UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELETRICA E INFORMATICA

DEPARTAMENTO DE ENGENHARIA ELETRICA

TRABALHO DE CONCLUSÃO DE CURSO – TCC

Extensão da representação em Realidade Virtual do Simulador do ambiente de operação de subestações de um sistema elétrico

Aluno: Erick de Miranda Lucena

Orientadora: Maria de Fátima Queiroz Vieira, Ph.D.

Campina Grande – Paraíba

Abril de 2010

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELETRICA E INFORMATICA

DEPARTAMENTO DE ENGENHARIA ELETRICA

Trabalho de conclusão de curso apresentado à coordenação do curso de Engenharia Elétrica da UFCG, como parte dos requisitos à obtenção do grau de bacharel em Engenharia Elétrica.

Erick de Miranda Lucena
Aluno

Fatima Vieira

Ph.D. Maria de Fátima Queiroz Vieira
Orientador

Campina Grande - PB

Abril de 2010

AGRADECIMENTOS

A Deus por me dar sabedoria, coragem e determinação para superar todos os desafios.

À minha mãe e meu pai que tudo fizeram para me apoiar e me tranqüilizar perante todas as situações adversas encontradas.

Aos colegas de apartamento Ricardo, Adhéus, Rafael, Geraldo por terem dividido comigo momentos de alegria e tristeza durante todo este tempo, dando apoio e torcida.

A todos os colegas de curso, por terem convivido comigo todos estes anos de vida acadêmica.

A professora Maria de Fátima Q. Vieira por toda ajuda e conhecimento que alicerçaram esse trabalho.

Agradeço também a todos aqueles, que não por menor importância, não foram citados, mas que também tiveram grande contribuição na realização do sonho de adquirir o título de Engenheiro Eletricista.

SUMÁRIO

ÍNDICE DE FIGURAS

1. INTRODUÇÃO

Os ambientes em realidade virtual tem se tornado uma alternativa interessante para o desenvolvimento de interfaces mais realistas e intuitivas para os usuários, de modo com que estes possam navegar e interagir com o ambiente de simulação da realidade.

O projeto do simulador consiste de uma descrição em realidade virtual da sala de comando da CGD, localizada no município de Campina Grande, que compreende tanto painéis de controle como também, uma representação do controle feito através de computadores. Devido a dificuldade de realização de treinamento utilizando o sistema real, torna-se interessante construir um sistema que o represente desconectado do ambiente real, tornando o treinamento mais seguro e flexível.

Dentre os aspectos a serem incorporados ao projeto foi detectada a falta de interação do usuário com o ambiente físico da SE, aumentando o nível de realismo do simulador. Assim foi proposta uma nova dimensão de interação com o ambiente físico e com os equipamentos na sala de comando da subestação. Além de portas interativas, interruptores da sala de comando, relógio de parede, foram animados os mostradores digitais e analógicos dos painéis de comando: medidores de corrente, tensão e potência.

Este relatório está organizado em seis capítulos, incluindo esta introdução. O Capítulo 2 (Fundamentação teórica) aborda os conceitos 3D, modelagem, JAVA, linguagem X3D e a escolha das ferramentas de desenvolvimento do projeto. O Capítulo 3 (Objetos construídos para o simulador) aborda os conceitos de animação, como foram criados os novos objetos e como devem ser instanciados. No Capítulo 4 (Integração dos objetos ao simulador) é discutido como os objetos criados foram inseridos no contexto do ambiente virtual do simulador. No Capítulo 5 (Conclusões) é feita uma reflexão sobre o trabalho, destacando os objetivos alcançados no projeto e propondo os próximos passos. Finalmente os Anexos A e B consistem na listagem do código gerado ao longo do projeto.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os conceitos que dão base a este trabalho: modelagem de mundos virtuais, a linguagem X3D e Java. Além de detalhar os editores e visualizadores utilizados no projeto.

2.1 MODELAGEM DE MUNDOS VIRTUAIS

A modelagem de mundos virtuais é de fundamental importância num sistema de realidade virtual, definindo a características dos objetos como: forma, aparência, movimento, restrições e mapeamento de dispositivos de E/S. Para isto, os sistemas de realidade virtual levam em conta os diversos aspectos de modelagem, mapeamento e simulação [1].

2.1.1 MODELAGEM GEOMÉTRICA

A modelagem geométrica abrange a descrição da forma dos objetos virtuais através de polígonos, triângulos ou vértices e, de sua aparência, usando textura, reflexão de superfície, cores. A forma poligonal dos objetos pode ser criada usando-se bibliotecas gráficas ou modelos prontos de bancos de dados ou digitalizadores tridimensionais. Os objetos também podem ser criados por programa CAD, como AutoCAD ou 3-D Studio, ou com o uso de editores de realidade virtual [1].

A aparência dos objetos está relacionada principalmente às características de reflexão da superfície e à sua textura. A primeira depende do modelo de iluminação de *Phong* e sombreamentos do tipo facetado, por interpolação de Gourad [1]. Já a segunda é obtida a partir do mapeamento de um padrão de textura do espaço bidimensional sobre os objetos tridimensionais. É como se um filme plástico, com seu padrão de textura, fosse ajustado e colocado sobre o objeto, fazendo parte integrante dele. Com a texturização dos objetos, aumenta-se o nível de detalhe e realismo da cena, fornece-se uma visão mais apurada de profundidade e permite-se a redução

substancial do número de polígonos da cena, propiciando o aumento da taxa de quadros por segundo [1].

2.1.2 MODELAGEM CINEMÁTICA

A modelagem geométrica de um objeto não é suficiente para criar uma animação. Para tanto, é necessário haver a interação, ou seja, deve ser possível pegar objetos, alterar sua posição, mudar a escala, detectar colisões e produzir deformações na superfície. Isso é faz possível através de coordenadas locais dos objetos e de coordenadas gerais, juntamente com matrizes de transformação [1].

2.1.3 MODELAGEM FÍSICA

Visando a obtenção de realismo dos mundos virtuais, os objetos devem se comportar como se fossem reais. Por exemplo, um objeto não pode atravessar outro objeto e deve mover-se respeitando um mínimo de leis físicas. Além disso, suas características físicas também podem ser consideradas (massas, peso, inércia, deformações) [1].

2.1.4 COMPORTAMENTO DO OBJETO

É possível modelar o comportamento de objetos, independentemente da interação com o usuário, tais como relógios, calendários, termômetros e outros agentes inteligentes, acessando quando necessário sensores externos [1].

2.2 TECNOLOGIA X3D

X3D (eXtensible 3D) é um padrão aberto de formato de arquivos e arquiteturas de execução, desenvolvido pelo Web3D Consortium [Web3D] para representar e comunicação de cenas e objetos 3D. É um padrão retificado pela norma ISO/IEC 19776 que provê um sistema para o armazenamento, recuperação e reprodução de conteúdo

3D embutido em aplicações. É um padrão baseado em XML (eXtensible Markup Language) e é considerado como a evolução do VRML [2].

Dado que é uma linguagem baseada em XML, X3D oferece vantagens como, uma melhor interoperabilidade entre aplicações (um dos principais objetivos para a criação do X3D) [2].

O padrão X3D apresenta uma arquitetura modular, que permite a criação de perfis ("profiles"), em camadas, os quais provêem mais funcionalidades para ambientes imersivos e melhor interatividade. Porém, a maioria dos domínios das aplicações não necessita de todos os recursos do padrão nem todas as plataformas existentes dão suporte à variedade de funcionalidades definidas na especificação [2].

Os recursos do padrão X3D são agrupados em componentes (blocos modulares de funcionalidade), definindo uma coleção específica de nós com um conjunto de funcionalidades. Os componentes podem ser estendidos individualmente ou modificados através da adição de novos níveis, ou ainda novos componentes podem ser adicionados para introduzir novos recursos. Um perfil é uma coleção de componentes para um nível específico de suporte. Assim, todos os arquivos X3D requerem a definição do perfil que está em uso [2].

Os perfis definidos atualmente para X3D são dos tipos:

- *Interchange*, que consiste em um perfil básico para comunicação entre aplicações, o qual da suporte às funcionalidades básicas tais como: geometria, texturas, iluminação básica e animação.
- *Interactive* , este perfil permite a interação básica com um ambiente em 3D por meio de nós sensores, como os nós de iluminação (por exemplo, SpotLight ou PointLight));
- *Immersive* este perfil inclui, entre outros, os nós de áudio, fumaça, colisão e scripts;
- *Full* este perfil inclui todos os tipos de nós definidos, tais como NURBS ou H-Anim – animação humanóide)[2].

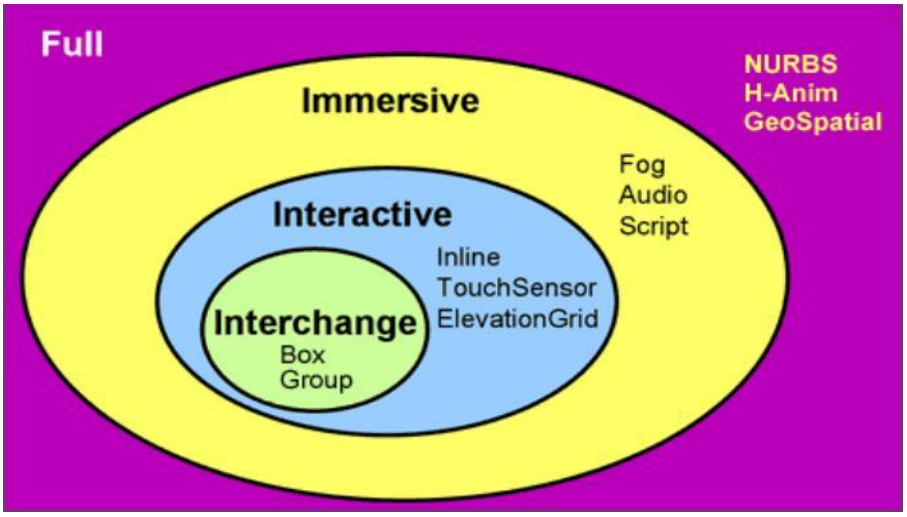

Figura 1 - Perfis (Profiles) definidos para X3D

Dado que X3D consiste na integração de VRML com XML, ela conta com todos os tipos de nós VRML descritos na seção anterior. Também conta com três representações de dados:

- binário, (x3db – sua vantagem é de ter uma representação de menor tamanho, além de facilitar o processamento por máquinas);
- XML padrão (x3d – mais legível para humanos, e por estar codificado em XML, permite interoperabilidade com outras aplicações que usem esse padrão) ;
- VRML clássico (x3dv – facilita a transição para usuários de VRML para X3D).

Na figura 2 é apresentado um exemplo de arquivo de cena X3D.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <X3D>
3.  <Scene>
4.      <Group >
5.          <Shape DEF="Column">
6.              <Cylinder height="10.0" radius="2.0"/>
7.               <Appearance >
8.                  <Material diffuseColor="0 0.19 1"/>
9.               </Appearance>
10.         </Shape>
11.         <Transform translation="20 0.0 0.0" rotation="0.0 0.0 1.0 3.5">
12.             <Transform DEF="cyl1">
13.                 <Shape USE="Column"/>
14.             </Transform>
15.         </Transform>
16.     </Group>
17. </Scene>
18. </X3D>
```

Figura 2 – Descrição de dois cilindros paralelos em X3D

2.2.1 A ESCOLHA DO X3D

A escolha do X3D foi baseada em uma análise realizada entre as linguagens, ferramentas e documentações disponíveis atualmente. Segundo Netto [3], foi constatado que: as linguagens 3DML e xVRML já estavam em desuso, a linguagem 3DXML foi descartada por ser uma linguagem proprietária e, enquanto 3DMLW possui apenas uma ferramenta de edição, uma documentação pobre dificultando seu uso.

As linguagens Java3D e VRML poderiam ter sido utilizadas, mas a descrição dos seus objetos não admite o formato XML. O VRML foi abandonado uma vez que X3D é uma evolução dessa linguagem [3].

A Tabela 1 apresenta o quadro comparativo das linguagens. O símbolo ✓ indica que a linguagem atende o requisito, vazio (em branco) indica que a linguagem não atende o requisito e, o símbolo * indica que o requisito não foi pesquisado dado que outros requisitos críticos não forma atendidos.

	VRML	X3D	Java3D	3DML	3DXML	xVRML	3DMLW
Versão/Ano	2.0/1997	3.0/2009	1.4/2007	2005	2006	2003	2009
Aceitação	✓	✓	✓				
Capacidade de representação	✓	✓	✓	*	*	*	*
Documentação/tutoriais	✓	✓	✓				✓
Ferramentas de edição	✓	✓	✓	*	*	*	✓
Ferramentas de visualização	✓	✓	✓	*	*	*	✓
Não proprietária	✓	✓	✓	✓		✓	✓
Formato em XML		✓			✓	✓	
Autor/Grupo	Web3D	Web3D	Sun	Flatland	Dassault	CMEG	R&D

Tabela 1 - Critérios x Linguagens

Legenda:

✓	Atende o critério.
	Não atende o critério.
*	Não foi pesquisado.

Portanto, depois de análise do quadro e demais fatores, concluiu-se que a linguagem escolhida pra representar RV seria o X3D, pois satisfazia a todos os requisitos.

2.3 JAVA

A linguagem de programação Java possui classes e métodos que permitem a criação de nós Script para interagir com cenas X3D. APIs *(Application Programming Interfaces*) podem ser usadas para manipular uma cena através de uma aplicação externa ao navegador.

A programação de scripts externos é realizada através do método X3D- SAI (*Scene Access Interface).* Nesse método, a aplicação Java se conecta a uma aplicação X3D. Assim, com o uso de métodos específicos, a aplicação Java tem acesso aos nós do programa escrito em X3D. Esse acesso permite que aplicações Java possam afetar o comportamento de qualquer nó, enviando eventos para dentro da aplicação X3D, ou recebendo eventos vindos da aplicação X3D. Para usar o método de script SAI, conta-se com o seguinte pacote:

