOULOS et al., 2011) e no apoio ao ensino, como proposto por (SILVA; NETO, 2012), (LEE; LIU; ZHANG, 2012) e (VILLAROMAN; ROWE; SWAN, 2011).

Para facilitar o desenvolvimento de aplicações computacionais utilizando NUI, nasceram as primeiras iniciativas implementadas como *frameworks*, conforme apresentado nos trabalhos de (SUMA et al., 2011), (PEDERSOLI et al., 2012), (KINECTJS, 2014) e (NETO; SANTOS; CARVALHO, 2013).

A despeito do seu pioneirismo na criação de interfaces, tais *frameworks* apresentam deficiências como dificuldades quanto a sua utilização em conjunto com movimentos complexos, embora existam abordagens que tenham demostrado seu bom funcionamento em cenários controlados (SUMA et al., 2011). Além disso, a modelagem nestas soluções pode ser complexa a usuários iniciantes, uma vez que baseia-se em regras. Outro ponto negativo é inexistência de *frameworks* voltados especificamente para o ambiente web.

1.1 Motivação

Muitas pesquisas tem sido realizadas no âmbito de Interfaces Naturais de Usuário a fim de estender esta tecnologia a outros cenários além de jogos digitais. Todavia, poucos estudos voltam-se à questão de modelagem de poses e gestos, um ponto importante para o desenvolvimento deste tipo de aplicação. Geralmente as aplicações desenvolvidas no contexto de NUI predefinem os gestos ou poses que o usuário deve utilizar, não permitindo

sua personalização.

A questão da modelagem é extremamente relevante em áreas como a fisioterapia ou educação física, onde são necessários movimentos específicos para cada sujeito. Neste sentido, permitir que poses e gestos sejam modelados exclusivamente para cada usuário torna-se um requisito indispensável.

Além disso, os trabalhos de modelagem de interfaces baseadas em gestos voltados para o ambiente web estão em fase embrionária, constituindo este um importante caminho para as pesquisas na área, haja vista o crescente número de aplicações desenvolvidas para a Internet.

Outrossim, a falta de padrões definidos e abertos para a troca de informações entre os diversos sistemas NUI gera um cenário de aplicações isoladas que não consideram questões importantes de interoperabilidade.

1.2 Objetivos

O objetivo geral desta dissertação é conceituar e desenvolver uma solução, denominada NuiMod (*Natural User Interface Modelling*) que permita a modelagem, compartilhamento e utilização de componentes gestuais (poses e gestos) no ambiente web. Como produtos desta solução, foram criados a *Engine*, para processamento dos componentes, o *Connector*, para interligar a *Engine* ao sistema operacional do usuário, permitindo o acesso ao dispositivo de interação natural e o *Editor*, ambiente web que permite a modelagem dos componentes. Além disso, foi especificado o padrão NMD, que permite a troca de informações sobre componentes modelados no *Editor*.

Para alcançar esse objetivo geral, os seguintes objetivos específicos foram traçados:

- identificar e analisar as soluções existentes para utilização de dispositivos de interação natural na web;
- definir um padrão aberto para troca de informações entre o *Editor* e sistemas terceiros, facilitando a interoperabilidade entre eles;
- projetar uma engine que faça o processamento de poses e gestos no navegador.
- conceber uma arquitetura conceitual da solução, definindo os módulos e sua relação de dependência;
- projetar uma arquitetura modular para compor o ambiente;
- mapear os elementos da arquitetura para as respectivas soluções tecnológicas.

1.3 Questões norteadoras

De acordo com os objetivos gerais, as indagações que esta pesquisa se propõe a responder são:

1. como permitir a personalização de movimentos pelo usuário em aplicações que utilizam interfaces naturais do usuário baseadas em gestos?
2. é possível utilizar interfaces naturais de usuário baseadas em gestos no ambiente web?
3. como trocar informações entre sistemas de modelagem de poses/gestos e aplicações que utilizam interfaces naturais de usuário baseadas em gestos?
4. como permitir o acesso que navegadores web acessem dados de sensores de movimento?

1.4 Metodologia de Pesquisa

Não há, evidentemente, regras fixas acerca da elaboração de um projeto. Sua estrutura é determinada pelo tipo de problema a ser pesquisado e também pelo estilo de seus autores. É necessário que o projeto esclareça como se processará a pesquisa, quais as etapas que serão desenvolvidas e quais os recursos que devem ser alocados para atingir seus objetivos (GIL, 2002). Nesse sentido, a metodologia de pesquisa adotada neste trabalho possui as seguintes características:

Tabela 1 – Características da metodologia científica aplicada ao trabalho.

Quanto à natureza	Pesquisa aplicada
Quanto aos objetivos	Exploratório e descritiva
Quanto às abordagens	Qualitativa
Quanto aos procedimentos	Bibliográfica e experimental

Fonte: Elaborada pelo autor

A primeira etapa desta pesquisa ocorreu a partir de estudos dirigidos que resultaram na definição e aproximação do tema de pesquisa. Foi realizado um levantamento bibliográfico sobre os trabalhos desenvolvidos com Kinect, a fim de definir objetivos e questões norteadoras.

À luz das indagações que a dissertação se propunha responder, foi realizado um novo levantamento bibliográfico, desta vez voltado a responder as questões norteadoras, fundamentar o referencial teórico e listar os trabalhos correlatos.

A seguir iniciou-se a etapa de prototipação, com a montagem do ambiente para desenvolvimento da solução. Foram realizados os experimentos iniciais, que comprovaram a viabilidade técnica da idéia proposta. A partir de então, começaram os ciclos de desenvolvimentos e testes de cada módulo idealizado.

Paralelamente ao desenvolvimento iniciou-se a dissertação, por meio da definição da estrutura inicial e cronograma. Em um processo cíclico, a cada 2 capítulos foram relizadas revisões parciais, com o propósito de facilitar ajustes e a revisão geral.

1.5 Estrutura do trabalho

Esta dissertação está organizada da seguinte forma: no presente capítulo (Capítulo 1) é feita uma introdução sobre conceitos importantes para a compreensão da temática abordada e uma descrição da motivação deste trabalho. São também abordados os objetivos do projeto e a metodologia de pesquisa empregada em seu desenvolvimento.

Após a fase introdutória, no Capítulo 2 são descritos os principais dispositivos de interação natural e as ferramentas de desenvolvimento existentes para sua utilização. Durante esta etapa são realizadas comparações entre os dispositivos a fim de compreender melhor suas características.

O problema que se pretende solucionar é descrito no Capítulo 3, seguido pela apresentação dos principais trabalhos correlatos.

A proposta e arquitetura da solução implementada é descrita no Capítulo 4, juntamente com as características e tarefas de cada um dos componentes do sistema. Em seguida, discute-se o processo de desenvolvimento e são apresentados detalhes sobre a implementação do projeto final.

O Capítulo 5 apresenta os cenários de uso criados para testar a solução concebida, bem como os resultados obtidos sobre a aplicação dos testes realizados.

As contribuições do projeto, suas limitações e trabalhos futuros são apresentados no Capítulo 6, seguido pela conclusão da pesquisa.

2 Fundamentos Teóricos, Metodológicos e Tecnológicos

Este capítulo faz uma introdução sobre tópicos importantes, estudados no desenvolvimento desta dissertação, a saber: Interação Humano-Computador, Interação Natural, Reconstrução tridimensional e Dispositivos de Interação Natural.

2.1 Interação humano-computador

A interação de uma pessoa com o mundo ocorre por meio de informações sendo enviadas e recebidas. Em uma interação com um computador, o usuário recebe a informação que é emitida por ele, e responde por meio de uma entrada para a máquina - a produção do usuário torna-se a entrada do computador e vice-versa.

Nos seres humanos, a entrada ocorre por meio dos sentidos - visão, audição, tato, paladar e olfato - e a saída ocorre por meio de atividades motoras e seus efeitos, como dedos, braços, movimentos de cabeça, voz, etc.

Dos sentidos de entrada, os três primeiros são os mais importantes para IHC. Gosto e cheiro não possuem um papel relevante e não está claro se poderiam ser explorados em sistemas computacionais em geral, embora possam desempenhar algum papel nos sistemas mais especializados (cheiros podem avisar sobre um mau funcionamento, por exemplo) ou em sistemas de realidade aumentada. (DIX, 2009)

Na outra ponta da interação, encontra-se o computador, tradicionalmente composto por teclado (para entrada de texto) mouse (para posicionamento) e monitor (para exibição de dados de saída). Existem outros inúmeros dispositivos de entrada e saída para computadores, tais como: tela sensível ao toque, scanner, leitor de código de barras, webcam, joystick, caixas de som e impressora. Embora numerosos, no caso dos computadores, os canais de entrada são muito mais simples.

Em relação ao uso dos dispositivos tradicionais de entrada, é necessário que o usuário se adeque à sua utilização, uma vez que seu uso não é natural ao ser humano. Uma área de pesquisa tem se dedicado à mudar este panorama, por meio de dispositivos que se adequem à linguagem humana: a Interação Natural.

2.2 Interação natural

No processo de comunicação, as pessoas geralmente empregam diferentes canais, como voz, gestos, expressões e movimentos. As pesquisas na área de Interação Natural buscam criar sistemas que compreendam tais ações e permitam envolver os indivíduos em um diálogo, interagindo naturalmente com o outro e com o ambiente.

A linguagem entre pessoas e máquinas tem sido determinada por restrições tecnológicas, e os humanos tiveram que se adaptar a esta linguagem. O surgimento de novas tecnologias tem possibilitado que as máquinas se adaptem à linguagem humana, em termos de sensação, apresentação e narração (NORMAN, 2002). Tais tecnologias têm dispensado, aos indivíduos, a necessidade de aprendizado de novos procedimentos, comandos, linguagens ou dispositivos eletrônicos.

A criação de novos paradigmas de interação e novas convenções de mídia, que explorem a capacidade de detecção das máquinas oferecidas pela tecnologia e compreendam a maneira espontânea dos humanos descobrirem o mundo real, é um grande desafio para os designers de hoje (VALLI, 2008).

O surgimento de novos dispositivos de interação natural e soluções de software para utilizá-los colaboram com este cenário, a medida em que popularizam a tecnologia e fomentam pesquisas na área.

2.3 Reconstrução tridimensional

Dispositivos de interação natural fornecem mapas de profundidade da cena visualizada, capturados em tempo real. Tais mapas constituem-se de imagens que contém valores de profundidade da cena associados para cada *pixel*.

Para se obter a forma tridimensional de um objeto podem ser empregados dois métodos: o ativo ou o passivo. No método passivo, utiliza-se a luz ambiente para extrair informações de profundidade da cena. Neste caso, para se obter uma boa definição, é necessário haver uma fonte de luz constante quando a luz natural não é suficiente.

Por outro lado, o método ativo utiliza projetores de padrões para reconhecimento dos objetos, substituindo a luz natural pela projeção de luz artificial simples ou codificada. Dos métodos ativos, três técnicas para se obter imagens de profundidade têm sido muito utilizadas atualmente: triangulação, *Time-of-Flight* (ToF ou Tempo de vôo) e luz estruturada.

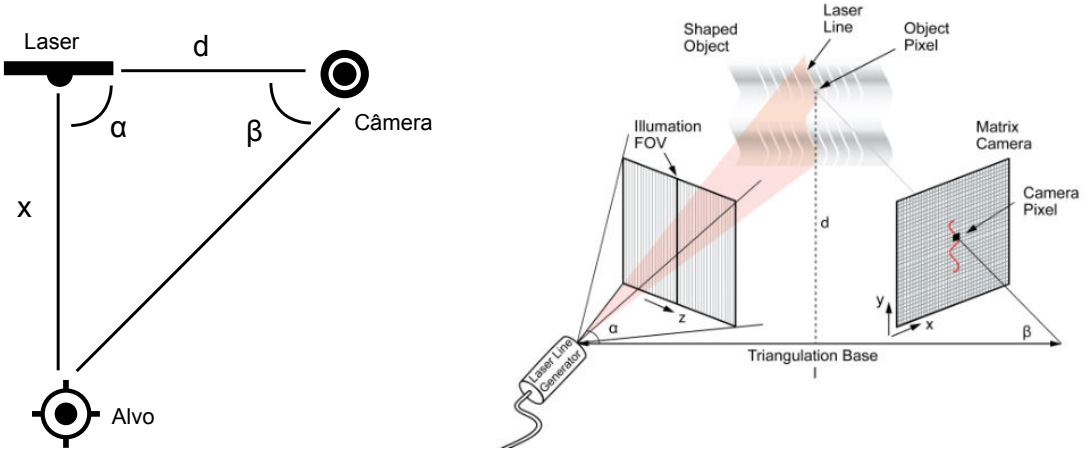
A medição de profundidade feita por triangulação utiliza as propriedades geométricas do triângulo para calcular a localização de um alvo, sendo considerado o método mais antigo (e também o mais comum) para medição de intervalo entre pontos remotos (BESL,

1988). O mapa de profundidade é obtido a partir da triangulação feita entre uma câmera receptora e um emissor infravermelho.

Neste caso, o comprimento de um lado do triângulo (a distância d entre a câmera e o emissor de laser) e o ângulo de canto do emissor de laser (α) são conhecidos, portanto o ângulo de canto da câmera (β) pode ser determinado observando a posição do ponto do laser no campo de visão da câmara (Figura 2).

Estas três partes de informação podem determinar completamente a forma e tamanho do triângulo e fornecer a localização do alvo desejado. Assim, para cada *pixel* da imagem, calcula-se a distância x até o alvo, formando a imagem de profundidade.

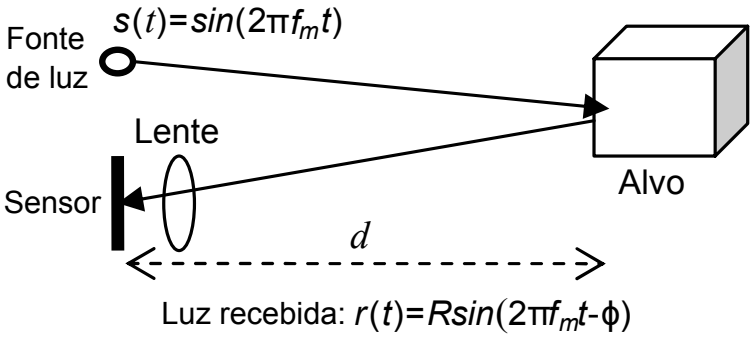
Figura 2 – Medição de profundidade por triangulação e luz estruturada.



Fonte: Elaborada pelo autor

No *Time-of-Flight*, a distância entre o objeto desejado e o sensor é calculada com base na velocidade da luz, medindo-se o tempo de vôo de um pulso luminoso entre a câmera e o objeto desejado para cada *pixel* da imagem (Figura 3).

Figura 3 – Medição do tempo de vôo.



Fonte: Adaptada de (GOKTURK; YALCIN; BAMJI, 2004)

Como a velocidade da luz (v) é conhecida, o tempo de ida e volta determina a

distância de percurso da luz, ou duas vezes a distância entre o *scanner* e a superfície. Se t é o tempo de ida e volta, então a distância d é determinada por:

$$d = \frac{(v \cdot t)}{2}$$

Uma vez que o laser detecta apenas a distância entre um ponto em sua direção de vista, para capturar todo o campo de visão é necessário calcular um ponto por vez, mudando a direção do laser para digitalizar pontos diferentes.

Os sistemas ToF possuem uma baixa precisão, com variação alta de profundidade e alcance contrastando com sistemas baseado em triangulação, que possuem grande precisão, porém o alcance e variação de profundidade pequenos.

Na técnica de luz estruturada (Figura 2), uma fonte de laser emite um único feixe de luz, dividido em múltiplos feixes por uma rede de difração a fim de criar um padrão constante de pontos projetados no ambiente (KHOSHELHAM, 2011).

O padrão é capturado por uma câmera infravermelha e comparado com um padrão de referência; caso a superfície seja plana, o padrão capturado será semelhante ao padrão projetado (referência) (BATLLE; MOUADDIB; SALVI, 1998). Se houver variação, a informação é processada como deformação do padrão capturado.

As variações são medidas para todas os pontos em um processo de correlação de imagens simples, produzindo uma imagem de disparidade. Para cada pixel, a distância até o sensor pode então ser recuperada a partir da disparidade correspondente, por meio de triangulação.

A vantagem desta técnica está na captação de multiplos pontos do campo de visão de uma única vez, conferindo maior precisão e velocidade quando comparado à técnica ToF (WEISE et al., 2011).

2.4 Dispositivos de Interação Natural

Nesta seção serão apresentados os principais dispositivos para Interação Natural por meio de poses e gestos disponíveis no mercado. Foram levantadas as principais características de cada sensor, bem como suas vantagens e desvantagens.

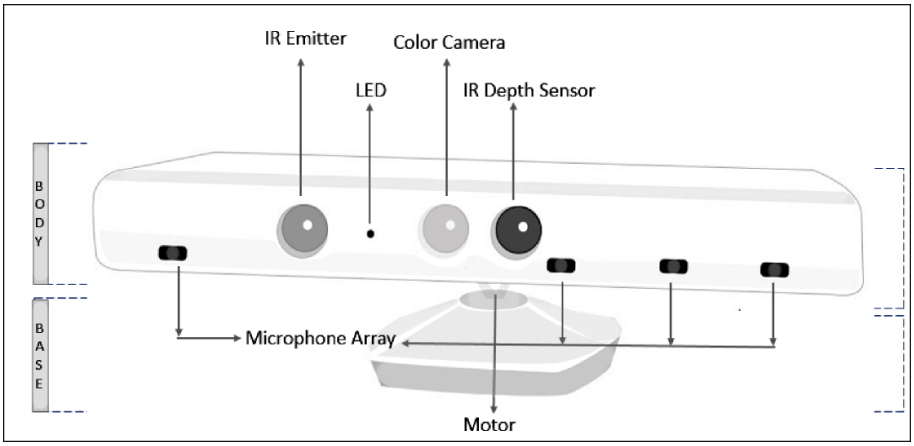
2.4.1 Kinect

O Kinect é um sensor de movimento desenvolvido pela Microsoft em parceria com a empresa Israelense PrimeSense. O dispositivo permite a interação entre jogadores e jogos digitais sem a necessidade de nenhum tipo de controle atrelado ao corpo do jogador, o que representa uma inovação no campo da jogabilidade (CATUHE, 2012).

O sensor incorpora vários *hardwares* de detecção avançada (Figura 4), sendo os principais:

- Sensor de profundidade
- Câmera a cores
- Conjunto de quatro microfones

Figura 4 – Estrutura básica do sensor Kinect



Fonte: (JANA, 2012)

O sensor de profundidade é composto por um projetor infravermelho e um sensor CMOS monocromático que trabalham juntos para "enxergar" o ambiente.

A câmera de cores ajuda no reconhecimento facial e na percepção de outras características ao detectar três cores componentes: vermelho, verde e azul. A Microsoft a denomina "câmera RGB", referindo-se aos componentes de cor que ela detecta.

O microfone multi-matriz é composto por uma série de quatro microfones que podem isolar as vozes dos jogadores de ruídos do ambiente. Ele possibilita ao jogador estar a poucos metros de distância do microfone e ainda usar controles por voz. A Tabela 2 apresenta as principais características do sensor.

Uma das grandes desvantagens do Kinect está na alimentação do sensor, que necessita de uma fonte de energia externa. Por outro lado, tendo em vista sua popularidade, se destaca por um maior número de soluções existentes.

2.4.2 ASUS Xtion PRO Live

O ASUS Xtion PRO Live integra a segunda geração de dispositivos lançados pela ASUS para criação de soluções com Interfaces Naturais de Usuário em computadores. O *hardware* é composto por uma câmera de detecção de profundidade, uma câmera RGB

Tabela 2 – Especificações do Kinect.

Característica	**Informações**
Distância de uso	entre 800mm e 4000mm (O hardware do Kinect for Windows possui o *Near Mode* que permite seu uso na faixa de 500mm a 3000mm)
Campo de visão	43º Vertical 57º Horizontal
Sensores	RGB& Profundidade& Microfone*4
Tamanho da imagem de profundidade	640 x 480 (30 FPS) 1280 x 960 (12 FPS) 640 x 480 (15 FPS)
Resolução	1280 x 960
Plataforma	Intel X86 & AMD
Suporte a sistema operacional	Oficialmente apenas Windows. Pode-se utilizar bibliotecas terceiras para suporte à outras plataformas, como libfreenect e OpenNI.
Interface	USB 2.0
Linguagem de programação	C#, C++ e VB
Dimensões	Altura: 73mm Largura: 283mm Profundidade: 72.8mm Peso: 564.5g
Faixa de inclinação vertical	±27º

Fonte: (CATUHE, 2012)

e dois microfones, tornando possível a captura de movimentos e áudio de seu utilizador (ASUS, 2013). Suas principais características são descritas na Tabela 3.

Um benefício do Xtion em relação ao Kinect é que o dispositivo pode ser alimentado diretamente pela USB, além de ser oficialmente aderente à especificação OpenNI. Em contrapartida, não possui motor de inclinação e é um dispositivo pouco conhecido e documentado, o que pode comprometer sua utilização por desenvolvedores independentes.

2.4.3 Carmine

O sensor Carmine (Figura 5) é um dispositivo de interação natural desenvolvido pela Primesense, empresa responsável pela criação do *hardware* do Kinect. Sua aparência é bastante semelhante ao ASUS Xtion PRO Live e também utiliza alimentação via USB para funcionar.

Uma das grandes vantagens deste dispositivo é ser totalmente aderente à especificação OpenNI, uma vez que seu fabricante liderava os trabalhos do grupo (SUAREZ;

Tabela 3 – Especificações do Xtion.

Característica	**Informações**
Distância de uso	Entre 0.8m e 3.5m
Campo de visão	58º Horizontal 45º Vertical 70º Diagonal
Sensores	RGB& Profundidade& Microfone*2
Tamanho da imagem de profundidade	VGA (640x480) : 30 fps QVGA (320x240): 60 fps
Resolução	SXGA (1280*1024)
Plataforma	Intel X86 & AMD
Suporte a sistema operacional	Win 32/64 : XP , Vista, 7, 8 Linux Ubuntu 10.10: X86, 32/64 Android (apenas por requisição)
Interface	USB2.0 / 3.0
Linguagem de programação	C++/C# (Windows) C++(Linux) JAVA
Dimensões	Altura: 50mm Largura: 180mm Profundidade: 35mm

Fonte: (ASUS, 2013)

Figura 5 – Imagens dos dispositivos Carmine e Capri da Primesense.

Fonte: *Google Images*

MURPHY, 2012). Outra ponto positivo está em seu tamanho reduzido e na melhor qualidade dos dados de profundidade. A tabela 4 apresenta as especificações detalhadas do dispositivo.

A Primesense produz ainda uma solução embarcada denominada Capri, que utiliza a tecnologia *System on Chip*, incluindo algoritmos melhorados com técnicas de sensoriamento 3D multimodais (PRIMESENSE, 2013). Foi criado para ser utilizado em computadores, PCs *All–in-One*, *tablets*, *laptops*, celulares, TVs e robôs. A Figura 5 mostra a solução

Tabela 4 – Especificações do Carmine.

Característica	**Informações**
Distância de uso	Entre 0.8m e 3.5m (V 1.08) 0.35m a 1.45m (V 1.09)
Campo de visão	57.5º Horizontal 45º Vertical 69º Diagonal
Sensores	RGB& Profundidade& Microfone*2
Tamanho da imagem de profundidade	VGA (640x480) : 30 fps
Resolução	640 x 480 (VGA)
Plataforma	Intel X86 & AMD & Arm
Suporte a sistema operacional	Win 32/64 : XP , Vista, 7, 8 Linux Ubuntu 10.10: X86, 32/64
Interface	USB2.0
Linguagem de programação	C++/C# (Windows) C++(Linux) JAVA
Dimensões	Altura: 35mm Largura: 180mm Profundidade: 25mm

Fonte: (PRIMESENSE, 2013)

utilizada em conjunto com um tablet.

Estes dispositivos tiveram suas vendas descontinuadas após a aquisição da PrimeSense pela Apple, o que torna sua utilização em projetos comerciais pouco viável.

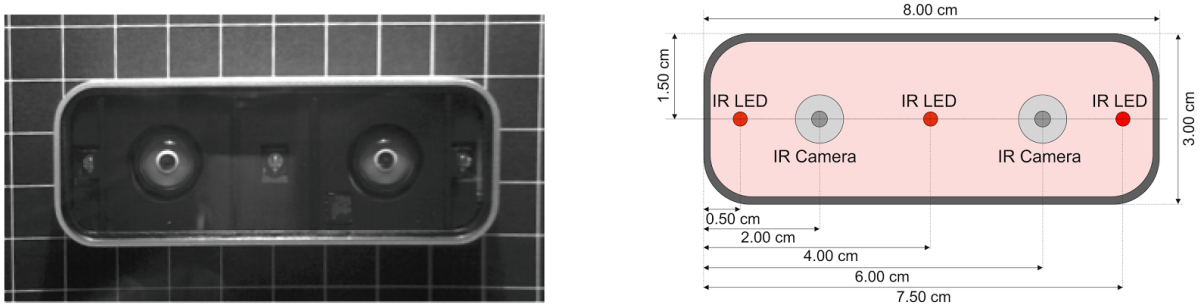
2.4.4 Leap Motion

O controlador *Leap Motion* é um sensor desenvolvido pela empresa homônima *Leap Motion*, projetado principalmente para detecção de gestos de mãos e dedos em aplicações de software interativos (WEICHERT et al., 2013). A Figura 6 apresenta uma visão esquemática da configuração de *hardware* do controlador, equipado com três emissores de infravermelhos e duas câmaras CCD. A precisão do sensor de detecção de posição dos dedos é de aproximadamente 0,01 mm.

O *Leap Motion* disponibiliza uma API, por meio da qual é possível obter as posições no espaço cartesiano de objetos pré-definidos, como as pontas dos dedos ou a ponta de uma caneta. A localização das posições é calculada em relação ao ponto central do dispositivo (Figura 6), localizado no segundo emissor de raios inframermelhos.

A versão 2 da API permite modelos mais robustos da mão, de modo que todas as informações possam ser inferidas mesmo havendo obstruções. A posição e a rotação de cada articulação é disponibilizada de maneira consistente e cada dedo é composto

Figura 6 – Visualização de um esquema real (a) e visualização esquemática (b) do *Leap Motion*.



(a) **(b)**

Fonte: (WEICHERT et al., 2013)

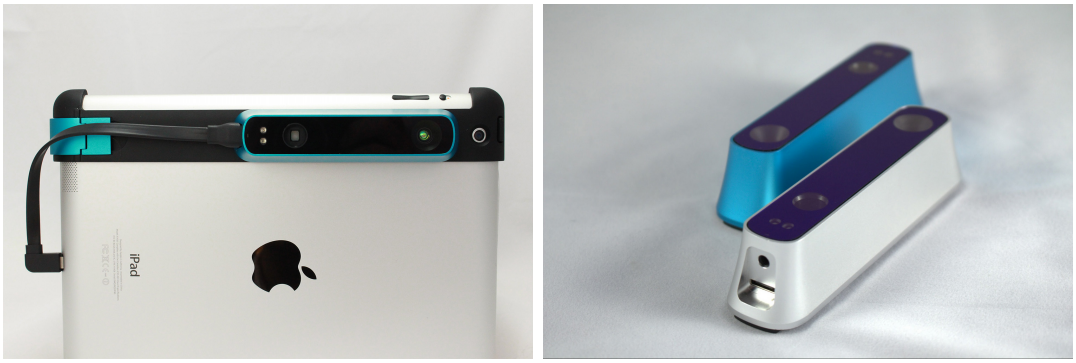
por quatro ossos precisamente nomeados. Além disso, podem ser detectados movimentos como pinça, indicando se um dedo está tocando em outro, em uma escala de zero a um; e apanhar, indicando quão perto do punho estão os dedos da mão.

