Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

DIEGO SOARES LOPES

CONSTRUÇÃO DE UMA BIBLIOTECA DE OBJETOS PARA EDIÇÃO DE AMBIENTES VIRTUAIS DE TREINAMENTO EM SUBESTAÇÕES DE SISTEMAS ELÉTRICOS

Campina Grande, Paraíba
Dezembro de 2011

DIEGO SOARES LOPES

CONSTRUÇÃO DE UMA BIBLIOTECA DE OBJETOS PARA EDIÇÃO DE AMBIENTES VIRTUAIS DE TREINAMENTO EM SUBESTAÇÕES DE SISTEMAS ELÉTRICOS

Trabalho de conclusão de curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Informações

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

Campina Grande, Paraíba
Dezembro de 2011

DIEGO SOARES LOPES

CONSTRUÇÃO DE UMA BIBLIOTECA DE OBJETOS PARA EDIÇÃO DE AMBIENTES VIRTUAIS DE TREINAMENTO EM SUBESTAÇÕES DE SISTEMAS ELÉTRICOS

Trabalho de conclusão de curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Informações

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professora Maria de Fátima Queiroz Vieira, Ph. D.
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho à minha mãe, companheira determinada, dedicada e corajosa graças a ela tudo isso foi possível.

AGRADECIMENTOS

Primeiramente agradeço a Deus, não importando de qual religião ele seja que me deu o dom da vida e a capacidade de perseverar conseguindo assim percorrer o longo caminho em busca do título de bacharel em engenharia elétrica.

Agradeço também à minha mãe, Arlene, que se esforçou sozinha na criação de sua família, que me deu forças pra continuar a estudar, que nas madrugadas de estudo me apoiou, enfim por tudo que ela fez e representou na minha vida e nessa conquista.

Agradeço também a toda minha família, em especial ao meu irmão Thiago, que sempre esteve presente nos momentos sejam eles bons ou ruins pelos quais passei.

Agradeço a Universidade Federal de Campina Grande e a professora Maria de Fátima, por me orientarem e me moldarem para melhor servir a sociedade com os conhecimentos adquiridos ao longo do curso e na elaboração deste trabalho.

E por fim agradeço a todos os amigos, principalmente aqueles que , assim como eu, ficaram noites sem dormir, finais de semana sem diversão e muitas outras privações em busca do conhecimento para melhorar a sociedade.

"O futuro pertence àqueles que acreditam
na beleza de seus sonhos"

Eleanor Roosevelt.

RESUMO

Uma das maneiras de se treinar operadores de subestações é através da utilização de simuladores computacionais, que representam o ambiente de trabalho onde irão atuar. Este trabalho tem como finalidade a elaboração de uma biblioteca de objetos para apoiar a representação de ambientes virtuais no simulador para o treinamento de operadores, desenvolvido no laboratório de interfaces homem máquina (LIHM) da UFCG. Esta biblioteca consiste de objetos extraídos do código original do simulador, os quais foram estruturados para permitir seu reuso e a modelagem de novos objetos, mais complexos. Neste trabalho também é especificada uma ferramenta para edição de mundos virtuais e de cenários de treinamento a partir da edição apoiada pela biblioteca de objetos.

Palavras-chave: Simuladores para treinamento de operadores, Ambientes virtuais, Biblioteca de objetos, Edição de mundos virtuais.

ABSTRACT

One strategy adopted in operator training is the use of computer simulators, which represents the work environment where they perform their task. This work aims to develop a library of objects to support the representation of virtual environments in the simulator for training operators, developed in the laboratory of human-machine interfaces (LIHM) UFCG. This library consists of objects extracted from the original simulation code, which were structured to allow their reuse and the modeling of new more complex objects. This work is also specified a tool for creating virtual worlds and training scenarios, supported by the object library.

Keywords: Simulators for operator training, Virtual world, object library, virtual world edition.

LISTA DE ILUSTRAÇÕES

Figura 1. Exemplo de código xml 17
Figura 2. Exemplo de código em X3D 18
Figura 3. Esfera modelada utilizando o X3D 18
Figura 4. Exemplo de classe Java que utiliza o SAI 19
Figura 5 Exemplo de um modelo em Rede de Petri Colorida 20
Figura 6 Vista do simulador 21
Figura 7 Arquitetura atula do SimuLIHM 22
Figura 8 diagrama unifilar do simulador da STPO da COELCE 24
Figura 9 Relatório gerado pelo simulador da plena 25
Figura 10 Interface do simulador da empresa Plena 25
Figura 11 Imagens do simulador FURNAS 26
Figura 12 alarme no espaço 3D 35
Figura 13 Leds com a base preta no espaço 3D 36
Figura 14 Mostrador analogico no espaço 3D 37
Figura 15 mostradores sete segmentos no espaço 3D 38
Figura 16 Quadro sinótico no espaço 3D 41
Figura 17 Chave punho no espaço 3D 43
Figura 18 Chave gpg no espaço 3D 44
Figura 19 Ctf tipo 1 no espaço 3D 46
Figura 20 Ctf tipo 2 no espaço 3D 46
Figura 21 Ctf tipo punho no espaço 3D 48
Figura 22 Chave Loc Tel no Espaço 3D 49
Figura 23 Copo maior no espaço 3D 50
Figura 24 Copo menor no espaço 3D 50
Figura 25 String de texto no espaço 3D 52
Figura 26 Armário no espaço 3D 53
Figura 27 Monitor no espaço 3D 54
Figura 28 Teclado no espaço 3D 55
Figura 29 Cadeira no espaço 3D 55
Figura 30 Diagrama de casos de uso do usuário projetista 81
Figura 31 Diagrama de casos de uso do usuário comum 82
Figura 32 Diagrama de classes proposto 83
Figura 33 Interface da ferramenta de desenvolvimento de mundos do LIHM 85
Figura 34 Objetos sendo inseridos em um mundo 85

LISTA DE ABREVIATURAS E SIGLAS

ANEEL – Agência Nacional de Energia Elétrica

API – Interface de Programação de Aplicativos

Application Programming Interface

CHESF – Companhia Hidroelétrica do São Francisco

COELCE – Companhia Energética do Ceará

CPN – Redes de Petri Coloridas

Coulored Petri Nets

GPL – Licença Pública Geral

General Public License

HTML – Linguagem de Marcação de Hipertexto

HyperText Markup Language

LGPL – Menor Licença Pública Geral

Lesser General Public License

LIHM – Laboratório de Interface Homem Máquina

JVM – Máquina Virtual Java

Java Virtual Machine

SAI – Interface de Acesso a Cena

Scene Access Interface

SAGE – Sistema Aberto de Supervisão e Controle

STL – Biblioteca de modelos padrões

Standard Template Library

STPO – Sistema de Treinamento em Proteção e Operação de sistemas elétricos

UFC – Universidade Federal do Ceará

UFCG – Universidade Federal de Campina Grande

UnB – Universidade de Brasília

VRML – Linguagem para Modelagem de Realidade Virtual

Virtual Reality Modeling Language

XML – Linguagem extensível de marcação

Extensible Markup Language

1 SUMÁRIO

1 Introdução ... 13
1.1 Motivação ... 13
1.2 Objetivos ... 14
2 Embasamento Teórico ... 15
2.1 Modelagem de mundos virtuais ... 15
2.1.1 Modelagem Geométrica ... 15
2.1.2 Modelagem Cinemática ... 16
2.1.3 Modelagem Física ... 16
2.1.4 Comportamento do Objeto ... 16
2.2 A Linguagem X3D ... 16
2.3 A Linguagem de programação Java ... 18
2.4 Redes de Petri Coloridas ... 20
3 Simuladores ... 21
3.1 Simulador do LIHM ... 21
3.2 Outros Simuladores ... 23
3.2.1 STPO desenvolvido pela COELCE ... 23
3.2.2 Simulador desenvolvido pela plena ... 24
3.2.3 Simulador FURNAS ... 25
4 Plataformas e Ferramentas utilizadas ... 27
4.1 Editores de ambientes X3D ... 27
4.1.1 X3D Edit ... 27
4.1.2 Notepad++ ... 27
4.2 Visualizadores X3D ... 28
4.2.1 Octaga Player ... 28
4.2.2 Vivaty Player ... 28
4.2.3 XJ3D ... 28
4.3 IDE Java ... 29
4.3.1 Net Beans IDE ... 29
4.4 Motor de simulação ... 29
4.4.1 CPN Tools ... 29
5 A Biblioteca ... 30
5.1 BIBLIOTECA ... 30
5.1.1 Descrição de Objeto Genérico ... 31
- Group ... 32
- Transform ... 32
- Shape ... 32
- Appearance ... 33
- Material ... 33
- TouchSensor ... 33