```
import org.web3d.x3d.sai.*;
```

Depois de importar o pacote mostrado acima, o desenvolvedor deve prover uma forma de a aplicação Java referenciar o programa X3D. Isso é feito através do método **getBrowser**. Depois de obter uma referência à aplicação X3D, é possível obter referências a qualquer um de seus nós, através do método **getNode** passando como parâmetro o nome do nó. Com a referência de um nó, pode-se obter referências para eventos de entrada e saída com os métodos **getEventIn** e **getEventOut**, respectivamente. Com as referências dos eventos de entrada e saída dos nós, pode-se atribuir parâmetros aos atributos desses nós, como o tamanho do raio de um Cone, ou *ouvir* alguma ação vinda do mundo virtual, como o clique do mouse em algum objeto virtual (e.g Cubo). Isso é feito usando os métodos **setValue** e **getValue**, respectivamente. Os eventos que chegam a classe Java são *ouvidos* graças à

implementação da interface **EventOutObserver**, que implica no uso do método **callback**.

A seguir é ilustrado um trecho de código que apresenta a estrutura de uma aplicação Java para tratamento de cenas X3D [4]:

```
import java.awt.BorderLayout;
import javax.swing.*;
import java.util.HashMap;
import org.web3d.x3d.sai.*;

public class CenaX3D extends JFrame {

  public CenaX3D() {
     JFrame frame = new JFrame();

     // Setup browser parameters
     HashMap requestedParameters = new HashMap();

     // Create an SAI component
     X3DComponent x3dComp = BrowserFactory.createX3DComponent(requestedParameters);

     // Add the component to the UI
     JComponent x3dPanel = (JComponent)x3dComp.getImplementation();
     frame.add(x3dPanel, BorderLayout.CENTER);

     // Get an external browser
     ExternalBrowser x3dBrowser = x3dComp.getBrowser();

     // Create an X3D scene by loading a file.  Blocks till the world is loaded.
     X3DScene mainScene = x3dBrowser.createX3DFromURL(new String[] { "arquivoX3D.x3d"
});

     // Replace the current world with the new one
     x3dBrowser.replaceWorld(mainScene);

     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setSize(1024,768);
     frame.setVisible(true);
  }

   public static void main(String[] args) {
     CenaX3D demo = new CenaX3D();
  }
}
```

2.4 ARQUITETURA DO SIMULADOR

Atualmente, a arquitetura do simulador é dividida em três camadas: (1) a camada de modelos, onde estão os modelos redes de Petri que trocam (enviam e recebem) mensagens com o mundo virtual; (2) a camada de comunicação que estabelece a interação entre os modelos formais e a representação X3D e ; (3) a camada de visualização, que corresponde à descrição X3D do simulador.

Essas três camadas são ligadas por interfaces, representadas na Figura 3 por setas bidirecionais, pelas quais elas são capazes de se comunicar uma com a outra.

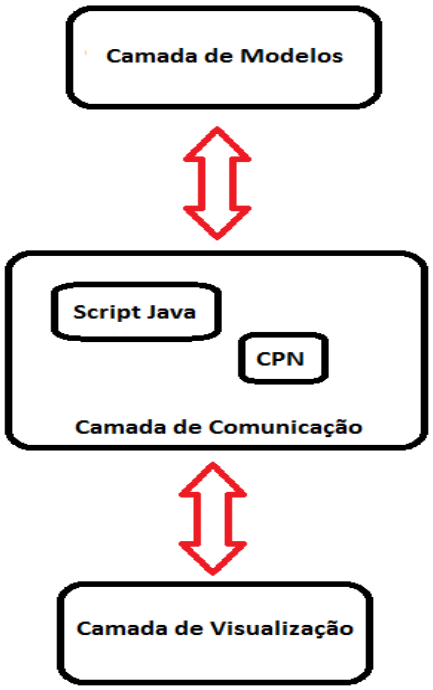

Figura 3 - Arquitetura atual do simulador

A arquitetura de camadas do simulador foi mantida neste projeto. Segundo o escopo deste trabalho, alterações deveriam ser feitas apenas nas camadas de visualização e de comunicação. Na camada de visualização, objetos seriam inseridos na descrição do mundo virtual para torná-lo mais realista. Já na camada de comunicação, foram criadas classes em Java para implementar *scripts* SAI para os novos objetos inserido no mundo virtual.

2.5 PLATAFORMAS E FERRAMENTAS X3D

Atualmente existem diversas plataformas para o desenvolvimento da aplicações X3D. Dentre as existentes foram selecionadas e testadas as mais acessíveis.

2.5.1 EDITORES

- Vivaty Studio

O Vivaty Studio é uma aplicação que utiliza a interface de programas de modelagem 3D aplicados à edição de VRML/X3D. Desenvolvido pela Vivaty como um componente complementar ao Vivaty Player, a atual versão 1.0 Beta, é baseada no software Flux Studio desenvolvido por Tony Parisi e publicado pela Media Machines (Vivaty Studio Player, 2009).

Esta aplicação destina-se a apoiar a criação de cenas em X3D ou VRML possibilitando: modelagem tridimensional, importação de modelos, aplicação de iluminação, fundo, pontos de vista, níveis de detalhe, gestão de navegação, animação, sensores interativos, modelos H-ANIM, extensão das linguagens VRML e X3D com scripts e declarações de protótipos. Além disso, gera arquivos VRML e X3D, com a possibilidade de exportar os objetos criados, em vários formatos 3D (Vivaty Studio Player, 2009).

- X3DEdit

X3D-Edit é um editor gráfico de arquivos para Extensible 3D (X3D) que permite a edição livre de erro, criação e validação de arquivos de cena em X3D ou VRML.

X3D-Edit usa o conjunto de construções XML definidos pelo *X3D Compact Document Type Definition* (DTD), juntamente com: o Java da Sun Microsystems e o editor de XML Xeena da IBM (X3D-Edit, 2009).

- SwirlX3D

SwirlX3D é uma ferramenta para criação e edição de X3D e VRML. A interface de edição gráfica de SwirlX3D dá pleno acesso às estruturas de dados em um arquivo X3D e é permite visualizar uma cena completa (Pinecoast Software, 2009).

2.5.2 VISUALIZADORES

- Vivaty Player

O Vivaty Player é um navegador, de uso gratuito, que implementa o Scene Access Interface (SAI). O Vivaty Player pode ser usado como *plugin* ou s*tandalone* (Studio Player, 2009).

- Xj3D Browser

É uma aplicação escrita em Java, aberta e de uso gratuito, que permite a visualização de mundos virtuais descritos em X3D. Faz parte do Xj3D Toolkit e pode ser embutido em aplicações Java, incluindo *applet* (Xj3D, 2009).

- Octaga player

Desenvolvido pela Octaga, consiste em um visualizador de descrições 3D em tempo real. Seus gráficos são de boa definição, porém o pacote completo do software é pago (Octaga, 2009).

- BS Contact VRML/X3D

É um software para visualização de ambientes 2D/3D, possui suporte para áudio e vídeo, aceita os padrões VRML e X3D, disponibiliza uma versão para teste, e foi desenvolvido pela empresa Bitmanagement Software. Trata-se da evolução da ferramenta Blaxxun (BS Contact, 2009).

- Instantreality

Foi desenvolvido pela Fraunhofer IGD Comments. O instan**treality**-framework apresenta um alto desempenho para sistemas que misturam realidade com realidade virtual *(mixed reality*). Apresenta uma interface simples e consistente (Instantreality, 2009).

2.5.3 FERRAMENTAS UTILIZADAS

Neste projeto foram utilizadas as seguintes ferramentas para a construção e modelagem e, uma ferramenta para visualização da descrição:

- X3DEdit 3.2 como aplicação de construção/modelagem dos objetos.
- Xj3D 2.0 como visualizador.

O X3DEdit 3.2 foi escolhido por ser a ferramenta com a qual o autor tem o melhor domínio. Já o Xj3D 2.0 foi escolhido como a ferramenta de visualização do projeto, por se tratar de um visualizador gratuito e de código aberto que implementa o SAI *(Scene Access Interface).*

No próximo capitulo será mostrado como os objetos foram criados e inseridos no simulador, a partir do X3DEdit 3.2.

3. NOVOS OBJETOS INTRODUZIDOS NO SIMULADOR

Para que o ambiente do simulador oferecesse um maior realismo foram introduzidos novos objetos no ambiente físico da sala de operação da subestação. Os objetos foram criados separadamente a partir do X3DEdit 3.2.

Para manter uma estrutura modular no código do simulador, os novos objetos não foram adicionados à descrição principal, mas foram organizados em arquivos .x3d secundários, o que permitirá o reaproveitamento de código.

3.1 PORTAS INTERATIVAS

A animação de portas, no ambiente físico da subestação, foi desenvolvida com o objetivo de aumentar o realismo da sala de comando, permitindo que o usuário pudesse navegar para o ambiente externo da sala. Este aspecto é importante na medida em que o ambiente externo também será inserido no mundo virtual.

3.1.1 Modelagem dos objetos

Inicialmente, foram modelados os objetos que formam a estrutura da porta: partes metálicas e partes de vidro. Em seguida, utilizando o campo *translation,* do nó *Transform*, estes objetos foram posicionados a fim de criar o objeto porta, como um todo.

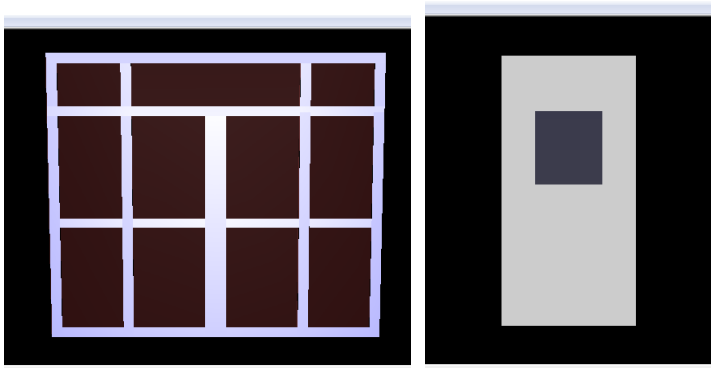

Figura 4 – Portas criadas para o ambiente da sala de operação

O código a seguir representa:

- A posição de uma das partes de vidro da porta, nos eixos x, y e z;
- O centro de rotação do objeto;
- A aparência do material;
- A forma geométrica do objeto;
- O nó *TouchSensor* que torna o objeto sensível ao clique do mouse.

```
<Transform translation='5.96368 -.0061 0'>
        <Shape containerField='children'>
<Appearance containerField='appearance'>
        <Material DEF='WinGlass' containerField='material' ambientIntensity='0.200'
        shininess='0.200' transparency='0.700' diffuseColor='.5 .1 .1' specularColor='.3 .3 .3'/>
</Appearance>
<Box containerField='geometry' size='4 14.65 .05'/>
</Shape>
</Transform>
  <Transform DEF='P_1' translation='-2.255 -1.325 0' center='-2.0 0 0'>
    <TouchSensor DEF='Touch' description='click to operate door'/>
     <Shape>
       <Appearance USE='WinGlass'/>
       <Box size='3.49 .5 .15'/>
     </Shape>
  </Transform>
```

Figura 4 – Trecho de código do objeto de vidro que compõe a porta

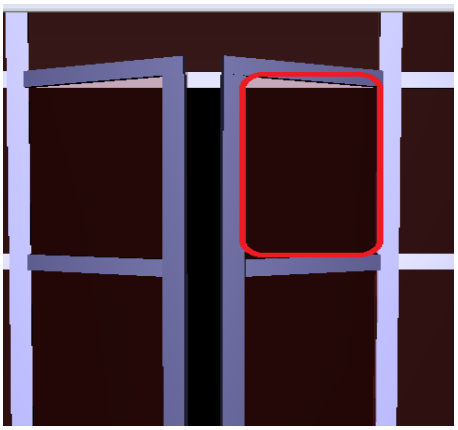

Figura 5 – Parte de vidro do objeto Porta

3.1.2 Comportamento

Nesta parte foram definidos quais objetos iriam sofrer a animação e qual a ação a ser tomada para desencadear a animação.

As partes centrais da porta foram escolhidas para serem animadas, de modo que estas façam um giro sobre o eixo Y, realizando o mesmo tipo de movimento de uma porta real.

Para desencadear a animação foram associados nós do tipo TouchSensor a todos os objetos que formam a porta, de modo que quando um usuário *clicar* sobre qualquer parte da porta a animação seja iniciada. Uma animação 3D é considerada uma modificação gráfica sobre um objeto em um intervalo de tempo. A modificação em cenas apresenta o comportamento de uma rota conectada a saída da informação de um nó e que passa um valor e estimula a entrada dessa informação em outro nó.

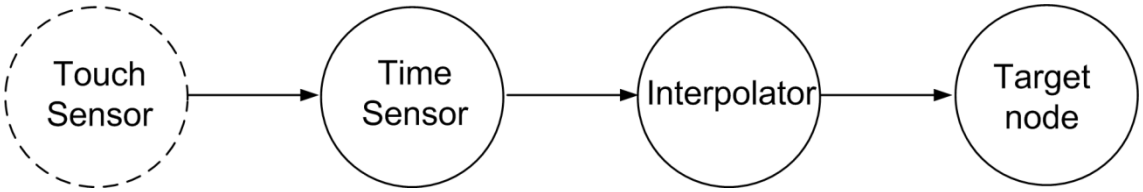

Figura 6 - Animação / Comportamento em uma rota

O *TimeSensor* controla quando a animação flui. Já o *Interpolator* produz a informação correta para uma mudança de valor (“value_changed”) de saída, por exemplo: *PositionInterpolator* produz eventos do tipo SFVec3f “value_changed”. Finalizando, o *Target node* é o objetivo de interesse para completar a animação.

```
<TimeSensor DEF='ClockNumber1' cycleInterval='2'/>
<ROUTE fromField='touchTime' fromNode='Click' toField='startTime' toNode='ClockNumber1'/>
<OrientationInterpolator DEF='DoorOpener' key='0 0.5 1' keyValue='0 1 0 0 0 1 0 4.7124 0 1 0 4'/>
<ROUTE fromField='fraction_changed' fromNode='ClockNumber1' toField='set_fraction' toNode='DoorOpener'/>
<ROUTE fromField='value_changed' fromNode='DoorOpener' toField='rotation' toNode='DoorTransform'/>
```

Figura 7 - Trecho de código / Interpolador e Rota

3.2 INTERRUPTOR DE LUZES DA SALA DE COMANDO

A animação dos interruptores da sala de comando foi desenvolvida de modo que o usuário pudesse interagir com a iluminação do ambiente.

3.2.1 Modelagem dos objetos

Primeiramente, foram modelados os objetos que formam o interruptor, compostos basicamente de duas caixas, de tamanhos e cores diferentes, que quando posicionados formam um botão.

Figura 8 – Interruptor de luzes da sala de comando

3.2.2 Comportamento

O interruptor possui um comportamento simples, resumido basicamente a um nó *TouchSensor* associado a figura geométrica de cor verde, como pôde ser observado na figura 8.

Ao *clicar* no *TouchSensor*, a forma não é alterada, porém dispara a modificação no campo *on* do nó *PointLight.* Este nó foi definido no teto da sala de comando e funciona como uma fonte de luz que irradia em todas as direções. O campo *on* determina quando a luz está habilitada, configurando *on*=”true” ou *on*=”false”.