Em situações de difícil reconhecimento, quando a mão está perto da zona limítrofe de reconhecimento do dispositivo ou quando uma mão sobrepõe a outra, a API disponibiliza métodos para a filtragem de dados, úteis especialmente para ações que não podem ser desfeitas. Neste caso, os dados indeferidos ou imprecisos podem ser descartados.

2.4.5 Structure Sensor

O *Structure Sensor* é um dispositivo de alumínio anodizado que pode ser utilizado em iPads (Figura 7) para captura 3D, fornecendo dados de profundidade do ambiente. Ele permite que dispositivos móveis não somente capturem imagens tridimensionais, mas também compreendam-as em três dimensões (OCCIPITAL, 2014).

Figura 7 – *Structure Sensor*, desenvolvido pela Occipital.



Fonte: (OCCIPITAL, 2014)

Embora tenha sido desenvolvido inicialmente para recriar espaços virtualmente, o *gadget* também pode ser utilizado para uma séries de outras aplicações, tais como:

- Mapeamento 3D de espaços internos (*indoor*) para medições instantâneas e redecoração virtual
- Jogos de Realidade Aumentada (AR) onde objetos virtuais interagem precisamente com geometrias do mundo real
- Scaneamento do corpo humano para rastreamento *fitness* e montagem de roupas virtuais
- Scaneamento de objetos tridimensionais para criação de conteúdo 3D

Diferentemente de outros sensores do gênero, projetados para conectar-se a *consoles* de jogos e computadores, o *Structure Sensor* foi projetado desde sua concepção para ser utilizado em dispositivos móveis, resultando em características como dimensões compactas, alcance otimizado e fornecimento de energia interno (Tabela 5).

Tabela 5 – Especificações do *Structure Sensor*.

Característica	**Informações**
Distância de uso	Entre 0.4m e 3.5m
Campo de visão	58 Horizontal 45º Vertical
Sensores	RGB& Profundidade& Mifophone*2
Tamanho da imagem de profundidade	VGA (640x480) : 30 fps QVGA (320x240): 60 fps
Plataforma	Intel X86 & AMD & Arm
Suporte a sistema operacional	Android, Windows, Linux, and OSX
Bateria	3-4 horas em funcionamento 1000+ horas em *standby*
Dimensões	Altura: 29mm Largura: 119.2mm Profundidade: 27.9mm

Fonte: (OCCIPITAL, 2014)

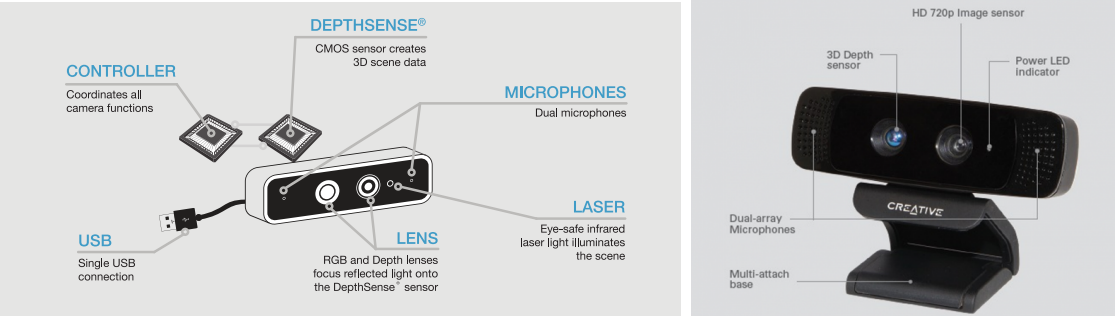
O dispositivo emprega a técnica de luz estruturada para captura dos dados de profundidade, utilizando um projetor a laser para lançar um padrão preciso de milhares de pontos infravermelhos invisíveis em objetos e espaços. Em seguida, usa uma câmera infravermelha pareada por freqüência para registrar como o padrão muda, compreeendendo assim a geometria dos objetos e espaços. Como resultado, o sensor pode gerar um fluxo de dados de profundidade VGA a 30 FPS, onde cada pixel representa uma distância exata de um ponto real.

2.4.6 DepthSense 325 (DS325)

O DS325 é um dispositivo NUI criado pela empresa SoftKinect que provê dados 3D de distância em tempo real para interação de curto alcance. É o dispositivo com a câmera de profundidade de melhor precisão do mercado e com a menor distância necessária para funcionamento (entre 15cm e 1m), voltado para o rastreamento de mãos e dedos (SOFTKINETIC, 2013).

A solução combina uma tecnologia patenteada da empresa, chamada *DepthSense*, que inclui uma câmera ToF (*time-of-flight*, ou tempo de vôo), um sensor RGB em HD (alta definição) e dois microfones (Figura 8). O dispositivo ainda conta com uma entrada USB, por onde ocorre a alimentação, dispensando o uso de adaptadores externos.

Figura 8 – Diagrama esquemático e foto do dispositivo *DepthSense 325*.



Fonte: (SOFTKINETIC, 2013)

O sensor mede o tempo que a luz infravermelha emitida por ele leva para retornar - o *time-of-flight* - e transforma, em tempo real, os dados posicionais ToF em imagens RGB 3D, bem como as imagens em tons de cinza e mapa de profundidade. Com uma taxa de atualização de 60 FPS, o dispositivo oferece rapidez e baixa latência na transferência de dados a fim de criar a melhor experiência para o usuário sem *lags* (atrasos). As especificações completas do DS325 podem ser conferidas na Tabela 6.

A empresa disponibiliza um SDK proprietário para desenvolvimento de aplicações, que inclui um *middleware* para reconhecimento de gestos denominado *iisu* (*The Interface is You*, ou "A Interface é Você", em português) e jogos e aplicativos da SoftKinect Studios[1], empresa que desenvolve soluções com dispositivos *DepthSense*.

O *iisu* está disponível em duas versões: a *Free* (gratuita), sem licença comercial e a *Pro* (profissional), que permite a comercialização de produtos. A versão *Pro* permite o rastreamento de até 4 usuários, enquanto a versão *Free* limita o uso a apenas 1 pessoa e possui limitação de 6 horas de uso.

[1] Informações: http://www.softkinetic-studios.com

Tabela 6 – Especificações do *DepthSense 325*.

Característica	**Informações**
Distância de uso	Entre 0.15m e 1m
Campo de visão	74º Horizontal 58º Vertical 87º Diagonal
Sensores	RGB & Profundidade Microfone*2 & Acelerômetro
Tamanho da imagem de profundidade	QVGA (320x240) (25 fps – 30 fps | QVGA)
Plataforma	Intel X86 & AMD & Arm
Suporte a sistema operacional	Windows 7 e 8, Linux e Android
Linguagem de programação	C++, C#, Flash e Unity 3D
Dimensões	Altura: 30mm Largura: 105mm Profundidade: 23mm

Fonte: (SOFTKINECT, 2013)

2.5 Frameworks para dispositivos de Interação Natural

2.5.1 OpenNI

A OpenNI (*Open Natural Interface*) foi uma organização sem fins lucrativos, criada em 2010, cujo objetivo era promover e certificar a compatibilidade e interoperabilidade entre dispositivos de interação natural. Um dos principais membros da organização era a PrimeSense, empresa responsável pelo desenvolvimento do *hardware* do Kinect. Após o anúncio da aquisição da Primesense pela Apple, em 2013, o site da OpenNI foi desligado, em 23 de abril de 2014.

A organização mantinha um SDK de desenvolvimento de aplicações para dispositivos de interação natural, cujo desenvolvimento era liderado pela PrimeSense com a colaboração de uma comunidade formada por desenvolvedores independentes. A biblioteca abrange a comunicação tanto com dispositivos de baixo nível (por exemplo, sensores de visão e áudio), como soluções de *middleware* de alto nível, para o acompanhamento visual utilizando visão computacional. Foi desenvolvida de modo a ser compatível com vários sistemas do tipo RGB-D e permite o desenvolvimento em várias linguagens de programação tais como: C, C++, Java e C#.

O SDK permite detectar até 20 articulações, com 2 utilizadores simultaneamente. Contudo, é necessário realizar uma calibração pré-utilização, o que condiciona a sua utilização para sistemas que tenham como foco a reabilitação motora.

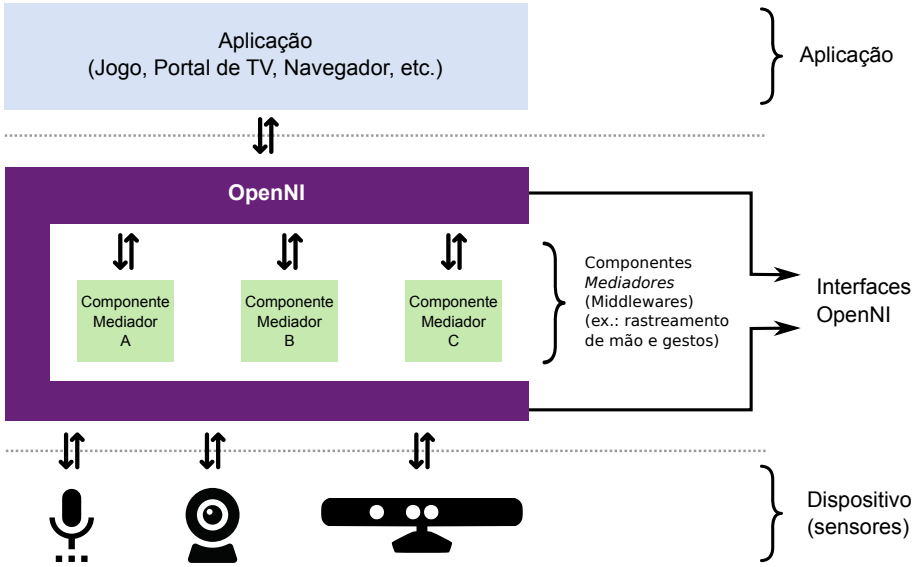
Embora não realize um seguimento preditivo de articulações, o OpenNI gera menos falsos positivos que o *Kinect for Windows SDK*, configurando-se como uma vantagem.

Também requer menos capacidade computacional, visto que os algoritmos a que recorre são mais rápidos e leves.

Só é possível obter imagens com uma resolução máxima de 800x600. Embora também seja possível acessar o array de microfones existentes nas plataformas, a API do OpenNI não inclui funções de gravação para os mesmos.

O principal ponto a favor do OpenNI é o fato de ser um *software opensource*, que pode ser usado em diversas plataformas (Figura 9), não limitando assim o desenvolvimento ao sistema operacional Windows. Contudo, os middlewares disponíveis para tarefas de alto nível, como rastreamento de esqueleto são proprietários.

Figura 9 – API do OpenNI



Fonte: (LORIOT, 2011)

Apesar de ter sido descontinuado após a aquisição pela Apple, o principal produto da OpenNI continua sendo mantido pela Occipital, empresa que desenvolve o projeto StructureIO[2], um *scanner* 3D para iPad que utiliza componentes de *hardware* desenvolvidos pela PrimeSense.

2.5.2 OpenKinect

O OpenKinect é um projeto de código aberto desenvolvido pela comunidade de pessoas interessadas em utilizar o *hardware* do Kinect em PC's e outros dispositivos, rodando Windows, Linux ou MacOS (OPENKINECT, 2014).

O principal componente do projeto é o software *libfreenect*, que provê suporte para operações como controlar o LED e o motor do dispositivo e acessar informações de imagem

[2] Ver: http://structure.io

RGB e dados de profundidade. A API do projeto é extensível para as linguagens C, C++, .NET, Java e Python.

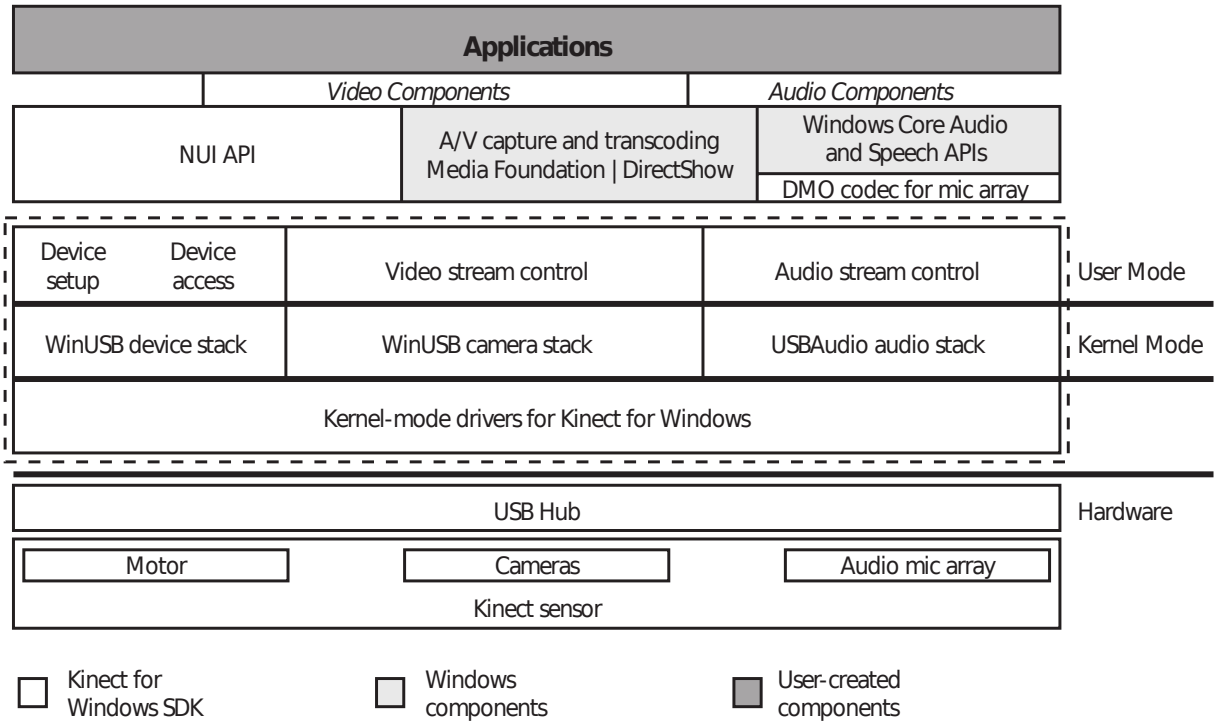
Apesar de suportar as principais funções para controle do *hardware* do Kinect, a *libfreenect* não oferece suporte à operações consideradas de alto nível, como rastreio do esqueleto e das mãos, geração e alinhamento de Nuvens de Pontos, reconstrução 3D e rastreio de movimento inercial com acelerômetro. Estas funcionalidades estão previstas no plano estratégico do projeto, porém sem data específica para serem disponibilizadas.

2.5.3 Kinect for Windows SDK

O *Kinect for Windows SDK* é um conjunto de ferramentas disponibilizados pela Microsoft para desenvolvimento de aplicações em conjunto com o Kinect. O SDK provê uma interface para interagir com o dispositivo via drivers do sistema operacional. O driver permite o acesso à câmera, sensor de produndidade, microfone e o motor de inclinação.

Integram a arquitetura do SDK (Figura 10) uma Interface de Programação de Aplicações (API) denominada NUI (Natural User Interface), os drivers para acesso aos sensores, bem como interfaces de alto nível, que permitem o rastreio de esqueleto.

Figura 10 – Arquitetura do Kinect SDK for Windows.



Fonte: (CATUHE, 2012)

As informações trafegam entre o dispositivo e o SDK por meio de fluxo de dados, que podem ser dos seguintes tipos:

- Fluxo de dados de profundidade
- Fluxo de dados de cor
- Fluxo de dados de áudio

A API NUI permite identificar a presença de até 6 pessoas e realizar o rastreamento de até 2 usuários simultaneamente. No tocante ao esqueleto humano, o SDK permite rastrear as coordenadas de 20 articulações corporais, expressas em metros (CATUHE, 2012).

Um dos diferenciais do SDK é permitir o rastreamento de usuários sentados, sendo rastreadas as 10 articulações da parte superior do corpo (cabeça, ombros, cotovelos, pulsos e mãos). Além disso é possível prever posições de articulações sobrepostas.

2.6 Considerações finais

Este capítulo apresenta os principais dispositivos e *frameworks* para sistemas de Interação Natural baseados em poses e gestos. Para o desenvolvimento desta dissertação, procurou-se analisar os *frameworks* a fim de verificar sua aderência aos seguintes requisitos:

- Multiplataforma - o *framework* deve funcionar em mais do que uma plataforma
- Código fonte aberto - o código fonte deve seguir a definição de *opensource* da *Open Source Iniciative (OSI)*
- Rastreio de esqueleto - o *framework* deve reconhecer as pessoas na cena de profundidade e seguir as seus movimentos

Dos requisitos elencados, apenas o terceiro (rastreio de esqueleto) é obrigatório, uma vez que sua ausência compromete o desenvolvimento do projeto. Com base em tais requisitos, foi construída a matriz de compatibilidade abaixo, que compara cada *framework* aos requisitos elencados.

Biblioteca	**Multiplataforma**	**Opensource**	**Rastreio de esqueleto**
OpenNI	Sim	Parcialmente	Sim
OpenKinect	Sim	Sim	Não
Kinect for Windows SDK	Não	Não	Sim

Dos *frameworks* analisados, a OpenNI disponibilizava a biblioteca com maior aderência requisitos elencados. Tendo em vista o fim da OpenNI, optou-se por descartar sua utilização, uma vez que os módulos de rastreio de esqueleto não possuem código fonte aberto. Como o OpenKinect não possui a funcionalidade rastreio de esqueleto e não

existia nenhuma alternativa disponível para esta operação, optou-se pelo uso do Kinect for Windows SDK.

3 Contexto do Problema e Trabalhos Correlatos

Neste capítulo é apresentada a descrição do problema no que tange a ambientes para modelagem de poses e gestos, bem como o compartilhamento de modelos gestuais. Para isto, foram levantadas pesquisas, em diferentes áreas do conhecimento, onde as questões que este trabalho se propõe resolver ficam evidenciadas, em contextos de navegação em mapas, jogos digitais e reabilitação de pacientes.

Ao final de cada seção discutem-se os problemas das propostas, em relação às soluções defendidas por esta pesquisa, bem como possíveis respostas que esta dissertação pode oferecer.

3.1 Navegação em mapas

O *Kinoogle* é uma Interface Natural de Usuário desenvolvida para navegação no Google Earth, um programa da empresa Google que apresenta um modelo tridimensional do globo terrestre, construido a partir de diversas imagens de satélite obtidas de diferentes fontes (KAMEL BOULOS et al., 2011). O *Kinoogle* permite ao usuário executar ações no *Google Earth*, por meio de uma série de gestos corporais.

O *hardware* escolhido para ser utilizado no projeto é o Kinect, controlado por meio do OpenNI. O *Kinoogle* utiliza o FAAST (abordado na Seção 3.4.1) para mapear os movimentos do usuário em ações do mouse e teclado, a fim de controlar o *Google Earth*. A arquitetura do Kinoogle está dividida em quatro objetos (*KinectControl*, *Kinoogle*, *EarthControl*, e *Kinoogle GUI*) que funcionam como conexão entre cada software terceiro utilizado. O sistema é baseado em entradas de hardware (mouse e teclado), seguindo uma arquitetura baseada em eventos.

O *KinectControl* realiza a comunicação entre o sistema e o OpenNI/NITE para coletar as informações do Kinect. O *loop* principal do Kinoogle chama continuamente o método `kinectRun` do *KinectControl*, que executa basicamente 3 ações:

- `context.WaitAndUpdateAll()`, que força o OpenNI a buscar um novo *frame* do Kinect com informações de profundidade e enviá-lo para o processamento por meio do NITE;
- `skeletonUpdater.run()` que verifica os dados de esqueleto recebidos envia-os ao *Kinoogle*, caso o esqueleto esteja ativo;

- `sessionManager.Update()` que força o gerenciador de sessão a processar os dados das mãos no contexto atual e atualizar todos os controles baseados em pontos por meio de *callbacks*

A interpretação dos dados recebidos do *KinectControl* é realizada pelo *Kinoogle*, que determina se o usuário executou determinado gesto ou pose específica para interagir com o *Google Earth* ou *Kinoogle GUI*. O *Kinoogle* ainda ocupa-se com a detecção de poses estáticas, por meio das quais o usuário pode alternar entre diferentes modos de navegação. As poses para cada um dos modos (panorâmica, zoom, rotação e inclinação) são estáticas e não podem ser personalizadas.

O *EarthControl* emprega ações simuladas do mouse e teclado para controlar o *Google Earth*. O módulo comunica-se com o *Kinoogle* por meio de troca de mensagens, e então interpreta-as para determinar a sequência correta de ações que devem ser executadas no mouse e teclado. Movendo o ponteiro em determinados eixos e mantendo certos botões do mouse pressionados, o *EarthControl* efetua as alterações necessárias para o visualizar as informações no mapa. Outras funções são executadas pressionando uma série de botões do teclado.

Por fim, o *Kinoogle GUI* é responsável pela interface gráfica do usuário, implementada como uma barra de menu no topo da aplicação (Figura 11). Por meio dela é possível controlar diferentes recursos do sistema disponiveis no modo de navegação atual e exibir as setas correspondentes à postura para ativar um determinado recurso.

O *Kinoogle* é uma excelente ferramenta que pode auxiliar diferentes áreas de conhecimento, como Educação, no ensino de Geografia; e saúde, onde globos são comumente empregados para visualizar e explorar epidemiologia espacial e dados de saúde pública.

Todavia, o fato de utilizar poses e gestos pré-definidos, alguns complexos a determinados públicos, pode inviabilizar sua adoção em determinados contextos, onde a ferramente seria de extrema importância. Outro fator limitante está em sua implementação, atrelada ao ambiente desktop, sendo necessário a instalação de diversos softwares de terceiros para apoiar sua execução.

A arquitetura proposta pelo Kinoogle pode ser implementada em ambiente web, utilizando a solução proposta por esta dissertação para controlar as ações por meio de poses e gestos modelados no *Editor*. O plugin para Greasemonkey serviria como um elo de ligação entre o Google Maps e o *Editor*, assim como foi realizado no experimento apresentado na Seção 5.3.

Figura 11 – Tela principal do Kinoogle

Fonte: captura de tela no sistema operacional Windows 8.1

3.2 Jogos digitais

Dificuldades na aprendizagem de matemática são frequentemente relatadas e conhecidas entre as crianças. A maneira de ensinar a disciplina praticamente não mudou ao longo do tempo e, com o surgimento de novas tecnologias como tablets, smartphones e sensores de movimento, as pessoas têm demonstrado enorme interesse e motivação em desenvolver novos métodos interativos para apoiar a aprendizagem (LEE; LIU; ZHANG, 2012).

Neste sentido, foi criado o *Xdigit*, um jogo para o ensino da matemática por meio de Interfaces Naturais de Usuário. Com a temática do espaço, o objetivo é combinar números por meio de operações aritméticas para conectar um número de destino antes que o tempo se esgote. O jogo possui diferentes níveis de dificuldade, oportunizando desafios a diferentes públicos ao aumentar a complexidade aritmética em cada fase.

A interface do usuário, apresentada na Figura 12, foi projetada para evitar confusão e sobrecarga de informação ao aluno, e fornece retornos visuais e auditivos para o usuário, sendo constituida pelos seguintes elementos:

- Número na roda: O número na roda representa o número atualmente selecionado. No caso da Figura 12, esse número é o “5”;
- Painéis laterais: Os painéis laterais indicam que os números que estão sendo utilizados para executar a equação. O número é apresentado tanto do lado esquerdo quanto do lado direito. A “?” é substituida por uma variável selecionada na roda pelo aluno.
- Barras de carga: A barra de carga oferece *feedback* visual a respeito de qual gesto operador o jogador está executando atualmente. O jogador precisa manter-se estático por 0,6 segundos para ativar o operador.
- Meteoros: O meteoro indica o número de destino que o jogador precisa acertar.
- Temporizador: O temporizador na parte superior da tela indica quanto tempo resta antes do meteoro atingir o jogador.
- Monitor de funcionamento: O monitor de saúde exibe quantas "vidas"o jogador perdeu. O jogador perde o jogo, se perder todos os corações. O estado atual da saúde do jogador também é mostrado visualmente pelo estado do espaçonave.
- Pontuação: A barra superior direita mostra a pontuação total do jogador.

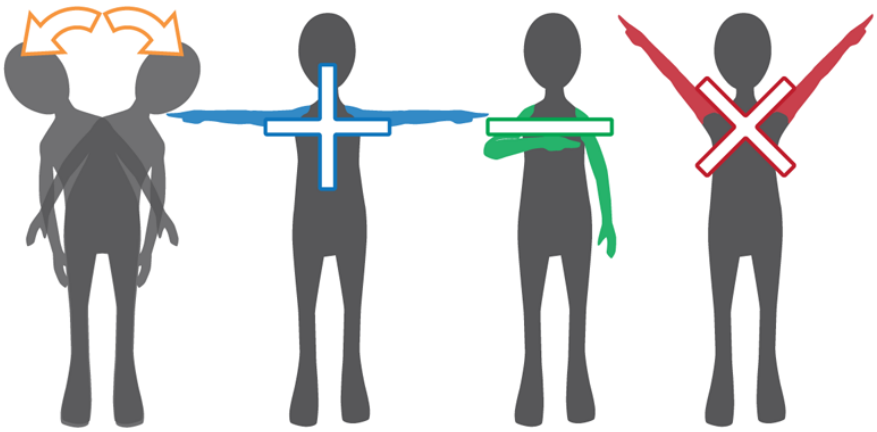
Figura 12 – Interface do jogo XDigit

Fonte: (XDIGIT, 2013)

As poses propostas pelo *Xdigit* foram concebidas para lembrar os operadores aritméticos, objetivando a fácil memorização por parte do aluno. A aplicação conta com 5 poses (Figura 13), que controlam o número na roda e as operações matemáticas (soma, subtração e multiplicação) que o jogador pode selecionar.

Oportunizar o ensino das disciplinas convencionais em conjunto com novas tecnologias que facilitem o processo de aprendizagem do aluno, tornando-o mais lúdico pode colaborar na formação de alunos mais engajados e criativos. Ferramentas como o *XDigit*

Figura 13 – Poses para executar ações no XDigit



Fonte: (XDIGIT, 2013)

colaboram com este processo ao unir novas tecnologias a assuntos que usualmente são desinteressantes ao aprendiz.

Jogos como o *Xdigit*, que exigem menor poder de processamento, poderiam ser acessíveis a um maior número de alunos se fossem implementados em HTML5, e disponibilizados na Internet.

Outrossim, apesar das poses propostas pelo *XDigit* serem de simples execução, não é permitido ao usuário personalizá-las. Destarte, alunos que tenham dificuldades motoras do lado esquerdo do corpo, por exemplo, não poderiam executar as poses sugeridas. A utilização do *Editor* pode resolver este problema, ao passo que cada aluno modelaria suas próprias poses levando em conta estas limitações.

3.3 Reabilitação de pacientes

O Acidente Vascular Cerebral (AVC) é a principal causa de incapacidade em adultos nos países desenvolvidos e totaliza cerca de 16 milhões de novos (primeiros) eventos de AVC por ano (MATHERS; LONCAR, 2006). Os custos para a reabilitação desses pacientes deve saturar os planos de saúde, forçando-os a encurtar a duração do apoio à reabilitação (BORGHESE et al., 2013). Apesar disso, os exercícios devem continuar fora do hospital, a fim de evitar a perda dos benefícios da reabilitação hospitalar e estabilizar as condições físicas e mentais do paciente.