5.2 Objetos – Representação visual 34
5.2.1 Objetos de Sinalização 34
• Alarme 34
• LED com base preta 35
• Mostrador Analógico 36
• Mostrador sete segmentos 38
• O Quadro Sinótico 40
5.2.2 Chaves 41
• Chave tipo punho 41
• Chave GPG (Giro, pressão giro) 43
• Chave de transferência 44
• Chave Local Telecomando 48
5.2.3 Acessórios das chaves 49
• Copos 49
5.2.4 Strings 51
• Strings de Texto 51
5.2.5 Objetos que compõem a sala de comando 52
• Armário 52
• Monitor 54
• Teclado 54
• Cadeira 55
5.3 Objetos – representação comportamental 56
• Conversorctf.java 56
• Conversoralarme.java 58
• Conversoloctel.java 61
• Conversorprotecao.java 63
• Parametrizador.java 64
• Mostrador.java 65
• Motordedisplay.java 66
• MotordePonteiro.java 68
• Conversorbotao.java 70
• Indicadorcampo.java 72
• Conversorsinótico.java 74
5.4 Exemplo de como instanciar um objeto 78
6 Ferramenta de edição de mundos Virtuais 80
6.1 Níveis de acessos à ferramenta 80
6.2 Arquitetura de classes da Ferramenta 82
6.3 Interface da Ferramenta 84
7 Conclusão 87
REFERÊNCIAS 88

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

A complexidade dos sistemas elétricos de potência vem aumentando ao longo do tempo devido ao crescimento e surgimento de zonas urbanas e dos parques industriais. Esse sistema é de fundamental importância, pois ele é responsável pelo fornecimento de energia elétrica e é necessário que o serviço oferecido não sofra interrupções. Para que isso seja possível existe uma equipe de operadores que supervisionam o sistema garantindo seu funcionamento.

Os operadores dos sistemas elétricos são capacitados através de treinamentos que consistem em aulas teóricas e práticas, onde a prática é obtida operando o sistema real. Devido às restrições de acesso ao sistema real, este treinamento é incompleto, além de dificilmente os operadores serem submetidos a todas as situações de emergência que eles possam enfrentar durante a operação do sistema. Uma alternativa para um melhor preparo desses operadores é a utilização de simuladores computacionais que possibilitam a criação de cenários encontrados nas salas de comando das subestações elétricas, com utilização de simuladores. Assim em um tempo menor pode-se treinar um número maior de operadores, com uma melhor qualidade. Desta forma, esta ferramenta se torna bastante útil nas empresas do setor elétrico, permitindo inclusive apoiar um processo de certificação de Operadores de Sistema, complementando a parte teórica e prática já adotada.

Além de fornecer um melhor treinamento, a utilização de simuladores possibilita a reciclagem de operadores que há muito tempo receberam treinamento. O simulador permite a realização de procedimentos menos freqüentes tais como contingências críticas, apagões, recomposição, etc. A realização de manobras comumente utilizadas em uma subestação é outra vantagem a ser explorada utilizando o simulador.

Existem diversos simuladores para treinamento de operadores em subestações elétricas em desenvolvimento ou concluídos tais como o Sistema para Treinamento de Proteção e Operação (STPO) desenvolvido pela Concessionária de Energia Elétrica do estado do Ceará (COELCE) em parceria com a Agência Nacional de Energia Elétrica

(ANEEL), outro simulador foi desenvolvido pela PLENA transmissora que é um grupo de empresas especializadas na área de sistemas elétricos. Há ainda o Simulador desenvolvido pela Universidade de Brasília (UnB), em parceria com a Behold Studios, ANEEL e Eletrobrás, para o treinamento de técnicos de uma subestação de energia elétrica. Este trabalho é voltado para o SimuLIHM, desenvolvido pelo LIHM da UFCG.

1.2 OBJETIVOS

Este trabalho vem contribuir com o desenvolvimento do simulador SimuLIHM do LIHM da Universidade Federal de Campina Grande (UFCG) através da criação de uma biblioteca de objetos modelados em X3D para apoiar a construção de novos mundos virtuais e de cenários de treinamento.

Um das limitações do simulador desenvolvido no LIHM é a dificuldade de construção de novos cenários no mundo virtual. Atualmente, a criação de um novo mundo virtual demanda diversas alterações no simulador, desde criações de novas classes até a implementação de novos objetos. Esta alteração demanda o conhecimento do seu funcionamento e da sua implementação. Surge a necessidade de uma ferramenta que realize essas alterações de forma mais fácil, ou seja, a partir de uma interface gráfica apoiada por uma biblioteca de objetos.

Portanto, os objetivos deste trabalho são: (a) estruturar a biblioteca de objetos do mundo virtual do ambiente de treinamento de operadores de sistemas elétricos e (b) especificar uma ferramenta de edição de mundos virtuais para apoiar a edição de cenários e contextos de treinamento no simulador SimuLIHM.

2 EMBASAMENTO TEÓRICO

O “mundo virtual” é um ambiente computacional onde podem ser realizadas interações com objetos e personagens (avatares) que simulem o comportamento de um ambiente real. Neste caso, o mundo virtual é o ambiente de trabalho do operador no qual ocorre o treinamento - uma subestação elétrica. Os cenários são a configuração destes objetos e personagens, os quais são monitorados e exibidos durante o treinamento. A seguir são apresentados os conceitos adotados neste trabalho.

2.1 MODELAGEM DE MUNDOS VIRTUAIS

Durante a modelagem de mundos virtuais diversas características relativas aos objetos que serão construídos são levadas em consideração, tais como forma, movimento, aparência e restrições. Para isto ser atendido os sistemas de realidade virtual devem levar em conta aspectos de modelagem, mapeamento e simulação (LUCENA E.M, 2010).

2.1.1 MODELAGEM GEOMÉTRICA

A modelagem geométrica é 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.

A aparência dos objetos está relacionada principalmente às características de reflexão da superfície e à sua textura. A texturização pode ser feita como se um filme plástico, com seu padrão de textura, fosse ajustado e colocado sobre o objeto, fazendo parte integrante dele. A texturização dos objetos serve para aumentar o nível de detalhe e realismo da cena, fornecendo uma visão mais apurada de profundidade e permitindo a redução substancial do número de polígonos da cena, propiciando o aumento da taxa de quadros por segundo (LUCENA E.M, 2010).

2.1.2 Modelagem Cinemática

Para ser criada a animação de um objeto a modelagem geométrica por si só não é suficiente, é necessário que exista a interação com os objetos presentes no mundo virtual, deve ser possível selecionar objetos, alterar sua posição, mudar sua escala, detectar colisões e produzir deformações na superfície. A modelagem cinemática consiste em como esses objetos serão animados em função dos outros objetos e da interação com o usuário (PASQUALOTTI. A, 2011).

2.1.3 Modelagem Física

Para a obtenção de um melhor realismo dos mundos virtuais, os objetos devem se comportar segundo as leis da física, por exemplo, objetos devem respeitar o principio da impenetrabilidade dos corpos e não penetrarem o espaço físico uns dos outros.e sim colidirem levando em consideração suas características (massas, peso, inércia, deformações) (PASQUALOTTI. A, 2011).

2.1.4 Comportamento do Objeto

Existem objetos do mundo virtual cujo comportamento independe da interação com o usuário, tais como relógios, calendários, termômetros dentre outros objetos *inteligentes*, acessando ou sendo acessados quando necessário, por sensores externos (PASQUALOTTI. A, 2011).