```
<Transform translation='10 -15 5'>
  <Transform DEF='dad_lamp' translation='0 8 -10'>
   <PointLight DEF='lamp' location="25.0 15.0 20.0" ambientIntensity='0.000' intensity='1' radius='100.000' attenuation='1 0 0' color='1 1 1' on='true'/>
  </Transform>
```

Figura 9 - Trecho de código / PointLight sala de comando

3.3 RELÓGIO DE PAREDE

O relógio digital foi desenvolvido com o objetivo de deixar o ambiente mais fiel ao ambiente real da sala de comando. Já que na sala da subestação CGD, existe um relógio posicionado no pilar central.

3.3.1 Modelagem dos objetos

Os objetos que compõem o relógio são basicamente uma caixa retangular de fundo preto, e um uma *string* sobreposta sobre a caixa.

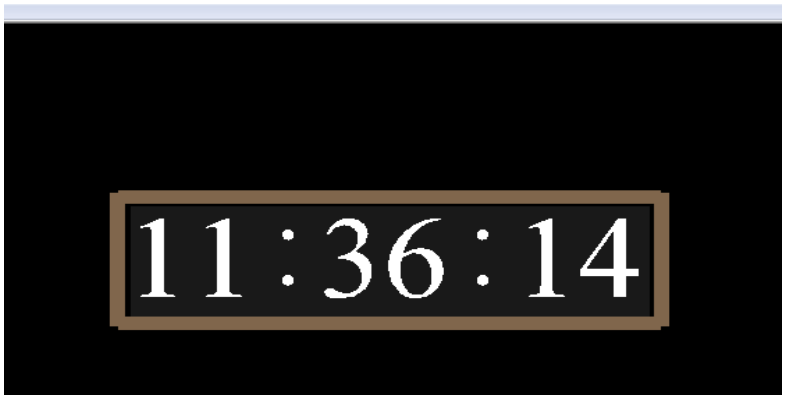

Figura 10 – Relógio digital de parede

No trecho de código abaixo, pode ser vista a definição do nó *Text* com a *string* inicial “00” , definida no campo especificado.

```
<Transform translation="-1 0 0">
 <Shape>
  <Appearance DEF="White">
      <Material diffuseColor="1 1 1"/>
  </Appearance>
  <Text DEF='HourText' string='00'>
   <FontStyle justify="END"/>
  </Text>
 </Shape>
</Transform>
```

Figura 11 - Trecho de código / String do relógio

3.3.2 Comportamento

Para dar funcionalidade ao relógio, foi necessário o uso de métodos em Java Script que permitissem a captura de hora, minutos e segundos, do sistema (gethours(), getminutes() e get_seconds(), respectivamente).

Como citado, a string exibida inicialmente no relógio é: “00:00:00”. Porém, à medida que o script é implementado, os valores são atualizados.

No trecho de código abaixo, é mostrada a utilização do método gethours(). Para visualizar o código completo, consulte o ANEXO A no final deste documento.

```
<Script DEF="HourScript">
      <field name="set_hour" type="SFTime" accessType="inputOnly"/>
      <field name="string_changed" type="MFString" accessType="outputOnly"/>
<![CDATA[javascript:

// set hour
function set_hour(time, eventTime) {
   var dat = new Date();
   var hour = dat.getHours();

   // for digital
   string_changed[0] = hour;
```

Figura 12 - Trecho de código / Método gethours()

3.4 MOSTRADORES DE CORRENTE, TENSÃO E POTÊNCIA

No mundo virtual do simulador as animações dos mostradores dos painéis eram feitas através da sobreposição de imagens. O funcionamento ocorria da seguinte forma: o usuário aciona uma chave/disjuntor abrindo ou fechando e no painel é feita a substituição da imagem da chave acionada e dos mostradores relacionados, de acordo com o tipo de ação realizada.

Este tipo de funcionamento acarretava em um transtorno já que era necessário ir à subestação CGD e tirar fotos dos mostradores nas mais diversas situações. Então, a

fim de deixar o ambiente dos painéis mais realista a solução encontrada foi construir representações para mostradores analógicos e digitais e, criar um script SAI.

Na nova versão, como a aplicação se baseia num *script* SAI, uma aplicação Java será executada enquanto interage com a aplicação do simulador escrita em X3D. De fato, a aplicação em Java deve obter referências das chaves e chamar métodos que alterem a orientação dos ponteiros e os valores visualizados nos mostradores de 7 segmentos. Além disso, foram desenvolvidas algumas classes adicionais em Java para permitir:

- Parametrização dos valores reais de tensão, corrente e potência para a rotação (radianos) dos ponteiros;
- Trabalhar com números negativos na referência mínima dos mostradores;
- Alterar os segmentos do mostrador digital.

3.4.1 Modelagem dos objetos

Os mostradores foram criados individualmente e são formados basicamente pelo objeto mostrador e pelo o objeto ponteiro. O objeto mostrador é formado por uma caixa "rasa" com a textura do mostrador real, sem ponteiros sobre a sua superfície.

O objeto ponteiro é definido no trecho de código ilustrado a seguir, com:

- O nó *TimeSensor* para controlar quando a animação flui;
- A posição do ponteiro, nos eixos x, y e z;
- O centro de rotação do ponteiro;
- A aparência do ponteiro;
- A forma geométrica do ponteiro;
- O nó *OrientationInterpolator* que torna possível a modificação da orientação do objeto dado um determinado evento ou rota.

```
<TimeSensor DEF='TIME_SENSOR' cycleInterval='5.0'></TimeSensor>
<Transform DEF='TG' translation="-5.5 5.8 37.65" center="-0.15 0 0" >
<Shape>
<Appearance>
<Material diffuseColor='1.0 0 0'/>
</Appearance>
<Box size="0.3 0.02 0.02"/>
</Shape>
</Transform>

<OrientationInterpolator DEF='PI' key='0.0 ,0.25 ,0.5 ,0.75 ,1.0'  keyValue='0 0 1 0 ,0 0 1 0.37 ,0 0 1
0.67 ,0 0 1 0.77 ,0 0 1 0.87'/>
```

Figura 13 – Definição do objeto ponteiro

Já o objeto mostrador é definido no trecho de código abaixo, com:

- A posição do mostrador, nos eixos x, y e z;
- A aparência do mostrador;
- A forma geométrica do mostrador.

```
<Transform DEF='Mostrador_Corrente' translation="-5.55 5.95 37.7">
   <Shape>
     <Appearance>
          <Material diffuseColor='0.8 0.8 0.8'></Material>
          <ImageTexture  url='"./medidorA.jpg"'/>
     </Appearance>
     <Box size="0.6 0.6 0.02"/>
   </Shape>
</Transform>
```

Figura 14 – Definição do objeto mostrador

Na definição da aparência do mostrador, a imagem real é passada como textura através do nó *ImageTexture*. Na figura abaixo, são ilustrados os dois objetos citados que compõem o mostrador como um todo.

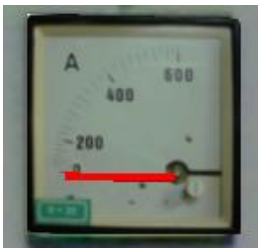

Figura 15 – Mostrador analógico completo

No caso do mostrador digital, foram criados os segmentos e estes forma posicionados de modo a formar a estrutura de 7 segmentos.

```
<Transform  translation="-4.15 2.6 37.65">
      <Shape >
          <Appearance >
              <Material  DEF="seg_11"  diffuseColor="0.2 0.2 0.2"/>
          </Appearance>
          <Box size="0.15 0.02 0.02"/>
      </Shape>
</Transform>
```

Figura 16 - Trecho de código que define um segmento do mostrador digital

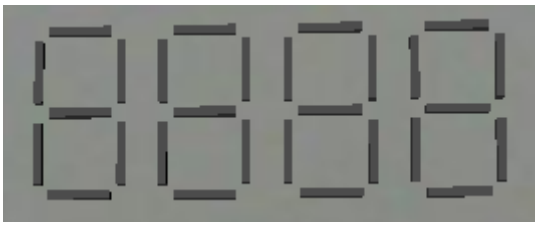

Figura 17 – Mostradores de 7 segmentos dispostos lado a lado

3.4.2 Comportamento dos mostradores analógicos

No caso dos mostradores analógicos, o comportamento consiste na rotação do ponteiro sobre o eixo especificado, em resposta a uma ação (fechamento/abertura) sobre uma chave no painel. Já os mostradores digitais devem mudar a cor dos segmentos, em função do valor configurado e da ação realizada pelo operador sobre uma chave no painel a qual está associada, no mundo virtual, a um *TouchSensor.*

4. INTEGRAÇÃO DOS OBJETOS AO SIMULADOR

Para integrar os objetos porta, interruptor e relógio ao simulador, foi utilizado o nó *Inline.* O *Inline* permite carregar nós de uma cena X3D externa, fazendo com que os objetos criados em arquivos ".x3d" individuais sejam integrados à descrição geral do simulador.

No trecho de código a seguir, é ilustrada a estrutura criada para carregar uma cena X3D externa, através do nó *Inline*.

```
<Transform  translation="75.0 2.7 49.82" rotation="0.0 1.0 0.0  1.57">
            <Inline  url='"onofflamp.x3d"'/>
</Transform>
<Transform  translation="47.75 0.375 60.0" rotation="0.0 1.0 0.0  0.0">
            <Inline  url='"porta2.x3d"'/>
</Transform>
<Transform  translation="47.75 0.375 -11.15" rotation="0.0 1.0 0.0  0.0">
            <Inline  url='"portamenor.x3d"'/>
</Transform>
<Transform   translation="31.3  4.8  20"  rotation="0.0  1.0  0.0  -1.57"
scale="0.7 0.8 0.3">
            <Inline  url='"relogio_digital.x3d"'/>
</Transform>
```

Figura 18 - Trecho de código / Inline

Abaixo são apresentadas imagens com os objetos inseridos na descrição do ambiente físico simulado (sala de operação).

Figura 19 - Porta principal abrindo

Figura 20 - Porta principal fechada

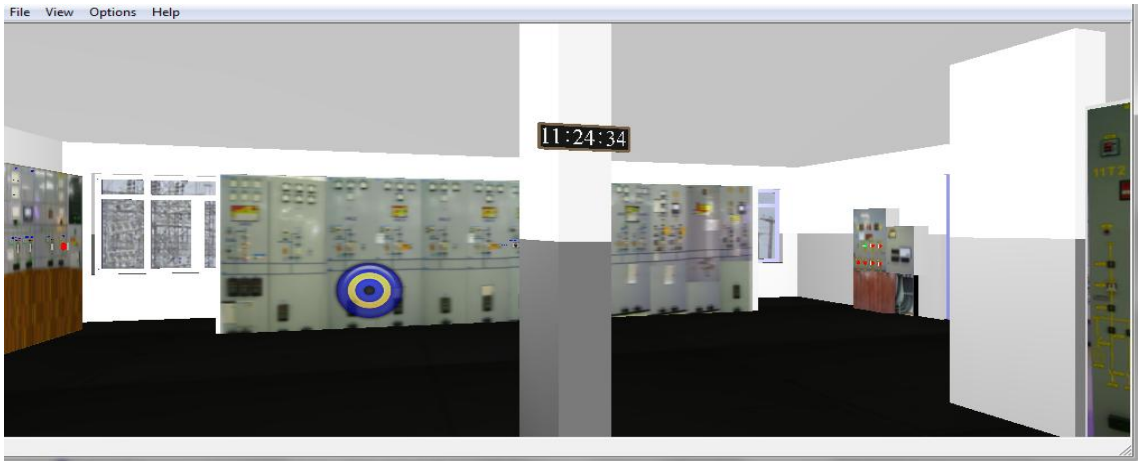

Figura 21 - Relógio digital

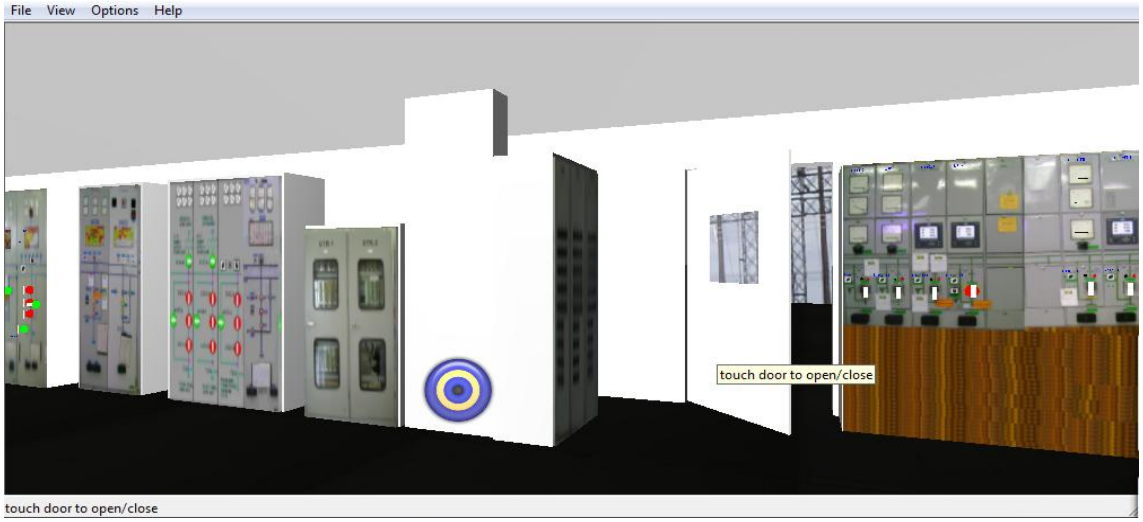

Figura 22 - Porta secundária

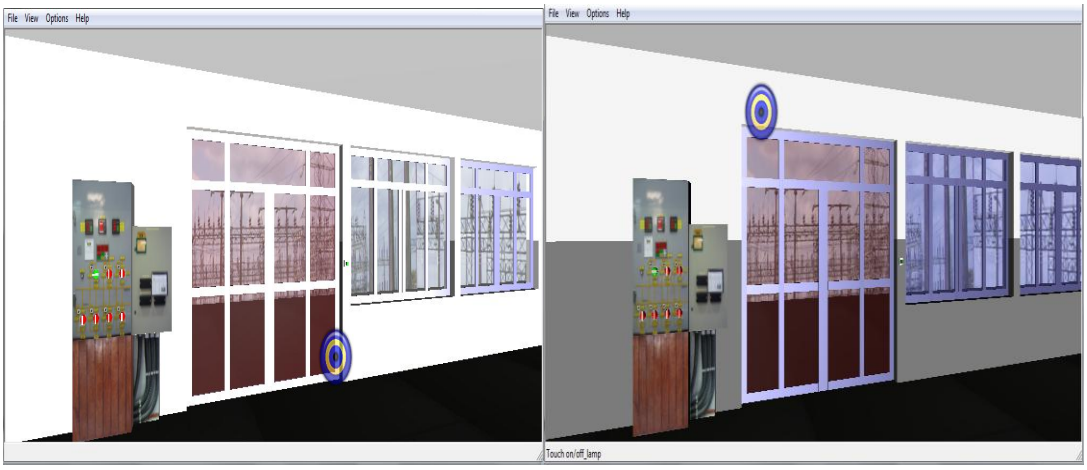

Figura 23 – Ambiente: iluminado e apagado

Para os mostradores analógicos e de 7 segmentos, seus respectivos códigos foram inseridos na descrição principal do simulador.

4.1 Mostrador analógico

Na estrutura atual, a aplicação “Browser.java”, chamada através do SAI, carrega a descrição do simulador. Ao *clicar* sobre uma chave, o nome do painel e da chave correspondente são enviados para a rede de painéis do CPNTools. Em seguida, as informações são retornadas do campo e as alterações são feitas na cena.

Através da classe criada “Parametrizador.java” foi possível parametrizar os valores das grandezas reais recebidas (tensão, corrente ou potência) em rotação dos ponteiros.

Como pode ser visto no trecho de código abaixo, nós e campos são capturados, parametrizados e retornados à cena. A classe “Parametrizador.java” completa e comentada se encontra no ANEXO B.