Neste sentido, a realidade virtual tem sido explorada como uma ferramenta viável para apoiar o processo de reabilitação (CASTIELLO et al., 2004). No entanto, os dispositivos de RA clássicos são caros, invasivos e inadequados para utilização em ambientes *indoor*. Não obstante, o surgimento de dispositivos como o Kinect, estimulou o desenvolvimento de novas interfaces, possibilitando o rastreio de movimentos do corpo humano de maneira

confiável em uma plataforma de baixo custo, que pode ser utilizada na casa do paciente.

Para auxiliar a reabilitação de pacientes em domicílio, foi criado o projeto Rewire (Religar, em português), uma plataforma supervisionada para estimular a prática de exercícios físicos por pacientes que sofreram AVC (BORGHESE et al., 2013). A plataforma é constituída por três componentes hierárquicos:

- uma estação do paciente, montada na casa do usuário
- uma estação hospitalar, disponível ao hospital que acompanha o caso
- uma estação de rede, instalado no site do provedor de saúde

A estação de paciente (PS) é usada pelo paciente e seus cuidadores em casa para realizar os exercícios de reabilitação. O paciente segue o seu plano de exercício através de uma série de jogos simples.

A estação hospitalar (HS) possui dois propósitos principais. Em primeiro lugar, ele é usado como uma ferramenta de gerenciamento de terapia que permite definir e monitorar o tratamento de reabilitação. Em segundo lugar, a estação do hospital administra a comunidade virtual de pacientes, cuidadores e médicos envolvidos no processo de reabilitação, e pode ser usada para reunir informações sobre a terapia e se comunicar com todos os sujeitos envolvidos no processo de reabilitação.

A estação de rede (NS) é instalada a nível regional pelo plano de saúde, sendo empregada para analisar todos os dados de entrada de pacientes individuais, no seu dia-a-dia da reabilitação e para comparar e interpretar os resultados em diferentes populações de pacientes.

A reabilitação de pacientes é um excelente contexto onde o NuiMod pode ser aplicado. O *Editor* disponibiliza ao médico as ferramentas necessárias para modelar os componentes específicos para cada paciente. Por meio da *Engine*, os jogos podem ser personalizados para empregar poses e gestos específicos a cada caso, modelados por um fisioterapeuta.

O paciente necessitaria apenas de um computador com acesso à Internet e o *Connector* disponível na máquina, que pode ser baixado diretamente do site *NuiMod*.

3.4 Trabalhos Correlatos

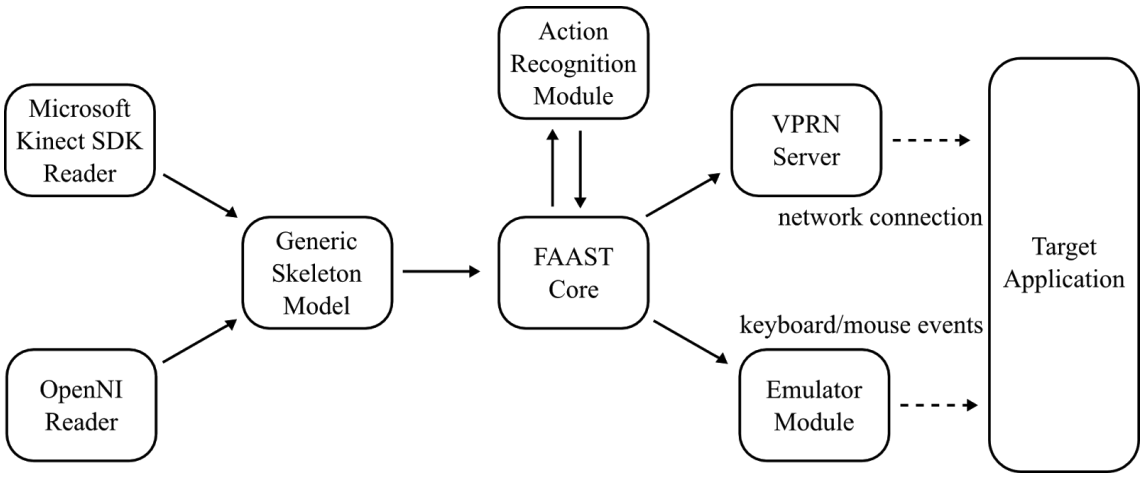
Esta seção aborda os trabalhos correlatos que foram utilizados para contextualizar esta pesquisa. Foram selecionados trabalhos que estão ligados ao tema de pesquisa, como modelagem de poses/gestos e utilização de NUI na web.

3.4.1 Flexible Action and Articulated Skeleton Toolkit

O FAAST (*Flexible Action and Articulated Skeleton Toolkit*) é um framework criado para facilitar a integração entre dispositivos de interação natural e aplicações de realidade virtual e jogos utilizando sensores de profundidade em conformidade com o padrão OpenNI (SUMA et al., 2011).

Em linhas gerais, o FAAST funciona em conjunto com dispositivos NUI para manipular duas grandes categorias de informação: ações e esqueletos articulados. A arquitetura do FAAST, observada na Figura 14, consiste em módulos que lêem dados de esqueleto por meio de uma biblioteca para câmeras de detecção de profundidade, um módulo de reconhecimento de ação, um módulo emulador que envia eventos simulados de teclado e mouse para a janela ativa do sistema operacional, e um servidor VRPN (*Virtual-Reality Peripheral Network*) para transmitir a posição e orientação de cada articulação esqueleto para aplicações em rede (SUMA et al., 2013).

Figura 14 – Arquitetura FAAST.



Fonte: Adaptado de (SUMA et al., 2011)

De maneira pormenorizada, os componentes da arquitetura (Figura 14) são (SUMA et al., 2013) (Taylor II et al., 2001) (SUMA et al., 2012):

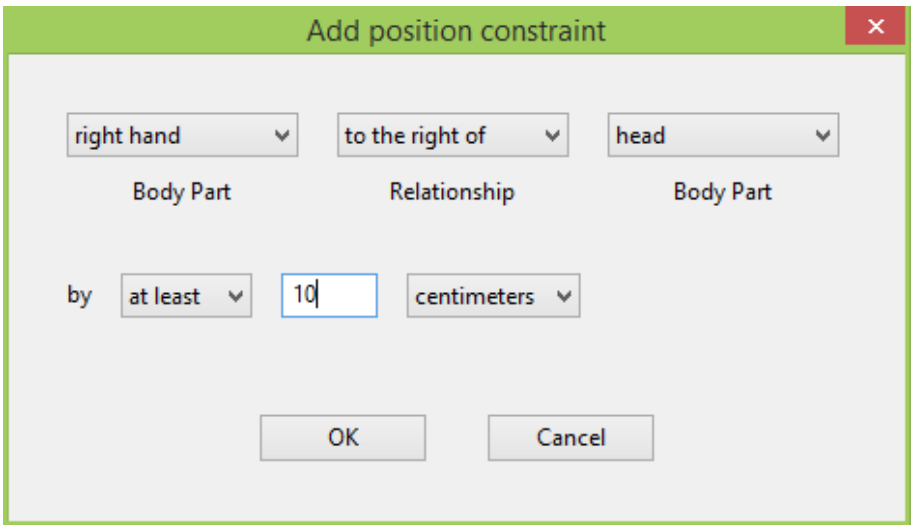
- *Microsoft Kinect SDK Reader*: é o elemento que representa os sensores proprietários do Kinect, responsáveis pela captura das imagens.
- OpenNI Reader: diz respeito aos sensores que fazem uso das bibliotecas OpenNI. Seu papel é semelhante ao item anterior, diferindo pelo fato deste trabalhar com código aberto, enquanto o segundo não. Trata-se de um componente chave da arquitetura do FAAST.
- *Generic Skeleton Model*: é o componente responsável por receber os dados lidos pelos sensores, do primeiro ou do segundo tipo mencionados, e posicioná-los num modelo

genérico de esqueleto, que será processado na sequência para reconhecimento de movimento.

- *FAAST Core*: É o núcleo da aplicação, que executa o papel de controlador. Ele aciona o módulo de Reconhecimento de Ação (*Action Recognition Module*) e repassa sua saída para o Módulo Emulador (*Emulator Module*) e para o VRPN Server.
- *VRPN Server*: o VRPN é uma biblioteca de software popular na comunidade de realidade virtual para interface com hardware de rastreamento de movimento. O FAAST inclui em seu código um servidor VRPN customizado, para que outras aplicações possam acessar os dados capturados por ele.
- *Action Recognition Module*: é o elemento que procede a operação de reconhecimento da ação, com base nos dados de movimento.
- *Emulator Module*: é o módulo que simula eventos como de teclado e mouse para enviar os sinais para a janela ativa da aplicação.
- *Target Application*: aplicação alvo que utiliza os sinais interpretados pelo FAAST em sua lógica de programação.

A modelagem de poses é feita por meio de regras, interligando duas articulações do corpo por meio de uma relação. As regras são definidas utilizando caixas de seleção, que formam expressões em Inglês. Para cada regra produzida, o usuário especifica um número limiar para ativação e a unidade de medida. Um exemplo de regra criada no FAAST pode ser visto na Figura 15.

Figura 15 – Regra criada com o FAAST



Fonte: Captura de tela no sistema operacional Windows 8.1

As regras podem ser dos seguintes tipos: restrição de posição, restrição angular, restrição de velocidade, restrição corporal e restrição de tempo. As **restrições de posição** referem-se a relação espacial relativa entre quaisquer duas articulações do corpo, descritas da seguinte maneira:

```
<parte_do_corpo> <relacao> <parte_do_corpo>
by <comparacao> [limiar] <unidade>
```

onde:

`<parte_do_corpo>` = cabeça, pescoço, tronco, cintura, ombro esquerdo, cotovelo esquerdo, pulso esquerdo, mão esquerda, ombro direito, cotovelo direito, pulso direito, mão direita, quadril esquerdo, joelho esquerdo, tornozelo esquerdo, pé esquerdo, quadril direito, joelho direito, tornozelo direito ou pé direito

`<relacao>` = para a esquerda de, para a direita de, em frente, por detrás, acima, abaixo ou para além de

`<unidade>` = centímetros, metros, polegadas ou pés

As **restrições angulares** são utilizadas nos casos em que o usuário flexiona um dos membros do corpo, sendo determinadas por meio do cálculo do ângulo de intersecção entre os dois vetores de ligação comuns do membro. São representadas da seguinte maneira:

```
<membro> flexed
by <comparacao> [limiar] <unidade>
```

onde:

`<membro>` = braço esquerdo, braço direito, perna esquerda, perna direita

`<unidade>` = graus, radianos

As **restrições de velocidade** referem-se a velocidade com que uma determinada parte do corpo se move, sendo representadas da seguinte maneira:

```
<parte_do_corpo> <direcao>
by <comparacao> [limiar] <unidade>
```

onde:

`<parte_do_corpo>` = cabeça, pescoço, tronco, cintura, ombro esquerdo, cotovelo esquerdo, pulso esquerdo, mão esquerda, ombro direito, cotovelo direito, pulso direito, mão direita, quadril esquerdo, joelho esquerdo, tornozelo esquerdo, pé esquerdo, quadril direito, joelho direito, tornozelo direito ou pé direito

`<direcao>` = para a esquerda, para a direita, para frente, para trás, para cima, para baixo, em qualquer direção

`<unidade>` = cm/seg, m/seg, inc/seg, ft/seg

As ações do corpo consideradas "globais", ou seja, que envolvem movimentos do corpo todo são denominadas **restrições corporais**. São divididas em inclinar, virar e saltar, representadas como se segue:

```
inclinar <esquerda, direita, para_frente, para_trás>
by <comparacao> [limiar] <unidade>
```

```
virar <esquerda, direita>
by <comparacao> [limiar] <unidade>
```

onde:

`<unidade>` = graus, radianos

Para detectar a ação pular, o FAAST compara a altura dos pés ao longo de uma janela de tempo (determinada como sendo 0,75 segundos), uma vez que a altura de cada junta é relativa ao sensor e não ao chão. Ações de salto são descritos da seguinte maneira:

```
saltar
by <comparacao> [limiar] <unidade>
```

`<unidade>` = centímetros, metros, polegadas, pés

As **restrições de tempo** representam o atraso temporal entre as ações individuais que constituem um gesto. No FAAST, gestos são considerados conjuntos de múltiplas regras de restrição (poses). Este tipo de restrição pode ser útil para definir ações temporais baseadas no início ou término de uma ação anterior ou parar o tempo, dependendo do gesto específico que está sendo modelado. A seguir, uma representação de restrição temporal:

```
wait for [minimum] to [maximum] seconds after action <starts, stops>
```

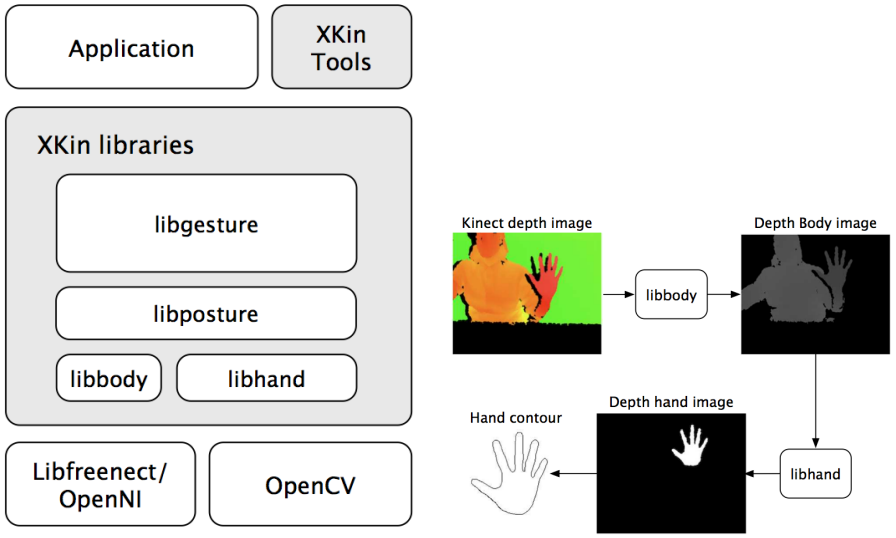
3.4.2 XKIN

O Xkin é um *framework* para Kinect que permite uma comunicação entre humanos e computadores por meio das mãos de maneira natural e intuitiva. O projeto possui código aberto, licenciado em FreeBSD e implementado na linguagem C (PEDERSOLI et al., 2012). Ele foi desenvolvido sobre a bibloteca *libfreenect* e faz uso do OpenCV para lidar com matrizes digitais, imagens e vídeos. Sua composiçao se baseia em quatro bibliotecas principais: *libbody*, *libhand*, *libposture* e *libgesture* (Figura 16).

A *libbody* é utilizada para detectar o corpo de uma pessoa na cena profundidade adquirida pelo dispositivo. Por meio dela, pode-se processar a imagem de profundidade da cena capturada para isolar a área correspondente ao corpo, definindo para zero todos os pixels que pertencem ao fundo.

Para detecção e extração do contorno das mãos, utiliza-se a *libhand*. Este processo é feito em três etapas: por meio da função *hand_detection*, obtém-se uma máscara binária

Figura 16 – Arquitetura do Xkin e detecção do contorno das mãos.



Fonte: (PEDERSOLI et al., 2012)

que representa a forma da mão e sua imagem de profundidade. Os dados brutos do contorno da mão são então extraídos por meio da função *get_hand_contour_basic*. Estes dados são processados pela função *get_hand_contour_advanced*, que utiliza uma abordagem mais sofisticada para conseguir um contorno mais definido da mão, conforme visto na Figura 16.

A *libposture* provê funções para classificar poses estáticas das mãos. O XKin fornece duas abordagens de classificação. A abordagem simples é extremamente rápida e robusta, mas limita-se ao reconhecimento de apenas duas poses: abrir e fechar as mãos. A classificação avançada distingue um conjunto maior de posturas, porém é menos robusta se comparado à técnica anterior.

A classificação da trajetória do movimento da mão é realizado por meio da *libgesture*, implementando uma estrutura flexível para lidar com Modelos Ocultos de Markov (HMM) e os algoritmos típicos para os modelos de treinamento e testes.

O código que descreve estas APIs é estruturado de forma modular, o que possibilita modificar facilmente qualquer um dos processo envolvidos. Este é um um fator-chave do *framework*, uma vez que o campo de pesquisa sobre o reconhecimento de gestos avança rapidamente (PEDERSOLI et al., 2012). Desta forma, é assegurada a possibilidade de adicionar recursos recém concebidos ao funcionamento das bibliotecas Xkin.

3.4.3 KinectJS

O projeto KinectJS combina JavaScript e Kinect para interação na web, com o objetivo final de prover controles de movimento em HTML5, resultando em uma biblioteca composta por dois módulos: Kinect.js e KinectSocketServer.

O Kinect.js é o módulo cliente que possui código aberto, desenvolvido em Javascript e que possibilita operar o Kinect por meio de uma API (KINECTJS, 2014). O KinectSocketServer (Figura 17) é módulo servidor disponibilizado sob licença proprietária que permite acesso direto ao harware por meio do Kinect for Windows SDK. A comunicação entre eles ocorre via websockets.

Figura 17 – Imagem do KinectSocketServer.

Fonte: Captura de tela no sistema operacional Windows 7

O projeto disponibiliza um demo[1], que exemplifica a utilização dos módulos. Para estabelecer a conexão entre o KinectSocketServer (servidor) e o Kinect.js (navegador), o usuário deve informar o endereço IP do servidor na página inicial (Figura 18).

É possível realizar o rastreamento de movimentos de até duas pessoas ao mesmo tempo, bem como controlar o motor do Kinect pelo navegador. O *framework* também possui gestos pré-definidos, como furtar, saltar e fugir.

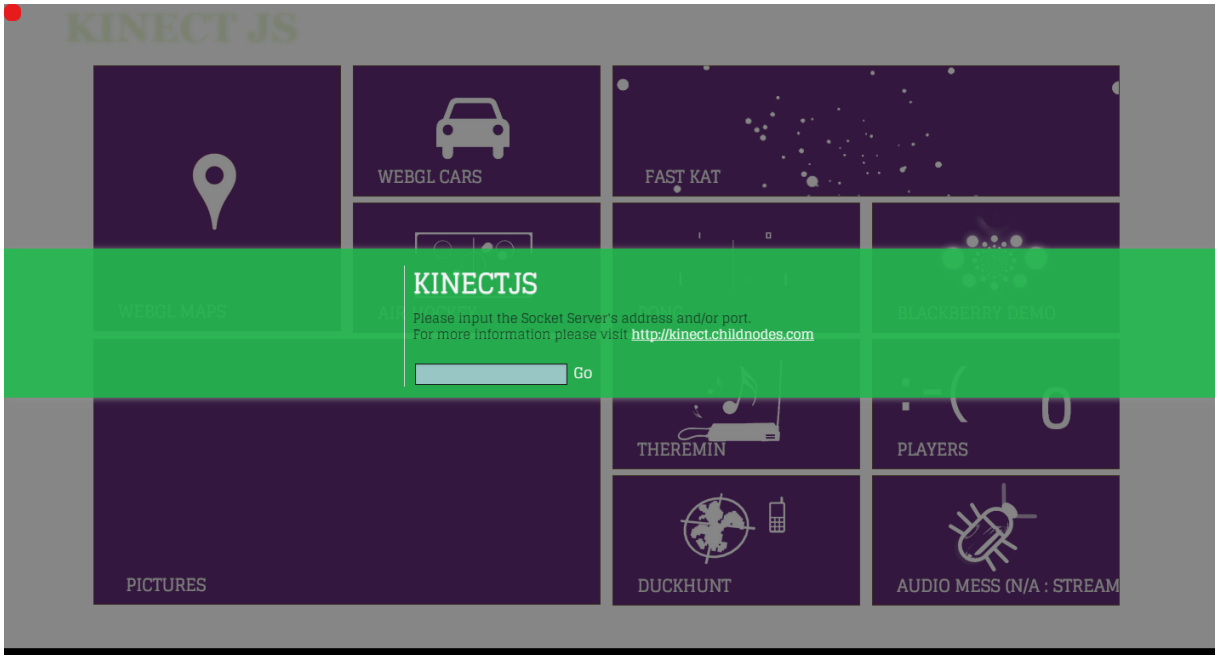
3.4.4 Kinected Browser

O *Kinected Browser* é um conjunto de ferramentas genéricas para integração de câmeras de profundidade e navegadores para Internet, com o objetivo de prover mais flexibilidade no design de interação (LIEBLING; MORRIS, 2012). O *framework* consiste em dois módulos (Figura 19): um *plugin* para o navegador que conecta-se ao Kinect SDK C++ e uma biblioteca JavaScript.

O *Kinect for Windows SDK* permite o acesso aos dados de baixo nível, incluindo o rastreio de esqueleto e imagens de cor e de profundidade. O *Kinected Browser* lê estes dados e interpreta-os como eventos de alto nível.

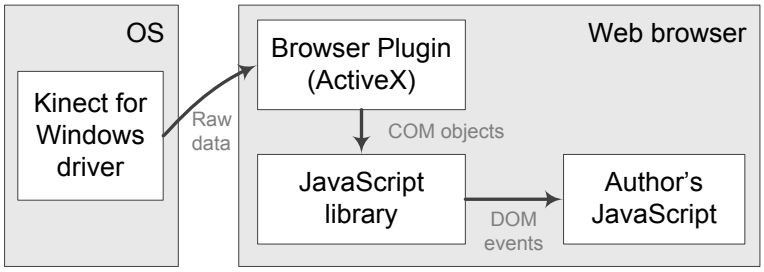
[1] Disponível em: http://kinect.childnodes.com/demo/

Figura 18 – Demo do Kinect.js.



Fonte: Captura de tela no navegador Google Chrome

Figura 19 – Arquitetura do Kinected Browser.



Fonte: (LIEBLING; MORRIS, 2012)

Quando os dados do Kinect estão prontos para leitura, o *plugin* do navegador captura-os, convertendo os dados brutos em estruturas de dados que podem ser interpretadas por ele. O sistema mapeia pontos do esqueleto e movimento em eventos que são disparados para elementos específicos da página.

Os desenvolvedores podem então escutar os eventos de movimentos de maneira similar ao que acontece com os eventos do mouse, adicionando ouvidores de eventos (*listeners*) para qualquer elemento visual da página web.

Similarmente ao que acontece com os eventos do mouse *mouseOver* e *mouseOut*, o sistema publica os eventos *jointOver* e *jointOut*, onde *joint* refere-se a cada uma das 18 articulações que o sistema pode seguir.

Embora a maior parte das funcionalidades tenha sido implementada em Javascript,

o *plugin* do navegador tem sua implementação em ActiveX, um *framework* para definição de componentes de software reutilizáveis, suportado oficialmente apenas pelo *Internet Explorer (IE)*, navegador da Microsoft. Esta característica restringe seu uso e possibilidades. O autor sugere que poderia-se implementar o *plugin* utilizando o protocolo Websocket, o que permitiria seu uso em outros navegadores além do IE.

3.4.5 XDKinect

O XDKinect é um conjunto de ferramentas para o Kinect, que permite mediar as interações entre vários dispositivos e os usuários. O *framework* é caracterizado por 3 aspectos principais (NEBELING et al., 2014a):

- Suporte a multiplataforma
- Arquitetura cliente/servidor flexível
- Abstrações de programação

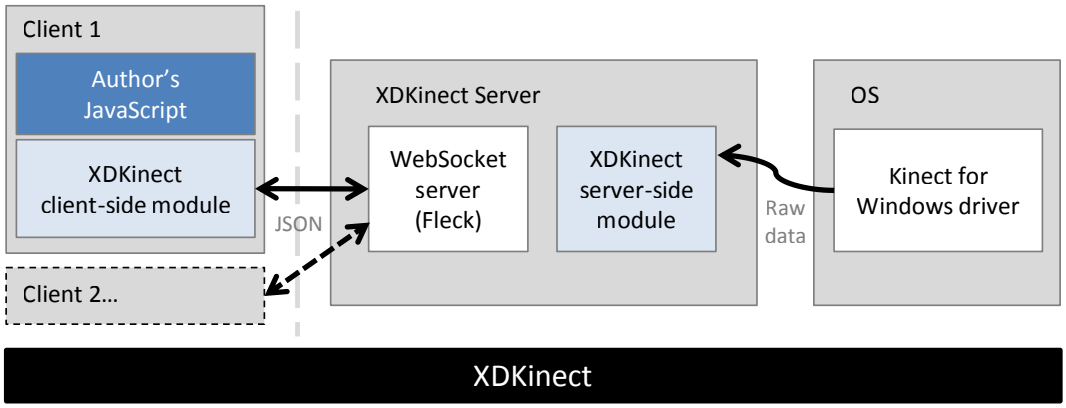
O primeiro aspecto visa suportar o desenvolvimento rápido de aplicações, dispensado ao desenvolvedor o conhecimento em várias plataformas, linguagens e tecnologias necessárias atualmente para se conseguir compatibilidade entre os vários tipos de dispositivos existentes no mercado.

A arquitetura cliente/servidor flexível (Figura 20), segundo aspecto importante no desenvolvimento do XDKinect, permite desenvolver aplicações que podem ser compartilhadas e distribuídas entre um ou vários clientes e um servidor, utilizado para hospedar o Kinect. Como o servidor do XDKinect é implementado com o *Kinect for Windows SDK*, este componente está disponível apenas para a plataforma Windows. No entanto, os sistemas operacionais suportados pelo XDKinect, do lado cliente, incluem Android, Windows e iOS. É possível utilizar este componente em qualquer navegador moderno, rodando em tablets, notebooks, paredes interativas e TV's.

Em terceiro lugar, o XDKinect proporciona abstrações que permitem ao desenvolvedor ignorar detalhes de implementação baixo-nível, por meio do *Kinect for Windows SDK*. Ele fornece uma API para configuração do Kinect, consulta e filtragem dos dados capturados do dispositivo, reconhecimento de poses, gestos e voz.

Um dos problemas com esta solução, segundo os próprios autores, é a inabilidade de criar gestos e poses customizadas e os poucos gestos disponíveis para uso com a ferramenta. Isto restringe as possibilidades de utilização em alguns cenários e até mesmo inviabiliza o uso em aplicações que necessitam de maior flexibilidade.

Figura 20 – Arquitetura do XDKinect.



Fonte: (NEBELING et al., 2014b)

3.5 Considerações finais

As NUI constituem-se uma área de pesquisa relativamente nova, que procura diminuir a curva de aprendizagem entre o usuário e as interfaces construídas, traduzindo movimentos corporais em ações e reutilizando habilidades existentes para interagir de maneira apropriada com o conteúdo.

Os recentes avanços na interação homem-computador e a criação de novos dispositivos têm facilitado o desenvolvimento de Interfaces Naturais de Usuário, proporcionando maneiras mais intuitivas de interagir com os computadores (VILLAROMAN; ROWE; SWAN, 2011).

Uma das dificuldades ao se desenvolver este tipo de interface é a modelagem de poses e gestos, uma vez que grande parte dos sistemas implementam gestos e poses pré-definidos (LEE; LIU; ZHANG, 2012) (KAMEL BOULOS et al., 2011), e não permitem sua personalização.

Além disso, as soluções existentes para modelagem são predominantemente limitadas ao ambiente desktop. As poucas soluções voltadas para a web, como o KinectJS e o XDKinect não buscam resolver a questão da modelagem, focando apenas em conectar os dispositivos NUI à web.

Outrossim, as soluções de modelagem, como o FAAST, requerem conhecimento expressivo do usuário, o que inviabiliza o seu uso por grande parte das pessoas. Deste modo, a construção de um ambiente para modelagem de poses e gestos preenche a lacuna existente neste universo de estudo.

Outra questão relevante neste contexto é a troca de dados entre as aplicações, assunto pouco abordado nos trabalhos pesquisados. Não há um padrão aberto que possibilite a troca de informações sobre poses, gestos e ações que representam em um sistema.