2.2 A Linguagem X3D

Por sua vez, a linguagem X3D representa uma arquitetura para construção e representação de gráficos computacionais em 3D e propõe um formato de codificação de arquivos. Ela é uma evolução do padrão internacional *Virtual Reality Modeling Language* (formalmente definido por ISO/IEC 14772-1:1997 mas freqüentemente denominado VRML 2 ou VRML 97).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<curriculo>
   <InformacaoPessoal>
      <DataNascimento>23-07-68</DataNascimento>
      <Nomecompleto>...</Nomecompleto>
      <Contatos>
         <Morada>
              <Rua>R.Topazio</Rua>
              <Num>111</Num>
              <Cidade>Porto</Cidade>
              <Pais>Portugal</Pais>
         </Morada>
         <Telefo9999-9999</Telefone>
         <CorreioEletronico>email@email.com</CorreioEletronico>
      </Contatos>
      <Nacionalidade>Portuguesa</Nacionalidade>
      <Sexo>M</Sexo>
   </InformacaoPessoal>
   <objetivo>Atuar na area de TI</objetivo>
   <Experiencia>
      <Cargo>Suporte técnico</Cargo>
      <Empregador>Empresa, Cidade - Estado</Empregador>
   </Experiencia>
   <Formacao>Superior Completo</Formacao>
</curriculo>
```

Figura 1. Exemplo de código xml

Para entender a linguagem X3D é necessário conhecer a linguagem *Extensible Markup Language* (XML).

O XML é uma linguagem concebida para o armazenamento e transporte de informações, diferentemente da *HyperText Markup Language* (HTML) que foi desenvolvida para possibilitar a exibição de informações na rede mundial de computadores(WEB3D.ORG, 2011). O código em XML é descrito por “tags” (etiquetas) definidas pelo programador, as quais são utilizadas para a descrição de algum objeto ou procedimento. Um exemplo de código em XML é ilustrado na Figura 1.

O X3D utiliza o formato XML para expressar a geometria e os aspectos comportamentais do VRML. O VRML é conhecido como um expressivo formato de intercâmbio 3D com suporte a muitas ferramentas e códigos base. Além de expressar diversas geometrias e comportamentos de animação, X3D permite a programação de rotinas (em ECMAScript ou Java) e a prototipagem de nós, provendo um excelente suporte para extensões de cenários gráficos com novas funcionalidades(BRUTZMAN, 2007). Um exemplo de código em X3D é ilustrado na Figura 2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"   "http://www.web3d.org/specifications/x3d-3.0.dtd">
3  <X3D profile='Immersive' version='3.0' xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
4    <Scene>
5        <Transform DEF = "Esfera" translation="0 -1 0" rotation="0 0 1 1.57">
6            <Shape>
7                <Sphere radius="1"/>
8                    <Appearance>
9                        <Material diffuseColor="1 0 0"/>
10                   </Appearance>
11               </Shape>
12       </Transform>
13   </Scene>
14 </X3D>
```

Figura 2. Exemplo de código em X3D

O resultado do trecho de código mostrado na Figura 2 é apresentado na Figura 3

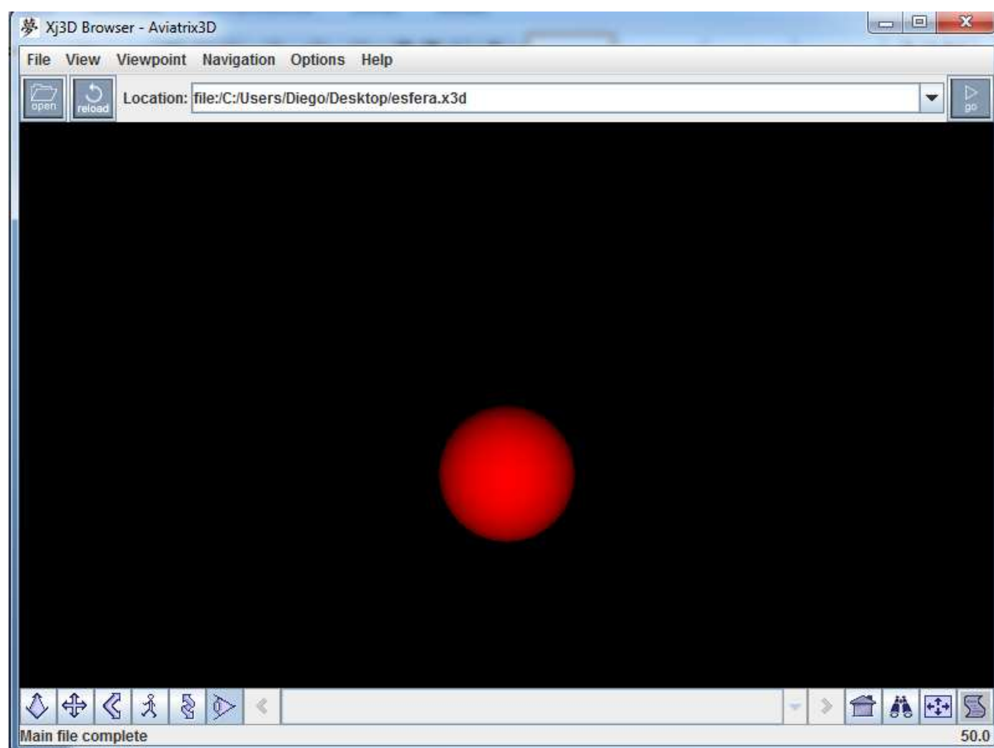


Figura 3. Esfera modelada utilizando o X3D

2.3 A LINGUAGEM DE PROGRAMAÇÃO JAVA

Java é um linguagem de programação de aplicação geral, baseada em classes, orientada a objetos a qual foi desenvolvida com a intenção de permitir que sejam desenvolvidos aplicativos "*write once, run anywhere*" (escreve uma vez, roda em qualquer lugar), com a utilização da *Java Virtual Machine* (JVM). Java é atualmente uma das linguagens de programação mais populares em uso, particularmente para aplicações cliente-servidor web (JAVA, 2011).

Utilizando a linguagem de programação Java, temos acesso a classes e métodos que permitem a criação de nós de rotina para iteração com cenas X3D. Algumas

aplicações *Application Programming Interface* (API) podem ser utilizadas para manipular uma cena através de uma aplicação externa ao navegador. A programação de scripts externos é realizada utilizando-se o X3D *Scene Access Interface* (SAI). Nesse método ocorre a conexão entre a aplicação Java e o cenário X3D, no qual a aplicação Java terá acesso aos nós do código X3D; podendo assim manipulá-los mudando sua estrutura e/ou comportamento. Um exemplo de um código de uma classe criada em Java é ilustrado na Figura 4:

```
1   package AV;
2
3   import org.web3d.x3d.sai.*;
4
5
6   /**
7    * Classe respons�vel pelo mostrador de 7 segmentos.
8    *
9    * @author Erick Lucena
10   * @version 1.0
11   */
12  public class Mostrador {
13      private X3DScene mainScene;
14      private int SEGMENTOS = 7;
15      private int numeroDeAlgarismos;
16      private SFColor[][] corDoSegmento;
17
18
19      /**
20       *
21       * @param
22       */
23      public Mostrador(X3DScene mainScene, String nome, int numeroDeAlgarismos) {
24          this.numeroDeAlgarismos = numeroDeAlgarismos;
25          this.mainScene = mainScene;
26          corDoSegmento = new SFColor[numeroDeAlgarismos][SEGMENTOS];
27          X3DNode[][] palitos = new X3DNode[numeroDeAlgarismos][SEGMENTOS];
28          for(int k = 0;  k < numeroDeAlgarismos; k++) {
29              for(int i = 0; i < SEGMENTOS; i++) {
30                  palitos[k][i] = mainScene.getNamedNode(nome + (k+1) + "" + (i+1));
31              }
32          }
33
```

Figura 4. Exemplo de classe Java que utiliza o SAI

Para a utilização do SAI é necessário importar o pacote *org.web3d.x3d.sai* que permite a obtenção de uma referência entre a aplicação Java e o cenário X3D através da função *getBrowser()*. Após obter essa referência, a aplicação Java tem acesso a qualquer nó do cenário X3D podendo assim alterar suas características.

2.4 REDES DE PETRI COLORIDAS

As Redes de Petri Coloridas (RPC) são um formalismo matemático para modelagem de sistemas (MOSSES P.D & ENGBERG U.H, 1998) Elas são uma extensão das Redes de Petri (PN) (CASSANDRAS C.G & LAFORTUNE S, 2008) com estruturas de dados encontradas em linguagens de programação de alto nível. As Redes de Petri Coloridas possuem uma representação gráfica e matemática.

Um modelo em CPN pode ser representado de forma hierárquica a partir de um conjunto de páginas contendo sub-redes interconectadas a partir de lugares e transições. A representação gráfica das Redes de Petri facilita a visualização da estrutura de um modelo construído nestas redes, como ilustrado na Figura 5.