```
      if(touchAtivado.equals("Chave_32J5_6_Sensor")){
            motorDoPonteiro("Chave_32J5_6_Sensor", "TIME_SENSOR",
            "PI","TG", valoresParametrizados);

      //Funcao de movimentacao dos ponteiros
      public void motorDoPonteiro(String nomeDoTouch, String
      nomeDoTime, String positionInterpolator, String
      nomeDoPonteiro, float[] valoresParametrizados) {
            float[] keyvalues = new float[20];
```

```
        X3DNode TCHS = mainScene.getNamedNode (nomeDoTouch);
        X3DNode TS = mainScene.getNamedNode(nomeDoTime);
        X3DNode PI =
mainScene.getNamedNode(positionInterpolator);

        ((OrientationInterpolator) PI).getKeyValue(keyvalues);

      //Configura os novos valores para o ponteiro.
        if (!(estadoDoPonteiro)) {
            keyvalues[3] = 0;
            keyvalues[7] = valoresParametrizados[0];
            keyvalues[11] = valoresParametrizados[1];
            keyvalues[15] = valoresParametrizados[2];
            keyvalues[19] = valoresParametrizados[3];
        } else {
             keyvalues[3] = valoresParametrizados[3];
             keyvalues[7] = valoresParametrizados[2];
             keyvalues[11] = valoresParametrizados[1];
             keyvalues[15] = valoresParametrizados[0];
             keyvalues[19] = 0;
        }
        ((OrientationInterpolator) PI).setKeyValue(keyvalues);

        X3DNode TG = mainScene.getNamedNode(nomeDoPonteiro);
        mainScene.addRoute(TCHS,"touchTime",TS,"startTime");

mainScene.addRoute(TS,"fraction_changed",PI,"set_fraction");
        mainScene.addRoute(PI,"value_changed",TG,"set_rotation");
```

Figura 24 - Trecho de código / MotordePonteiro

Os valores “0” e “200” indicam o número mínimo e máximo das grandezas reais lidas (limites do mostrador), respectivamente. Portanto, os valores podem ser facilmente modificados e replicados para quaisquer tipos de mostradores. Lembrando que valores negativos também são aceitos. Veja o trecho de código abaixo:

```
    private static Parametrizador parametrizador = new
Parametrizador(((float)1.57),0, 200);
```

Figura 25 - Trecho de código / Valores mínimos e máximos suportados pelo mostrador

O método MotordePonteiro também pode ser replicado para os demais painéis. Para isso, é necessário chamar o método passando os parâmetros dos mostradores desejados: nomedoTouch, nomedoTime, positionInterpolator, nomedoPonteiro e, o array de valores parametrizados.

```
      motorDoPonteiro("Chave_32J5_6_Sensor", "TIME_SENSOR",
"PI","TG", valoresParametrizados);
      motorDoPonteiro("Chave_32J6_6_Sensor", "TIME_SENSOR2",
"PI2","TG2", valoresParametrizados2);
      motorDoPonteiro("Chave_32J6_7_Sensor", "TIME_SENSOR3",
"PI2","TG3", valoresParametrizados3);
```

Figura 26 - Trecho de código / Chamando método MotordePoteiro

Abaixo pode ser visto o funcionamento do mostrador digital em um painel da sala de comando:

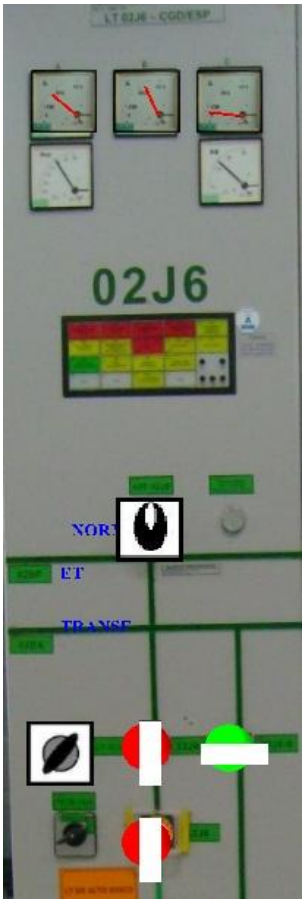

Figura 27 - Painel 12J6 com mostrador analógico

A inserção dos objetos ao simulador destacou a importância em observar o posicionamento dos painéis com relação ao eixo. Na definição da cena X3D, os respectivos campos *rotation* dos objetos (chaves e ponteiros) serão diferentes à medida que estes sejam inseridos em painéis que se encontrem perpendiculares na sala de comando.

4.2 Mostrador de 7 segmentos

Para animar o mostrador de 7 segmentos, foram criadas as seguintes classes: “ConverteNumeroparaDisplay.java” (responsável pelo modificação dos segmentos do mostrador para o valor escolhido), “Mostrador.java” (responsável pela criação do mostrador) e “MotordeDisplay.java” (responsável pela atualização de um valor para o display).

Figura 28 - Display 7 segmentos mostrando um valor

No trecho de código abaixo, é criado um mostrador digital dentro do “Browser.java”. O parâmetro “seg_” indica a parte inicial do nome do objeto na cena X3D. Já o parâmetro “4” indica quantos mostradores de 7 segmentos estão sendo criados para o “display1”. Assim, torna-se fácil gerar os mostradores para quaisquer painéis.

O mostrador digital também representa números negativos, lembrando que sempre o algarismo mais significativo ficará reservado para o sinal.

```
if(touchAtivado.equals("Chave_32J5_6_Sensor")){
      display1 = new Mostrador(mainScene, "seg_", 4);
```

Figura 29 - Trecho de código / Criação de um mostrador digital

Abaixo é ilustrado o funcionamento do mostrador digital em um painel da sala de comando:

Figura 30 - Conjunto de três mostradores de 7 segmentos inseridos em um painel

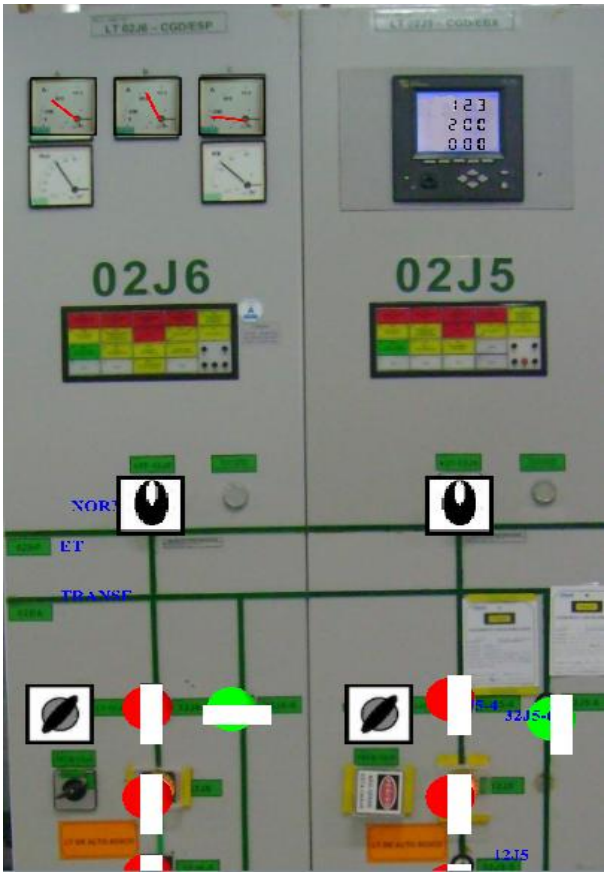

Figura 31 - Painel 12J5 com mostrador digital

Os códigos completos e comentados das classes citadas se encontram no ANEXO B.

5. CONCLUSÕES

O trabalho de conclusão de curso possibilitou adquirir conhecimento sobre o processo de construção de objetos de realidade virtual, tridimensional, em X3D, além da fundamentação teórica necessária para a criação de scripts SAI para modelos de realidade virtual. Além disso, tendo estudado a linguagem X3D, foi possível compreender o funcionamento de uma animação.

Durante o trabalho foram criados objetos cujo código foi tratado como módulos individuais em relação à descrição do simulador de modo a facilitar sua inclusão em outros cenários de simulação, a partir do reaproveitamento de código.

A utilização de classes em Java permitiu a parametrização dos valores das grandezas reais (tensão, corrente e potência) representadas nos instrumentos digitais e analógicos, a exemplo dos valores de rotação dos ponteiros nos medidores analógicos. Além disso, tornou simples a representação de valores nos mostradores digitais de 7 segmentos, (tanto positivos como negativos).

Apesar dos aplicativos Java funcionarem corretamente, durante o projeto surgiram problemas com a falta de suporte a alguns scripts no Xj3D, escolhido como a ferramenta de visualização no projeto.

Com os resultados deste projeto foi possível substituir a estratégia de utilização de imagens sobrepostas para sua animação em função das ações dos usuários sobre o mundo virtual.

Como trabalho futuro sugere-se melhorar a estrutura dos scripts, de forma a corrigir o problema do não funcionamento de JavaScript no Xj3D, facilitando a expansão do mundo virtual para acomodar a representação do ambiente externo à sala de comando. Sugere-se que seja construída uma biblioteca contendo a representação e comportamento dos objetos representados no mundo virtual do ambiente simulado.

6. BIBLIOGRAFIA

[1] PASQUALOTTI, A., *Introdução a Realidade Virtual.* Acessado em 19 de Mar. 2010. Disponível em: http://usuarios.upf.br/~pasqualotti/ccc053/intr_rv/

[2] Web 3D Consortium, *Open Standards for Real-Time 3D Communication.* Acessado em 10 de Fev. 2010. Disponível em: http://www.web3d.org/x3d/

[3] NETTO, A. V. S., *Proposta de uma arquitetura para representação em Realidade Virtual de ambientes de automação e supervisão industrial*, Dezembro de 2009.

[4] Hoss A., *Tutorial rápido de configuração Xj3D + Java*. Acessado em 10 de Fev. de 2010. Disponível em: www.docstoc.com/docs/26200913/Tutorial-rápido-X3D--Java

Brutzman, D.; Daly, L.; *X3D: Extensible 3D Graphics for Web Authors*, Ed. Morgan Kaufmann, April 2007, 468 pages.

Castilho, E. P. J. Trabalho de conclusão de curso de engenharia elétrica. Titulo: *Integração de Mundos Virtuais relativos ao Ambiente de Operação de uma Subestação Elétrica*, desenvolvido na UFCG, em Dezembro de 2007.

Aquino, M. S., *VEPersonal - Uma Infra-estrutura para Gerenciamento de Componentes de Ambientes Virtuais Adaptativos*, Tese de doutorado UFPE, 2007.

[X3D-Edit] Disponível em: https://savage.nps.edu/X3D-Edit/. Acessado em Novembro, 2009.

[Octaga, Player] Disponível em: http://www.octaga.com. Acessado em Novembro,2009.

[Xj3D] Disponível em: http://www.web3d.org/x3d/xj3d/. Acessado em Novembro, 2009.

[Studio Player, Vivaty] Disponível em: http://mediamachines.com/downloads.php. Acessado em Novembro, 2009.

[Studio, Vivaty] Disponível em: http://mediamachines.com/downloads.php. Acessado em Novembro, 2009.

[Pinecoast Software] Disponível em: http:// *www.pinecoast.com/*

ANEXO A - Códigos X3D

PORTA PRINCIPAL

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"
"http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D profile='Immersive' version='3.1' xmlns:xsd='http://www.w3.org/2001/XMLSchema-
instance'
xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d3.1.xsd>

<head>
   <meta content='Portap.x3d' name='title'/>
 </head>

 <Scene>

<Viewpoint description='TimeSensor chaining example' position='0 1 22'/>

<WorldInfo title='Porta'/>

// Definição dos objetos que formam a porta

<Transform translation='5.96368 -.0061 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance'>
   <Material DEF='WinGlass' containerField='material' ambientIntensity='0.200'
shininess='0.200' transparency='0.700' diffuseColor='.5 .1 .1' specularColor='.3 .3 .3'/>
  </Appearance>
  <Box containerField='geometry' size='4 14.65 .05'/>
 </Shape>
</Transform>

<Transform translation='-7.75 0 0'>
 <Shape containerField='children'>
  <Appearance DEF='Aluminum' containerField='appearance'>
   <Material DEF='material0' containerField='material' ambientIntensity='0.200'
shininess='0.100' diffuseColor='.3 .3 .5' specularColor='.7 .7 .8'/>
  </Appearance>
  <Box containerField='geometry' size='.5 14.65 .15'/>
 </Shape>
</Transform>

<Transform translation='7.75 0 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='.5 14.65 .15'/>
 </Shape>
```

```
</Transform>

<Transform translation='-4.25 0 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='.5 14.65 .15'/>
 </Shape>
</Transform>

<Transform translation='4.25 0 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='.5 14.65 .15'/>
 </Shape>
</Transform>

<Transform translation='-6 -7.075 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='4 .5 .15'/>
 </Shape>
</Transform>

<Transform translation='0 7.075 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='16 .5 .15'/>
 </Shape>
</Transform>

<Transform translation='-6 4.425 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='4 .5 .15'/>
 </Shape>
</Transform>

<Transform translation='-6 -1.325 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='3 .5 .15'/>
 </Shape>
</Transform>

<Transform translation='6 -1.325 0'>
 <Shape containerField='children'>
  <Appearance containerField='appearance' USE='Aluminum'/>
  <Box containerField='geometry' size='3 .5 .15'/>
 </Shape>
</Transform>
```

```
<Transform DEF='dad_Al_cen_sup' translation='0 4.42333 0'>
 <Shape DEF='Al_esquerda_sup'>
  <Appearance>
   <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5' specularColor='.7
.7 .8'/>
  </Appearance>
  <Box containerField='geometry' size='8 .5 .15'/>
 </Shape>
</Transform>

  <Transform translation='6 4.40752 0'>
 <Shape containerField='children'>
  <Appearance>
   <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5' specularColor='.7
.7 .8'/>
  </Appearance>
  <Box containerField='geometry' size='4 .5 .15'/>
 </Shape>
</Transform>

<Transform translation='-.00029 5.74515 0'>
 <Shape>
  <Appearance>
   <Material USE='WinGlass'/>
  </Appearance>
  <Box size='8 3 .05'/>
 </Shape>
</Transform>

<Transform translation='-6.05465 -.00698 0'>
 <Shape>
  <Appearance>
   <Material USE='WinGlass'/>
  </Appearance>
  <Box size='4 14.65 .05'/>
 </Shape>
</Transform>

<Transform translation='6 -7.075 0'>
 <Shape>
  <Appearance>
   <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5' specularColor='.7
.7 .8'/>
  </Appearance>
  <Box size='4 .5 .15'/>
 </Shape>
</Transform>
```