Ante ao exposto, evidencia-se a carência ambientes que permitam ao usuário personalizar seus próprios gestos ou poses e compartilhá-los por meio de um padrão de dados bem definido, que possa ser facilmente validado por aplicações terceiras consumidoras.

4 Proposta de solução

Diante da limitação das propostas atuais voltadas para o desenvolvimento de sistemas com interface natural no que diz respeito à modelagem gestual, foi concebido o NuiMod, um ambiente capaz de realizar as seguintes tarefas:

1. Modelar componentes
2. Capturar dados de sensores de movimento
3. Processar dados de esqueleto

Na etapa de **modelagem de componentes**, o utilizador define poses e gestos e quais gatilhos serão executados no momento em que estes componentes forem localizados. O gatilho representa um procedimento da aplicação, executado sempre que o evento associado ocorrer. Os componentes devem ser modelados segundo um formato pré-definido.

Os dados de esqueleto capturados pelo sensor de movimento são disponibilizados para a aplicação na etapa de **captura de dados**. Por fim, na etapa de **processamento de dados**, as informações capturadas são comparados com os componentes modelados. Caso sejam iguais, dispara-se um gatilho para que a ação relacionada a ele seja executada.

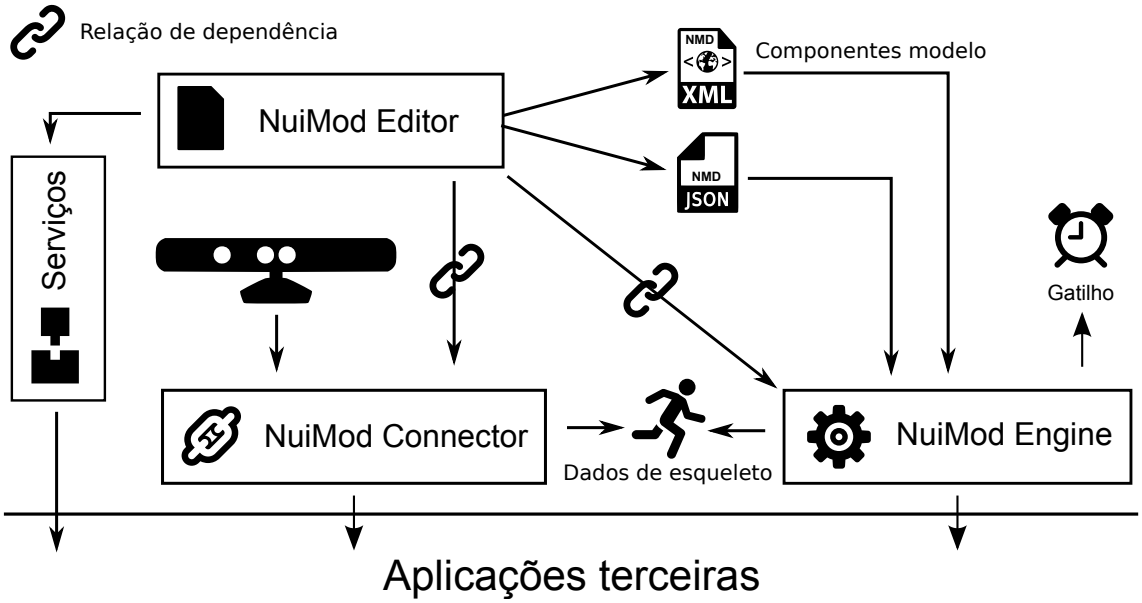
Para lidar com cada uma das etapas mencionadas, foram criados três componentes: o *NuiMod Editor*, o *NuiMod Connector* e o *NuiMod Engine*, além de uma especificação: o modelo NMD. A arquitetura do ambiente foi concebida para ser facilmente extensível, permitindo personalizar todas as etapas do processo, desde o formato de captura de dados até as estratégias de reconhecimento dos componentes. A Figura 21 representa o ambiente e suas dependências.

O NuiMod especifica ainda um modelo para compartilhamento e validação de dados entre aplicações terceiras, permitindo o uso da *Engine* em cenários além do *Editor*. As seções a seguir descrevem cada um dos componentes que integram o NuiMod e suas funcionalidades.

4.1 Modelagem de componentes

Na etapa de modelagem de componentes, o usuário especificará quais poses e gestos serão utilizados para executar ações dentro de um determinado sistema. Para isto, utiliza-se o modelo NMD (Figura 22), cujo objetivo é representar componentes no NuiMod, bem como definir a relação entre projetos e seus componentes.

Figura 21 – Arquitetura do NuiMod



Fonte: Elaborada pelo autor

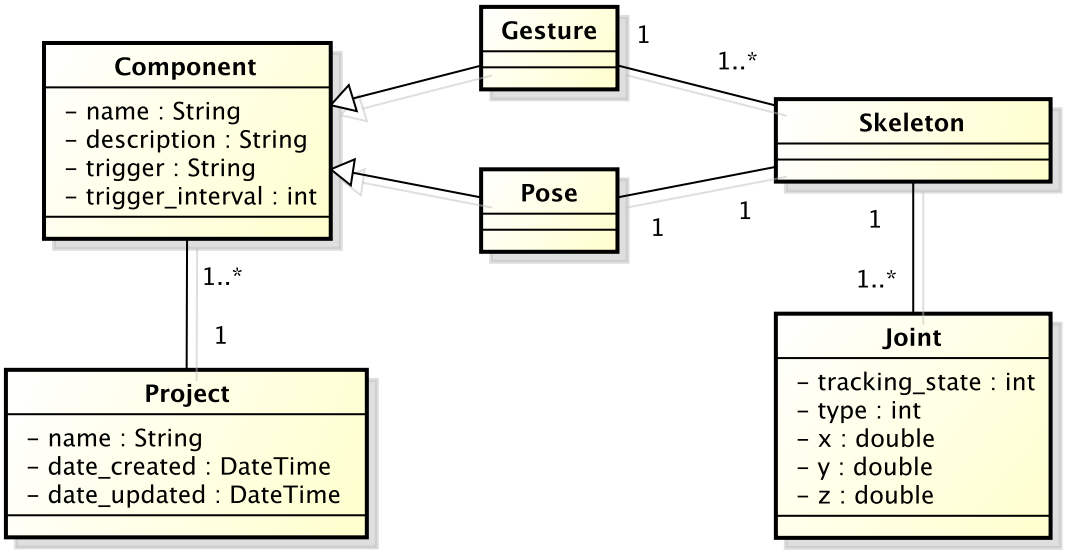
No modelo NMD cada projeto possui vários componentes, divididos em poses ou gestos. Cada componente possui um nome, descrição, gatilho e intervado do gatilho. O nome e a descrição ajudam na identificação do componente e são obrigatórios. O gatilho indica um rótulo para o evento a ser disparado quando o componente for encontrado pela *Engine*. Seu preenchimento é obrigatório e pode conter somente letras, números, sublinhados ou hífens. O intervalo do gatilho indica, em milisegundos, quanto tempo deve-se aguardar até disparar novamente o mesmo evento. Este campo é útil para ações não contínuas e, quando nulo, indica que o gatilho deve ser disparado ininterruptamente enquanto o usuário permanecer na mesma posição.

A respeito dos componentes, uma pose possui apenas um modelo de esqueleto e um gesto dispõe de um a vários esqueletos. Cada esqueleto é constituído de articulações, que armazenam o tipo, estado e suas posições x, y e z (expressas em metros) no plano cartesiano. As articulações podem ser de 20 tipos diferentes e possuem 3 estados: Não rastreado, Inferido e Rastreado.

As articulações não rastreadas indicam aquelas que não puderam ser encontradas pelo sensor. O estado inferido designa as articulações que, apesar de não encontradas pelo sensor, puderam ser inferidas a partir da posição de articulações vizinhas. Por fim, o estado rastreado assinala aquelas articulações que puderam ser monitoradas pelo dispositivo.

O modelo de esqueleto adotado pelo NuiMod (Figura 23) foi escolhido com base no modelo utilizado pelo Kinect. Esta escolha representa uma restrição para a solução, uma vez que cada dispositivo NUI possui um modelo de esqueleto diferente. No futuro, espera-se manipular diferentes modelos de esqueleto por meio de metamodelos. Uma vez que o

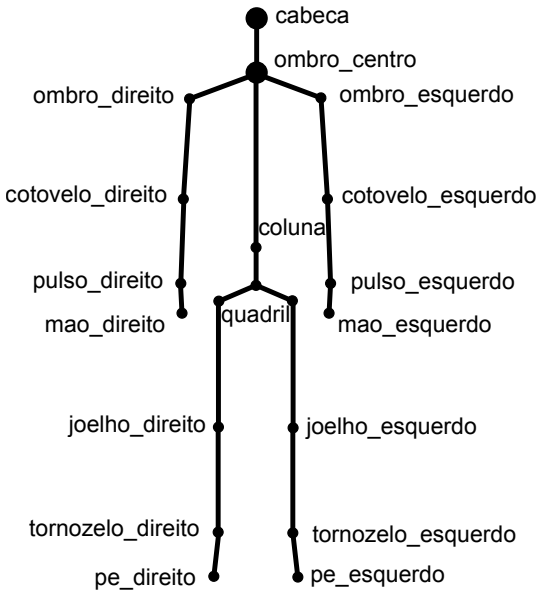
Figura 22 – Representação do modelo NMD



Fonte: Elaborada pelo autor

metamodelo do esqueleto for definido, modelos de esqueleto para diferentes dispositivos NUI deverão estar de acordo com este metamodelo, permitindo ao NuiMod lidar com cada um deles.

Figura 23 – Modelo de esqueleto com 20 articulações adotado pelo NuiMod



Fonte: Elaborada pelo autor

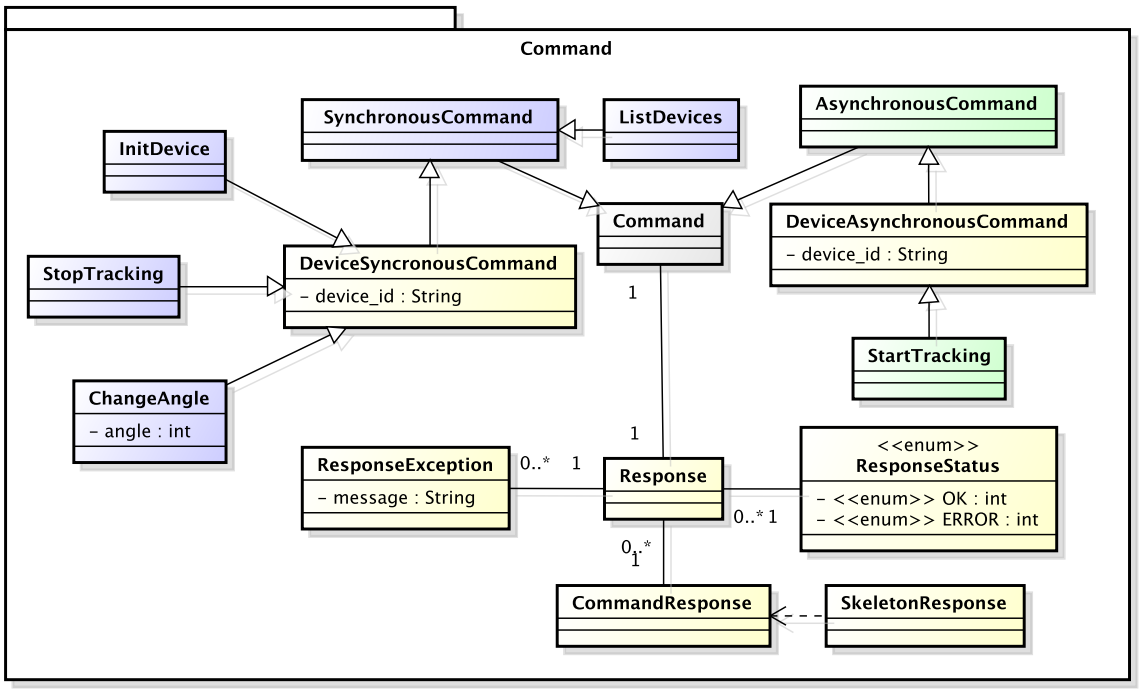
4.2 Captura dos dados

As informações sobre a posição do usuário (rastreio de esqueleto) são capturadas e disponibilizadas para a aplicação web na etapa de captura de dados. Para isto, emprega-se

o *NuiMod Connector*, componente responsável por receber um comando, executar a ação desejada e retornar as informações capturadas por meio do sensor de movimentos.

Os comandos a serem enviados ao *NuiMod Connector* podem ser de dois tipos: síncronos ou assíncronos. Os comandos síncronos incluem aqueles cuja resposta é enviada ao cliente assim que executado. Nesta categoria estão os comandos de listagem e inicialização de dispositivos, e término de rastreio do esqueleto. O comando de inicio de rastreio do esqueleto é assíncrono, uma vez que irá enviar respostas ao cliente até que o comando de término de rastreio do esqueleto seja recebido pelo servidor. A Figura 24, ilustra os comandos e respostas emitidas pelo *NuiMod Connector*.

Figura 24 – Tipos de comandos e respostas do NuiMod Connector



Fonte: Elaborada pelo autor

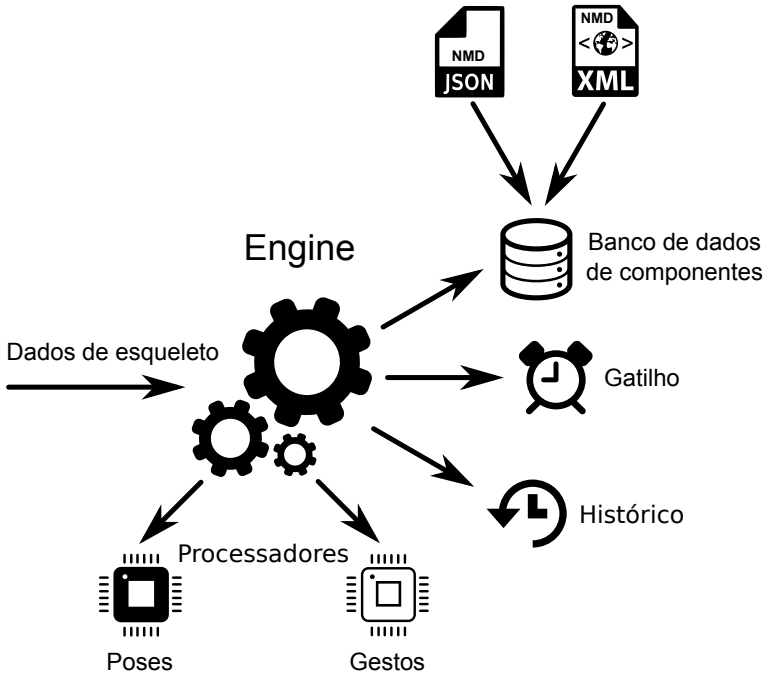
4.3 Processamento dos dados

A etapa de processamento de dados é responsável pelo processamento das informações de esqueleto. O componente empregado para esta tarefa é o *NuiMod Engine*, cuja arquitetura flexível permite personalizar e estender os recursos disponibilizados, desde os formatos de entrada até os processadores de componentes (poses e gestos). Sua arquitetura é apresentada na Figura 25.

Os principais elementos que integram o *NuiMod Engine* são:

- Banco de dados de componentes

Figura 25 – Arquitetura que representa os elementos do *NuiMod Engine*



Fonte: Elaborada pelo autor

- Histórico
- Gatilho
- Processadores

O banco de dados de componentes armazena os componentes-modelo: poses, gestos e seus gatilhos, que serão utilizados para comparação com os dados de entrada. Os dados de entrada devem seguir o formado NMD, especificado na seção anterior.

No histórico ficam armazenados os dados de esqueleto recebidos pela *Engine*. Por padrão, é suportado o armazenamento de 300 poses, ou 10 segundos de dados com o sensor Kinect, cuja taxa de transmissão padrão é de 30 quadros por segundo. Isto é suficiente para o processamento da maior parte das poses e gestos, no entanto este valor pode ser modificado de acordo com a necessidade do desenvolvedor. Quando a pilha de armazenamento chega ao limite, os dados mais antigos são eliminados.

Os processadores são utilizados para reconhecimento de componentes. Ao receber um novo dado de esqueleto, a *Engine* o envia aos processadores que verificam sua similaridade com o banco de componentes. As estratégias para reconhecimento de poses e gestos serão discutidas nas seções 4.3.1 e 4.3.2.

Muito embora os dados a serem processados sejam normalmente recebidos do *Connector*, a *Engine* não possui uma dependência direta deste componente, sendo possível

receber os dados de qualquer outro local. Isto facilita, por exemplo, a criação de casos de testes, que podem ser realizados com dados capturados do sensor e armazenados em um banco de dados.

No tocante aos eventos, a *Engine* segue o padrão de projeto *observer*. Segundo (FAISON, 2011):

> Na linguagem comum, um evento é uma ocorrência de algum tipo, enquanto a notificação é uma mensagem informando o destinatário que algo (presumivelmente importante) aconteceu. No contexto de software, as definições de eventos e notificações são inextricavelmente ligados, com um definido em termos do outro. Os eventos são uma causa; notificações são um efeito.

Assim, quando um processador constata similaridade entre um dado recebido e um modelo no banco de componentes (evento), a informação é retornada à *Engine*, que dispara um gatilho (notificação). Todos os observadores são então notificados, para que possam ou não executar uma ação.

4.3.1 Detecção de estabilidade

Para detecção de estabilidade, a cada novo dado de esqueleto recebido, o sistema calcula a similaridade entre as posições das articulações da última pose disponível no histórico e a posição das articulações adquiridas.

Em termos matemáticos, simetria consiste em um espaço métrico (X, d), onde X é um conjunto munido de uma métrica, ou seja, uma função $d : XxX \rightarrow \mathbb{R}$ tal que, para quaisquer $x, y, z \in \mathbb{R}$:

- (x, y) é um número real, não negativo
- $d(x, y) = 0 \iff x = y$
- $d(x, y) = d(y, x) = 0$ (simetria)
- $d(x, z) \leqslant d(x, y) + d(y, z)$

É possível determinar as poses similares e diferentes a partir da definição da função distância. Aquelas que possuem distâncias pequenas são consideradas similares. A escolha da função d encerra um grau de liberdade muito grande, o que dificulta sua escolha (de Oliveira Marin, 2006). O conjunto de articulações para o qual foi definida a função d é determinado como espaço métrico e representado pelo par $\{conjunto, distancia\}$.

Assim, para efetuar o reconhecimento de similaridade é necessário especificar um espaço métrico, em símbolos: $< J, d >$; onde J é o conjunto de articulações consideradas e d a função distância do espaço métrico conhecido como função de similaridade.

A abordagem mais simples para esta finalidade é empregar uma métrica de Minkowski, tais como a distância euclidiana (PARVINI; SHAHABI, 2007). A distância euclidiana entre dois vetores (neste caso, pode-se interpretar cada vetor como a posição de uma articulação do esqueleto) $|a_1, a_2, a_3, a_n| T$ e $|e_1, e_2, e_3, a_n| T$ é definida por:

$$\sqrt{\sum_{n}^{i=1}(a_i - e_i)^2}$$

Como os dados de articulação são tridimensionais (horizontal, vertical e de profundidade), a distância euclidiana é computada como:

$$\sqrt{(a_x - e_x)^2 + (a_y - e_y)^2 + (a_z - e_z)^2}$$

onde a é a articulação anterior e e é a articulação atual. Logo, para cada frame recebido calcula-se a distância euclidiana para cada uma das 20 articulações rastreadas. A *Engine* considera similares aquelas articulações cuja distância euclidiana esteja dentro do desvio máximo configurado. Pode-se ainda configurá-la para ignorar os cálculos seguintes assim que encontrar uma articulação cuja distância euclidiana esteja além dos limites estabelecidos no *Editor*.

4.3.2 Reconhecimento de poses

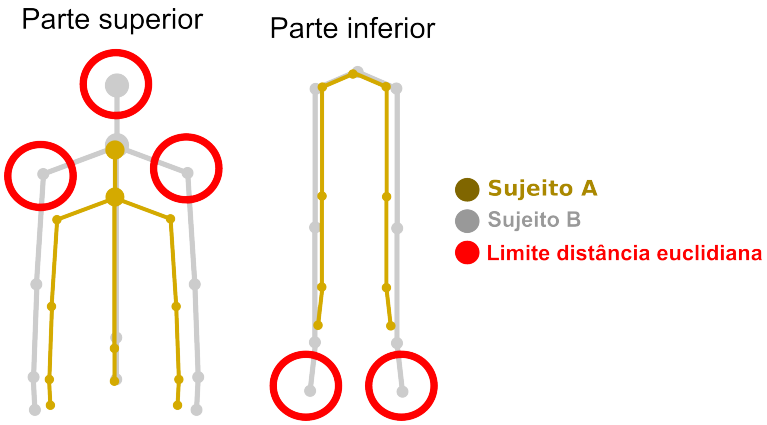
Para a *Engine*, uma pose é composta de n articulações, cada uma contendo uma posição no plano cartesiano, representadas nos três eixos espaciais (x, y e z). O reconhecimento de poses é realizado de maneira análoga à detecção de estabilidade, podendo-se empregar o mesmo algoritmo utilizado àquele propósito. Todavia, a distância euclidiana leva em consideração as coordenadas x, y e z no plano cartesiano, que podem variar de sujeito para sujeito, uma vez que essas posições estão relacionadas à características de cada pessoa, como a altura.

Assim, quando uma pessoa de determinada altura modelar seus componentes e outra pessoa de altura díspar utilizá-los, caso a *Engine* empregue a estratégia da distância euclidiana para comparar os dados de esqueleto, poderá não ser detectada nenhuma similaridade entre eles, uma vez que a diferença de altura pode exceder o desvio máximo informado pelo desenvolvedor.

A Figura 26 ilustra o problema; apesar dos sujeitos A e B estarem na mesma posição, a *Engine* não dispararia um gatilho, uma vez que o limite da distância euclidiana nas articulações demarcadas não englobariam as juntas comparadas.

Para resolver esta questão, foi empregado o cálculo de ângulo entre vetores, levando-se em consideração cada vetor que compõe o esqueleto. Um vetor (geométrico) no espaço $\mathbb{R}$

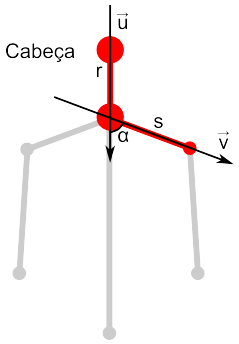
Figura 26 – Problema da distância euclidiana em sujeitos de tamanhos diferentes



Fonte: Elaborada pelo autor

é uma classe de objetos matemáticos (segmentos de reta) que tem a mesma direção, mesmo sentido e mesma intensidade. Para cálculo do ângulos, foram utilizadas as coordenadas de pontos que possuem ligação com, pelo menos, dois outros pontos, formando dois seguimentos de reta, conforme observado na Figura 27.

Figura 27 – Seguimentos de reta do esqueleto para cálculo do ângulo



Fonte: Elaborada pelo autor

Desta forma, para cada tripla de coordenadas, teremos duas retas. Com duas retas em um espaço, podem ocorrer os seguintes casos:

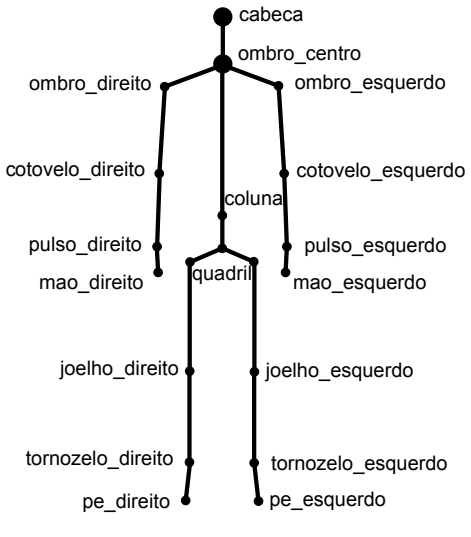
- As retas são concorrentes, ou seja, se interceptam em um ponto;
- As retas são paralelas (ou coincidentes);
- As retas são inversas, isto é, não são palalelas, mas também não se interceptam.

Se as retas são concorrentes, como neste caso, então elas determinam quatro ângulos, dois a dois opostos pelo vértice. O ângulo entre elas é definido então pelo menor destes valores. Considerando os vetores diretores das retas r e s, e aplicando a relação que determina o ângulo entre os vetores diretores, temos:

$$cos\alpha = \frac{|\vec{u} * \vec{v}|}{|\vec{u}| \cdot |\vec{v}|}, \text{ onde } 0 \leqslant \alpha \leqslant 90^{o}.$$

Assim, para o modelo de esqueleto com 20 articulações utilizado pela *Engine*, foram definidas 17 triplas de coordenadas, conforme determinado na Figura 28.

Figura 28 – Triplas utilizadas para cálculo de similaridade entre esqueletos



Tripla 1: cabeca, ombro_centro, ombro_direito
Tripla 2: cabeca, ombro_centro, ombro_esquerdo
Tripla 3: ombro_centro, coluna, quadril_centro
Tripla 4: ombro_centro, ombro_direito, cotovelo_direito
Tripla 5: ombro_direito, cotovelo_direito, pulso_direito
Tripla 6: cotovelo_direito, pulso_direito, mao_direito
Tripla 7: ombro_centro, ombro_esquerdo, cotovelo_esquerdo
Tripla 8: ombro_esqurdo, cotovelo_esquerdo, pulso_esquerdo
Tripla 9: cotovelo_esquerdo, pulso_esquerdo, mao_esquerdo
Tripla 10: coluna, quadril_centro, quadril_direito
Tripla 11: quadril_centro, quadril_direito, joelho_direito
Tripla 12: quadril_direito, joelho_direito, tornozelo_direito
Tripla 13: joelho_direito, tornozelo_direito, pe_direito
Tripla 14: coluna, quadril_centro, quadril_esquerdo
Tripla 15: quadril_centro, quadril_esquerdo, joelho_esquerdo
Tripla 16: quadril_esquerdo, joelho_esquerdo, tornozelo_esquerdo
Tripla 17: joelho_esquerdo, tornozelo_esquerdo, pe_esquerdo

Fonte: Elaborada pelo autor

Caso seja necessário, o NuiMod permite que seja implementada outra estratégia de reconhecimento de poses, de acordo com a necessidade da aplicação.

4.3.3 Reconhecimento de gestos

O reconhecimento de gestos é realizado de maneira similar ao reconhecimento de poses. A *Engine* considera que um gesto g é formado por n poses p que precisam ser executadas sequencialmente, em um intervalo de tempo t predefinido (Figura 29).

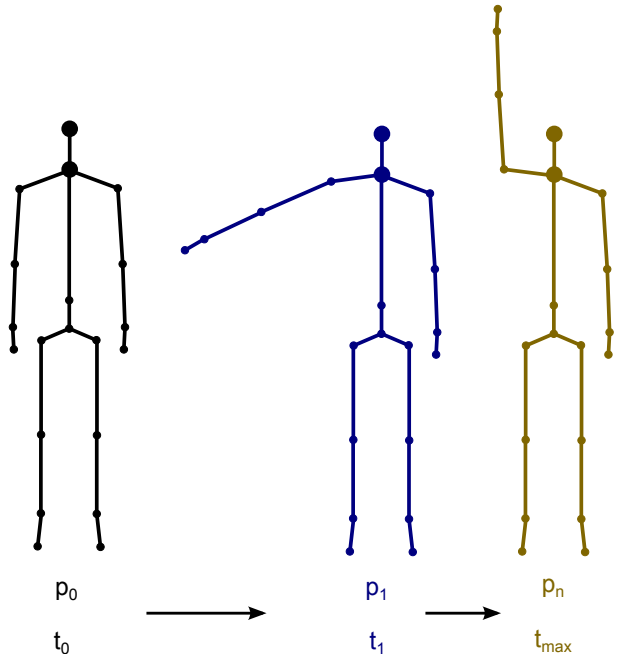
Neste caso, a *Engine* processa cada novo dado recebido comparando-a com a pose atual do gesto desejado. Caso sejam similares, a *Engine* armazena a pose e passa à seguinte até que todas as poses do gesto tenham sido executadas.