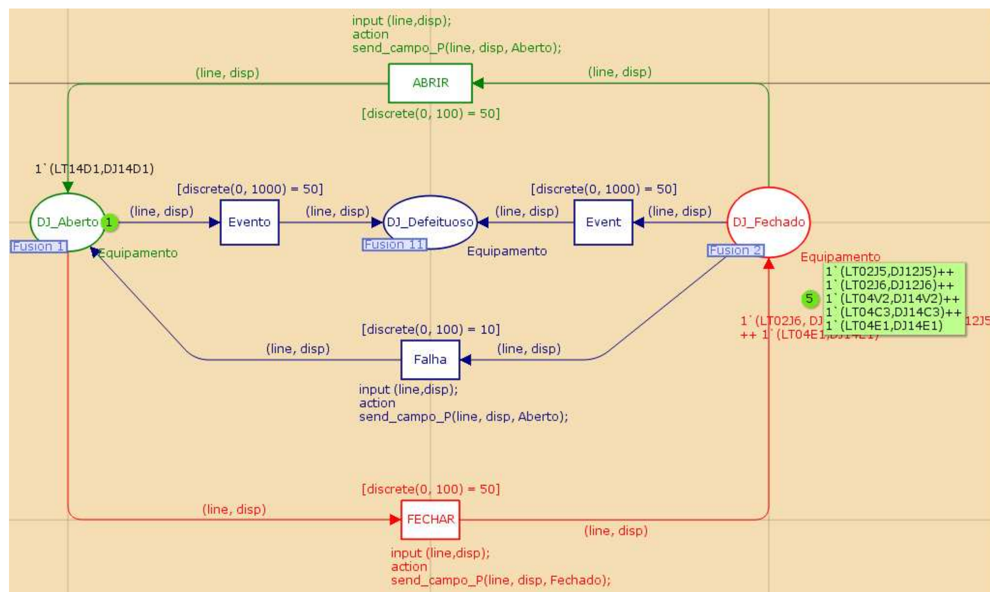


Figura 5 Exemplo de um modelo em Rede de Petri Colorida

As redes de Petri coloridas possuem uma representação matemática formal, com semântica e sintaxe bem definidas (JENSEN. K & KRISTENSEN L.M, 2009). Essa representação é de fundamental importância para definição das diferentes propriedades comportamentais as quais são passíveis de verificação. A construção e verificação das redes de Petri demandam ferramentas como CPNtools e o domínio de sua semântica e sintaxe. (CASSANDRAS C.G & LAFORTUNE S, 2008).

Atualmente o simulador utiliza basicamente dois conjuntos de Redes de Petri Coloridas, uma que modela o estado das chaves dos painéis e outro que modela os comportamentos dos equipamentos no pátio da subestação além da rede do supervisório.

3 SIMULADORES

3.1 SIMULADOR DO LIHM

O simulador SimuLIHM está em desenvolvimento no Laboratório de Interfaces Homem-Máquina no DEE da UFCG. O projeto é desenvolvido com o propósito de apoiar o treinamento de operadores em subestações elétricas. Ele contém uma interface gráfica que representa na forma de um mundo virtual o ambiente de operação de uma sub-estação elétrica possibilitando a imersão do usuário no ambiente virtual, semelhantemente a um jogo de computador, como mostrado na Figura 6.

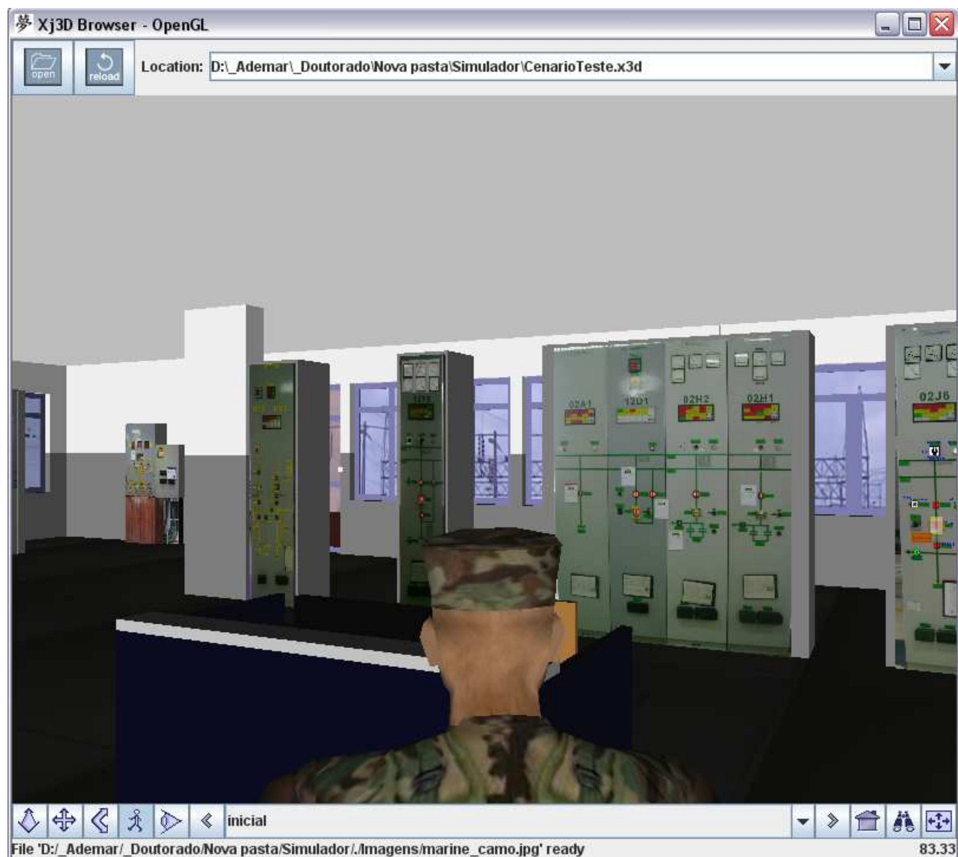


Figura 6 Vista do simulador

Imerso nesse mundo virtual o usuário tem a possibilidade de interagir com diversos botões, botoeiras, chaves, medidores, anunciadores, alarmes e ter acesso a um sistema supervisório semelhante ao utilizado em empresas do setor elétrico, permitindo

assim a realização de manobras sobre o sistema, tanto em nível de painéis quanto ao nível do supervisório.

O simulador SimuLIHM oferece a possibilidade de interação multi-usuário, onde mais de um usuário, normalmente um tutor e um treinando, utilizam o simulador de tal forma que toda ação realizada pelo tutor é vista pelo treinando e todas as atividades realizadas pelo treinando são vistas pelo tutor. Essa ferramenta torna a experiência de simulação mais semelhante àquela vivenciada no dia-a-dia do operador. Todas as ações realizadas no simulador são arquivadas em um log.

O mundo virtual é modelado utilizando o X3D e o visualizador utilizado é o Xj3D, já as animações dos objetos são realizadas utilizando o Java que envia mensagens que serão processadas pelo motor de simulação composto por Redes de Petri Coloridas através do motor de simulação do ambiente CPNTools. Este, devolve as ações a serem realizadas no mundo virtual através de mensagens para o aplicativo Java, o qual tem acesso ao mundo virtual através do SAI. A arquitetura atual do simulador é ilustrada na Figura 7.

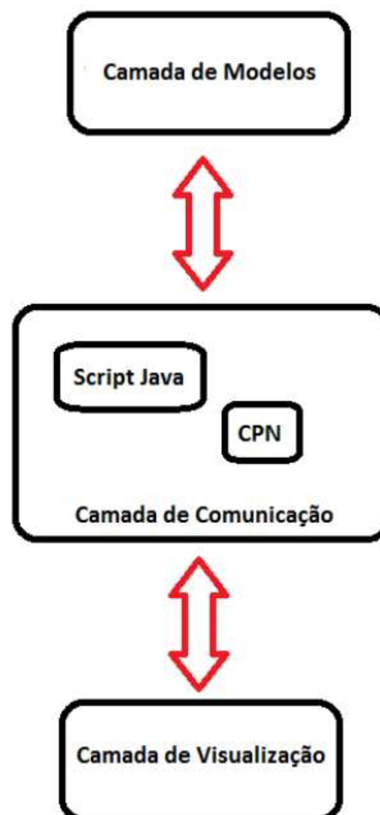


Figura 7 Arquitetura atula do SimuLIHM

A arquitetura do simulador é organizada nas seguintes camadas: A camada de visualização, a camada de comunicação e a camada de modelos. A camada de visualização é referente à modelagem física do simulador através da utilização do X3D. A camada de comunicação estabelece a interação entre os modelos formais e a representação X3D e a camada de modelos compreendem os modelos das Redes de Petri Coloridas que trocam mensagens com o mundo virtual. Segundo a proposta original deste trabalho, alterações foram 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, seriam criadas classes em Java para implementar scripts SAI para os novos objetos inseridos no mundo virtual.