// Objetos da porta que são animados

```
  <Transform DEF='DoorTransform' translation='-2.255 -1.325 0' center='-2.0 0 0'>
    <TouchSensor DEF='Click' description='click to operate door'/>
    <Shape>
      <Appearance USE='Aluminum'/>
      <Box size='3.49 .5 .15'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform2' translation='-.25618 -1.57669 0' center='-4.0 0 0'>
    <Shape>
      <Appearance USE='Aluminum'/>
      <Box size='.5 11.5 .15'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform3' translation='-2.03285 4.42333 0' center='-2.22 0 0'>
    <Shape>
      <Appearance>
        <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5'
specularColor='.7 .7 .8'/>
      </Appearance>
    <Box size='4 .5 .15'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform4' translation='-2 -1.47594 0' center='-2 -1.47594 0'>
    <Shape DEF='Glass_esquerda' containerField='children'>
      <Appearance>
        <Material USE='WinGlass'/>
      </Appearance>
    <Box size='4 11.65 .05'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform5' translation='-2.255 -7.075 0' center='-2 -7.075 0'>
    <Shape DEF='Al_esquerda_inf' containerField='children'>
      <Appearance>
        <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5'
specularColor='.7 .7 .8'/>
      </Appearance>
    <Box size='3.49 .5 .15'/>
    </Shape>
  </Transform>
```

```
  <Transform DEF='DoorTransform6' translation='2 -1.31686 0' center='2 0 0'>
    <Shape DEF='Glass_direita' containerField='children'>
      <Appearance>
        <Material USE='WinGlass'/>
      </Appearance>
    <Box size='4 11.65 .05'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform7' translation='2.10 -7.075 0' center='2 0 0'>
    <Shape DEF='Al_direita_inf' containerField='children'>
      <Appearance>
        <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5'
specularColor='.7 .7 .8'/>
      </Appearance>
    <Box size='4.1 .5 .15'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform8' translation='2.00674 4.42551 0.1' center='2.260 0 0'>
    <Shape DEF='Al_direita_sup' containerField='children'>
      <Appearance>
        <Material ambientIntensity='0.200' shininess='0.100' diffuseColor='.3 .3 .5'
specularColor='.7 .7 .8'/>
      </Appearance>
    <Box size='4 .5 .15'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform9' translation='2.10 -1.325 0' center='2.0 0 0'>

    <Shape DEF='Al_direita_mid'>
      <Appearance USE='Aluminum'/>
    <Box size='4.1 .5 .15'/>
    </Shape>
  </Transform>


  <Transform DEF='DoorTransform10' translation='.24641 -1.5714 0.1' center='4.0 0 0'>
    <Shape containerField='children'>
      <Appearance USE='Aluminum'/>
    <Box containerField='geometry' size='.5 11.5 .15'/>
    </Shape>
  </Transform>
```

// Animação da porta

```
<TimeSensor DEF='ClockNumber1' cycleInterval='2'/>
  <ROUTE fromField='touchTime' fromNode='Click' toField='startTime' toNode='ClockNumber1'/>
  <OrientationInterpolator DEF='DoorOpener' key='0 0.5 1' keyValue='0 1 0 0 0 1 0 4.7124 0 1 0 4'/>
  <OrientationInterpolator DEF='DoorOpener2' key='0 0.5 1' keyValue='0 1 0 0 0 1 0 -4.7124 0 1 0 -4'/>
  <ROUTE fromField='fraction_changed' fromNode='ClockNumber1' toField='set_fraction' toNode='DoorOpener'/>
  <ROUTE fromField='fraction_changed' fromNode='ClockNumber1' toField='set_fraction' toNode='DoorOpener2'/>


  <ROUTE fromField='value_changed' fromNode='DoorOpener' toField='rotation' toNode='DoorTransform'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener' toField='rotation' toNode='DoorTransform2'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener' toField='rotation' toNode='DoorTransform3'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener' toField='rotation' toNode='DoorTransform4'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener' toField='rotation' toNode='DoorTransform5'/>

  <ROUTE fromField='value_changed' fromNode='DoorOpener2' toField='rotation' toNode='DoorTransform6'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener2' toField='rotation' toNode='DoorTransform7'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener2' toField='rotation' toNode='DoorTransform8'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener2' toField='rotation' toNode='DoorTransform9'/>
  <ROUTE fromField='value_changed' fromNode='DoorOpener2' toField='rotation' toNode='DoorTransform10'/>

  <!-- ===== door is now open ===== -->
  <BooleanFilter DEF='DoorOpenComplete'/>
  <ROUTE fromField='isActive' fromNode='ClockNumber1' toField='set_boolean' toNode='DoorOpenComplete'/>


  <!-- ===== shut the door ===== -->
  <!--<TimeSensor DEF='ClockNumber3' cycleInterval='2' enabled='false'/>
  <ROUTE fromField='touchTime' fromNode='Click' toField='startTime' toNode='ClockNumber3'/>-->
  <!-- enable ClockNumber3 once door is fully open -->
```

```
  <!--<ROUTE fromField='inputFalse' fromNode='DoorOpenComplete' toField='enabled' toNode='ClockNumber3'/>

  <ROUTE fromField='isActive' fromNode='ClockNumber3' toField='enabled' toNode='ClockNumber3'/>
  <ScalarInterpolator DEF='ClockReversal' key='0 1' keyValue='1 0'/>
  <ROUTE fromField='fraction_changed' fromNode='ClockNumber3' toField='set_fraction' toNode='ClockReversal'/>
  <ROUTE fromField='value_changed' fromNode='ClockReversal' toField='set_fraction' toNode='DoorOpener'/>-->

 </Scene>
</X3D>
```

PORTA MENOR

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile='Immersive' version='3.0' xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance' xsd:noNamespaceSchemaLocation=' http://www.web3d.org/specifications/x3d-3.0.xsd '>
 <head>
  <meta name="Porta menor" content="portamenor.x3d"/>
 </head>

<Scene>
```

// Definição dos objetos que formam a porta

```
<Transform  DEF='part_door1' translation="0.0 2.325 0.0" rotation="0.0 1.0 0.0  0.0" center='-4 2.325 0'>
    <Shape DEF='Al1'>
      <Appearance >
        <Material DEF="WinGlass" transparency="0.7" diffuseColor="0.5 0.5 1.0" specularColor="0.3 0.3 0.3"/>
      </Appearance>
      <Box  size="4.0 4.0 0.05"/>
    </Shape>
</Transform>


<!-- %% -->

  <Transform  DEF='part_door2' translation="0.0 -3.5 0.0" rotation="0.0 1.0 0.0  0.0" center="-4.0 -3.5 0.0">
    <TouchSensor DEF='Click' description='click to operate door'/>
    <Shape DEF='Al2'>
```

```
        <Appearance  DEF="Madeira" >
          <Material  diffuseColor="0.8 0.8 0.8"/>
        </Appearance>
        <Box  size="4.0 7.65 0.15"/>
    </Shape>
   </Transform>


  <!-- %%% -->


 <Transform DEF='part_door3' translation="0.0 5.825 0.0" rotation="0.0 1.0 0.0  0.0" center="-
4.0 5.825 0.0">
    <Shape DEF='Al3'>
       <Appearance USE="Madeira"/>
       <Box  size="4.0 3.0 0.15"/>
    </Shape>
 </Transform>


<!-- %%%% -->


<Transform DEF='part_door4' translation="-3.0 0.0 0.0" rotation="0.0 1.0 0.0  0.0" center="-
1.0 0.0 0.0">
    <Shape DEF='Al4'>
       <Appearance USE="Madeira"/>
       <Box  size="2.0 14.65 0.15"/>
    </Shape>
</Transform>


<!-- %%%%% -->


<Transform DEF='part_door5' translation="3.0 0.0 0.0" rotation="0.0 1.0 0.0  0.0" center="-7.0
0.0 0.0">
    <Shape DEF='Al5'>
       <Appearance USE="Madeira"/>
       <Box  size="2.0 14.65 0.15"/>
    </Shape>
</Transform>
```

// Animação da porta

```
<TimeSensor DEF='Clock' cycleInterval='2'/>
<ROUTE fromField='touchTime' fromNode='Click' toField='startTime' toNode='Clock'/>
```

```
<OrientationInterpolator DEF='OpenDoor' key='0 0.5 1' keyValue='0 1 0 0 0 1 0 4.7124 0 1 0 4'/>

  <ROUTE fromField='fraction_changed' fromNode='Clock' toField='set_fraction' toNode='OpenDoor'/>

  <ROUTE fromField='value_changed' fromNode='OpenDoor' toField='rotation' toNode='part_door1'/>
  <ROUTE fromField='value_changed' fromNode='OpenDoor' toField='rotation' toNode='part_door2'/>
  <ROUTE fromField='value_changed' fromNode='OpenDoor' toField='rotation' toNode='part_door3'/>
  <ROUTE fromField='value_changed' fromNode='OpenDoor' toField='rotation' toNode='part_door4'/>
  <ROUTE fromField='value_changed' fromNode='OpenDoor' toField='rotation' toNode='part_door5'/>


// Porta aberta
  <BooleanFilter DEF='DoorOpenComplete'/>
  <ROUTE fromField='isActive' fromNode='Clock' toField='set_boolean' toNode='DoorOpenComplete'/>

 </Scene>
</X3D>
```

RELÓGIO DIGITAL

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"
"http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D version="3.1" profile="Immersive">

<head>
  <meta name="Relógio Digital" content="Clock"/>
 </head>

<Scene>

 <Group>
 <Viewpoint description="Fit View" position="0 0 5" />
 <TimeSensor DEF="Clock" loop="true"/>
```

// Definição dos objetos que formam o relógio

```
    <Transform translation='-0.1 0.3 -0.1'>
        <Shape>
          <Appearance>
            <Material diffuseColor='0.1 0.1 0.1'/>
          </Appearance>
          <Box size='3.6 1 0.1'/>
        </Shape>
    </Transform>

    <Transform translation='-0.1 0.75 0'>
        <Shape>
          <Appearance>
            <Material diffuseColor='0.5 0.4 0.3'/>
          </Appearance>
          <Box size='3.6 0.1 0.1'/>
        </Shape>
    </Transform>

    <Transform translation='-0.1 -0.2 0'>
        <Shape>
          <Appearance>
            <Material diffuseColor='0.5 0.4 0.3'/>
          </Appearance>
          <Box size='3.6 0.1 0.1'/>
        </Shape>
    </Transform>

    <Transform translation='-1.9 0.28 0'>
        <Shape>
          <Appearance>
```

```
            <Material diffuseColor='0.5 0.4 0.3'/>
           </Appearance>
           <Box size='0.1 1 0.1'/>
          </Shape>
      </Transform>

      <Transform translation='1.7 0.28 0'>
          <Shape>
           <Appearance>
            <Material diffuseColor='0.5 0.4 0.3'/>
           </Appearance>
           <Box size='0.1 1 0.1'/>
          </Shape>
      </Transform>

<!-- %%%%% -->

      <Transform translation="-1 0 0">
       <Shape>
        <Appearance DEF="White">
            <Material diffuseColor="1 1 1"/>
        </Appearance>
        <Text DEF='HourText' string='00'>
         <FontStyle justify="END"/>
        </Text>
       </Shape>
      </Transform>


      <Transform translation="-0.9 0.1 0">
       <Shape>
        <Appearance USE="White" />
        <Text string=':'>
         <FontStyle justify="BEGIN"/>
        </Text>
       </Shape>
      </Transform>


      <Transform translation="0.3 0 0">
       <Shape>
        <Appearance USE="White" />
        <Text DEF='MinuteText' string='00'>
          <FontStyle justify="END"/>
        </Text>
       </Shape>
      </Transform>


      <Transform translation="0.4 0.1 0">
```

```
      <Shape>
       <Appearance USE="White"/>
        <Text string=':'>
          <FontStyle justify="BEGIN"/>
        </Text>
       </Shape>
     </Transform>


     <Transform translation="1.6 0 0">
       <Shape>
        <Appearance USE="White"/>
         <Text DEF='SecondText' string='00'>
           <FontStyle justify="END"/>
         </Text>
       </Shape>
     </Transform>
```