Se o gesto exceder o tempo limite *tmax* indicado pelo desenvolvedor para ser executado, a *Engine* inicia novamente a escuta por um novo gesto ou pose. A cada pose p similar encontrada, é disparado um gatilho parcial, indicando que a pose atual foi reconhecida. Os gatilhos parciais possuem a seguinte sintaxe:

```
"<nome_do_gatilho>:<pose_atual>"
```

Onde `nome_do_gatilho` indica qual o gatilho será disparado pela *Engine* e `pose_atual` é um número (de 0 a n) designando a pose encontrada p do gesto em execução.

Figura 29 – Exemplo de gesto no NuiMod.



Fonte: Elaborada pelo autor

Outra estratégia para reconhecimento de gestos seria o processamento em bloco de quadros recebidos. Neste caso, poderia-se selecionar amostras no bloco, evitando o processamento de todos os frames, como acontece no método adotado. Todavia, apesar deste método ser mais otimizado, acarretaria um atraso no reconhecimento do gesto, uma vez que seria necessário armazenar os quadros para processamento posterior. Por este motivo, optou-se por processar os quadros em tempo real, à medida em que forem recebidos pela *Engine*.

4.4 Solução tecnológica para o NuiMod

Neste capítulo apresentamos a implementação dos componentes especificados na seção anterior, bem como as tecnologias utilizadas em cada um deles.

4.4.1 Modelo NMD

O modelo NMD utiliza o formato XML (*eXtensible Markup Language*) para representação dos dados. A Figura 30 ilustra parcialmente um projeto criado no *NuiMod Editor* e exportado no padrão NMD. Como alternativa, o formato NMD também pode ser codificado em JSON, um formato legível para humanos e compatível com JavaScript. Esta codificação é comumente utilizada em projetos web, e utilizado como padrão pelo *Editor* para comunicação com a *Engine*.

Figura 30 – Exemplo parcial de projeto no formato NMD

```
<?xml version="1.0" encoding="utf-8"?>
<project date_created="2014-09-25T01:16:08" date_updated="2014-09-25T01:16:08"
    name="Projeto Levantamento de Teste">
     <!--Generated with NuiMod Editor-->
     <components>
          <pose description="Posição inicial" name="Posição inicial"
              trigger="posicao-inicial" trigger_interval="1000">
               <skeleton>
                    <joints>
                         <joint tracking_state="2" type="0">
                              <x>-0.153762236</x>
                              <y>0.04709116</y>
                              <z>2.047703</z>
                         </joint>
                         ...
                         <joint tracking_state="2" type="19">
                              <x>-0.0175785273</x>
                              <y>-0.7952746</y>
                              <z>1.99070609</z>
                         </joint>
                    </joints>
               </skeleton>
          </pose>
     </components>
</project>
```

Fonte: Elaborada pelo autor

Uma tarefa de grande importância ao compartilhar dados com outros sistemas é garantir sua conformidade com o modelo definido. Quando uma validação é realizada, pressupõe-se que qualquer outro sistema, empregando as mesmas regras estruturais, pode utilizar o XML criado (HOSKINS, 2013).

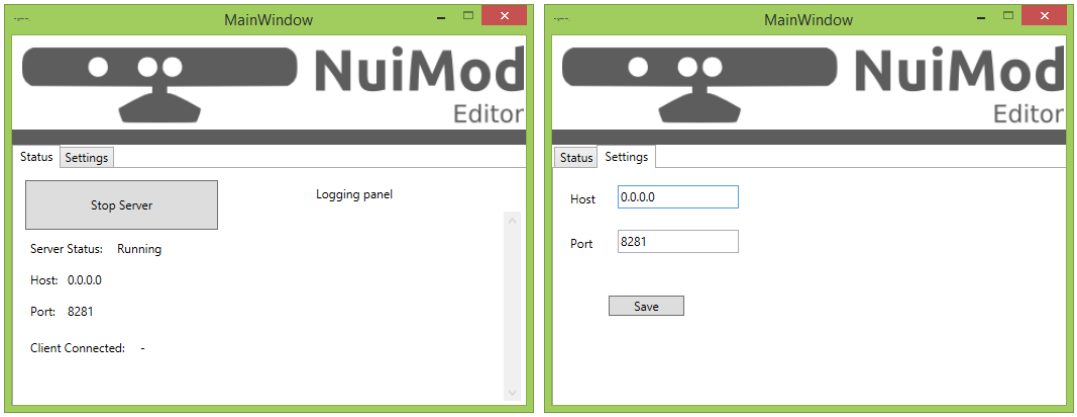
Para este propósito, o NuiMod fornece um *XML Schema* para validação de arquivos exportados pelo projeto. A descrição completa do *schema* pode ser encontrada no Apêndice A.

4.4.2 NuiMod Connector

O *NuiMod Connector* é o componente responsável por transmitir os dados do sensor de movimentos para a aplicação web (Figura 31). Do lado servidor, é implementado em linguagem C# e comunica-se com o Kinect via por meio do *Kinect for Windows SDK*. Para interagir com o módulo cliente, faz uso de um protocolo baseado em troca de mensagens, realizada via websockets, tecnologia que permite a comunicação bidirecional entre um cliente executando um código não confiável, em um ambiente não controlado, e um host remoto (FETTE; MELNIKOV, 2011).

Os comandos são transmitidos entre o cliente e o servidor por meio de mensagens JSON, um formato leve para intercâmbio de dados computacionais. O cliente deve informar, ao enviar uma mensagem, qual o tipo de comando e, quando houver, quais os parâmetros

Figura 31 – Tela principal e tela de configuração do *NuiMod Connector*



Fonte: Captura de tela no sistema operacional Windows 8.1

para a execução daquela instrução. A Figura 32 apresenta um exemplo de mensagem para o comando *ChangeAngle*, que altera a inclinação do Kinect:

Figura 32 – Exemplo de mensagem para envio de comando ao NuiMod Connector

Fonte: Elaborada pelo autor

Em relação ao projeto de software, o *Connector* está dividido em 4 pacotes, a saber:

- **Models** - define as estruturas de dados necessárias para enviar e receber comandos. Isto inclui as informações sobre o dispositivo, articulações (com seus tipos e estados, baseados no modelo NMD) e dados do esqueleto.
- **Command** - inclui as classes com os comandos a serem interpretados pelo *Connector*. É responsável pela transformação de dados JSON em instruções C# e vice-versa.
- **Device** - armazena as interfaces para implementação de comandos em diferentes sensores.

Apesar de estar voltado especificamente para o Kinect, o *Connector* define as interfaces necessárias para implementação com outros dispositivos. Isto permitirá, no futuro, estender a sua utilização em conjunto com sensores além daquele desenvolvido pela Microsoft. Além disso, por possuir estruturas de dados bem definidas, o módulo pode ser

reimplementado em diferentes linguagens, como Java ou Python, permitindo sua expansão para dispositivos que não possuem SDK's em C#.

No tocante à distribuição, o *Connector* pode ser instalado na máquina cliente a partir de um arquivo executável ou inicializado diretamente pelo navegador, por meio do *ClickOnce Deployment*, tecnologia de implantação que permite a criação de aplicativos AutoAtualizáveis, que podem ser instalados e executados com mínima interação do usuário, por meio de um link na web (NOYES, 2006).

Do lado cliente, o *Connector* é implementado em Javascript, sendo utilizado pela aplicação web como uma ponte para receber e enviar dados ao servidor. O módulo cliente possui as estruturas de dados necessárias para processar as respostas do servidor e os conversores para transformar objetos Javascript em JSON.

4.4.3 NuiMod Engine

O *NuiMod Engine* é o módulo, desenvolvido em Javascript, responsável por processar os dados de esqueleto. Ao instanciar um objeto deste tipo, deve-se informar quais são os componentes modelo, que devem seguir o modelo NMD e podem estar codificados em XML ou JSON:

```
engine = new NuiMod.Engine(projeto_nmd);
```

Os componentes serão então armazenados em um banco de dados, que será utilizado posteriormente para comparação com os dados obtidos do *NuiMod Connector*. A seguir, os observadores devem subscrever-se para escutar os eventos, por meio do método `on` da *Engine*:

```
NuiMod.Engine.on("<nome_do_gatilho>", function() { ...})
```

O `nome_do_gatilho` indica qual o gatilho será disparado pelo componente e o segundo parâmetro do método aponta a função a ser executada quando o evento acontecer. Caso seja necessário observar um gatilho parcial de uma pose, os observadores podem subscrever-se da seguinte maneira:

```
NuiMod.Engine.on("<nome_do_gatilho>:<pose_atual>", function() { ...})
```

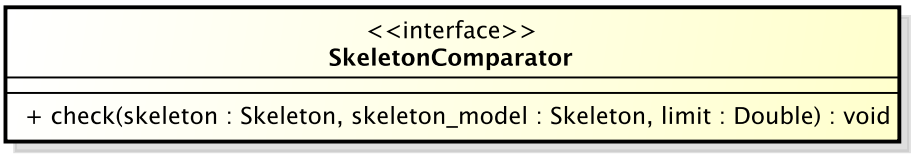
É possível ainda que um observador subscreva-se para escutar todos os gatilhos disparados pela *Engine*, bastando utilizar o operador `*` como `nome_do_gatilho`, conforme exemplificado abaixo:

```
NuiMod.Engine.on("*", function(<nome_do_gatilho>) { ...})
```

Neste caso, o nome do gatilho (`nome_do_gatilho`) é informado como parâmetro para a função executada assim que o gatilho for disparado pela *Engine*.

Caso seja necessário, o desenvolvedor pode implementar sua própria estratégia de reconhecimento de componentes, por meio da interface `NuiMod.Engine.SkeletonComparator` (Figura 33) disponibilizada pela *Engine* para este propósito. Esta interface define o método *check* que deverá receber como parâmetros os esqueletos a serem comparados, o limite máximo de desvio e retornar *True* ou *False* caso os esqueletos sejam similares ou não, respectivamente.

Figura 33 – Inteface para comparação de poses



Fonte: Elaborada pelo autor

4.4.4 NuiMod Editor

O *NuiMod Editor* é um ambiente web para a modelagem de poses e gestos que podem ser exportados no formato NMD. Ele oferece uma interface gráfica de usuário para manipulação do modelo NMD, facilitando a criação de componentes neste formato. A subseção a seguir explanará sobre as tecnologias utilizadas para o desenvolvimento desta solução.

4.4.4.1 Tecnologias empregadas

Para o desenvolvimento da solução, optou-se por utilizar a linguagem de programação Python em conjunto com o *framework* Django. Esta decisão de projeto levou em consideração questões como produtividade, simplicidade e organização. O *framework* Django já inclui, por padrão, funcionalidades comuns a projetos desta natureza, como suporte a vários bancos de dados, evolução de esquema (migrações), templates e controle de acesso.

Python é uma linguagem de programação orientada a objetos, interpretada, fortemente tipada e dinâmica, focada em legibilidade de código com uma sintaxe clara e concisa (LUTZ, 2013). O *framework* Django é importante implementação em Python, bastante utilizado para desenvolvimento rápido de aplicações web (SANTANA; GALESI, 2010). Ele segue o padrão de desenvolvimento *Model-Template-View* (MTV), que possui semelhanças com o padrão MVC - *Model-View-Controller*, fornecendo separação clara de tarefas e responsabilidades entre os aspectos importantes de um aplicativo (ALCHIN; KAPLAN-MOSS; VILCHES, 2009).

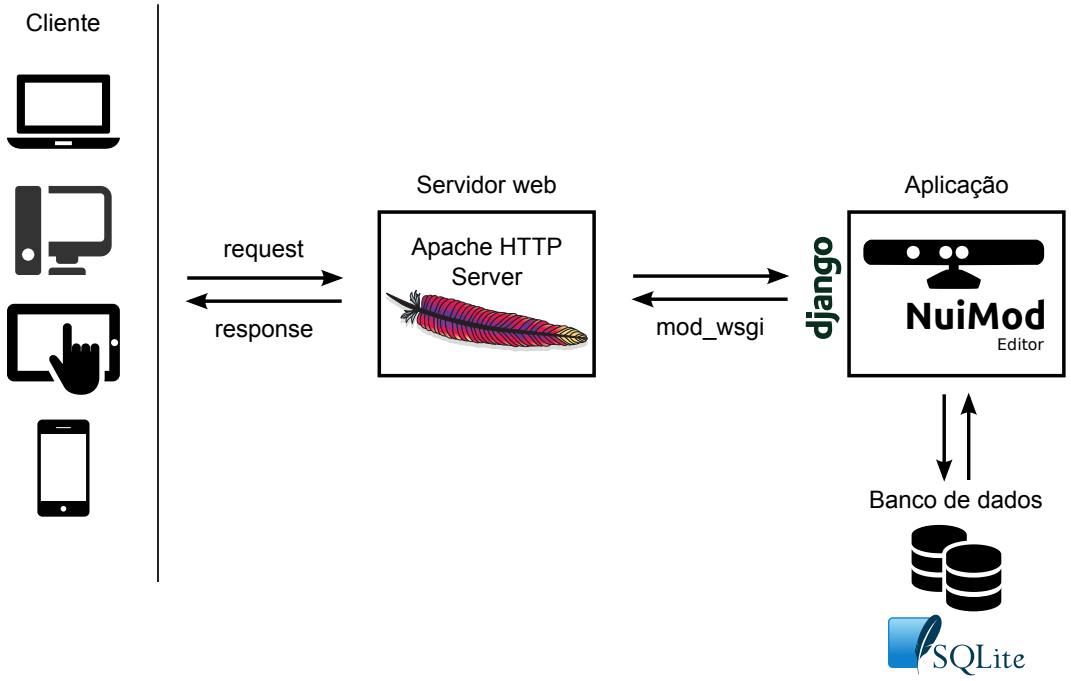
Do lado cliente, para a estruturação e apresentação do conteúdo optou-se por utilizar HTML5 em conjunto com o Bootstrap, *framework* CSS para *design* de aplicações que permite a adequação do layout de acordo com a tela dos dispositivos, característica conhecida como *design* resposivo. Muito embora o *Connector* ainda não esteja disponível em versão móvel, esta caraterística permite que os usuários acessem os projetos criados no *Editor* em dispositivos como tablets e celulares.

Para manipulação de elementos DOM, criação de animações, gerencimento de eventos e AJAX, foi escolhido o *framework* jQuery, biblioteca JavaScript *cross-browser* (resolução da incompatibilidade entre os navegadores), desenvolvida para simplificar o desenvolvimento de scripts que rodam do lado cliente e interagem com o HTML.

4.4.4.2 Arquitetura do *NuiMod Editor*

A Figura 34 apresenta a arquitetura do ambiente de modelagem de poses e gestos, bem como as tecnologias empregadas em cada elemento integrante. Isto permite a visualização das dependências do *Editor*, vinculadas a cada uma das tecnologias adotadas.

Figura 34 – Arquitetura do *NuiMod Editor*



Fonte: Elaborada pelo autor

A arquitetura está dividida em duas partes principais: o lado servidor, composto pelos módulos desenvolvidos com o *framework* Django, e o lado cliente, onde localizam-se as páginas HTML5 que serão visualizadas pelo navegador do usuário.

Como servidor web, optou-se por utilizar o *Apache HTTP Server*, projeto comumente utilizado para distribuição de páginas web em servidores Linux. Para comunicação

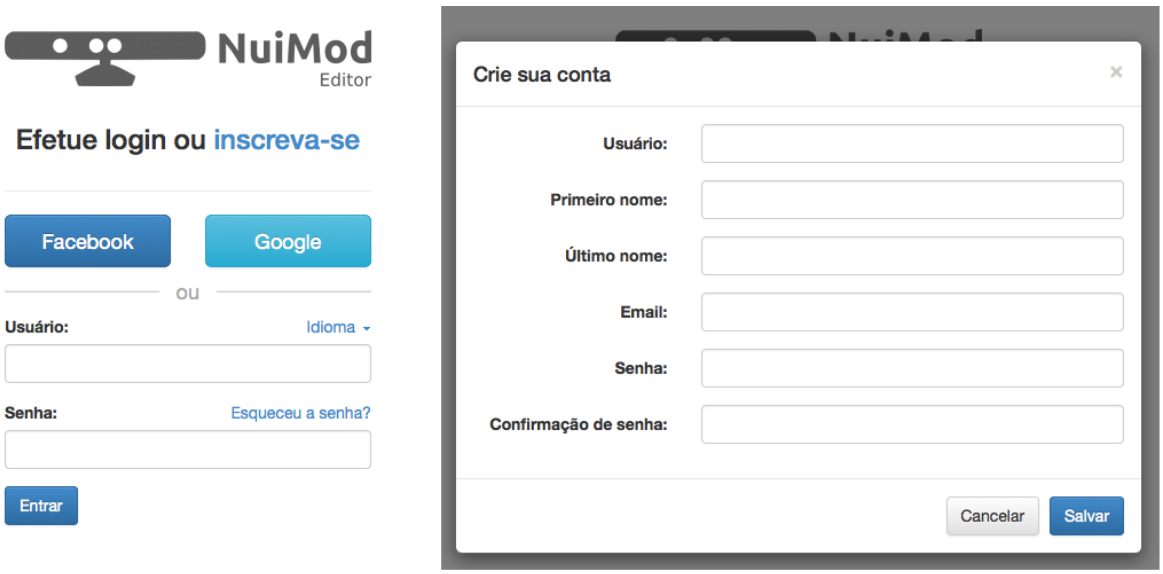
com a aplicação Django, foi empregado o módulo `mod_wsgi`, especificamente projetado para hospedar aplicações WSGI (*Web Server Gateway Interface*) e recomendado para implantação de aplicações desenvolvidas com o *framework*.

Como banco de dados adotou-se o SQLite, que permite acesso a banco de dados SQL sem executar um processo SGBD separado, proporcionando simplicidade na implantação do projeto. Apesar disso, o Django permite utilizar outros bancos de dados, como MySQL ou PostgreSQL, podendo ser empregados em contextos que necessitem de maior robustez.

4.4.4.3 Acesso ao sistema

Ao acessar o ambiente, o usuário é redirecionado para a tela de login, por meio da qual pode autenticar-se no sistema utilizando seus dados cadastrais (Figura 35). Caso ainda não esteja registrado, o modelador pode inscrever-se por meio da opção "increva-se". Um formulário solicitando os dados cadastrais é apresentado ao usuário, que pode finalizar o processo de criação de conta por meio da opção "Salvar" (Figura 35).

Figura 35 – Tela de login e registro do *NuiMod Editor*



Fonte: Captura de tela no navegador Google Chrome

Para facilitar o acesso à ferramenta, o *NuiMod Editor* permite que o usuário realize login social, por meio de suas credenciais no Google ou Facebook. Nesta tela também podem ser recuperados o acesso à conta e alterado o idioma do sistema. Atualmente, o *Editor* está disponível em Português do Brasil e Inglês.

Após realizar a autenticação, o modelador visualiza a tela principal do ambiente. Na parte superior é apresentado o menu principal, por meio do qual o usuário pode acessar as funcionalidades do sistema. Na parte direita do menu encontram-se as opções de notificação

de problemas e logout do ambiente. Por fim, a parte central da tela exibe informações sobre cada componente do NuiMod e permite que o utilizador realize o *download* do *Connector*.

4.4.4.4 Edição de projetos

Dois conceitos fundamentais para a modelagem no ambiente são os projetos e os componentes. No *NuiMod Editor*, o usuário pode criar um projeto e vincular componentes a ele. Tais componentes podem ser de dois tipos: poses e gestos.

Por meio da opção Projetos -> Meus projetos, o modelador pode acessar os projetos criados por ele dentro do ambiente. Nesta tela, é possível criar e editar projetos existentes, bem como removê-los do sistema. Ao clicar sobre o título de um projeto, o usuário é redirecionado para a tela de Edição de Projetos.

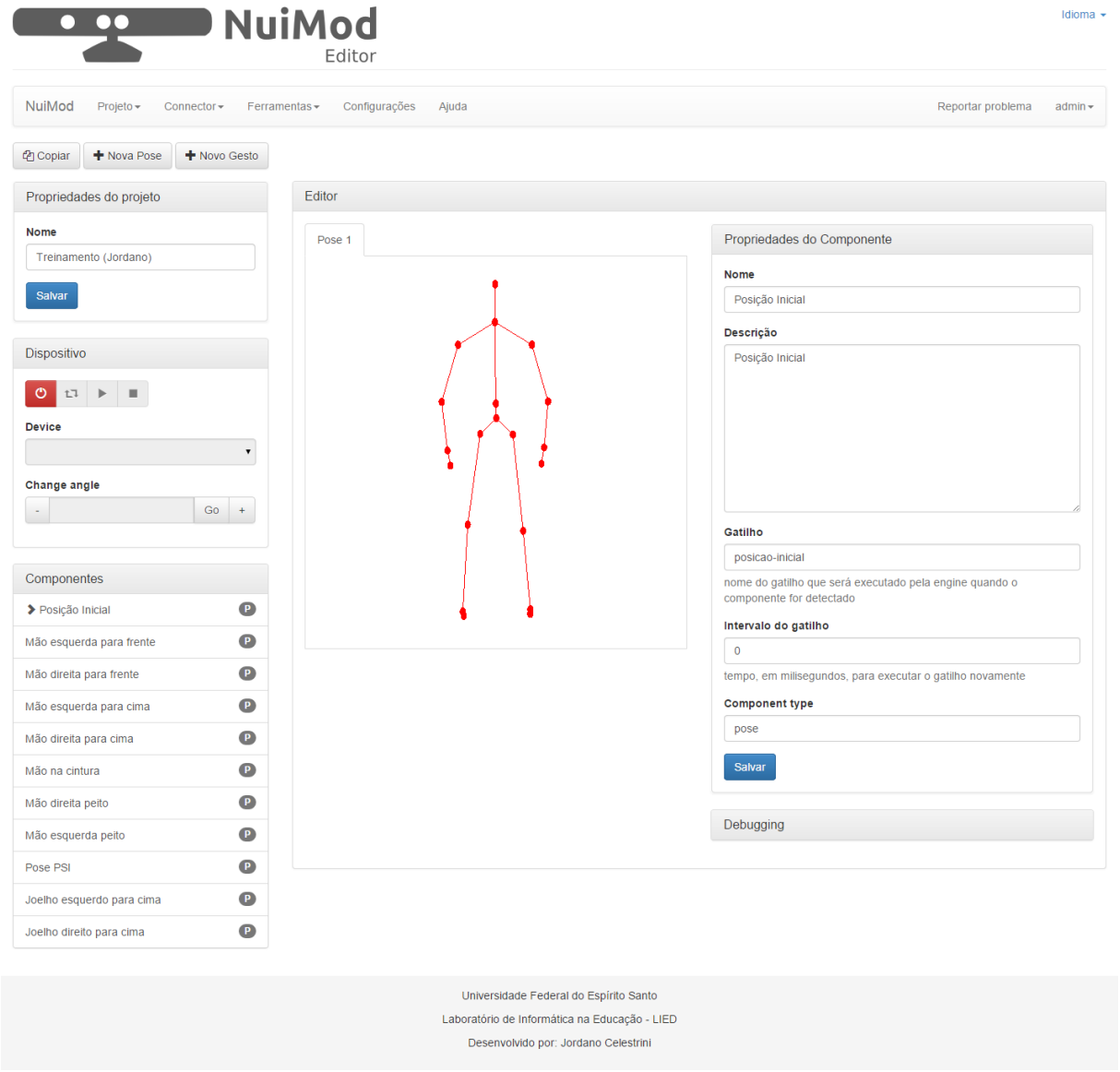
Na tela de Edição de Projeto encontra-se o cerne do *NuiMod Editor*, onde o usuário pode modelar seus componentes (Figura 36). A modelagem funciona de modo guiada, ou seja, o utilizador emprega o sensor de movimentos para manipular um avatar articulado e armazenar os dados de esqueleto. A página está dividida em 6 painéis:

- Propriedades do projeto - neste painel o usuário pode alterar os dados gerais do projeto.
- Controle de dispositivos - utilizado pelo modelador para controle do sensor de interação natural.
- Componentes - painel que exibe todos os componentes que existem no projeto e seus respectivos tipos (poses ou gestos).
- Visualização de esqueleto - exibe o esqueleto que está sendo modelado.
- Propriedades do componente - permite ao usuário alterar as propriedades de determinado componente como nome, descrição, gatilho e intervalo do gatilho.
- Painel de depuração - exibe dados de depuração, para fins de *debug*.

Para iniciar a captura de um componente, é necessário utilizar o painel de controle de dispositivos (Figura 37). Este painel interliga o *Editor* ao *Connector*, permitindo acesso aos dados do dispositivo de interação natural. Por meio dele, o usuário pode executar 4 ações: Estabelecer uma conexão com o *Connector*, listar os dispositivos NUI e iniciar e parar a captura de dados. Caso seja permitido, exite a opção de ajustar o ângulo do sensor que, no caso do Kinect, pode estar entre 27 e -27 graus.

O *Editor* possui duas instâncias da *Engine*: a primeira para tratar os dados que estão sendo modelados e a outra para controlar o processo de edição. A segunda *Engine* possui os seguintes gatilhos:

Figura 36 – Tela de edição de projetos no *NuiMod Editor*

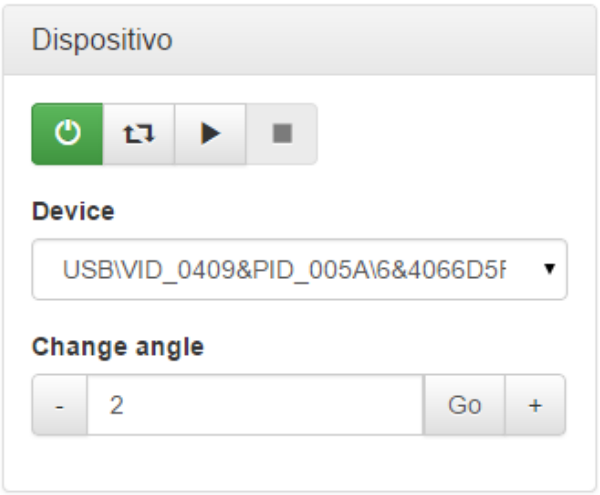


Fonte: Captura de tela no navegador Google Chrome

- `editor:restart` (Reiniciar modelagem) - Utilizado para reiniciar a modelagem caso o usuário não fique satisfeito com a pose/gesto capturado
- `editor:save` (Salvar componente) - Utilizado para persistir os dados capturados pelo modelador
- `editor:next` (Próximo componente) - Navega até o próximo componente do projeto
- `editor:prev` (Componente anterior) - Retorna ao componente anterior

Após iniciar a captura, o esqueleto permanecerá na cor vermelha até que o usuário persista na mesma posição por meio segundo, tempo padrão para detecção de estabilidade. Ao estabilizar, o esqueleto muda automaticamente para a cor verde.

Figura 37 – Painel de controle do *NuiMod Connector*



Fonte: Captura de tela no navegador Google Chrome

Se o usuário continuar na mesma posição pelos próximos dois segundos, o esqueleto muda para a cor azul, indicando o encerramento da captura. Neste momento, a *Engine* de captura é pausada, e a *Engine* de edição entra em ação.