3.2 OUTROS SIMULADORES

3.2.1 STPO DESENVOLVIDO PELA COELCE

O Simulador para Treinamento de Proteção e Operação de Sistemas Elétricos STPO é um ambiente composto de uma tela com um diagrama unifilar sistêmico contendo os componentes de proteção, tais como disjuntores, religadores, transformadores e relés. Este simulador foi desenvolvido pela Universidade Federal do Ceará (UFC) em parceria com a COELCE. No simulador o usuário pode simular faltas e re-configurar o sistema, atuando sobre relés e outros dispositivos. (BARROSO G.C & et al).. A tela do simulador é mostrada na Figura 8

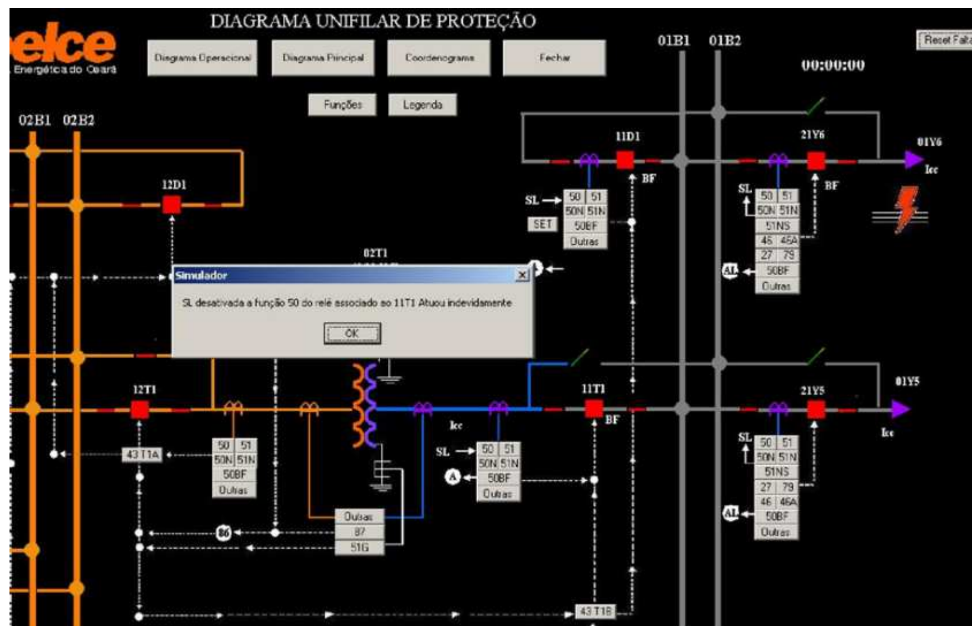


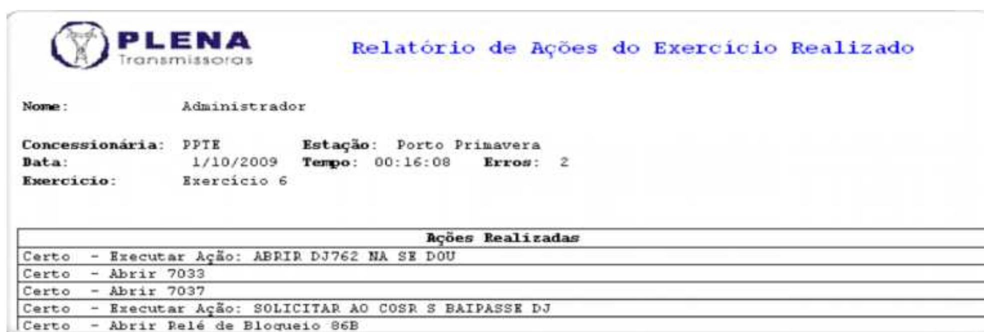
Figura 8 diagrama unifilar do simulador da STPO da COELCE

Dentre as funcionalidades do STPO estão: geração de faltas em alimentadores com efeitos visuais indicativos de curtos-circuitos e atuação de proteção 50/51 (relés de sobrecorrente de fase), 50/51N(relés de sobrecorrente de neutro), 27(relé de supervisão) dentre outras. Este simulador oferece uma quantidade de dados sobre o sistema em simulação (tensão, corrente, potência dentre outros), porém a falta da imersão em um mundo virtual torna esse simulador limitado com relação a aprendizagem e ao treinamento de operadores.

3.2.2 SIMULADOR DESENVOLVIDO PELA PLENA

Este simulador, desenvolvido pela empresa Plena, é bem mais simples que o da UFC e que o em desenvolvimento no LIHM. Consiste em um aplicativo "*stand-alone*" que executa uma série de manobras e exercícios configuráveis. Ele pode ser utilizado por um tutor, e tem associado um diagrama unifilar.(MACEDO R. R, 2010)

Este simulador tem por finalidade auxiliar no treinamento do operador de subestações na realização de suas tarefas básicas e no seu desempenho durante recomposições do sistema após situações de perturbação. O usuário, baseado em seu conhecimento e instruções de operação, executa os exercícios e roteiros de simulação desenvolvidos por um especialista neste sistema. Ao final do treinamento gera um relatório do desempenho do candidato como mostrado na Figura 9.



PLENA Transmissoras

Relatório de Ações do Exercício Realizado

Nome: Administrador

Concessionária: PPTE **Estação:** Porto Primavera
Data: 1/10/2009 **Tempo:** 00:16:08 **Erros:** 2
Exercício: Exercício 6

Ações Realizadas
Certo - Executar Ação: ABRIR DJ762 NA SE DOU
Certo - Abrir 7033
Certo - Abrir 7037
Certo - Executar Ação: SOLICITAR AO COSR S BAIPASSE DJ
Certo - Abrir Relé de Bloqueio 86B

Figura 9 Relatório gerado pelo simulador da plena

Assim como o simulador da STPO esse simulador não possibilita a imersão do treinando no ambiente de trabalho. As ações do operador são realizadas sobre um diagrama unifilar que representa o sistema elétrico, como ilustrado na Figura 10.

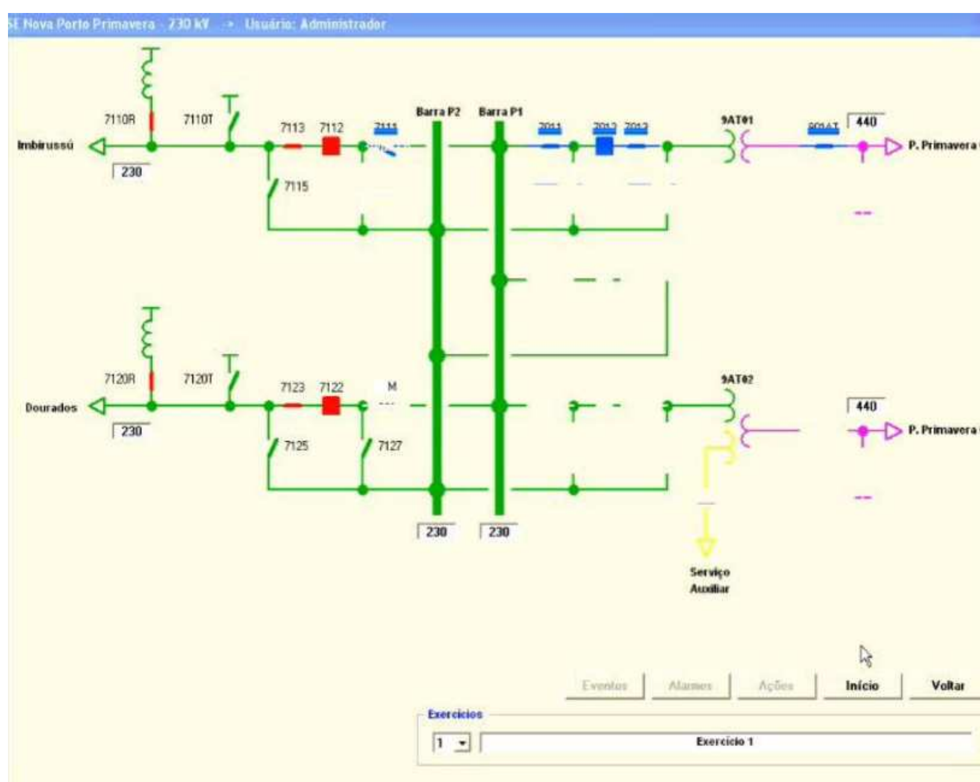


Figura 10 Interface do simulador da empresa Plena

3.2.3 SIMULADOR FURNAS

Este simulador foi desenvolvido pela UnB, em parceria com a Behold Studios, ANEEL e Eletrobras, para o treinamento de técnicos de uma subestação de energia elétrica no ano de 2010. (ALVES A.A.S et AL, 2010).