// Java script para configuração de horas, minutos e segundos

```
<Script DEF="HourScript">
      <field name="set_hour" type="SFTime" accessType="inputOnly"/>
      <field name="string_changed" type="MFString" accessType="outputOnly"/>
<![CDATA[javascript:

// set hour
function set_hour(time, eventTime) {
   var dat = new Date();
   var hour = dat.getHours();

   // for digital
   string_changed[0] = hour;

}

// set minute
function set_minute(time, eventTime) {
   var dat = new Date();
   var min = dat.getMinutes();

   // for digital
   if (min < 10) string_changed[0] = "0" + min;
   else         string_changed[0] = min;

}

// set second
function set_second(time, eventTime) {
   var dat = new Date();
   var sec = dat.getSeconds();
```

```
   // for digital
   if (sec < 10) string_changed[0] = "0" + sec;
   else         string_changed[0] = sec;

}

]]>
</Script>


<Script DEF="MinuteScript">
      <field name="set_minute" type="SFTime" accessType="inputOnly"/>
      <field name="string_changed" type="MFString" accessType="outputOnly"/>

<![CDATA[javascript:

// set hour
function set_hour(time, eventTime) {
   var dat = new Date();
   var hour = dat.getHours();

   // for digital
   string_changed[0] = hour;

}

// set minute
function set_minute(time, eventTime) {
   var dat = new Date();
   var min = dat.getMinutes();

   // for digital
   if (min < 10) string_changed[0] = "0" + min;
   else         string_changed[0] = min;

}

// set second
function set_second(time, eventTime) {
   var dat = new Date();
   var sec = dat.getSeconds();

   // for digital
   if (sec < 10) string_changed[0] = "0" + sec;
   else         string_changed[0] = sec;

}

]]>
```

```
</Script>

<Script DEF="SecondScript">
      <field name="set_second" type="SFTime" accessType="inputOnly"/>
      <field name="string_changed" type="MFString" accessType="outputOnly"/>

<![CDATA[javascript:

// set hour
function set_hour(time, eventTime) {
   var dat = new Date();
   var hour = dat.getHours();

   // for digital
   string_changed[0] = hour;

}

// set minute
function set_minute(time, eventTime) {
   var dat = new Date();
   var min = dat.getMinutes();

   // for digital
   if (min < 10) string_changed[0] = "0" + min;
   else         string_changed[0] = min;

}

// set second
function set_second(time, eventTime) {
   var dat = new Date();
   var sec = dat.getSeconds();

   // for digital
   if (sec < 10) string_changed[0] = "0" + sec;
   else         string_changed[0] = sec;

}

]]>
</Script>
</Group>


<ROUTE fromNode="Clock" fromField="time" toNode="SecondScript" toField="set_second"/>
<ROUTE fromNode="Clock" fromField="time" toNode="MinuteScript" toField="set_minute"/>
<ROUTE fromNode="Clock" fromField="time" toNode="HourScript" toField="set_hour"/>
<ROUTE     fromNode="SecondScript"     fromField="string_changed"     toNode="SecondText"
toField="set_string"/>
```

```
<ROUTE fromNode="MinuteScript" fromField="string_changed" toNode="MinuteText" toField="set_string"/>
<ROUTE fromNode="HourScript" fromField="string_changed" toNode="HourText" toField="set_string"/>

 </Scene>
</X3D>
```

ILUMINAÇÃO SALA DE COMANDO

// Definição do PointLight

```
 <Transform translation='10 -15 5'>
  <Transform DEF='dad_lamp' translation='0 8 -10'>
   <PointLight DEF='lamp' location="25.0 15.0 20.0" ambientIntensity='0.000' intensity='1' radius='100.000' attenuation='1 0 0' color='1 1 1' on='true'/>
  </Transform>

  <Shape>
   <Appearance>
    <Material DEF='BulbColor' ambientIntensity='0.200' shininess='0.200' diffuseColor='0 0 0' emissiveColor='1 1 1'/>
   </Appearance>
   <Sphere radius='1.000'/>
  </Shape>
 </Transform>
```

// Interruptor das luzes

```
    <Transform translation="39.5 0.375 59.0" rotation="0.0 1.0 0.0 -3.14">
     <Shape>
      <Appearance >
        <Material shininess="0.0" diffuseColor="1.0 1.0 1.0"/>
      </Appearance>
      <Box size="0.4 0.4 0.2"/>
    </Shape>
    <TouchSensor DEF='onofflamp'/>
    </Transform>

    <Transform translation="39.5 0.375 59.05">
     <Shape>
       <Appearance>
         <Material shininess="0.0" diffuseColor="0.1 0.5 0.1"/>
       </Appearance>
```

```
      <Box  size="0.2 0.2 0.2"/>
  </Shape>
</Transform>
```

MOSTRADOR ANALÓGICO

// Definição do ponteiro

```
<TimeSensor DEF='TIME_SENSOR' cycleInterval='5.0'></TimeSensor>

<Transform DEF='TG' translation="-5.5 5.8 37.65" center="-0.15 0 0" >
      <Shape>
        <Appearance>
          <Material diffuseColor='1.0 0 0'/>
        </Appearance>
        <Box size="0.3 0.02 0.02"/>
      </Shape>
</Transform>
    <OrientationInterpolator DEF='PI' key='0.0 ,0.25 ,0.5 ,0.75 ,1.0'
    keyValue='0 0 1 0 ,0 0 1 0.37 ,0 0 1 0.67 ,0 0 1 0.77 ,0 0 1 0.87'/>
```

// Definição do mostrador

```
<Transform DEF='Mostrador_Corrente' translation="-5.55 5.95 37.7">
   <Shape>
     <Appearance>
          <Material diffuseColor='0.8 0.8 0.8'></Material>
          <ImageTexture  url='"./medidorA.jpg"'/>
     </Appearance>
     <Box size="0.6 0.6 0.02"/>
   </Shape>
</Transform>
```

MOSTRADOR DIGITAL DE 7 SEGMENTOS

// Definição dos segmentos

```
<Transform  translation="-4.15 2.6 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_11"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.15 0.02 0.02"/>
    </Shape>
</Transform>

<Transform  translation="-4.25 2.7 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_12"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.02 0.153 0.02"/>
    </Shape>
</Transform>

<Transform  translation="-4.25 2.9 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_13"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.02 0.15 0.02"/>
    </Shape>
</Transform>

<Transform  translation="-4.15 3 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_14"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.15 0.02 0.02"/>
    </Shape>
</Transform>

<Transform  translation="-4.05 2.9 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_15"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.02 0.15 0.02"/>
    </Shape>
</Transform>
```

```
<Transform  translation="-4.05 2.7 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_16"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.02 0.15 0.02"/>
    </Shape>
</Transform>

<Transform  translation="-4.15 2.8 37.65">
    <Shape >
      <Appearance >
        <Material  DEF="seg_17"  diffuseColor="0.2 0.2 0.2"/>
      </Appearance>
      <Box size="0.15 0.02 0.02"/>
    </Shape>
</Transform>
```

ANEXO B - Códigos Java

PARAMETRIZADOR.JAVA

```
/**
 *
 * @author Erick Lucena
 * @version 1.0
 * Descrição: Classe que parametriza dados como tensao, corrente,
entre outros
 * para um ângulo a ser usado no ponteiro.
 */
public class Parametrizador {

        private float graus;
        private int numeroMaximoDaUnidade;
        private int numeroMinimoDaUnidade;
        private int delta;

        /**
         *
         * @param graus Número máximo de ângulo que o ponteiro
desloca(em radianos).
         * @param numeroMaximoUnidade Número máximo da unidade a ser
calculada.
         */
        public Parametrizador(float graus,int numeroMinimoDaUnidade,
int numeroMaximoDaUnidade) {
                this.graus = graus;
                this.numeroMinimoDaUnidade = numeroMinimoDaUnidade;
                this.numeroMaximoDaUnidade = numeroMaximoDaUnidade;
                delta = numeroMaximoDaUnidade - numeroMinimoDaUnidade;


        }
        /**
         *
         * @param unidade Unidade em que se dejeja encontrar o
equivalente em radianos.
         * @return Retorna o ângulo equivalente ao valor em radianos.
         */
        public float calcula(int unidade) {
                  if ( numeroMinimoDaUnidade < 0) {
                        return ((((unidade +
numeroMaximoDaUnidade)*graus) / delta));
                  } else {
                        return ((unidade*graus) / delta);
                  }
        }
        /**
         *
         * @param unidade Unidade que o ponteiro deve marcar
         * @return Retorna o array com os valores a serem seguidos
pelo ponteiro.
         */
        public float[] getArrayForInterpolation(int unidade) {
            float angulo = calcula(unidade);
```

```
            float intervalo = (angulo / 4);
            float[] arrayDeRetorno = new float[4];
            for(int k = 0; k < 4; k++) {
                  arrayDeRetorno[3-k] = (angulo - (intervalo * k));
            }
            return arrayDeRetorno;
        }
        /**
        public static void main(String[] args) {
            Parametrizador p = new Parametrizador(-1.57f,-100, 100);
            float[] array = p.getArrayForInterpolation(-50);
            for(int k = 0; k < 4; k++) {
                  System.out.print(array[k] + ", ");
            }

            }
            */
}
```

MOSTRADOR.JAVA

```
import java.awt.*;
import java.util.HashMap;
import java.util.Scanner;

import javax.swing.*;
import java.io.File;

import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.interpolation.OrientationInterpolator;

/**
 * Classe responsável pelo mostrador de 7 segmentos.
 *
 * @author Erick Lucena
 * @version 1.0
 */
public class Mostrador {
   private X3DScene mainScene;
   private int SEGMENTOS = 7;
   private int numeroDeAlgarismos;
   private SFColor[][] corDoSegmento;


   /**
    *
    * @param
    */
   public Mostrador(X3DScene mainScene, String nome, int numeroDeAlgarismos) {
         this.numeroDeAlgarismos = numeroDeAlgarismos;
         this.mainScene = mainScene;
         corDoSegmento = new SFColor[numeroDeAlgarismos][SEGMENTOS];
         X3DNode[][] palitos = new X3DNode[numeroDeAlgarismos][SEGMENTOS];
         for(int k = 0;  k < numeroDeAlgarismos; k++) {
               for(int i = 0; i < SEGMENTOS; i++) {
                     palitos[k][i] = mainScene.getNamedNode(nome + (k+1) + "" + (i+1));
               }
         }

         for(int k = 0;  k < numeroDeAlgarismos; k++) {
               for(int i = 0; i < SEGMENTOS; i++) {
                     corDoSegmento[k][i] = (SFColor) palitos[k][i].getField("diffuseColor");
               }
         }

   }

   public void mudarCorDoSegmento(int numero) {
         MotorDeDisplay motor = new MotorDeDisplay(numeroDeAlgarismos);
```

```
            try {
                motor.modificaSegmento(numero,corDoSegmento);
            } catch(Exception e) {
                System.out.println(e.getMessage());
            }
    }
}
```

MOTORDEDISPLAY.JAVA

```
import org.web3d.x3d.sai.*;

/**
 * Descrição: Classe responsável pela atualização de um número para um
display.
 * @author Erick
 * @version 1.0
 */
public class MotorDeDisplay {
	private Object[] arraysDoDisplay;
	private int tamanhoDoDisplay;
	public MotorDeDisplay(int tamanhoDoDisplay) {
		this.tamanhoDoDisplay = tamanhoDoDisplay;
	}
	/**
	 *
	 * @param numero Número a ser mostrado no display.
	 * @param segmento Array com as o valores do display de 7
segmentos.
	 * @throws Exception Caso o número não seja suportado pelo
display.
	 */
	public void modificaSegmento(int numero, SFColor[][] segmento)
throws Exception {
		boolean negativo = (numero < 0);
		if(negativo) {
			numero = -1 *numero;
		}
		String auxiliar = Integer.toString(numero);
		/**
		 * Se o número tiver mais algarismos que o display suporta,
é lançada uma exceção.
		 * Ex de números suportados para um diplay de 3 algarismos:
-99 a 99
		 * O algarismo mais significativo do display é reservado
para o sinal.
		 */
		if (((auxiliar.length() >= tamanhoDoDisplay) &&
(!(negativo))) ||
				((auxiliar.length() > tamanhoDoDisplay-1) &&
(negativo))) {
			Exception e = new Exception("Número não suportado
pelo display.");
			throw e;
		} else {
			/**
			 * Corrige números menores que o suporte do display.
			 * Exemplo, em um display de 3 algarismos, números
como 2, -2 serão mostrados
			 * como: 02, -02
			 */
			int diferenca = ((tamanhoDoDisplay) -
auxiliar.length());
			for(int i = 0; i < diferenca; i++) {
				auxiliar = "0" + auxiliar;
			}
```

```
                    if(negativo) {
                          auxiliar = "-" + auxiliar.substring(1);
                    } else {
                          auxiliar = "x" + auxiliar.substring(1);
                    }
              }
              // Atualiza a string formatada pro display.
              for(int k=0; k<auxiliar.length(); k++) {
                    arraysDoDisplay =
ConverteNumeroParaDisplay.converterInteiro(auxiliar.charAt(k));
                    for(int l=0; l < arraysDoDisplay.length; l++) {
                          float[] arrayAuxiliar = new float[3];
                          arrayAuxiliar = (float[]) arraysDoDisplay[l];
                          segmento[k][l].setValue(arrayAuxiliar);
                    }
              }
        }

}
```