Para cada componente criado deve-se vincular um gatilho, que será disparado pela Engine assim que o componente for detectado. É possível especificar o tempo de intervalo, em milisegundos, entre a execução dos gatilhos, previnindo que uma mesmo evento seja executado ininterruptamente caso o usuário permaneça estático em uma determinada posição.

O usuário pode ainda estipular quais articulações deverão ser rastreadas em determinado componente, uma vez que usualmente um componente não utiliza todas as juntas do corpo. Além disso, rastrear articulações desnecessárias a determinado componente pode diminuir a taxa de acerto da *Engine*. Para selecionar quais juntas não devem ser rastreadas, basta clicar duas vezes sobre a articulação no avatar da tela de edição do projeto. A cor da articulação deverá aparecer na cor amarela, indicando que não está sendo rastreada.

Ao finalizar o projeto, o usuário pode exportá-lo por meio do menu Ferramentas - Salvar no computador. O sistema gera um arquivo XML, com a extensão .nmd, contendo os componentes criados pelo modelador e os eventos que devem ser disparados assim que o usuário realizar a pose ou gesto modelado.

4.4.4.5 Serviços

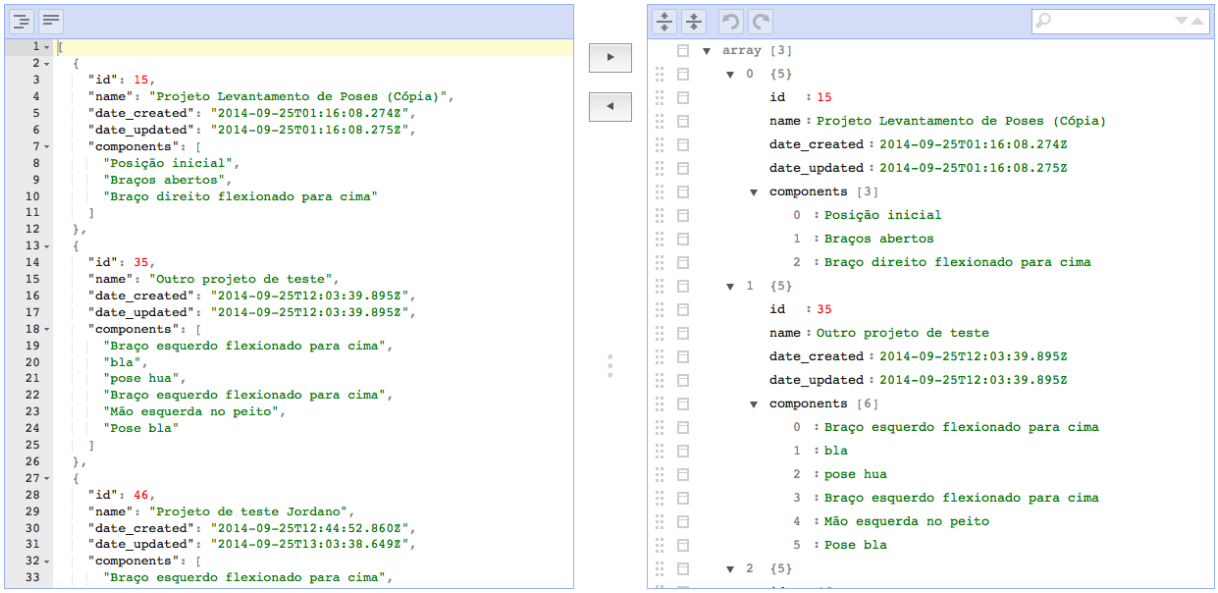
O *Editor* disponibiliza ainda uma API para acesso aos dados por meio de Transferência de Estado Representativo (REST). Isto possibilita que aplicações terceiras consumam os componentes modelados pelo editor sem a necessidade de exportação/importação do

arquivo NMD.

Para troca de dados, a API utiliza o formato JSON. Atualmente, este é um formato popular que é comumente empregado em design de API REST, e que possui alguns benefícios herdados de JavaScript, como a integração com o ambiente de execução nativo do browser. Isso permite que os dados trocados entre o cliente e o servidor sejam entendindos pelo navegador sem a necessidade de qualquer conversão.

Um objeto JSON é um conjunto desordenado de pares nome-valor. A sintaxe de objeto JSON define nomes como *strings*, sempre delimitadas por aspas dupla (MASSE, 2011). Note-se que esta é uma regra de formatação menos branda do que a de literais de objeto em JavaScript, e essa diferença muitas vezes leva a um JSON malformado. A Figura 38 ilustra um JSON bem-formatado, que representa parcialmente uma lista de projetos do *NuiMod Editor*.

Figura 38 – Exemplo de projetos exportados pela API do *NuiMod Editor*



Fonte: Elaborada pelo autor

Para controlar o acesso à API, o *Editor* utiliza OAuth 2.0, um padrão aberto que fornece autenticação segura, utilizando uma abordagem consistente para todos os clientes (MASSE, 2011). O padrão é amplamente utilizado por grandes empresas como Google[1] e Facebook[2] para permitir o compartilhamento de seus recursos privados, tais como documentos, fotos ou lista de contatos, sem a necessidade do usuário revelar seu login ou senha.

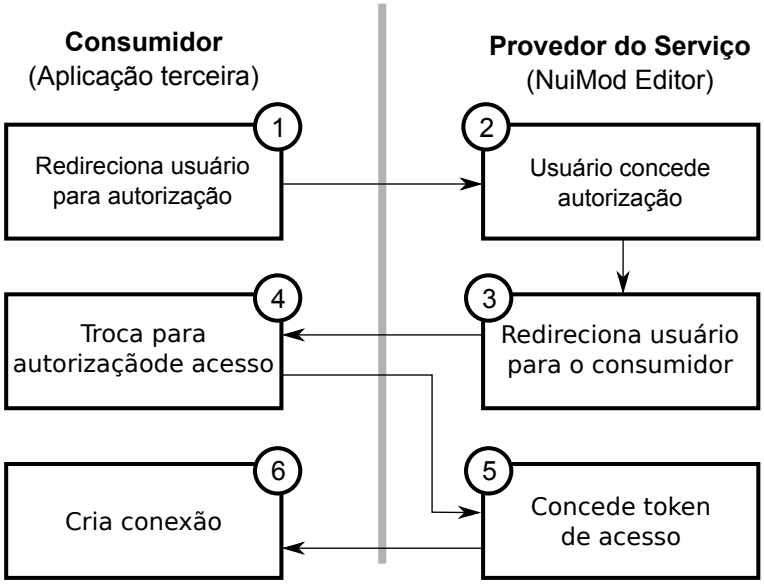
A especificação define 4 fluxos diferentes para autenticação: *Authorization Code*, *Implici*, *Resource Owner Password Credentials* e *Client Credentials*, cada um deles voltado

1 Ver: https://developers.google.com/accounts/docs/OAuth2
2 Ver: https://developers.facebook.com/docs/reference/dialogs/oauth/

para um cenário específico. No caso do *NuiMod Editor*, o fluxo adotado é o do tipo *Authorization Code*. Este fluxo para obtenção de autorização de acesso à conta do usuário (Figura 39) segue 6 etapas:

1. O fluxo começa quando a aplicação consumidora redireciona o usuário para a URL de autorização do *NuiMod Editor*. O *Editor* então exibe uma página para que o usuário possa logar-se (caso ainda não o tenha feito) e, em seguida, exibe a solicitação de acesso da aplicação para que ele possa autorizar ou não.
2. O usuário então autoriza o acesso.
3. O *Editor* o redireciona à aplicação consumidora, retornando um código de autorização.
4. A aplicação consumidora troca o código de autorização por um token de autenticação.
5. O *Editor* autoriza o acesso ao serviço, retornando o token.
6. A aplicação utiliza o token de acesso para criar uma conexão.

Figura 39 – Fluxo de autenticação com OAuth 2.0



Fonte: Elaborada pelo autor

Após realizar o fluxo de autenticação, a aplicação parceira utiliza o token de acesso para criar uma conexão e recuperar os dados de projeto do usuário. Para tal, basta fazer uma solicitação HTTP GET, conforme exemplificado abaixo:

```
GET editor/api/v1/projects/ HTTP/1.1
Host: nuimod.com.br
```

Por padrão, a API retorna apenas dados parciais dos projetos, incluindo nome e tipo dos componentes. Assim diminui-se a quantidade de dados trafegados entre cada requisição, previnindo o envio desnecessário de informações ao cliente. Todavia, caso o desenvolvedor considere necessário, pode-se incluir o parâmetro `f=full` à URL, a fim de retornar os dados completos dos componentes.

O desenvolvedor pode ainda realizar uma pesquisa nos dados, por meio do parâmetro `q`. Este parâmetro irá pesquisar os campos nome e descrição do projeto. Além disso, pode-se pesquisar por gatilhos específicos por meio do parâmetro `t`. Esta operação é especialmente útil quando a aplicação consumidora faz uso de gatilhos específicos e, portanto, está interessada em recuperar apenas os projetos que contenham esta informação.

A requisição de exemplo abaixo retornará todos os projetos que contenham os gatilhos 'editor:reiniciar' e 'editor:parar'. Além disso, todas as informações relativas aos componentes serão retornadas, uma vez que o parâmetro `f=full` foi incluido à URL:

```
GET editor/api/v1/projects/?f=full&t=editor:reiniciar|editor:parar HTTP/1.1
Host: nuimod.com.br
```

Os serviços fornecidos pelo *Editor* permitem tornar transparente a comunicação entre o NuiMod e as aplicações terceiras. Muito embora a exportação dos projetos no formato NMD oportunize o compartilhamento dos componentes modelados, o processo de exportar e importar é trabalhoso para o usuário. Neste sentido, fornecer um serviço de compartilhamento retrata uma vantagem em relação à exportação dos dados.

4.5 Considerações finais

Este capítulo apresentou o ambiente NuiMod, solução proposta por esta dissertação para a construção de sistemas na web utilizando NUI. Foram descritos os componentes (*Engine*, *Editor* e *Connector*) que integram sua arquitetura, bem como o modelo NMD, que especifica um formato para troca de dados entre aplicações NUI.

Diferente das outras propostas para modelagem de poses e gestos, o *Editor* apresenta-se como uma solução descomplicada e intuitiva, podendo ser utilizado por usuários que não possuem conhecimentos profundos no assunto. A manipulação de um avatar virtual facilita esta tarefa, simplificando o processo de modelagem.

Por outro lado, a proposta também contempla o universo de desenvolvedores de aplicações, apresentando uma proposta para integração de NUI a ambientes web, por meio da *Engine* e do *Connector*. Além disso, os diversos pontos de extensão do NuiMod permitem que sejam alteradas várias configurações, tornando a solução robusta e, ao mesmo tempo, flexível.

Outro ponto levado em consideração nesta proposta foi a interoperabilidade entre

aplicações NUI. Poucas pesquisas focam este problema e buscam como tratá-lo e as soluções existentes não atendem aos requisitos do NuiMod, uma vez que não apresentam pontos de extensão. Além disso, foi incluido um serviço de compartilhamento de dados no *Editor*, o que permite aplicações terceiras acessarem e utilizarem os modelos criados pela ferramenta.

Algumas questões ainda precisam ser resolvidas e não foram foco desta dissertação. Destacam-se a falta de um metamodelo para representação dos modelos de esqueleto para diferentes dispositivos NUI e a impossibilidade de modelagem de cenários que exigem dados do ambiente (como saltar), demandando a utilização de uma engine baseada em regras. Estes pontos serão detalhados e discutidos na Seção 6.2.

5 Experimentações com o NuiMod

Esta seção abordará os experimentos realizados a fim de validar a solução criada. Pretende-se, desta forma, validar a modelagem realizada por meio da textitEngine; o recebimento dos dados do sensor, por meio do *Connector*; bem como o processamento das informações, realizado pela *Engine*. Para esta finalidade, foram realizados 3 experimentos:

1. uma base de dados de poses e gestos para testes com a *Engine*, a fim de validar as estratégias de detecção de estabilidade e de reconhecimento de poses e gestos;
2. um editor de apresentações online, para validar os serviços providos pelo *Editor*, bem como o uso do *Connector* e da *Engine* além das fronteiras da solução proposta;
3. um plugin baseado no Greasemonkey, extensão disponível para o Firefox e Google Chrome que permite a injeção de códigos customizados em páginas na web.

5.1 Experimento 1 - *Engine*

A fim de testar o algoritmo implementado para o reconhecimento de poses e gestos, foram criados 5 projetos no *NuiMod Editor*, por pessoas de tamanho, idade e altura diferentes. Cada projeto possui 11 componentes modelados, num total de 55 poses capturadas.

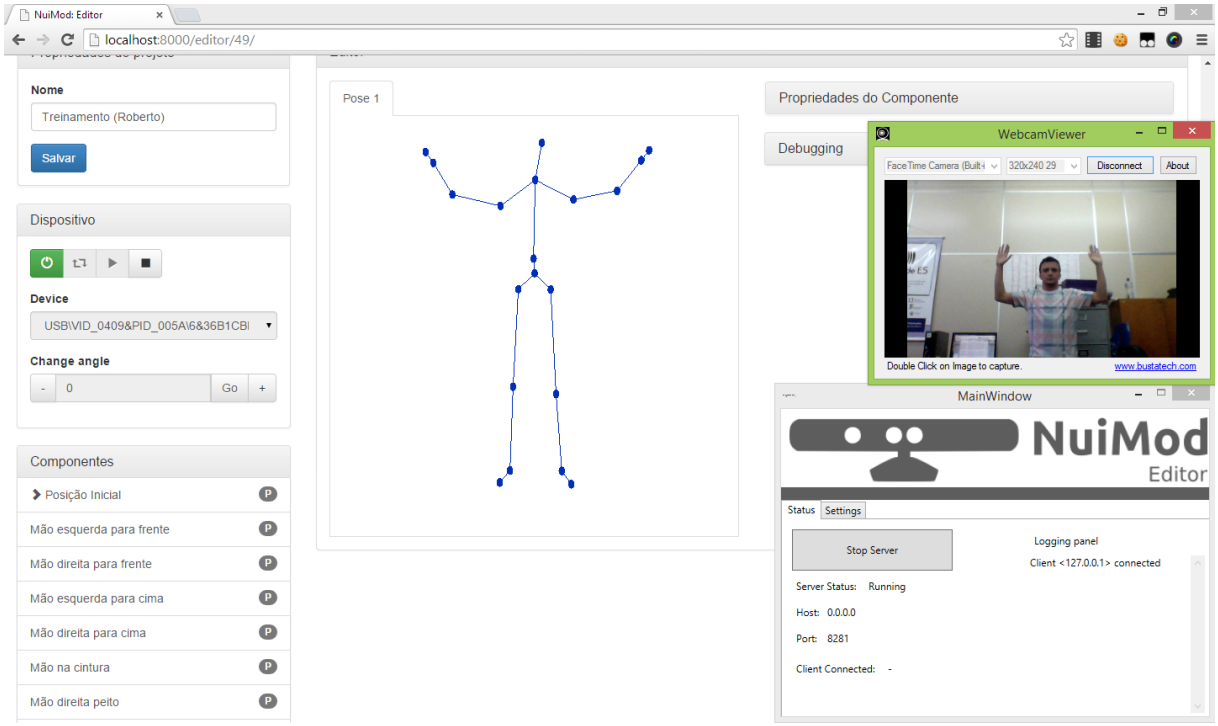
As 5 pessoas selecionadas, com faixa etária entre 19 e 29 anos, iniciaram o experimento realizando um projeto de treinamento por 2 vezes, a fim de conhecer cada uma das 11 poses a serem executadas. Foi então solicitado que cada participante modelasse novamente cada pose, agora em um novo projeto.

Para comparação, foram realizadas 5 etapas de testes (Figura 40), cada uma comparando um projeto de referência, com os outros 4 projetos e com ele mesmo, em um total de 3025 diferentes combinações. Os dados de cada participante podem ser conferidos na Tabela 7. Os componentes modelados e os resultados detalhados do experimento estão no Apêndice 54.

O tempo mínimo para a conclusão da tarefa foi de 5 minutos e 49 segundos e o tempo máximo foi de 11 minutos e 27 segundos. Todos os participantes completaram a atividade até o fim.

Os dados mostram uma taxa de acerto da *Engine* de 89,09%, considerando-se apenas os cenários em que a *Engine* deveria reconhecer similaridade (destacadas com fundo preto e vermelho). Em nenhuma das comparações a *Engine* retornou falsos positivos, o que

Figura 40 – Etapa de teste realizada durante os experimentos com o NuiMod



Fonte: Captura de tela no sistema operacional Windows 8.1

Tabela 7 – Tabela de comparação entre projetos para validação da *Engine*

Projeto	**Sexo**	**Idade**	**Altura**
Projeto 1	Masculino	27	1,72
Projeto 2	Masculino	19	1,73
Projeto 3	Feminino	28	1,58
Projeto 4	Masculino	27	1,67
Projeto 5	Feminino	29	1,69

Fonte: Elaborada pelo autor

demonstra a eficácia do algoritmo de comparação. Todavia, vale ressaltar que não foram realizados experimentos com movimentos aleatórios, o que poderia alterar os resultados do experimento.

5.2 Experimento 2 - Editor de apresentações

Apresentações constituem-se uma das maneiras mais efetivas de comunicar idéias para pessoas interessadas em determinado assunto. Uma apresentação atraente conquistará a audiência do público, ao passo que uma apresentação ruim pode prejudicar a audiência ou arruinar a reputação do apresentador. (RATNAYAKE, 2013)

Durante muito tempo esta foi uma área dominada por aplicações *desktop*, tais

como o Microsoft PowerPoint e o Libreoffice Impress. O avanço da Internet tem provocado mudanças neste cenário, favorecendo o aparecimento de soluções para apresentações *on-line*.

O *impress.js* é um dos *frameworks* que nasceram a partir deste movimento. Com ele é possível criar apresentações utilizando HTML5, CSS e JavaScript que executam em navegadores modernos sem a necessidade de instalação de qualquer *plugin*. Baseado em outra aplicação bastante conhecida denominada Prezi[1], o *impress.js* foi desenvolvido empregando a tecnologia de transição e transformação do CSS3, a fim de disponibilizar as seguintes funcionalidades:

- **Posicionamento** - os elementos podem ser colocados em certas áreas do navegador, permitindo a transição entre os slides;
- **Dimensionamento** - permite que elementos sejam ampliados ou reduzidos para exibir uma visão geral ou detalhes dos elementos que compõe a apresentação;
- **Rotação** - elementos podem ser rotacionados em qualquer eixo determinado.
- **Espaço 3D** - as apresentações não estão limitadas ao espaço 2D. Todos os efeitos mencionados acima podem ser aplicados no espaço tridimensional com o eixo Z.

O *impress.js Editor* é um *front-end* para o *impress.js* que permite criar e executar presentações não-lineares na web (Figura 41). A aplicação utiliza o serviço de autenticação disponibilizado pelo *NuiMod Editor* para acessar os componentes criados pelo usuário, permitindo personalizar poses e gestos para controle da apresentação.

Para o desenvolvimento desta experimentação, além do *impress.js*, foram utilizados as seguintes tecnologias:

- Django - *framework* para desenvolvimento da aplicação web
- Django Outh Toolkit - ferramenta para autenticação com Django em conjunto com Outh 2.0
- Bootstrap - *framework* CSS para criação de interfaces em HTML
- jQuery - *framework* JavaScrip para manipulação de páginas HTML
- NuiMod Connector - *plugin* NuiMod para acesso ao *Connector*
- NuiMod Engine - *Engine* de reconhecimento de poses e gestos do NuiMod

[1] Ver: http://prezi.com/

Figura 41 – Página inicial do *impress.js Editor*

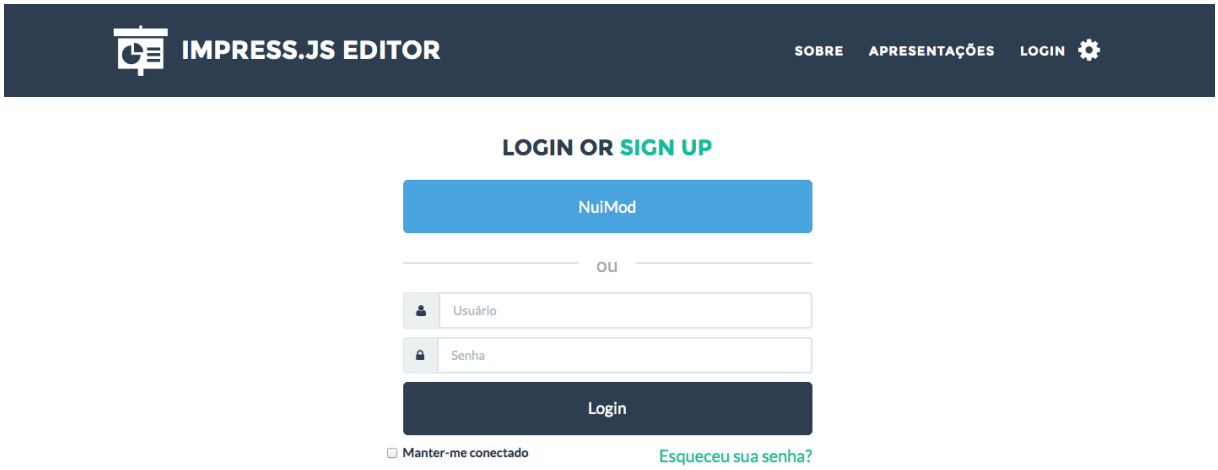
Fonte: Captura de tela no navegador Google Chrome

Para acessar a ferramenta, a primeira ação que o usuário deve executar é efetuar login, conforme exemplifica a Figura 42. O utilizador pode identificar-se com as mesmas credenciais de acesso ao *NuiMod Editor*, por meio do serviço de autenticação, ou criar uma conta local. Caso opte por uma conta local, não será concedido acesso ao serviço de importação de componentes, ficando sob responsabilidade do usuário exportar e importar os arquivos NMD, caso necessite personalizar poses e gestos para a apresentação.

Após logar-se, o usuário será redirecionado para a tela de apresentações, onde serão listados todos os projetos aos quais ele possui acesso. Neste momento, pode-se optar pela criação de uma nova apresentação ou pela edição de uma apresentação existente. Em ambos os casos, o usuário será redirecionado à tela de edição. As apresentações criadas pela ferramenta podem ser públicas, cujo acesso é liberado a qualquer usuário, ou privadas, ficando disponível apenas ao autor.

Na edição de apresentações (Figura 43), o usuário pode alterar informações como nome e tipo do projeto (público ou privado). É disponibilizado ainda o acesso à recursos externos, como CSS, Javascript ou fontes - por meio da opção **Recursos Externos**. Caso seja necessário fazer upload de arquivos o usuário pode utilizar a opção **Resources**,

Figura 42 – Tela de login do *impress.js Editor*

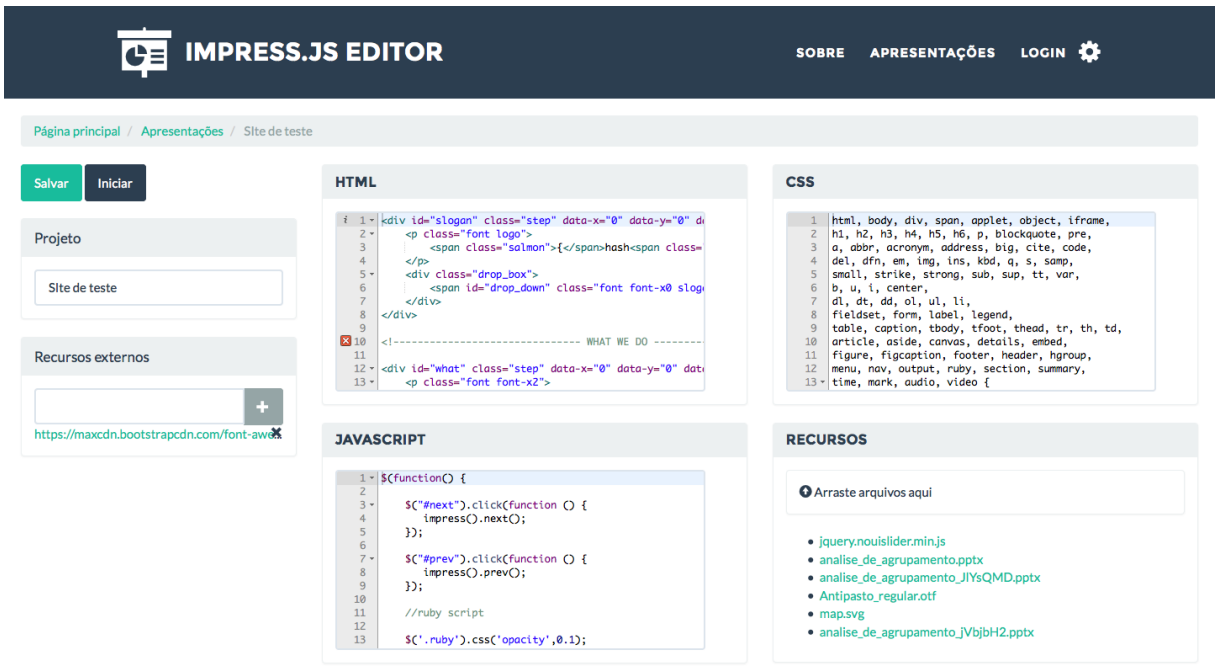


Fonte: Captura de tela no navegador Google Chrome

arrastando e soltando os arquivos de seu computador que são enviados ao servidor do *impress.js Editor*.

O conteúdo da apresentação é alterado no painel HTML, enquanto o layout e os efeitos podem ser preparados nos painéis CSS e Javascript. Ao finalizar a edição do projeto, o usuário deve selecionar a opção **Salvar** no menu principal para enviar as alterações ao servidor.

Figura 43 – Tela de edição de apresentações do *impress.js Editor*



Fonte: Captura de tela no navegador Google Chrome

Para iniciar a apresentação, o utilizador deve acessar a opção **Play**. Uma nova aba é aberta no navegador e o usuário pode habilitar o modo tela cheia por meio dos controles que aparecem do lado inferior da apresentação. O *player* detecta automaticamente se o *NuiMod Connector* está disponível e se existe algum dispositivo NUI conectado ao computador do usuário. Em caso positivo, são carregados os componentes para controle da apresentação, que se inicia em sequência.

Na seção de configuração da aplicação, o utilizador pode personalizar os componentes e ações para a execução da apresentação, importando estes dados do *NuiMod Editor* por meio do serviço disponibilizado ou carregando os arquivos NMD com as informações sobre componentes. Os arquivos enviados ao *impress.js Editor* são validados por meio do *XML Schema* disponibilizado pelo NuiMod. Também é permitido ao usuário configurar as informações relativas ao *NuiMod Connector*, como url e porta de acesso.

5.3 Experimento 3 - Script para Greasemonkey

Greasemonkey é uma extensão disponível para o navegador Firefox, que permite escrever *scripts* para alterar páginas na Internet em tempo de execução (PILGRIM, 2005). Tais scripts, chamados *user scripts* constituem-se de pequenos arquivos contendo pedaços de código em JavaScript, que podem realizar todas as tarefas às quais a linguagem oferece suporte, como alterar um trecho da página ou, neste caso, incluir a API NuiMod para utilização de poses e gestos em sites na Internet.

Cada *user script* possui uma seção chamada *metadata* (Figura 44), onde o desenvolvedor fornece ao Greasemonkey informações sobre ele, como nome, descrição, localização e quando executá-lo. O *metadata* também inclui dados sobre onde o *script* deve ser executado: em uma única página, em todo um domínio ou uma seleção de múltiplos sites.