Assim como o SimuLIHM ele permite a simulação em um mundo virtual que representa uma subestação, porém diferentemente do simuLIHM tanto os equipamentos

de campo quanto os equipamentos da sala de operação estão representados, aumentando o realismo da representação e a imersão do operador em treinamenteo. No entanto, diferentemente dos equipamentos do SimuLIHM seus equipamentos não estão funcionais a partir dos painéis de controle.

Esse simulador utiliza o Sistema Aberto de Supervisão e Controle (SAGE) como supervisório e permite a interação com outros avatares presentes na subestação, além da geração de um relatório final de desempenho do usuário e das atividades realizadas. Algumas imagens deste simulador estão na Figura 11.

Figura 11 Imagens do simulador FURNAS

4 PLATAFORMAS E FERRAMENTAS UTILIZADAS

Para o desenvolvimento deste trabalho algumas ferramentas e plataformas foram utilizadas para edição e visualização de ambientes e objetos utilizando o X3D e para a animação dos mesmos desenvolvidos em Java e processados pelas Redes de Petri Colorida.

4.1 EDITORES DE AMBIENTES X3D

Os editores de ambientes X3D são softwares com modo texto que permite a criação de um código em X3D, além de facilitar a visualização do código que está sendo escrito e dependendo do visualizador até funcionalidades como indicar que um nó está incompleto dentre outros.

4.1.1 X3D EDIT

O X3D Edit é um editor gráfico de X3D que permite a edição livre de erros, autorização e validação de arquivos de cenários gráficos de X3D ou VRML. Possui ferramentas que resumem concisamente cada nó ou atributo do X3D. De maneira mais geral é um editor XML, customizado para editar cenários X3D que também pode ser utilizado para gerar outros tipos de arquivos, tais como HTML e VRM97(BRUTZMAN, 2007).

4.1.2 NOTEPAD++

O Notepad++ é um editor de código que suporta diversas linguagens. Em execução no ambiente MS Windows, a sua utilização é regida pela licença *General Public License* (GPL). Baseado em um poderoso editor de componentes Scintilla, o Notepad++ é escrito em C++ e utiliza as APIs *Standard Template Library* (STL) e Win32 API as quais garantem uma alta velocidade de execução e um pequeno tamanho do arquivo.(NOTEPAD ORGANIZATION, 2011)

O Notepad++ não possui suporte para desenvolvimento em X3D, porém suporta edição utilizando XML, logo no momento de salvar o arquivo bastava mudar a extensão do mesmo para que fosse possível a edição e criação de arquivos em X3D.

4.2 VISUALIZADORES X3D

Os Visualizadores são softwares que “montam” a descrição de um arquivo X3D, ou seja, eles modelam visualmente aquilo que está descrito em um documento de cenário X3D.

4.2.1 OCTAGA PLAYER

O Octaga Player é um visualizador que suporta os perfis de núcleo e interativo de X3D e dá aos usuários uma infinidade de efeitos visuais, como sombreamento de pixel e multi-texturização, além de ser um visualizador de alto desempenho, pode ser executado como um aplicativo independente ou como um plug-in para os navegadores da Internet, porém o pacote completo do software é pago (SOFTPEDIA, 2011).

4.2.2 VIVATY PLAYER

O Vivaty Player é um navegador, de uso gratuito, que implementa o SAI. O Vivaty Player pode ser usado como plugin ou standalone (VIVATY STUDIO PLAYER, 2011).

4.2.3 XJ3D

Xj3D é um projeto de código aberto *Lesser General Public License* (LGPL), do Grupo de Trabalho Web3D Consórcio focado na criação de um kit de ferramentas para VRML97 e X3D de conteúdo totalmente escrito em Java. Possui basicamente dois propósitos, ser uma base de código experimental para experimentar novas áreas da especificação do X3D e como uma biblioteca para desenvolvedores de aplicativos que utilizam dentro de sua própria aplicação como suporte da tecnologia X3D(Xj3D ORG, 2009).

4.3 IDE JAVA

4.3.1 NET BEANS IDE

O NetBeans IDE é um ambiente de desenvolvimento - uma ferramenta para programadores escrever, compilar, depurar e implantar programas. É escrito em Java - mas pode suportar qualquer linguagem de programação. Existem também um enorme número de módulos para aprimorar o NetBeans IDE tais como o mobility8 e o ruby. O NetBeans IDE é um produto gratuito sem restrições de como ser utilizado(NET BEANS ORGANIZATION, 2011).

4.4 MOTOR DE SIMULAÇÃO

O motor de simulação é responsável pelas animações e comportamento dos objetos em um cenário e como eles se relacionam entre si.

4.4.1 CPN TOOLS

CPN Tools é uma ferramenta de alto nível para contrução e simulação de redes de Petri. Ele suporta os tipos mais básicos de Redes de Petri tais como a temporizada e as redes de Petri coloridas. Além de possuir um simulador e uma ferramenta de análise de espaço de estados incluídos.

CPN Tools foi originalmente desenvolvido pelo Grupo de CPN em Aarhus University 2000-2010. Os Principais arquitetos por trás da ferramenta são Kurt Jensen, Soren Christensen, Lars M. Kristensen, e Michael Westergaard. A partir do outono de 2010, CPN Tools é transferido para o grupo de AIS, Eindhoven University of Technology, na Holanda.

CPN Tools é composto por dois componentes principais, um editor gráfico e um componente de simulador de "*backend*"(CPN TOOLS ORGANIZATION, 2010)

5 A BIBLIOTECA

Esta seção dará inicio a descrição da biblioteca desenvolvida para o SimuLIHM, mais informações sobre o que é a biblioteca, quais os objetos que a compõem atualmente e como é modelado cada objeto serão abordados a seguir.

5.1 BIBLIOTECA

A biblioteca de objetos do LIHM consiste no conjunto dos objetos que compõem o mundo virtual de uma subestação, descritos utilizando a linguagem X3D. Originalmente a descrição dos objetos do mundo virtual era parte integrante do código do SimuLIHM dificultando sua extensão e reuso.

Nas próximas subseções serão apresentados os códigos em X3D relativos aos aspectos visuais dos objetos representados na biblioteca e os códigos em Java que representam os aspectos comportamentais dos objetos que compõem essa biblioteca. Será apresentada uma meta descrição da estrutura dos códigos dos objetos, que permitirá compreender e identificar os elementos da descrição que deverão ser manipulados por ocasião do uso do objeto no processo de edição de cenários. Esta descrição possibilitará editar os objetos existentes e modelar e inserir novos objetos nessa biblioteca.

Objetos

A seguir são apresentados os objetos que compõem a biblioteca:

- Objetos de sinalização
    - Alarme
    - LED com base preta
    - Mostrador analógico
    - Mostrador de 7 segmentos
    - Quadro sinótico
- Chave
    - Chave tipo punho

    - Chave GPG(giro – pressão - giro)
    - Chave de transferência
    - Chave local e telecomando
    - Acessórios das chaves
        - Copos
- Strings
    - Strings de texto
- Objetos que compõem a sala de comando
    - Armário
    - Monitor
    - Teclado
    - Cadeira

E as classes que são utilizadas para modelar o comportamento dos objetos

- ConversorCTF.java
- ConversorAlarme.java
- ConversorLoctel.java
- ConversorProtecao.java
- Parametrizador.java
- Mostrador.java
- MotorDeDisplay.java
- MotorDePonteiro.java
- ConversorBotao.java
- IndicadorCampo.java
- ConversorSinotico.java

5.1.1 Descrição de Objeto Genérico

Do ponto de vista da representação visual em X3D os objetos modelados possuem campos e características que podem ser editados pelo usuário através do uso da ferramenta de edição de mundos do LIHM. Em geral os objetos são compostos dos seguintes campos editáveis:

- GROUP

O nó <*Group*> é responsável por delimitar a descrição do objeto, é um campo que pode ser editável definindo um nome para este nó que para identificação do objeto no mundo virtual. O modo de edição desse nó é feito inserindo a palavra *DEF* e o nome que será atribuído ao objeto pelo usuário entre aspas da seguinte maneira <*Group DEF="ObjetoGenerico"*>. Para representar o fim da descrição de um objeto e mesmo grupo utilizasse a tag </*Group*>.