CONVERTENUMEROPARADISPLAY.JAVA

```
public class ConverteNumeroParaDisplay {
      private static float[] ACESO = {0, 0, 0};
      private static float[] APAGADO = {0.8f, 0.8f, 0.8f};
      private static int[] configuracao = new int[7];

      public ConverteNumeroParaDisplay() {
      }

      private static void converterInteiroMenorQue10(char numero) {
                  switch(numero) {
                  case '0': { configuracao[0] = 1;
                                    configuracao[1] = 1;
                                    configuracao[2] = 1;
                                    configuracao[3] = 1;
                                    configuracao[4] = 1;
                                    configuracao[5] = 1;
                                    configuracao[6] = 0;
                                    break;
                        }
                  case '1': { configuracao[0] = 0;
                                    configuracao[1] = 1;
                                    configuracao[2] = 1;
                                    configuracao[3] = 0;
                                    configuracao[4] = 0;
                                    configuracao[5] = 0;
                                    configuracao[6] = 0;
                                    break;
                        }
                  case '2': { configuracao[0] = 1;
                                    configuracao[1] = 0;
                                    configuracao[2] = 1;
                                    configuracao[3] = 1;
                                    configuracao[4] = 0;
                                    configuracao[5] = 1;
                                    configuracao[6] = 1;
                                    break;
                        }
                  case '3': { configuracao[0] = 1;
                                    configuracao[1] = 1;
                                    configuracao[2] = 1;
                                    configuracao[3] = 1;
                                    configuracao[4] = 0;
                                    configuracao[5] = 0;
                                    configuracao[6] = 1;
                                    break;
                        }
                  case '4': { configuracao[0] = 0;
                                    configuracao[1] = 1;
                                    configuracao[2] = 1;
                                    configuracao[3] = 0;
                                    configuracao[4] = 1;
                                    configuracao[5] = 0;
                                    configuracao[6] = 1;
                                    break;
                        }
                  case '5': { configuracao[0] = 1;
```

```
                    configuracao[1] = 1;
                    configuracao[2] = 0;
                    configuracao[3] = 1;
                    configuracao[4] = 1;
                    configuracao[5] = 0;
                    configuracao[6] = 1;
                    break;
        }
case '6': { configuracao[0] = 1;
                    configuracao[1] = 1;
                    configuracao[2] = 0;
                    configuracao[3] = 1;
                    configuracao[4] = 1;
                    configuracao[5] = 1;
                    configuracao[6] = 1;
                    break;
          }
case '7': { configuracao[0] = 0;
                    configuracao[1] = 1;
                    configuracao[2] = 1;
                    configuracao[3] = 1;
                    configuracao[4] = 0;
                    configuracao[5] = 0;
                    configuracao[6] = 0;
                    break;
        }
case '8': { configuracao[0] = 1;
                    configuracao[1] = 1;
                    configuracao[2] = 1;
                    configuracao[3] = 1;
                    configuracao[4] = 1;
                    configuracao[5] = 1;
                    configuracao[6] = 1;
                    break;
        }
case '9': { configuracao[0] = 1;
                    configuracao[1] = 1;
                    configuracao[2] = 1;
                    configuracao[3] = 1;
                    configuracao[4] = 1;
                    configuracao[5] = 0;
                    configuracao[6] = 1;
                    break;
        }
case 'x': { configuracao[0] = 0;
                    configuracao[1] = 0;
                    configuracao[2] = 0;
                    configuracao[3] = 0;
                    configuracao[4] = 0;
                    configuracao[5] = 0;
                    configuracao[6] = 0;
break;
}
case '-': { configuracao[0] = 0;
                    configuracao[1] = 0;
                    configuracao[2] = 0;
                    configuracao[3] = 0;
                    configuracao[4] = 0;
                    configuracao[5] = 0;
```

```
                                    configuracao[6] = 1;
                        break;
                        }
                        }
                  }
            public static Object[] converterInteiro(char numero) {
                  Object[] resultado = new Object[7];
                  converterInteiroMenorQue10(numero);
                  for(int k=0; k < configuracao.length; k++) {
                        if (configuracao[k] == 1) {
                              resultado[k] = ACESO;
                        } else {
                              resultado[k] = APAGADO;
                        }
                  }
                  return resultado;
            }
}
```