Figura 44 – Metadata do plugin NuiMod no Greasymonkey

```
// ==UserScript==
// @name       NuiMod Plugin
// @namespace  http://nuimod.com.br/
// @version    0.1
// @description  Google Drive Presentation controladapelo NuiMod
// @match      https://docs.google.com/presentation/d/*/present*
// @copyright  2012+, NuiMod Project
// @require    https://code.jquery.com/jquery-2.1.1.min.js
// @require    http://cdn.nuimod.com.br/nuimod/nuimod.utils.js
// @require    http://cdn.nuimod.com.br/nuimod/nuimod.engine.js
// @require    http://cdn.nuimod.com.br/nuimod/nuimod.connector.js
// @require    http://app.nuimod.com.br/api/v1/projects/?format=json
// ==/UserScript==
```

Fonte: Elaborada pelo autor

Um *metadata* tipicamente inclui as seguintes informações:

- **Wrapper (Invólucro)** - utilizados pelo Greasemonkey para sinalizar o início e término do *metadata*. Esta seção pode ser colocada em qualquer local do *script*, mas orienta-se colocá-lo próximo ao início do arquivo;
- **Name (Nome)** - é o nome do *user script*, exibido quando o usuário solicita sua instalação e posteriormente ao acessar o Gerenciador de Scripts;
- **Namespace** - URL utilizada pelo Greasemonkey para distinguir *user scripts* que possuem o mesmo nome, porém foram desenvolvidos por autores diferentes. Geralmente apontam para o site (ou subdiretório) do desenvolvedor.
- **Description (Descrição)** - descrição legível do *script*, informando qual a sua finalidade. É exibido ao instalar a aplicação, bem como Gerenciador de Scripts.
- **URL Directives (Diretivas de URL)** - informam ao Greasemonkey onde o script deve ser executado. Os comandos *@include* e *@exclude* são utilizados para informar as URLs onde o *script* deve ser incluido ou excluído, respectivamente.

Para exemplificar a utilização do NuiMod com o Greasemonkey, foi desenvolvido um *user script* que interage com o Google Docs, controlando apresentações criadas na ferramenta por meio de poses e gestos. O *plugin* simula o clique do mouse a fim de avançar ou retroceder a apresentação (Figura 45).

Figura 45 – Execução de apresentação no Google Drive Presentation utilizando o NuiMod

Fonte: captura de tela no sistema operacional Windows 8.1

O fluxo seguido pelo *plugin* divide-se em 3 etapas: primeiro instancia-se o *Connector* e a *Engine*. Então, é iniciada a escuta pelo componente desejado, carregado diretamente do *Editor*. Caso o usuário não tenha indicado um projeto para personalizar as poses ou gestos ou o projeto indicado não contenha os gatilhos necessários ao *plugin*, um projeto modelo é carregado.

O último passo é conectar-se ao cliente, a fim de buscar os dispositivos NUI conectados à máquina. O plugin seleciona o primeiro dispositivo encontrado e inicia a apresentação, aguardando os comando do usuário.

Em relação aos navegadores, o Google Chrome também fornece suporte à *scripts* criados no Greasemonkey por meio de uma extensão denominada Tampermonkey[2]. As versões mais novas do *browser* suportam nativamente este recurso, identificando cada *user script* como uma extensão.

5.4 Considerações finais

Este capítulo apresentou aplicações práticas do NuiMod e os resultados para o reconhecimento de poses e gestos alcançados pela *Engine*. O primeiro experimento evidencia a facilidade com que, usuários sem nenhum conhecimento prévio de modelagem de componentes, conseguem utilizar o *Editor*.

O segundo experimento mostra como é possível que desenvolvedores integrem, em novas soluções, a utilização de NUI com o suporte da arquitetura desenvolvida nesta dissertação. Além disso, este experimento faz uso dos serviços providos pela plataforma, permitindo que sejam personalizadas as poses e gestos para executar ações dentro do sistema.

Por fim, o terceiro experimento demonstra ser possível utilizar interfaces naturais de usuário até mesmo em soluções já existentes e de código fechado, por meio da injeção de códigos JavaScript no ambiente de destino. Desta forma, abre-se caminho para que esta arquitetura possa integrar soluções já conhecidas e que ainda não possuem suporte à tal funcionalidade.

[2] Disponível para download em: http://tampermonkey.net/

6 Conclusões e Perspectivas

Este capítulo apresenta as conclusões desta dissertação. São evidenciadas as principais contribuições do trabalho e suas limitações. Além disso, são apresentadas as lições aprendidas e os possíveis caminhos para pesquisas futuras.

6.1 Contribuições

O cenário atual no desenvolvimento de dispositivos NUI aponta uma tendência para o crescimento da área, com o surgimento de novos sensores, voltados para dispositivos móveis, e soluções embarcadas, que logo estarão presentes em computadores e sob a forma de acessórios. As primeiras iniciativas neste sentido já são percebidas com o anuncio da próxima geração de notebooks de grandes empresas, como HP[1], que virão equipados com sensores de movimento.

Todavia, a despeito de seu pioneirismo, estes dispositivos ainda necessitam de um maior suporte para o desenvolvimento de aplicações, principalmente voltadas ao ambiente web. Como evidenciado no Capítulo 3, existem poucas soluções voltadas especificamente para este cenário. Além disso, grande parte das soluções existentes não permitem que o usuário modele de maneira simples e intuitiva seus próprios componentes, um fator limitante em diversas áreas de aplicação.

A proposta do NuiMod colabora com a resolução destes problemas, ao propor uma solução modularizada, que permite desde a modelagem até o compartilhamento de componentes, disponibilizando aos desenvolvedores as ferramentas necessárias para consumir modelos gestuais em aplicações além das fronteiras do NuiMod.

O *Editor* provou ser possível aliar sensores de movimento ao ambiente web, proporcionando maior iteratividade ao usuário final. Os experimentos realizados realçaram estes resultados, provando ser possível levar a experiência de controlar ações em páginas web inclusive a sistemas que não foram criados sob tal perspectiva, como demonstrado com o Google Drive Presentation.

A especificação de um modelo bem definido de dados colabora para a iteroperabilidade entre as aplicações que utilizam sensores de movimento, conforme apresentado no Experimento 5.2. No experimento, intitulado *impress.js Editor*, é possível personalizar poses e gestos que disparam ações para controlar apresentações, por meio de arquivos NMD, formato originado a partir deste trabalho.

[1] Veja: http://www8.hp.com/us/en/ads/envy-leap-motion/overview.html

6.2 Limites da proposta

A edição de poses e gestos baseada em posições de articulações do corpo humano permitiu a criação de uma ferramenta intuitiva, por meio da qual é possível modelar componentes de modo guiado, utilizando dispositivos NUI. Todavia, este modo compromete a modelagem de componentes que dependem de dados do ambiente, como posicionamento de articulações em relação à superficie onde se encontra o modelador.

Assim, impede-se a modelagem de gestos como saltar, onde é necessário levar em conta a distância entre o pé do sujeito e o solo. Para solucionar a questão é necessário implementar uma edição baseada em regras, o que traria maior complexidade à modelagem, mas permitiria abarcar estes cenários. Como a proposta inicial fora facilitar ao máximo o processo de criação de componentes, a questão foi deixada fora do escopo da solução proposta.

Além disso, não foram levadas em consideração questões de colaboração, importantes na modelagem de componentes para cenários complexos, que exigem a participação de várias pessoas. Não está claro de que modo esta funcionalidade pode ser incorporada ao ambiente, haja vista a quatidade de dados trafegados entre o *Connector* e o *Editor* durante a modelagem.

Neste contexto, uma possível solução seria a edição individual de componentes e uma posterior fusão entre eles. É preciso levar em consideração que estratégias seriam adotadas para lidar com gatilhos repetidos e componentes similares.

Outro limite importante desta proposta está no modelo de esqueleto, fixado em 20 articulações pré-definidas, conforme pode ser conferida no Capítulo 4. É importante que a plataforma permita generalizar estas articulações, a fim de suportar dispositivos que não trabalham como modelos completos do corpo humano, como o Leap Motion, que realiza o rastreamento apenas de articulações da mão. Assim, o *Editor* poderia atuar não apenas na modelagem de componentes que utilizam todas as articulações do corpo, mas também daqueles voltados apenas para partes específicas, como mãos e dedos.

6.3 Lições aprendidas

Esta pesquisa oportunizou a troca de idéias e novos aprendizados, seja por meio da iteração com outros colegas que trabalham em diferentes linhas de pesquisas, seja através dos levantamentos bibliográficos realizados em todas as etapas de execução do trabalho. Além disso, a implementação do projeto permitiu aprimorar conhecimentos no desenvolvimento de aplicações para web e interoperabilidade entre elas.

Em relação ao referencial teórico, a criação de um ambiente de estudo estabelecido desde o início desta jornada viabilizou um local para concentrar todo o material que seria

utilizado para a dissertação, facilitando o seu desenvolvimento. O ambiente escolhido foi o PBworks, dividido em diferentes seções para organizar o conteúdo produzido. Foram criados dois espaços: um para trabalhos correlatos e outro voltado a dados técnicos sobre dispositivos NUI.

Uma das dificuldades nesta etapa de levantamento foi encontrar trabalhos científicos relevantes no tocante à modelagem de poses e gestos. Embora existam muitas pesquisas relacionadas à dispositivos NUI, a questão da modelagem ainda é pouco explorada. O principal trabalho encontrado com este viés foi o FAAST, apresentado na Seção 3.4.1.

Na implementação da solução, o maior desafio enfrentado foi o fim da OpenNI durante a execução deste trabalho, em 2013. A primeira versão do *Connector*, um dos principais componentes da solução proposta, foi implementada em Java, utilizando o OpenNI e os *middlewares* Nite. Após a compra da PrimeSense pela Apple, optou-se por descontinuar esta versão do *Connector*, reimplementando toda a solução em C#, por meio do *Kinect for Windows SDK*.

Esta decisão não impactou diretamente no cronograma para execução do trabalho, uma vez que na etapa de desenvolvimento do *Connector* havia alguma folga, que pode ser utilizada em sua reimplementação. Todavia, o fato serve de referência e alerta à pesquisas futuras que utilizem componentes proprietários, disponibilizados por fornecedores sem quaisquer garantias. Sempre que possível, é preferivel optar por soluções de código aberto que possam ser mantidas independentemente de fornecedores.

Por fim, o planejamento realizado previamente, levando em consideração os pilares do gerenciamento de projetos, com as definições de cada etapa e seus prazos de entrega, permitiu que os envolvidos tivessem uma visão geral das fases de execução da dissertação. Este planejamento conduziu a uma concretização equilibrada, com o inicio antecipado da prototipação e sem intercorrências que prejudicassem seu desenvolvimento. O cronograma elaborado para o desenvolvimento desta dissertação pode ser apreciado no Apêndice C.

6.4 Trabalhos futuros

Uma das principais necessidades encontradas durante o desenvolvimento do NuiMod foi a utilização do `Connector` em outros dispositivos além dos computadores. Por ter sido desenvolvido em C# e em conjunto com o *Kinect for Windows SDK*, a execução do `Connector` limitou-se ao sistema operacional Windows. Todavia, esta decisão de projeto foi a mais acertada, em um momento de incertezas em relação ao OpenNI, cujo principal componente de rastreio de esqueleto estava disponível apenas sob licença proprietária.

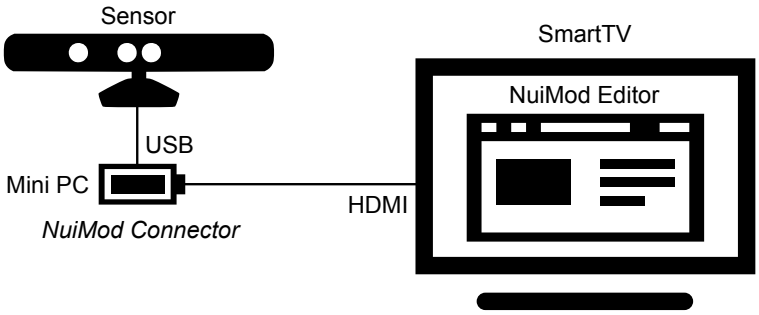
A maior dificuldade em levar o `Connector` a outros ambientes reside no fato de não existirem bibliotecas de código-aberto estáveis que possam ser utilizadas para rastreio

do esqueleto humano. Muito embora existam algumas iniciativas neste sentido, como a biblioteca Skeltrack [2], elas ainda não podem ser consideradas maduras para serem empregadas em projetos como o NuiMod, haja vista a baixa precisão com que operam.

Permitir que o `Connector` seja executado em dispositivos embarcados, oportunizaria a execução do `NuiMod Editor`em diversos outros ambientes, como TV's e tablets. Neste sentido, esta pesquisa realizou alguns experimentos com mini PC's, a fim de verificar a viabilidade desta proposta. Chegou-se a conclusão de que é possível obter acesso aos dados de profundidade do Kinect, por meio da *libfreenect* (Seção 2.5.2), porém ainda não há uma maneira eficiente de processá-los a fim de recuperar os pontos de articulação do esqueleto.

A visão de futuro desta pesquisa pode ser conferida na Figura 46. Neste cenário, o *Connector* estaria embarcado em um dispositivo de pequenas dimensões, acoplado ao próprio dispositivo NUI, o que proporcionaria maior mobilidade à solução. O *Editor*, disponível na nuvem, poderia ser acessado por meio de diferentes dispositivos.

Figura 46 – Visão de futuro do projeto NuiMod



Fonte: Elaborada pelo autor

6.5 Conclusão

As Interfaces Naturais de Usuário são uma área de estudo promissora. O lançamento de novos dispositivos para desenvolvimento de aplicações com esta tecnologia revela uma tendência nas interface humano-computador, e o avanço desta tecnologia para dispositivos móveis, como tablets e celulares, evidencia sua importância, desbravando novos contextos de utilização.

Um dos problemas encontrados nos sitemas que utilizam NUI é a impossibilidade de personalizar poses e gestos. Normalmente os sistemas pré-definem estas opções, não permitindo ao usuário alterá-las em tempo de execução.

Este trabalho apresentou os resultados de uma pesquisa sobre modelagem de poses e gestos em ambiente web e troca de modelos gestuais a partir de um padrão de dados

[2] Disponível em: https://github.com/joaquimrocha/Skeltrack

bem definido. Os resultados produzidos por ela impactam não somente na computação, mas também em diversas outras áreas do conhecimento, como educação e saúde, onde esta solução poderia ser aplicada.

É proposto um ambiente que permita aos próprios usuários definirem quais poses ou gestos, sob sua ótica, são mais intuitivos para executar determinadas ações dentro de um sistema computacional. Outrossim, a arquitetura disponibilizada permite sua utilização por desenvolvedores, a fim de integrar Interfaces Naturais de Usuário em páginas na web.

A disponibilização de um modelo de dados para troca de informações gestuais facilita a comunicação entre os sistemas que porventura utilizarem a arquitetura proposta, permitindo-os validar os dados intercambiados, por meio de um *schema* também fornecido nesta solução.

Estas são lacunas encontradas nos trabalhos pesquisados, conforme apresentado no Capítulo 3. Poucos trabalhos possuem foco na modelagem intuitiva, sendo voltados à desenvolvedores de sistemas e não ao usuário final. A proposta do NuiMod é ser tão simples quanto possível, para que seja acessível a qualquer tipo de utilizador.

O desenvolvimento da solução foi realizada em três módulos distintos, a fim de facilitar o desenvolvimento de novas funcionalidades e com pontos de extensão, para simplificar a personalização operações executadas. Desta forma, é possível alterar desde o formato de entrada de dados até a forma de processamento de poses e gestos.

O *Connector* é o componente responsável pela comunicação entre o dispositivo e a aplicação web. Ele disponibiliza, por meio do protocolo websocket, um serviço para recuperação dos dados de esqueleto. As mensagens são enviadas e recebidas por meio de JSON, em um formato de dados bem definido. Desta forma, permite-se que o navegador se comunique com o sensor de movimentos e obtenha acesso aos dados de rastreio do esqueleto.

A *Engine* realiza o reconhecimento dos componentes e dispara gatilhos para a aplicação. Para o reconhecimento de estabilidade foi utilizado o algoritmo de distância euclidiana, que se mostrou eficiente para esta finalidade. O reconhecimento de poses e gestos foi realizado por meio do cálculo de ângulo das articulações.

A modelagem dos componentes é realizada por meio do *Editor*, um ambiente web desenvolvido em Python e Django. Ele disponibiliza os controles necessários para a comunicação com o *Connector*, e faz uso da *Engine* para controlar a edição dos componentes. A modelagem guiada, permite aos usuários personalizar facilmente as ações dentro dos sistemas que fazem uso do NuiMod. É possível ainda acessar os projetos criados por meio de uma API, ou exportá-los no formato NMD.

Os testes realizados com a ferramenta demonstram a eficácia da *Engine* em reconhecer os componentes projetados pelo *Editor*. Além disso, os experimentos descritos no

Capítulo 4, demonstram a possibilidade de utilização da ferramente tanto em soluções concebidas para utilizarem sensores de movimento, quanto na adaptação de aplicações já existentes.

Referências

ALCHIN, M.; KAPLAN-MOSS, J.; VILCHES, G. *Pro Django*. [S.l.]: Springer, 2009. Citado na página 68.

ASUS. *Especificações Xtion PRO Live*. 2013. Disponível em: <http://www.asus.com/Multimedia/Xtion_PRO_LIVE/specifications/>. Acesso em: 20.6.2014. Citado 2 vezes nas páginas 28 e 29.

BATLLE, J.; MOUADDIB, E.; SALVI, J. Recent progress in coded structured light as a technique to solve the correspondence problem: a survey. *Pattern Recognition*, v. 31, n. 7, p. 963–982, 1998. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320397000745>. Citado na página 26.

BESL, P. J. Active, optical range imaging sensors. *Machine vision and applications*, Springer, v. 1, n. 2, p. 127–152, 1988. Citado na página 25.

BLAKE, J. The natural user interface revolution. *Natural User Interfaces in .NET, Manning*, p. 1–35, 2011. Citado na página 19.

BORGHESE, N. A. et al. An intelligent game engine for the at-home rehabilitation of stroke patients. In: *Serious Games and Applications for Health (SeGAH), 2013 IEEE 2nd International Conference on*. [s.n.], 2013. p. 1–8. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6665318>. Citado 2 vezes nas páginas 43 e 44.

CASTIELLO, U. et al. Improving left hemispatial neglect using virtual reality. *Neurology*, AAN Enterprises, v. 62, n. 11, p. 1958–1962, 2004. Citado na página 43.

CATUHE, D. *Programming with the Kinect for Windows Software Development Kit: Add gesture and posture recognition to your applications*. Microsoft Press, 2012. (Developer Reference). ISBN 9780735667815. Disponível em: <http://books.google.com.br/books?id=FMFrpkD91MQC>. Citado 4 vezes nas páginas 26, 28, 36 e 37.

de Oliveira Marin, L. Métodos Estatísticos no Reconhecimento de Faces. *Revista Eletrônica de Sistemas de Informação ISSN 1677-3071 doi: 10.5329/RESI*, v. 5, n. 2, 2006. Citado na página 60.

DIX, A. *Human-computer interaction*. [S.l.]: Springer, 2009. Citado na página 23.

FAISON, T. *Event-Based Programming: Taking Events to the Limit*. 1st. ed. Berkely, CA, USA: Apress, 2011. ISBN 1430243260, 9781430243267. Citado na página 60.

FETTE, I.; MELNIKOV, A. The websocket protocol. 2011. Disponível em: <http://tools.ietf.org/html/rfc6455>. Citado na página 65.

GIL, A. C. Como elaborar projetos de pesquisa. *São Paulo*, v. 5, 2002. Citado na página 21.

GOKTURK, S. B.; YALCIN, H.; BAMJI, C. A time-of-flight depth sensor-system description, issues and solutions. In: IEEE. *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*. [S.l.], 2004. p. 35–35. Citado na página 25.

GUEDES, G. Interface humano computador: prática pedagógica para ambientes virtuais. *Teresina: EDUFPI*, 2008. Citado na página 18.

HOSKINS, D. J. *XML and InDesign*. [S.l.]: "O'Reilly Media, Inc.", 2013. Citado na página 65.

JANA, A. *Kinect for Windows SDK Programming Guide*. [S.l.]: Packt Publishing Ltd, 2012. Citado na página 27.

KAMEL BOULOS, M. et al. Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation. *International Journal of Health Geographics*, v. 10, n. 1, p. 45, 2011. Disponível em: <http://www.ij-healthgeographics.com/content/10/1/45>. Citado 3 vezes nas páginas 19, 39 e 53.

KHOSHELHAM, K. Accuracy analysis of kinect depth data. In: *ISPRS workshop laser scanning*. [S.l.: s.n.], 2011. v. 38, n. 5, p. W12. Citado na página 26.

KINECTJS. *Portal de Documentação do KinectJS*. 2014. Disponível em: <kinect.childnodes.com/docs>. Acesso em: 17.7.2014. Citado 2 vezes nas páginas 19 e 50.

LEE, E.; LIU, X.; ZHANG, X. Xdigit: An arithmetic kinect game to enhance math learning experiences. *Retrieved February*, Citeseer, v. 14, p. 2013, 2012. Citado 3 vezes nas páginas 19, 41 e 53.

LIEBLING, D.; MORRIS, M. R. Kinected browser: Depth camera interaction for the web. In: *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*. New York, NY, USA: ACM, 2012. (ITS '12), p. 105–108. ISBN 978-1-4503-1209-7. Disponível em: <http://doi.acm.org/10.1145/2396636.2396652>. Citado 2 vezes nas páginas 50 e 51.

LORIOT, Y. *How to install and use OpenNI on Windows*. 2011. Disponível em: <http://yannickloriot.com/2011/03/kinect-how-to-install-and-use-openni-on-windows-part-1/>. Acesso em: 20.6.2014. Citado na página 35.

LUTZ, M. *Learning python*. [S.l.]: "O'Reilly Media, Inc.", 2013. Citado na página 68.

MASSE, M. *REST API design rulebook*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 74.

MATHERS, C. D.; LONCAR, D. Projections of global mortality and burden of disease from 2002 to 2030. *PLoS medicine*, Public Library of Science, v. 3, n. 11, p. e442, 2006. Citado na página 43.

NEBELING, M. et al. Xdkinect: development framework for cross-device interaction using kinect. In: ACM. *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*. [S.l.], 2014. p. 65–74. Citado na página 52.

NEBELING, M. et al. XDKinect: Development Framework for Cross-device Interaction Using Kinect. In: *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York, NY, USA: ACM, 2014. (EICS '14), p. 65–74. ISBN 978-1-4503-2725-1. Disponível em: <http://doi.acm.org/10.1145/2607023.2607024>. Citado na página 53.

NETO, A. N. R.; SANTOS, C. A. S.; CARVALHO, L. A. de. Touch the Air: An Event-driven Framework for Interactive Environments. In: *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2013. (WebMedia '13), p. 73–80. ISBN 978-1-4503-2559-2. Disponível em: <http://doi.acm.org/10.1145/2526188.2526216>. Citado na página 19.

NORMAN, D. A. *The design of everyday things*. [S.l.]: Basic books, 2002. Citado na página 24.

NOYES, B. *Smart Client Deployment with ClickOnce: Deploying Windows Forms Applications with ClickOnce*. [S.l.]: Pearson Education, 2006. Citado na página 67.

OCCIPITAL. *Structure.IO Developer Portal*. 2014. Disponível em: <http://structure.io/developers>. Acesso em: 10.4.2014. Citado 2 vezes nas páginas 31 e 32.

OPENKINECT. *Website do projeto OpenKinect*. 2014. Disponível em: <http://openkinect.org/wiki/Main_Page>. Acesso em: 2.7.2014. Citado na página 35.

PARVINI, F.; SHAHABI, C. An algorithmic approach for static and dynamic gesture recognition utilising mechanical and biomechanical characteristics. *International journal of bioinformatics research and applications*, Inderscience, v. 3, n. 1, p. 4–23, 2007. Citado na página 61.

PEDERSOLI, F. et al. *XKin -: eXtendable hand pose and gesture recognition library for kinect*. Nara, Japan: ACM, 2012. 1465–1468 p. Citado 3 vezes nas páginas 19, 48 e 49.

PILGRIM, M. *Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox*. [S.l.]: "O'Reilly Media, Inc.", 2005. Citado na página 83.

PRIMESENSE. *PrimeSense 3D Sensors DataSheet*. 2013. 4 p. Disponível em: <http://www.i3du.gr/pdf/primesense.pdf>. Acesso em: 20.7.2014. Citado 2 vezes nas páginas 29 e 30.

RATNAYAKE, R. N. *Building Impressive Presentations with impress. js*. [S.l.]: Packt Publishing Ltd, 2013. Citado na página 79.

SANTANA, O.; GALESI, T. Python e Django - Desenvolvimento ágil de aplicações web. *São Paulo: Novatec*, 2010. Citado na página 68.

SILVA, A. J. da; NETO, E. B. C. Utilização do Kinect como ferramenta de apoio ao processo de ensino e aprendizado. In: *VII CONNEPI - Congresso Norte Nordeste de Pesquisa e Inovação*. [S.l.: s.n.], 2012. Citado na página 19.

SOFTKINECT. *SoftKinect DepthSense Cameras*. 2013. Disponível em: <http://www.softkinetic.com/en-us/products/depthsensecameras.aspx>. Acesso em: 1.8.2014. Citado na página 34.

SOFTKINETIC. *DS325 Datasheet*. 2013. Disponível em: <http://www.softkinetic.com/Portals/0/Documents/PDF/WEB_20130527_SK_DS325_Datasheet_V4.0.pdf>. Acesso em: 10.8.2014. Citado na página 33.

SOUZA, C. S. de et al. Projeto de Interfaces de Usuário: perspectivas cognitivas e semióticas. In: *Anais da Jornada de Atualização em Informática, XIX Congresso da Sociedade Brasileira de Computação, Rio de Janeiro*. [S.l.: s.n.], 1999. Citado na página 18.

SUAREZ, J.; MURPHY, R. R. Hand gesture recognition with depth images: A review. In: *RO-MAN, 2012 IEEE*. [S.l.: s.n.], 2012. p. 411–417. ISSN 1944-9445. Citado na página 29.

SUMA, E. A. et al. Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit. *Computers & Graphics*, Elsevier, v. 37, n. 3, p. 193–201, 2013. Citado na página 45.

SUMA, E. A. et al. FAAST-R: Defining a Core Mechanic for Designing Gestural Interfaces. In: *The 3rd Dimension of CHI: Touching and Designing 3D User Interfaces*. [s.n.], 2012. p. 35–42. Disponível em: <http://people.ict.usc.edu/~suma/papers/suma-3dchi2012.pdf>. Citado na página 45.

SUMA, E. A. et al. FAAST: The Flexible Action and Articulated Skeleton Toolkit. In: *2011 IEEE Virtual Reality Conference*. IEEE, 2011. p. 247–248. ISBN 978-1-4577-0039-2. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5759491>. Citado 2 vezes nas páginas 19 e 45.

Taylor II, R. M. et al. VRPN: A Device-independent, Network-transparent VR Peripheral System. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA: ACM, 2001. (VRST '01), p. 55–61. ISBN 1-58113-427-4. Disponível em: <http://doi.acm.org/10.1145/505008.505019>. Citado na página 45.

VALLI, A. The design of natural interaction. *Multimedia Tools and Applications*, Springer, v. 38, n. 3, p. 295–305, 2008. Citado na página 24.

VILLAROMAN, N.; ROWE, D.; SWAN, B. *Teaching natural user interaction using OpenNI and the Microsoft Kinect sensor*. West Point, New York, USA: ACM, 2011. 227–232 p. Citado 2 vezes nas páginas 19 e 53.

WEICHERT, F. et al. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 13, n. 5, p. 6380–6393, 2013. Citado 2 vezes nas páginas 30 e 31.