- TRANSFORM

Semelhante o nó <*Group*> o nó <*Transform*> também pode definir um nome da seguinte maneira <*Transform DEF = "ObjetoGenericoTrans"*> , porém esse nome será utilizado pela ferramenta através do SAI para alteração dos parâmetros pertencentes ao nó tais como *center*, *translation* e *rotation*. Existem outros parâmetros relacionados ao nó *Transform* mas os citados são os mais importantes para este documento, o *translation* localiza fisicamente o objeto no mundo virtual através das coordenadas x,y e z. O *rotation* especifica a rotação do objeto com relação ao seu centro e ao sistema de coordenadas através de quatro campos x,y,z e a o ângulo de rotação em radianos. O campo *center* define o centro de rotação do objeto com relação ao sistema de coordenadas através das coordenadas x,y e z. A edição completa de um nó Transform pode ser realizada da seguinte maneira <*Transform DEF = "ObjetoGenericoTrans" translation =" 0 0 0" rotation = "0 1 0 1.57" Center = "0 0 0"* > para identificação do fim de um nó *transform* deve ser inserido no código a tag </*Transform*>, caso um dos parâmetros não estejam definidos por padrão é atribuído 0 0 0 e no caso do parâmetro *center* os seus valores são os valores referentes ao centro geométrico do objeto .

- SHAPE

O nó <*Shape*> deve esta presente no escopo de um nó <*Transform*> e é o nó onde são definidas as formas do objeto, para a biblioteca presente neste documento não é um campo editável, pois não se pode mudar nem a forma nem o tamanho dos objetos já definidos na biblioteca, porém a presença deste nó é necessária, pois se deve obedecer

a hierarquia do X3D para que os nós de níveis inferiores sejam utilizados, para identificação do fim de um nó *Shape* deve ser inserido no código a tag </*Shape*>.

- APPEARANCE

Define a aparência do objeto e deve estar no escopo de um nó <*Shape*>. Assim como o nó *Shape* não possui parâmetro editáveis, porém é de fundamental importância para que seja respeitada a hierarquia do X3D. Para identificação do fim de um nó *Appearance* deve ser inserido no código a tag </*Appearance*>.

- MATERIAL

O nó <*Material*> deve está inserido no escopo de um nó <*Appearance*> e nele são definidas as cores, o brilho, opacidade, o índice de reflexão do objeto dentre outros, para este documento é interessante o parâmetro *diffuseColor* que representa a cor do objeto que está sendo modelado através da combinação das cores vermelho, verde e azul respectivamente. Caso haja a necessidade desse objeto mudar de cor durante a execução do simulador é necessário especificar um nome para este nó, assim ele poderá ser localizado através da utilização do SAI e ser alterado durante a execução do simulador. da seguinte maneira <*Material DEF =”ObjetoGenericoMat” diffuseColor =”1 1 1”*> . Para identificação do fim de um nó *Material* deve ser inserido no código a tag </ *Material* >.

- TOUCHSENSOR

O <*TouchSensor*> é o nó que é responsável por identificar se o objeto está sendo apontado pelo cursor do *mouse* e se ele recebeu algum clique para que seu estado seja alterado, caso ele tenha recebido um clique ele muda de estado e essa mudança é capturada pelo motor de simulação do simulador e tratada. Este nó deve estar dentro do escopo de um nó <*Group*> e necessita ter um nome definido pelo usuário caso este objeto possua este nó sensor da seguinte maneira <*TouchSensor DEF = “ObjetoGenerico_Sensor”*/> a “/” indica o fim do nó sensor.

O nosso objeto genérico seria modelado como é mostrado no Código 1.

```
<Group DEF = “ObjetoGenerico”>
	<Transform DEF = “ObjetoGenericoTrans  translation ="0 0 0" rotation = "0 1 0 1.57"
	center  = "0 0 0">
		<Shape>
		<Appearance >
			<Material DEF = "ObjetoGenericoMat" diffuseColor="1.0 1.0 1.0"/>
		</Appearance>
		<Sphere  radius="0.22986"/>
		</Shape>
	</Transform>
<TouchSensor DEF = “ObjetoGenerico_Sensor />
</Group>
```

Código 1 Descrição de um objeto genérico

5.2 OBJETOS – REPRESENTAÇÃO VISUAL

Os objetos presentes na biblioteca possuem um modelo físico tridimensional descrito utilizando a linguagem X3D, nesta seção são apresentados como os objetos são modelados.

5.2.1 OBJETOS DE SINALIZAÇÃO

- ALARME

O alarme representa um dos sinais sonoro e visual que se encontra em painéis na subestação, é modelado apenas por uma esfera branca que possui um nó sensor para captura de cliques, assim pode-se atuar nesse objeto. Existe uma classe Java responsável pela animação e comportamento desses alarmes. O código em X3D que modela um alarme pode ser visto Código 2.

```
<Group>
	<Transform translation ="0 0 0" rotation = "0 1 0 0">
		<Shape>
			<Appearance >
				<Material diffuseColor="1.0 1.0 1.0"/>
			</Appearance>
		<Sphere  radius="0.22986"/>
		</Shape>
	</Transform>
<TouchSensor />
</Group>
```

Código 2 Alarme.x3d

Figura 12 alarme no espaço 3D

O resultado da descrição realizada no código acima pode ser visto na Figura 12. No Código 2 Alarme.x3d algumas características como o nome, a cor e a translação serão definidos ou alterados pelo usuário que estará inserindo o objeto no seu mundo virtual através de uma ferramenta que será especificada na seção 6.

- LED COM BASE PRETA

Este objeto é composto de um prisma da cor preta que compõe a base em que o LED se encontra e um cilindro que representa o diodo emissor de luz. Os campos referentes a cor, rotação e translação do LED será definido segundo o usuário através da ferramenta de edição de cenários do LIHM. O LED em um painel representa o estado de uma chave no que não possui sinaleiros incorporados ou uma chave que esteja no pátio. O código referente a esse objeto se encontra no Código 3.

```
<Group>
        <Transform translation="0 0 0" rotation="0.0 1.0 0.0 0">
                <Shape>
                        <Appearance >
                                <Material diffuseColor="0.0 0.0 0.0"/>
                        </Appearance>
                <Box  size="0.1 0.2 0.02"/>
                </Shape>
        </Transform>
        <Transform translation="0 0 0" rotation="0.0 1.0 0.0 1.57">
                <Shape>
                        <Appearance >
                                <Material diffuseColor="0.0 1.0 0.0"/>
                        </Appearance>
                <Cylinder  height="0.05" radius="0.05"/>
                </Shape>
        </Transform>
</Group>
```

Código 3 ledcomfundopreto.x3d

A saída da descrição mostrada no Código 3 encontra-se na Figura 13.

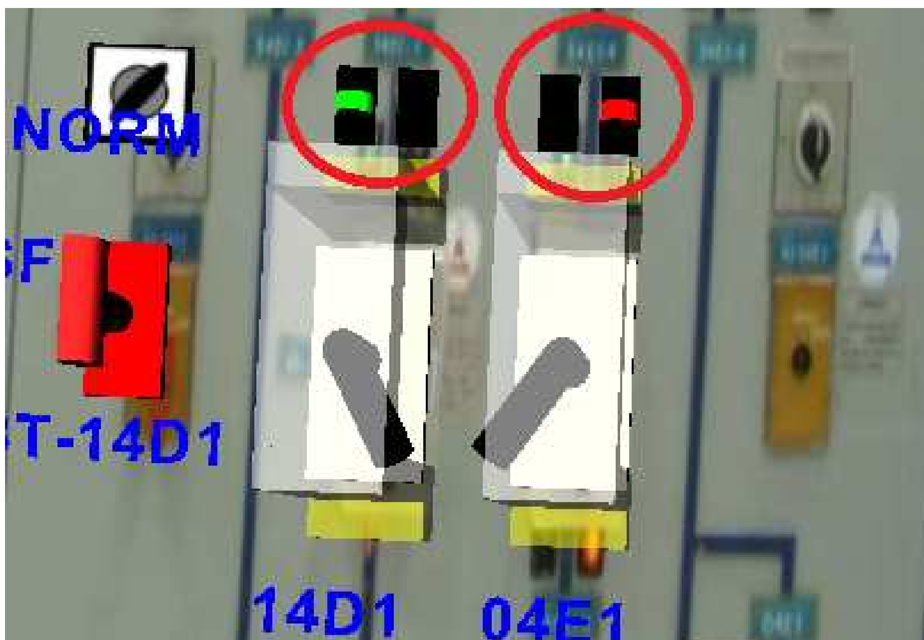


Figura 13 Leds com a base preta no espaço 3D

- MOSTRADOR ANALÓGICO

O mostrador analógico é um instrumento que exibe o valor de uma determinada grandeza utilizando uma graduação e um ponteiro. Esse objeto é modelado com um prisma maior que é coberto por uma textura que representa o medidor analógico, juntamente com outro prisma que simula o comportamento do ponteiro. Assim como outros objetos esse mostrador possui uma classe que modela seu comportamento e

realiza a animação dos ponteiros via Java chamada de parametrizador.java e constará nos anexos deste trabalho. O código referente ao mostrador analógico pode ser visualizado em Código 4.

```
<Group>
    <TimeSensor cycleInterval="5.0"></TimeSensor>
        <Transform translation="0.02 -0.135 -0.025" rotation="0.0 0.0 1.0 0.47"
        center="-0.15 0 0" >
            <Shape>
                <Appearance>
                    <Material diffuseColor="0 0 0"/>
                </Appearance>
            <Box size="0.3 0.02 0.02"/>
            </Shape>
        </Transform>
    <OrientationInterpolator 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"/>
    <Transform translation="0 0 0">
        <Shape>
            <Appearance>
                <Material diffuseColor="0.8 0.8 0.8"></Material>
                <ImageTexture  url="./Imagens/medidorA.jpg"/>
            </Appearance>
        <Box size="0.6 0.6 0.02"/>
        </Shape>
    </Transform>
</Group>
```