BROWSER.JAVA

```
import java.awt.BorderLayout;
import java.awt.Container;
//import java.net.SocketException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Scanner;
import java.util.Timer;

import javax.swing.JComponent;
import javax.swing.JFrame;
import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.interpolation.OrientationInterpolator;
import org.web3d.x3d.sai.grouping.Transform;

//Pacote interno
//import comunicacao.EncodeDecode;
//import comunicacao.NovaThread;
import comunicacao.Servidor;
//import comunicacao.JavaCPN;

public class Browser extends JFrame implements X3DFieldEventListener {

	private static X3DScene mainScene;
	private MFString horas;
	private MFString minutos;
	private MFString segundos;
	private Mostrador display1;
	private Mostrador display2;
	private Mostrador display3;


	//Criacao de mais mostradores de 7 segmentos

	private static int novaunidade;
	private static int novaunidade2;
	private static int novaunidade3;

	private static float[] valoresParametrizados = new float[4];
	private boolean estadoDoPonteiro = false;

	//Criacao de arrays de valores parametrizados
	private static float[] valoresParametrizados2 = new float[4];
	private static Parametrizador parametrizador = new
Parametrizador(((float)1.57),0, 200);

	//Altera o valor maximo da grandeza a ser parametrizada. Ex:
Abaixo, o valor 100 indica o
	//valor maximo da grandeza seja tensao, corrente ou potencia.
	private static float[] valoresParametrizados3 = new float[4];
	private static Parametrizador parametrizador2 = new
Parametrizador(((float)1.57),-100, 100);


	public Servidor conector;
```

```
	private static ArrayList<String> nome;
	private Boolean eventTouch = false;

	float[] GREEN = {0, 1.0f, 0};
	float[] RED = {1.0f, 0, 0};
	float[] YELLOW = {1, 1, 0};
	float[] FECHADO_V = {0, 1, 0, 0};
	float[] ABERTO_V = {0, 0, 1, 1.57f};
	float[] FECHADO_H = {0, 0, 1, 1.57f};
	float[] ABERTO_H = {0, 1, 0, 0};

	float[] FECHADO_V2 = {0, 1, 0, 0};
	float[] ABERTO_V2 = {1, 0, 0, 1.57f};
	float[] FECHADO_H2 = {1, 0, 0, 1.57f};
	float[] ABERTO_H2 = {0, 1, 0, 0};

	float[] tr = new float[4];


	public Browser() {
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container contentPane = getContentPane();
//		Setup browser parameters
		HashMap requestedParameters = new HashMap();

//		Create an SAI component
		X3DComponent x3dComp =
BrowserFactory.createX3DComponent(requestedParameters);

//		Add the component to the UI
		JComponent x3dPanel =
(JComponent)x3dComp.getImplementation();
		contentPane.add(x3dPanel, BorderLayout.CENTER);

//		Get an external browser
		ExternalBrowser x3dBrowser = x3dComp.getBrowser();

		setSize(500,500);
		show();


//		Create an X3D scene by loading a file
		mainScene = x3dBrowser.createX3DFromURL(new String[] {
"Armario 12J5.x3d" });

		X3DNode horas = mainScene.getNamedNode("HourText");
		X3DNode minutos = mainScene.getNamedNode("MinuteText");
		X3DNode segundos = mainScene.getNamedNode("SecondText");

		this.horas = (MFString)horas.getField("string");
		this.minutos = (MFString)minutos.getField("string");
		this.segundos = (MFString)segundos.getField("string");
		atualizaHoras();
		X3DNode time = mainScene.getNamedNode("Clock_Sensor");
//		Replace the current world with the new one
		x3dBrowser.replaceWorld(mainScene);

//		Instancia a classe Servidor()
		conector = new Servidor();
```

```
            conector.conectar();


//          Cria a lista de sensores touch
            nome = adicionarChave();

            listen(nome);

            verificaCampo();


      }


      /**
       * Define a lista de sensores do tipo touch
       * @return nome uma lista de array com o nome dos sensores
       */
      public static ArrayList<String> adicionarChave(){
            nome = new ArrayList<String>();

            nome.add("Chave_32J5_6_Sensor");
            nome.add("Chave_32J5_5_Sensor");
            nome.add("Chave_32J5_4_Sensor");
            nome.add("Chave_32J5_7_Sensor");
            nome.add("Disjuntor_12J5_Sensor");
            nome.add("Botao_Loctel_12J5_Sensor");

            nome.add("Chave_32J6_6_Sensor");
            nome.add("Chave_32J6_5_Sensor");
            nome.add("Chave_32J6_4_Sensor");
            nome.add("Chave_32J6_7_Sensor");
            nome.add("Disjuntor_12J6_Sensor");
            nome.add("Botao_Loctel_12J6_3_Sensor");

            nome.add("onofflamp");

            nome.add("Disjuntor_14C3_Sensor");

            return nome;
      }
      /**
       * Cria o monitoramento dos sensores
       * @param lista Contem a lista com o nome dos sensores
       */
      private void atualizaHoras() {

            Date data = new Date(System.currentTimeMillis());
            int auxsegundos = data.getSeconds();
            int auxminutos = data.getMinutes();
            int auxhoras = data.getHours();

            if(auxsegundos < 10) {
                  segundos.set1Value(0, "0" + auxsegundos);
            } else {
                  segundos.set1Value(0, "" + auxsegundos);
            }
```

```
            if(auxminutos < 10) {
                  minutos.set1Value(0, "0" + auxminutos);
            } else {
                  minutos.set1Value(0, "" + auxminutos);

            }if(auxhoras < 10) {
                  horas.set1Value(0, "0" + auxhoras);
            } else {
                  horas.set1Value(0, "" + auxhoras);
            }

      }

      public void listen(ArrayList<String> lista){

            Iterator<String> it = lista.iterator();
            X3DNode generic;
            SFTime genericTF;
            String nomeSensor;

//          Varre a lista de string para saber o nome dos touchsensors
            while(it.hasNext()){
                  nomeSensor=it.next();
                  generic = mainScene.getNamedNode(nomeSensor);
//                System.out.println("e  "+generic.toString());
                  genericTF = (SFTime) generic.getField("touchTime");
                  System.out.println("Nome do sensor:  "+nomeSensor);
//                Envia qual sensor foi ativado
                  genericTF.setUserData(nomeSensor);
//                Aguardo um sensor ser ativado
                  genericTF.addX3DEventListener(this);
            }
      }

      /**
       * Função de monitoramento dos touchsensors.
       * É necessario adicionar a condição para ativar a chave.
       * @param evt Recebe o nome do touchsensor ativado
       */
      public void readableFieldChanged(X3DFieldEvent evt) {

            Object touchAtivado;

            if(eventTouch){
                  verificaCampo();
            }
//          Capta o nome do sensor que foi ativado
            touchAtivado = evt.getData();
            System.out.println("O touch ativado foi:  "+touchAtivado);


//          Adicionar aqui a função de comparação para ativar uma
chave.

            //Armario 12J5
            if(touchAtivado.equals("Chave_32J5_6_Sensor")){


                  //Chama a funcao de movimentacao dos ponteiros.
```

```
                  //Especifica a criacao dos mostradores de 7 segmentos
                  //Ex: seg_ -> nome do mostrador;
                  //    3 -> numeros de mostradores

                  display1 = new Mostrador(mainScene, "seg_", 4);
                  display2 = new Mostrador(mainScene, "seg2_", 4);
                  display3 = new Mostrador(mainScene, "seg3_", 4);

                  motorDoPonteiro("Chave_32J5_6_Sensor", "TIME_SENSOR",
"PI","TG", valoresParametrizados3);
                  motorDoPonteiro("Chave_32J5_6_Sensor",
"TIME_SENSOR2", "PI2","TG2", valoresParametrizados2);
                  motorDoPonteiro("Chave_32J5_6_Sensor",
"TIME_SENSOR3", "PI3","TG3", valoresParametrizados);

                  if (!(estadoDoPonteiro)) {
                        //Modifica display especifico com a unidade
desejada
                        motorDoDisplay(display1, novaunidade2);
                        motorDoDisplay(display2, novaunidade);
                        motorDoDisplay(display3, novaunidade3);

                        estadoDoPonteiro = true;
                  } else {
                        motorDoDisplay(display1, 0);
                        motorDoDisplay(display2, 0);
                        motorDoDisplay(display3, 0);

                        estadoDoPonteiro = false;
                  }
                  conversorBotao("32J5_6", "Chave_32J5_6",
"LT02J5,SC32J56", "vertical");
            }else
            if(touchAtivado.equals("Chave_32J5_5_Sensor")){
                  conversorBotao("32J5_5", "Chave_32J5_5",
"LT02J5,SC32J55", "vertical");
            }else
            if(touchAtivado.equals("Chave_32J5_4_Sensor")){
                  conversorBotao("32J5_4", "Chave_32J5_4",
"LT02J5,SC32J54", "vertical");
            }else
            if(touchAtivado.equals("Chave_32J5_7_Sensor")){
                  conversorBotao("32J5_7", "Chave_32J5_7",
"LT02J5,SC32J57", "horizontal");
            }else
            if(touchAtivado.equals("Disjuntor_12J5_Sensor")){
                  conversorBotao("12J5", "Disjuntor_12J5",
"LT02J5,DJ12J5", "vertical");
            }else
            if(touchAtivado.equals("Botao_Loctel_12J5_Sensor")){
                  conversorLoctel("Botao_Loctel_12J5",
"LT02J5,LocTel");
            }else

            // 12J6
            if(touchAtivado.equals("Chave_32J6_6_Sensor")){
                  conversorBotao("32J6_6", "Chave_32J6_6",
"LT02J6,SC32J66", "vertical");
            }else
```

```
            if(touchAtivado.equals("Chave_32J6_5_Sensor")){
                  conversorBotao("32J6_5", "Chave_32J6_5",
"LT02J6,SC32J65", "vertical");
            }else
            if(touchAtivado.equals("Chave_32J6_4_Sensor")){
                  conversorBotao("32J6_4", "Chave_32J6_4",
"LT02J6,SC32J64", "vertical");
            }else
            if(touchAtivado.equals("Chave_32J6_7_Sensor")){
                  conversorBotao("32J6_7", "Chave_32J6_7",
"LT02J6,SC32J67", "horizontal");
            }else
            if(touchAtivado.equals("Disjuntor_12J6_Sensor")){
                  conversorBotao("12J6", "Disjuntor_12J6",
"LT02J6,DJ12J6", "vertical");
            }else
            if(touchAtivado.equals("Botao_Loctel_12J6_Sensor")){
                  conversorLoctel("Botao_Loctel_12J6_3",
"LT02J6,LocTel");
            }else



            // Armario Extra

            if(touchAtivado.equals("Disjuntor_14C3_Sensor")){

                  motorDoPonteiro("Disjuntor_14C3_Sensor",
"TIME_SENSOR4", "PI4","TG4", valoresParametrizados2);
                  motorDoPonteiro("Disjuntor_14C3_Sensor",
"TIME_SENSOR5", "PI5","TG5", valoresParametrizados2);
                  motorDoPonteiro("Disjuntor_14C3_Sensor",
"TIME_SENSOR6", "PI5","TG6", valoresParametrizados3);

                  if (!(estadoDoPonteiro)) {


                        estadoDoPonteiro = true;
                  } else {

                        estadoDoPonteiro = false;
                  }
                  conversorBotao("14C3", "Chave_14C3", "14C3,SC14C3",
"vertical");
            }


            //Acendimento de lampadas da sala
            else
                  if(touchAtivado.equals("onofflamp")){
                        X3DLightNode lamp = (X3DLightNode)
mainScene.getNamedNode("lamp");
                        boolean aceso = true;
                        boolean apagado = false;
                        boolean estado = lamp.getOn();
                        if(estado == aceso) {
                              lamp.setOn(apagado);
                        } else {
                              lamp.setOn(aceso);
```

```
                        }

                }


        }
        /**
                * MOTOR
                * @param display
                * @param numero
                */

        public void motorDoDisplay(Mostrador display, int numero) {
                display.mudarCorDoSegmento(numero);
        }

        /**
         *
         * @param nomeDoTouch Nome do TouchSensor representado no X3D
Ex.: Chave_Transferencia_14C3_Sensor
         * @param nomeDoTime Nome do TimeSensor associado na cena X3D
Ex.: TIME_SENSOR
         * @param positionInterpolator Nome do PositionInterpolator Ex.:
PI
         * @param nomeDoPonteiro Nome do ponteiro a ser animado
         * @param valoresParametrizados Array de valores da grandeza a
serem parametrizados para rotação do ponteiro
         */
        //Funcao de movimentacao dos ponteiros
        public void motorDoPonteiro(String nomeDoTouch, String
nomeDoTime,
                                String positionInterpolator, String
nomeDoPonteiro, float[] valoresParametrizados) {
                float[] keyvalues = new float[20];
                X3DNode TCHS = mainScene.getNamedNode(nomeDoTouch);
          X3DNode TS = mainScene.getNamedNode(nomeDoTime);
          X3DNode PI = mainScene.getNamedNode(positionInterpolator);
          ((OrientationInterpolator) PI).getKeyValue(keyvalues);
          //Configura os novos valores para o ponteiro.
          if (!(estadoDoPonteiro)) {
                keyvalues[3] = 0;
                keyvalues[7] = valoresParametrizados[0];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[2];
                keyvalues[19] = valoresParametrizados[3];
          } else {
                keyvalues[3] = valoresParametrizados[3];
                keyvalues[7] = valoresParametrizados[2];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[0];
                keyvalues[19] = 0;
          }
          ((OrientationInterpolator) PI).setKeyValue(keyvalues);
          X3DNode TG = mainScene.getNamedNode(nomeDoPonteiro);
          mainScene.addRoute(TCHS,"touchTime",TS,"startTime");
          mainScene.addRoute(TS,"fraction_changed",PI,"set_fraction");
          mainScene.addRoute(PI,"value_changed",TG,"set_rotation");
        }
```

```
      /**
       * Converte o sinal de um touchsensor em uma ação.
       * @param SphereColor Nome do objeto que representa a esfera da
chave no X3D Ex.: SphereColor = "32J5_6".
       * @param nomeChave Nome da Chave representada no X3D que sera
manipulada Ex.:nomeChave = "Chave_32J5_6".
       * @param nomeRede Nome da painel e da chave na rede CPN Ex.:
nomeRede = "LT02J5,SC32J66".
       * @param posicaoChave posicao de Fechamento da chave
(vertical/horizontal)
       */
      public void conversorBotao(String SphereColor, String nomeChave,
String nomeRede, String posicaoChave){

            float[] estadoChave = new float[4];
//          Recebe mensagem do campo
            String recebido;

            X3DNode mat =
mainScene.getNamedNode("SphereColor_"+SphereColor);
            if (mat == null) {
                  System.out.println("Couldn't find material named:
SphereColor_"+SphereColor);
                  return;
            }
            SFColor color = (SFColor) mat.getField("diffuseColor");


            X3DNode  tipoChave= mainScene.getNamedNode(nomeChave);
            if (tipoChave == null) {
                  System.out.println("Nó: "+nomeChave+" não
encontrado.");
                  return;
            }
            else{
                  SFRotation pos = (SFRotation)
tipoChave.getField("rotation");
//                Salva a posicao de rotacao do objeto e armazena em
estadoChave
                  pos.getValue(estadoChave);

                  X3DNode DJ = mainScene.getNamedNode(nomeChave);
                  ((Transform) DJ).getTranslation(tr);
                  System.out.println("Nó: "+nomeChave+" não
encontrado."+tr[1]+" "+tr[2]);

                  if(tr[1]==0.6f){
                        // float [] aberto = ABERTo;
                        System.out.println("Passei por aqui");

//                      Compara o valor atual de rotacao (estadoChave)
com valor da constante ABERTO/FECHADO

                        if(posicaoChave.equals("vertical")){
                              if(Arrays.equals(estadoChave,
ABERTO_V2)){
//                                  Muda o valor de rotacao
```

```
                                    pos.setValue(FECHADO_V2);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(YELLOW);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(RED);
                                    }
                              }else if(Arrays.equals(estadoChave,
FECHADO_V2)){
//                                  Muda o valor de rotacao
                                    pos.setValue(ABERTO_V2);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(GREEN);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(YELLOW);
                                    }
                              }

                        }if(posicaoChave.equals("horizontal")){
                              if(Arrays.equals(estadoChave,
ABERTO_H2)){
//                                  Muda o valor de rotacao
                                    pos.setValue(FECHADO_H2);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(YELLOW);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(RED);
                                    }
                              }else if(Arrays.equals(estadoChave,
FECHADO_H2)){
//                                  Muda o valor de rotacao
                                    pos.setValue(ABERTO_H2);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(GREEN);
```

```
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(YELLOW);
                                    }
                              }
                        }

                  }



                  if(tr[2]==37.5){
                        System.out.println("Passei por aqui");
//                      Compara o valor atual de rotacao (estadoChave)
com valor da constante ABERTO/FECHADO

                        if(posicaoChave.equals("vertical")){
                              if(Arrays.equals(estadoChave, ABERTO_V)){
//                                  Muda o valor de rotacao
                                    pos.setValue(FECHADO_V);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(YELLOW);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(RED);
                                    }
                              }else if(Arrays.equals(estadoChave,
FECHADO_V)){
//                                  Muda o valor de rotacao
                                    pos.setValue(ABERTO_V);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(GREEN);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(YELLOW);
                                    }
                              }

                        }if(posicaoChave.equals("horizontal")){
                              if(Arrays.equals(estadoChave, ABERTO_H)){
//                                  Muda o valor de rotacao
                                    pos.setValue(FECHADO_H);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
```

```
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(YELLOW);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(RED);
                                    }
                              }else if(Arrays.equals(estadoChave,
FECHADO_H)){
//                                  Muda o valor de rotacao
                                    pos.setValue(ABERTO_H);
//                                  Envia o nome do painel e chave para
a rede de paineis do CPNTools
                                    conector.envia(nomeRede);
                                    recebido = conector.recebe();

      if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                          color.setValue(GREEN);
                                    }
                                    else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                          color.setValue(YELLOW);
                                    }
                              }
                        }

                  }









            }

            }
      /**
       * Conversor para a Chave do tipo Local/Telecomando
       * @param nomePainel Nome da chave loctel
       * @param nomeRede Nome da Ficha na rede de petri
       */
      public void conversorLoctel(String nomePainel, String nomeRede){

            float[] estadoChave = new float[4];

            X3DNode  tipoChave= mainScene.getNamedNode(nomePainel);
            if (tipoChave == null) {
                  System.out.println("Nó: "+nomePainel+" não
encontrado.");
                  return;
            }
            else{
                  SFRotation pos = (SFRotation)
tipoChave.getField("rotation");
//                Salva a posicao de rotacao do objeto e armazena em
estadoChave
                  pos.getValue(estadoChave);
```

```
//                Compara o valor atual de rotacao (estadoChave) com
valor da constante ABERTO/FECHADO
                  if(Arrays.equals(estadoChave, ABERTO_V)){
//                      Muda o valor de rotacao
                        pos.setValue(FECHADO_V);
//                      Envia o nome do painel e chave para a rede de
paineis do CPNTools
                        conector.envia(nomeRede);

                  }
//                Compara o valor atual de rotation (estadoChave) com
valor da constante ABERTO/FECHADO
                  else if (Arrays.equals(estadoChave, FECHADO_V)){
                        pos.setValue(ABERTO_V);
                        conector.envia(nomeRede);
                  }
            }

      }
      private void Reminder(int seconds) {
            verificaCampo();
    }


      /**
       * Abre canal de recebimento de fichas vindas do campo
       */
      private void verificaCampo() {
            String msgRecCampo;
            System.out.println("Esperando nova mensagem: ");
            msgRecCampo = conector.recebeCampo();
            String s[] = msgRecCampo.split(",");
//          conversorCampoX3D(String SphereColor, String nomeChave,
      String posicaoChave, String nomeRede,String msgRecCampo)
            if(s[1].equals("SC32J56")){

      conversorCampoX3D("32J5_6","Chave_32J5_6","vertical","LT02J05,SC
32J56",msgRecCampo);
            }else if(s[1].equals("SC32J55")){

      conversorCampoX3D("32J5_5","Chave_32J5_5","vertical","LT02J05,SC
32J55",msgRecCampo);
            }else if(s[1].equals("SC32J54")){

      conversorCampoX3D("32J5_4","Chave_32J5_4","vertical","LT02J05,SC
32J54",msgRecCampo);
            }else if(s[1].equals("SC32J57")){

      conversorCampoX3D("32J5_7","Chave_32J5_7","horizontal","LT02J05,
SC32J57",msgRecCampo);
            }else if(s[1].equals("DJ12J5")){

      conversorCampoX3D("12J5","Disjuntor_12J5","vertical","LT02J5,DJ1
2J5",msgRecCampo);
            }

            else if(s[1].equals("SC32J66")){
```

```
      conversorCampoX3D("32J6_6","Chave_32J6_6","vertical","LT02J06,SC
32J66",msgRecCampo);
            }else if(s[1].equals("SC32J65")){

      conversorCampoX3D("32J6_5","Chave_32J6_5","vertical","LT02J06,SC
32J65",msgRecCampo);
            }else if(s[1].equals("SC32J64")){

      conversorCampoX3D("32J6_4","Chave_32J6_4","vertical","LT02J06,SC
32J64",msgRecCampo);
            }else if(s[1].equals("SC32J67")){

      conversorCampoX3D("32J6_7","Chave_32J6_7","horizontal","LT02J06,
SC32J67",msgRecCampo);
            }else if(s[1].equals("DJ12J6")){

      conversorCampoX3D("12J6","Disjuntor_12J6","vertical","LT02J6,DJ1
2J6",msgRecCampo);
            }else if(s[1].equals("DJ12J6")){

      conversorCampoX3D("12J6","Disjuntor_12J6","vertical","LT02J6,DJ1
2J6",msgRecCampo);
            }


            eventTouch = true;
            Reminder(30);
      }
      public void conversorCampoX3D(String SphereColor, String
nomeChave,  String posicaoChave, String nomeRede,String msgRecCampo){
            float[] estadoChave = new float[4];
            X3DNode mat =
mainScene.getNamedNode("SphereColor_"+SphereColor);

            if (mat == null) {
                  System.out.println("Couldn't find material named:
SphereColor_"+SphereColor);
                  return;
            }
            SFColor color = (SFColor) mat.getField("diffuseColor");

            X3DNode  tipoChave= mainScene.getNamedNode(nomeChave);
            if (tipoChave == null) {
                  System.out.println("Nó: "+nomeChave+" não
encontrado.");
                  return;
            }else{
                  SFRotation pos = (SFRotation)
tipoChave.getField("rotation");
                  //Salva a posicao de rotacao do objeto e armazena em
estadoChave
                  pos.getValue(estadoChave);
                  if(posicaoChave.equals("vertical")){
                        if(Arrays.equals(estadoChave, ABERTO_V)){

      if(msgRecCampo.equalsIgnoreCase(nomeRede+",Aberto")){
                                    color.setValue(GREEN);
                              }else{
```

```
                                    color.setValue(YELLOW);
                              }
                        }else if(Arrays.equals(estadoChave,
FECHADO_V)){

      if(msgRecCampo.equalsIgnoreCase(nomeRede+",Fechado")){
                                    color.setValue(RED);
                              }else{
                                    color.setValue(YELLOW);
                              }
                        }
                  }else if(posicaoChave.equals("horizontal")){
                        if(Arrays.equals(estadoChave, ABERTO_H)){

      if(msgRecCampo.equalsIgnoreCase(nomeRede+",Aberto")){
                                    color.setValue(GREEN);
                              }else{
                                    color.setValue(YELLOW);
                              }
                        }else if(Arrays.equals(estadoChave,
FECHADO_H)){

      if(msgRecCampo.equalsIgnoreCase(nomeRede+",Fechado")){
                                    color.setValue(RED);
                              }else{
                                    color.setValue(YELLOW);
                              }
                        }
                  }
            }
      }
      /**
       * Main method.
       *
       * @param args None handled
       */
      public static void main(String[] args) {
            System.out.println("Digite um valor de tensão (1): ");
            Scanner sc = new Scanner(System.in);
            novaunidade = sc.nextInt();
        valoresParametrizados =
parametrizador.getArrayForInterpolation(novaunidade);

            System.out.println("Digite um valor de tensão (2): ");
            novaunidade2 = sc.nextInt();
        valoresParametrizados2 =
parametrizador.getArrayForInterpolation(novaunidade2);

        System.out.println("Digite um valor de tensão (3): ");
            novaunidade3 = sc.nextInt();
            valoresParametrizados3 =
parametrizador2.getArrayForInterpolation(novaunidade3);

        Browser Inicia = new Browser();


      }

}
```