WEISE, T. et al. Online loop closure for real-time interactive 3D scanning. *Computer Vision and Image Understanding*, v. 115, n. 5, p. 635–648, 2011. ISSN 1077-3142. Disponível em: <http://www.sciencedirect.com/science/article/pii/S107731421000264X>. Citado na página 26.

XDIGIT. *XDigit Website*. 2013. Disponível em: <http://www.elwinlee.com/portfolio/game/xdigit/>. Acesso em: 11.6.2014. Citado 2 vezes nas páginas 42 e 43.

Apêndices

APÊNDICE A – NuiMod XML Schema

Este apêndice apresenta o XML Schema criado para validar os arquivos NMD gerados pelo NuiMod. O *schema* foi dividido em dois *namespaces*, a fim de reaproveitar o vocabulário comum que descreve esqueletos humanos rastreados por dispositivos de interação natural.

O *namespace* `http://nuimod.com.br/2014/CommonTypes` armazena os tipos comuns: *Skeleton* (Esqueleto), *Joint* (Articulação), *JointType* (Tipo de articulação), *TrackingState* (Estado de rastreio), *Pose* (Pose) e *Gesture* (Gesto). Este namespace pode ser utilizado por aplicações terceira que desejarem representar apenas dados de esqueleto, sem levar em consideração informações sobre projetos desenvolvidos no NuiMod.

O *namespace* `http://nuimod.com.br/2014/NuiModSchema` é especifico para o NuiMod, utilizado para representar os projetos criados com a ferramenta. Por meio dele são especificados os seguintes tipos: *Slug* (Slug), *Component* (Componente), *Pose* (Pose), *Gesture* (Gesto) e *Project* (Projeto).

Uma vez que *XML Schema* não suporta herança múltipla, Poses e Gestos precisaram ser especificadas duas vezes: a primeira (no *namespace* CommonTypes), para representá-las apenas como estruturas do corpo humano e a segunda (no *namespace* NuiModSchema) para representá-las como componentes do NuiMod. No *namespace* NuiModSchema, são incluidas informações como nome, descrição e dados do gatilho, que não interessam ao primeiro contexto.

A primeira parte do apêndice especifica a sintaxe e semântica necessárias para representar dados de esqueleto genéricos. A seguir, apresentamos a segunda parte da especificação, que determina as estruturas relacionadas ao escopo do NuiMod.

A.1 Parte I - Dados Comuns

Esta seção descreve os tipos comuns a mais de uma parte da especificação NuiMod, incluindo as convenções de esquema e tipos de dados básicos, relacionados com a representação do esqueleto humano. Os tipos definidos nesta subdivisão podem ser utilizados em combinação com a Parte II da especificação ou instanciados individualmente, para representar dados genéricos de esqueleto. A sintaxe definida nesta seção assume o cabeçalho apresentado na Figura 47 a fim de formar um documento de *XML Schema* válido.

Figura 47 – Cabeçalho definido para o *namespace* CommonTypes

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="http://nuimod.com.br/2014/CommonTypes"
xmlns='http://nuimod.com.br/2014/CommonTypes'
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
...
</xs:schema>
```

Fonte: Elaborada pelo autor

A.1.1 Articulações

Esta tópico descreve a estrutura dos tipos de dados para representar o estado de rastreio e as articulações do corpo humano. A sintaxe está especificada na Figura A.1.1 e a semântica descrita na tabela 8.

Tabela 8 – Semântica de articulações

Nome	**Definição**
TrackingStateType	Este tipo descreve o estado de rastreamento de uma articulação. Os possíveis valores para este tipo são: Não rastreado (0), Inferido (1) e Rastreado (2)
JointType	Descreve os possíveis tipos de articulações de um esqueleto. São definidas 20 diferentes articulações: HipCenter (0), Spine (1), ShoulderCenter (2), Head (3), ShoulderLeft (4), ElbowLeft (5), WristLeft (6), HandLeft (7), ShoulderRight (8), ElbowRight (9), WristRight (10), HandRight (11), HipLeft (12), KneeLeft (13), AnkleLeft (14), FootLeft (15), HipRight (16), KneeRight (17), AnkleRight (18), FootRight (19)
Joint	Descreve uma articulação, com as coordenadas x, y e z no plano cartesiano, expressas em metros. Devem ser informados ainda o tipo de articulação e seu estado de rastreio (campos obrigatórios).

Fonte: Elaborada pelo autor

Figura 48 – Sintaxe para especificação de articulações

```
<!-- ########################################################## -->
<!-- Tracking State Datatype -->
<!-- ########################################################## -->
<xs:simpleType name="TrackingStateType">
     <xs:restriction base="xs:integer">
          <xs:enumeration value="0" />
          <xs:enumeration value="1" />
          <xs:enumeration value="2" />
     </xs:restriction>
</xs:simpleType>

<!-- ########################################################## -->
<!-- Joint Type Datatype -->
<!-- ########################################################## -->
<xs:simpleType name="JointTypeType">
     <xs:restriction base="xs:integer">
          <xs:minInclusive value="0" />
          <xs:maxInclusive value="19" />
     </xs:restriction>
</xs:simpleType>

<!-- ########################################################## -->
<!-- Joint Datatype -->
<!-- ########################################################## -->
<xs:complexType name="JointType">
     <xs:sequence>
          <xs:element name="x" type="xs:double" />
          <xs:element name="y" type="xs:double" />
          <xs:element name="z" type="xs:double" />
     </xs:sequence>
     <xs:attribute name="tracking_state" type="TrackingStateType"
          use="required" />
     <xs:attribute name="type" type="JointTypeType" use="required" />
</xs:complexType>
```

Fonte: Elaborada pelo autor

A.1.2 Esqueletos, Poses e Gestos

Esta subseção descreve as estruturas de dados necessárias para representar esqueletos humanos, poses e gestos. Cada esqueleto é representado como um conjunto de até 20 articulações. Poses são definidas como uma representação estática de um esqueleto e gestos representados como um agrupamento de `n` posições de esqueleto. A Figura 49 especifica a sintaxe e a semântica é descrita na tabela 9.

Figura 49 – Sintaxe para especificação de esqueletos, poses e gestos

```
<!-- ########################################################## -->
<!-- Skeleton Datatype -->
<!-- ########################################################## -->
<xs:complexType name="SkeletonType">
      <xs:sequence>
            <xs:element name="joints" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:sequence>
                              <xs:element name="joint" type="JointType"
                                    MinOccurs="20" maxOccurs="20" />
                        </xs:sequence>
                  </xs:complexType>
            </xs:element>
      </xs:sequence>
</xs:complexType>

<!-- ########################################################## -->
<!-- Pose Datatype -->
<!-- ########################################################## -->
<xs:complexType name="PoseType">
      <xs:sequence>
            <xs:element name="skeleton" type="SkeletonType" />
      </xs:sequence>
</xs:complexType>

<!-- ########################################################## -->
<!-- Gesture Datatype -->
<!-- ########################################################## -->
<xs:complexType name="GestureType">
      <xs:sequence>
            <xs:element name="skeletons" minOccurs="1" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:sequence maxOccurs="unbounded">
                              <xs:element name="skeleton" type="SkeletonType"
                                    MinOccurs="2" maxOccurs="unbounded" />
                        </xs:sequence>
                  </xs:complexType>
            </xs:element>
      </xs:sequence>
</xs:complexType>
```

Fonte: Elaborada pelo autor

Tabela 9 – Semântica de esqueletos, poses e gestos

Nome	**Definição**
SkeletonType	Este tipo descreve um esqueleto humano com 20 articulações obrigatórias.
PoseType	Tipo de dado que representa uma pose. Cada pose deve possuir apenas um elemento do tipo esqueleto.
GestureType	Tipo de dado que descreve um gesto, que deve possuir, pelo menos dois elementos do tipo esqueleto

Fonte: Elaborada pelo autor

A.2 Parte II - NuiMod

A Parte II da especificação descreve os tipos de dados exclusivos do NuiMod, a serem utilizados para representar e validar projetos exportados pela ferramenta. A sintaxe definida nesta seção assume o cabeçalho apresentado na Figura 50 a fim de formar um documento de *XML Schema* válido.

Figura 50 – Cabeçalho definido para o *namespace* NuiModSchema

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="http://nuimod.com.br/2014/NuiModSchema"
      xmlns='http://nuimod.com.br/2014/NuiModSchema'
      xns:nmc='http://nuimod.com.br/2014/CommonTypes'
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
...
</xs:schema>
```

Fonte: Elaborada pelo autor

A Figura 52 representa a sintaxe dos tipos de dados utilizados para representar componentes modelados no NuiMod. Os tipos *Pose* e *Gesture* herdam as características de *Component*, que especifica nome, descrição e informações sobre o gatilho.

Os projetos, cuja sintaxe está representada na Figura 51, podem conter de `0` a `n` componentes, divididos em poses e gestos. A estrutura de dados de projeto armazena o nome do projeto, a data de criação e última atualização realizada. A semântica para descrever componentes e projetos está descrita na Tabela 10.

Tabela 10 – Semântica de projetos no NuiMod

Nome	**Definição**
SlugType	Este tipo descreve um rótulo curto, contendo somente letras, números, sublinhados ou hífens.
ComponentType	Descreve um componente modelado no *NuiMod Editor*. Cada componente possui um nome, descrição, gatilho (SlugField) e intervalo do gatilho.
PoseType	Representa uma pose modelada no sistema. Cada pose deve conter apenas uma definição de esqueleto.
GestureType	Descreve um gesto modelado no *NuiMod Editor*. Cada gesto deve possuir pelo menos dois esqueletos.
ProjectType	Este tipo descreve um projeto do NuiMod. O nome do projeto, sua data de criação e última alteração são campos obrigatórios. Cada projeto pode ter vários componentes.

Fonte: Elaborada pelo autor

Figura 51 – Sintaxe definida pra representar projetos no NuiMod

```
<!-- ############################################################ -->
<!-- Project Datatype -->
<!-- ############################################################ -->
<xs:complexType name="ProjectType">
     <xs:all>
          <xs:element name="components" maxOccurs="1">
               <xs:complexType>
                    <xs:choice maxOccurs="unbounded">
                         <xs:element name="pose"
                              type="PoseType" minOccurs="0"
                              maxOccurs="unbounded" />
                         <xs:element name="gesture"
                              type="GestureType" minOccurs="0"
                              maxOccurs="unbounded" />
                    </xs:choice>
               </xs:complexType>
          </xs:element>
     </xs:all>
     <xs:attribute name="name" type="xs:string" />
     <xs:attribute name="date_created" type="xs:dateTime" />
     <xs:attribute name="date_updated" type="xs:dateTime" />
</xs:complexType>
```

Fonte: Elaborada pelo autor

Figura 52 – Sintaxe definida pra representar componentes no NuiMod

```
<!-- ############################################################ -->
<!-- Slug Datatype -->
<!-- ############################################################ -->
<xs:simpleType name="SlugType">
      <xs:restriction base="xs:string">
            <xs:pattern value="[-a-zA-Z0-9_]+" />
      </xs:restriction>
</xs:simpleType>

<!-- ############################################################ -->
<!-- Component Datatype -->
<!-- ############################################################ -->
<xs:complexType name="ComponentType">
      <xs:attribute name="description" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="trigger" type="SlugType" />
      <xs:attribute name="trigger_interval" type="xs:positiveInteger" />
</xs:complexType>

<!-- ############################################################ -->
<!-- Pose Datatype -->
<!-- ############################################################ -->
<xs:complexType name="PoseType">
      <xs:complexContent>
            <xs:extension base="ComponentType">
                  <xs:sequence>
                        <xs:element name="skeleton"
                                    type="nmc:SkeletonType" />
                  </xs:sequence>
            </xs:extension>
      </xs:complexContent>
</xs:complexType>

<!-- ############################################################ -->
<!-- Gesture Datatype -->
<!-- ############################################################ -->
<xs:complexType name="GestureType">
      <xs:complexContent>
            <xs:extension base="ComponentType">
                  <xs:sequence>
                        <xs:element name="skeletons" minOccurs="1"
                                          maxOccurs="unbounded">
                              <xs:complexType>
                                    <xs:sequence maxOccurs="unbounded">
                                          <xs:element name="skeleton"
                                                type="nmc:SkeletonType"
                                                minOccurs="2"
                                                maxOccurs="unbounded" />
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                  </xs:sequence>
            </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

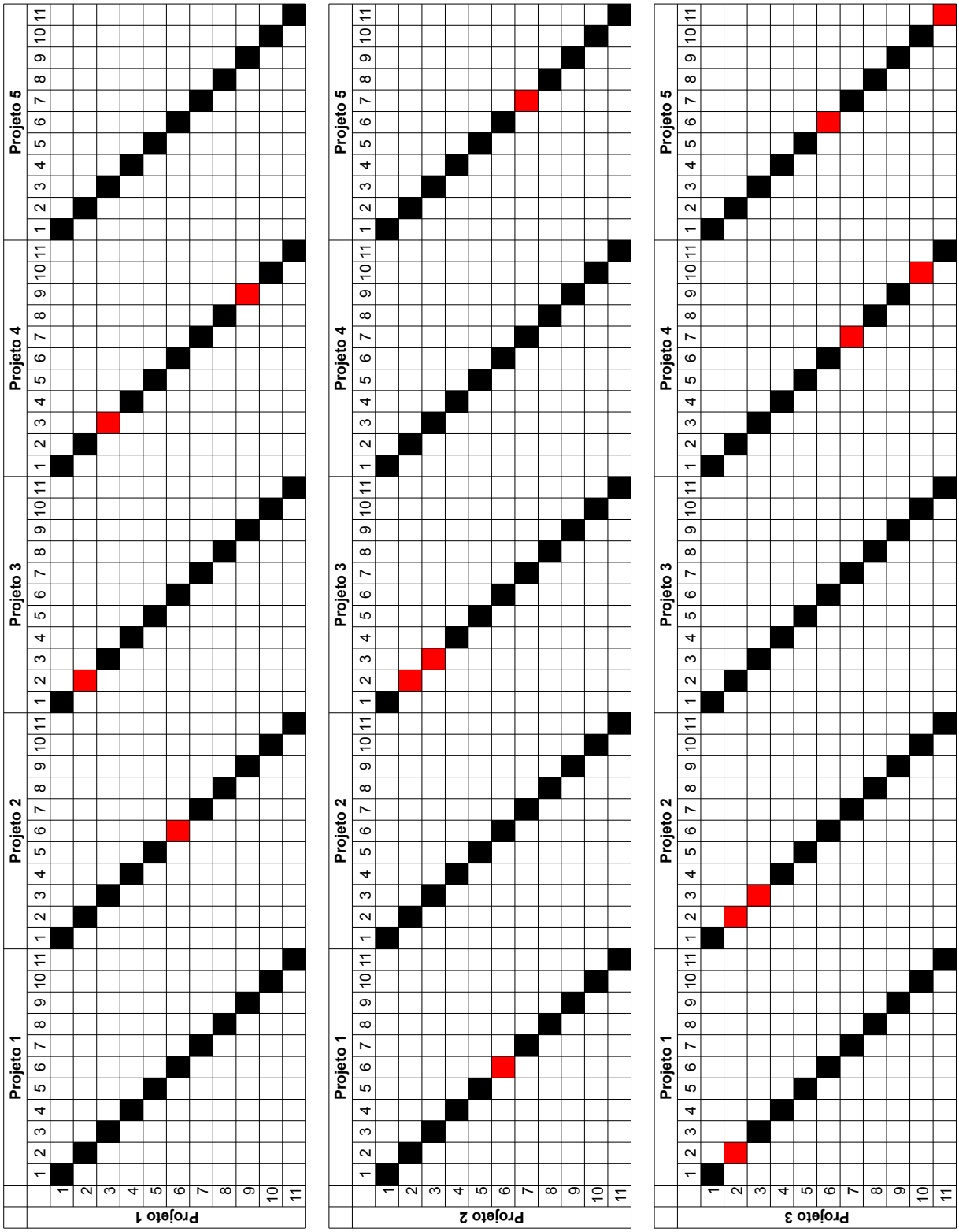
Fonte: Elaborada pelo autor

APÊNDICE B – Dados de precisão da *Engine*

Este apêndice apresenta os resultados de comparação entre poses obtidos a partir do experimento realizado na seção 5.1. Cada projeto foi comparado com outros 5 projetos, incluindo ele mesmo. Os resultados podem ser conferidos nas Figuras 53 e 54.

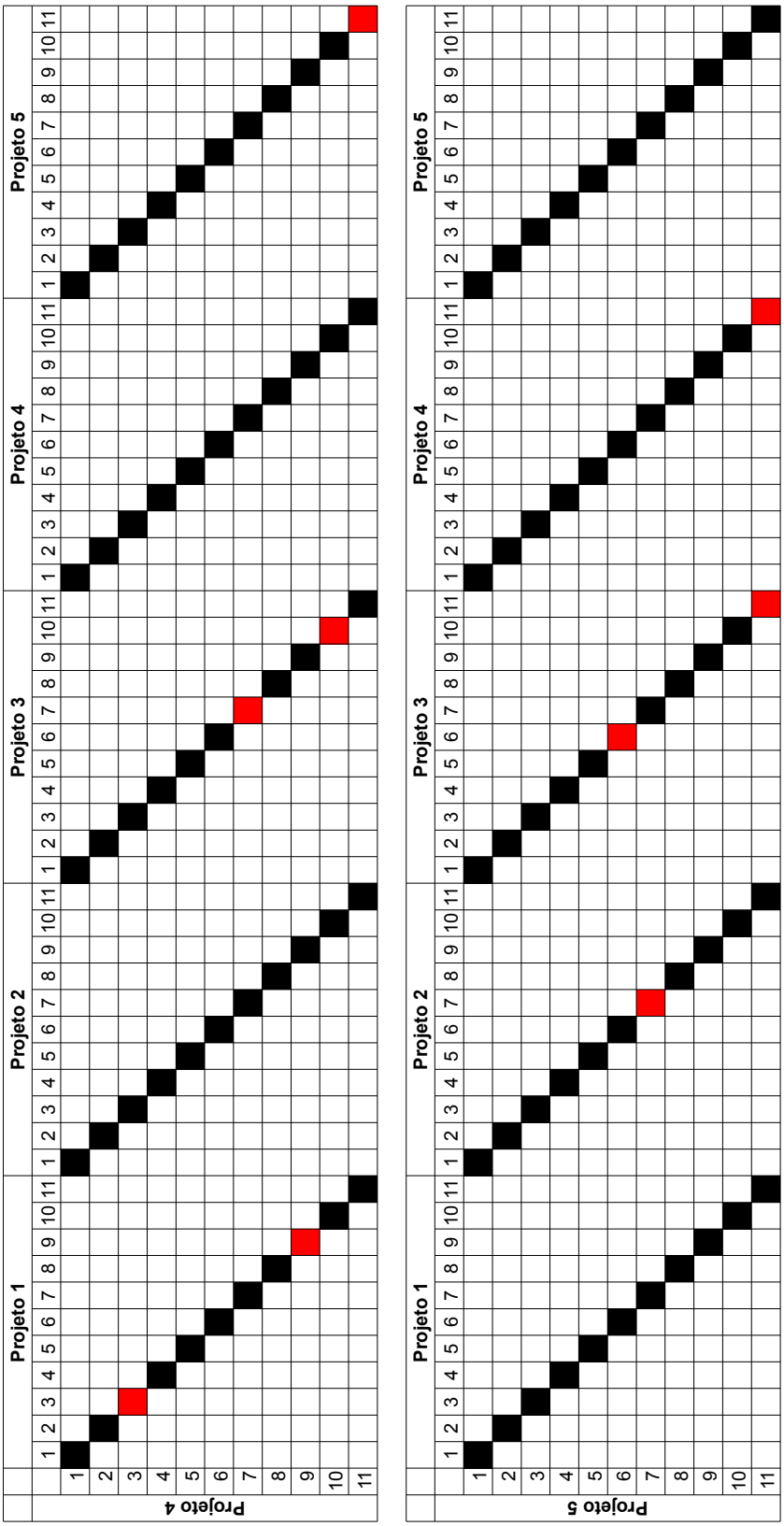
As celulas marcadas em vermelho indicam poses que deveriam ter sido reconhecidas (uma vez são similaridade) e não foram. As celulas marcadas com o fundo preto, indicam as ocorrências corretas, ou seja, aquelas em que realmente as poses são similares. Por sua vez, as celular sem qualquer marcação (fundo branco) indicam que não foi reconhecido similaridade entre as poses comparadas, onde este seria o resultado esperado.

Figura 53 – Dados de comparação para os Projetos 1, 2 e 3



Fonte: Elaborada pelo autor

Figura 54 – Dados de comparação para os Projetos 4 e 5



Fonte: Elaborada pelo autor

APÊNDICE C – Cronograma da dissertação

A fim de visualizar e acompanhar todas as etapas de desenvolvimento desta dissertação foi elaborado um cronograma, dividido em 5 marcos principais: Projeto, Pesquisa, Prototipação, Experimentação e Dissertação.

O cronograma foi elaborado levando em consideração as etapas elencadas na metodologia de pesquisa. A Fase 1 - Projeto, representa a etapa de definição objetivos e questões norteadoras da dissertação.

Na Fase 2 - Pesquisa foi realizado o levantamento bibliográfico seguido pela prototipação (Fase 3). Paralelamente, iniciou-se a escrita da dissertação (Fase 5). A fase de experimentação (Fase 4) foi realizada no final do projeto.

A Figura 55 ilustra cada etapa do desenvolvimento deste trabalho, com suas restrições e duração estimadas. Para lidar com possíveis mudanças de prazo, foi criado um fluxograma (Figura56) que detalha as ações contingenciais que devem ser tomadas em caso de atraso no cronograma.

O fluxograma de controle de mudanças determina que apenas as ações que implicarem em desvios no cronograma devem ser analisadas. Nesta hipótese, são analisadas os prazos das atividades impactadas, elaborado um plano de ação e encaminhado para o orientador para parecer.

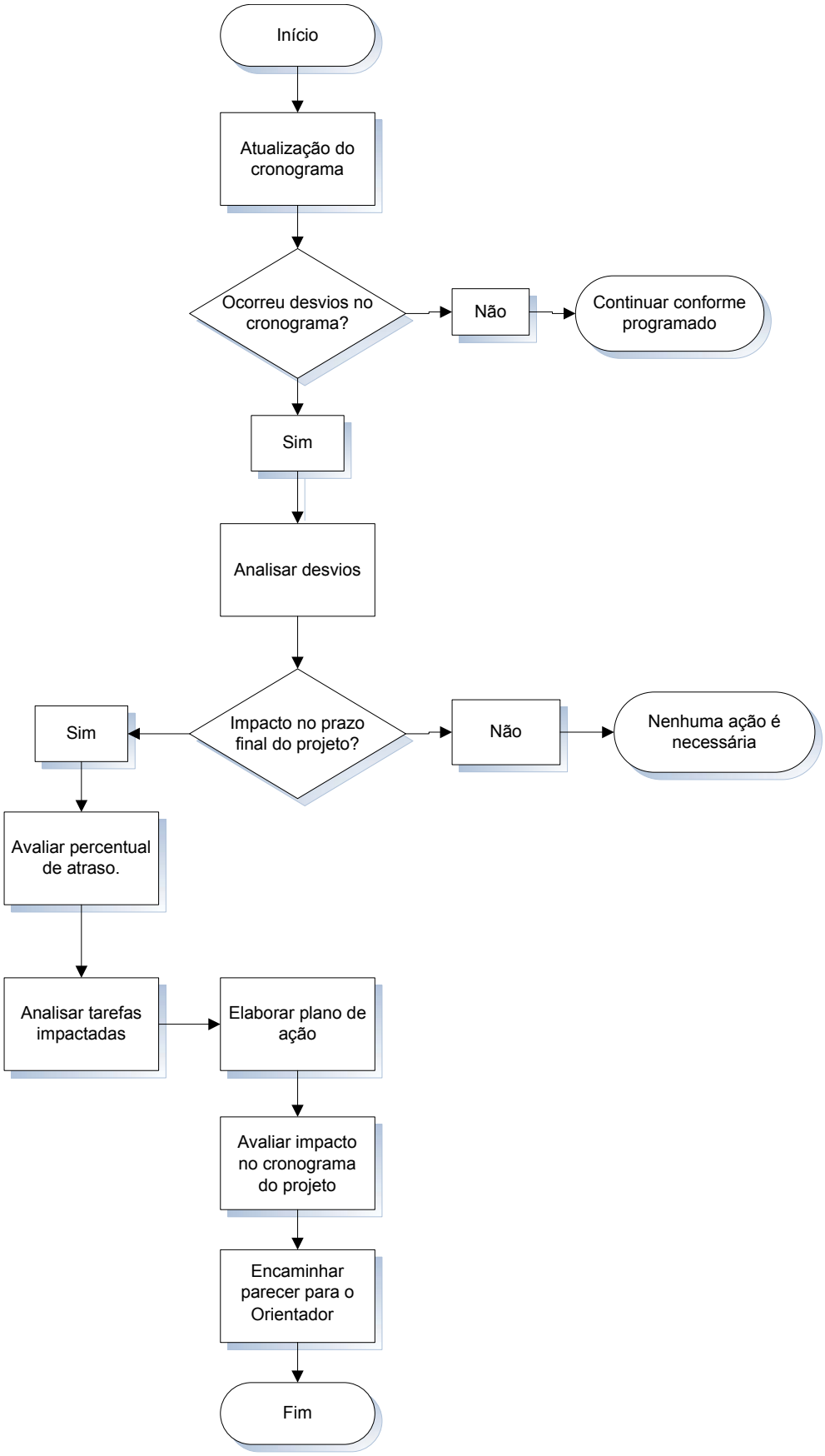
Este recurso foi utilizado apenas uma vez durante o desenvolvimento da dissertação, na fase de Prototipação, quando houve a aquisição da PrimeSense pela Apple, que impactou no desenvolvimento do *Connector*. Todavia, como não houve mudança no prazo final do projeto, nenhuma ação precisou ser tomada.

Figura 55 – Cronograma de desenvolvimento da dissertação.

		Nome	Duração	Predecessores
1		**Pesquisa NuiMod**	**597 dias**	
2		**Projeto**	**140 dias**	
3		Pesquisa de trabalhos na área	120 dias	
4		Definição de objetivos	10 dias	3
5		Definição de questões norteadoras	10 dias	4
6		**Pesquisa**	**211 dias**	**2**
7		Revisão bibliográfica	120 dias	
8		Resposta às questões norteadoras	30 dias	7
9		Referencial teórico	30 dias	8
10		Trabalhos correlatos	30 dias	9
11		Cronograma preliminar	1 dia	10
12		**Prototipação**	**156 dias**	**6**
13		Cronograma final do protótipo	1 dia	
14		Montagem de ambiente	5 dias	13
15		Experimentos iniciais	30 dias	14
16		Desenvolvimento e testes	120 dias	15
17		**Experimentação**	**90 dias**	**16**
18		Definição de cenários	20 dias	
19		Criação de roteiro	10 dias	18
20		Desenvolvimento e testes	60 dias	19
21		**Dissertação**	**64 dias**	**6**
22		Definição de estrutura	5 dias	
23		Cronograma final da dissertação	2 dias	22
24		**Escrita**	**26 dias**	
25		Capítulo 1 e 2	10 dias	23
26		Revisão parcial orientador	3 dias	25
27		Capítulo 3 e 4	10 dias	23
28		Revisão parcial orientador	3 dias	27
29		Capítulo 5 e 6	10 dias	23
30		Revisão parcial orientador	3 dias	29
31		Capítulo 7 e 8	10 dias	30
32		Revisão parcial orientador	3 dias	31
33		Revisão geral	5 dias	24
34		Análise da banca	15 dias	33
35		Defesa	1 dia	34
36		Revisão pós-defesa	10 dias	35

Fonte: Elaborada pelo autor

Figura 56 – Controle de mudanças no cronograma.



Fonte: Elaborada pelo autor