Código 4 mostradoranalogico.x3d

A saída do Código 4 pode ser vista na Figura 14

Figura 14 Mostrador analogico no espaço 3D

- MOSTRADOR SETE SEGMENTOS

O mostrador de sete segmentos é composto de sete pequenos prismas que estão dispostos de maneira a formar uma estrutura para representação de números decimais ou hexadecimais, esses mostradores são utilizados para exibição de grandezas nos painéis assim como os mostradores analógicos. Um mostrador de sete segmentos pode ser visualizado na Figura 15.

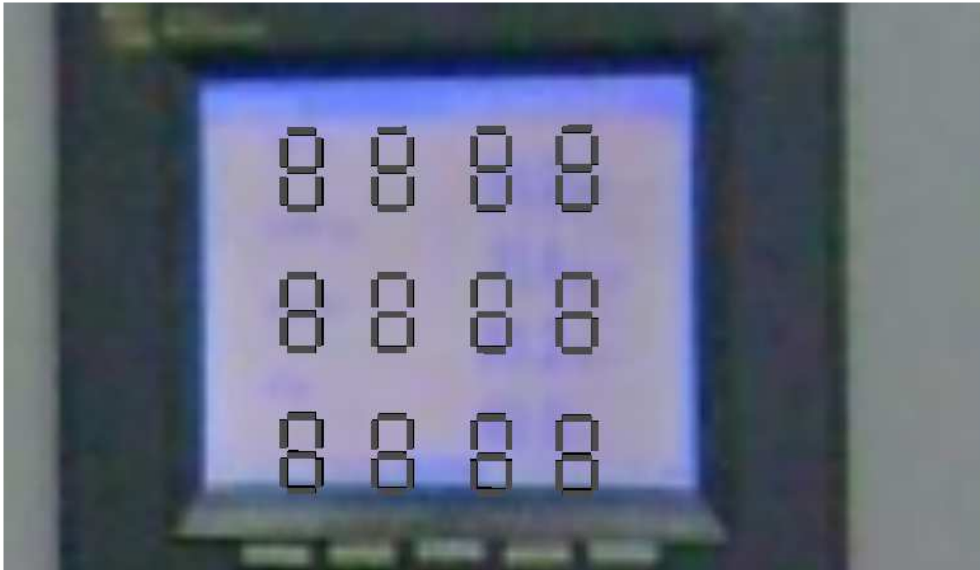


Figura 15 mostradores sete segmentos no espaço 3D

Para que se obtenham mais casas decimais para representação das grandezas associadas basta colocarmos a quantidade de mostradores que necessitamos, isso pode ser feito através da ferramenta que será especificada na próxima sessão.

A descrição referente a Figura 15 pode ser visualizado no Código 5

```
<Group>
	<Transform translation="0 -0.05 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.04 0.01 0.01"/>
		</Shape>
	</Transform>
	<Transform translation="-0.025 -0.025 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.01 0.04 0.01"/>
		</Shape>
	</Transform>
	<Transform translation="-0.025 0.03 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.01 0.04 0.01"/>
		</Shape>
	</Transform>
	<Transform translation="0 0.055 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.04 0.01 0.01"/>
		</Shape>
	</Transform>
	<Transform translation="0.025 0.03 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.01 0.04 0.01"/>
		</Shape>
	</Transform>
	<Transform translation="0.025 -0.025 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.01 0.04 0.01"/>
		</Shape>
	</Transform>
	<Transform translation="0 0 0" rotation = "0 0 0 0">
		<Shape >
			<Appearance >
				<Material diffuseColor="0.2 0.2 0.2"/>
			</Appearance>
		<Box size="0.04 0.01 0.01"/>
		</Shape>
	</Transform>
</Group>
```

Código 5 setesegmentos.x3d

- O QUADRO SINÓTICO

O quadro sinótico também é conhecido como quadro de anunciadores é nele onde as mensagens de erro, alerta ou defeito de algum equipamento são exibidos. A variedade de quadros sinóticos é considerável então ficaria inviável, de modelar todas as possibilidades de quadros sinóticos existentes, então na verdade o que é modelado pelo Código 6 é apenas um prisma que é considerado a unidade básica para a construção do quadro sinótico pela ferramenta de criação de mundos e cenários do LIHM, onde o usuário é que será responsável por colocar a quantidade de pontos (unidades básicas), as cores de cada uma e a mensagem de erro associada a ele.

```
<Group>
    <Transform translation = "0 0 0" rotation="0.0 1.0 0.0  0">
        <Shape>
            <Appearance >
                <Material diffuseColor = "0.3 0 0" />
            </Appearance>
        <Box size="0.296 0.2125 0.01"/>
        </Shape>
    </Transform>
    <Transform translation="-0.022 0.1 -0.01" scale="0.04 0.04 0.04"
rotation="0.0 1.0 0.0  3.14">
        <Shape>
            <Appearance>
                <Material  diffuseColor="0.0 0.0 0.0"/>
            </Appearance>
            <Text  string="02J5">
                <FontStyle style="BOLD" horizontal="true"
justify="MIDDLE" family="ARIAL"  size="1" language="pt"/>
            </Text>
        </Shape>
    </Transform>
    <Transform translation="0.108 0.1 -0.01" scale="0.04 0.04 0.04"
rotation="0.0 1.0 0.0  3.14">
        <Shape>
            <Appearance>
                <Material  diffuseColor="0.0 0.0 0.0"/>
            </Appearance>
            <Text  string="01">
                <FontStyle style="BOLD" horizontal="true"
justify="MIDDLE" family="ARIAL"  size="1" language="pt"/>
            </Text>
        </Shape>
    </Transform>
</Group>
```

Código 6 sinotico.x3d

Um exemplo de sinótico completo pode ser observado na Figura 16

Figura 16 Quadro sinótico no espaço 3D

5.2.2 CHAVES

- CHAVE TIPO PUNHO

A chave do tipo punho é composta por três partes, um prisma que funciona como a base da chave e dois cilindros, um menor que serve de conexão entre o prisma e o cilindro maior que é onde fica o nó sensor para capturar cliques sobre a chave. O Código 7. A classe conversorbotao.java é a responsável pela animação desse objeto no simulador do LIHM essa classe assim como outras constarão no apêndice deste trabalho.

```
<Group>
	<Group>
	<Transform translation="-0.225 0.15 0" center="0.0 -0.15 0.0"
rotation="0.53501f,-0.26238f,-0.80307f, 3.01f">
			<Shape >
				<Appearance >
					<Material diffuseColor="0.0 0.0 0.0"/>
				</Appearance>
			<Cylinder  height="0.3" radius="0.05"/>
			</Shape>
		</Transform>
	<TouchSensor/>
	</Group>
			<Transform translation="-0.15 0 0" rotation="0.0 0.0 1.0
1.57">
		<Shape>
			<Appearance >
				<Material />
			</Appearance>
		<Cylinder  height="0.15" radius="0.05"/>
		</Shape>
	</Transform>
	<Transform translation="-0.225 0 0">
		<Shape>
			<Appearance >
				<Material diffuseColor="0.0 0.0 0.0"/>
			</Appearance>
		<Sphere radius="0.05"/>
		</Shape>
	</Transform>
	<Transform translation="0 0 0" rotation="0.0 1.0 0.0 1.57">
		<Shape>
			<Appearance >
				<Material  diffuseColor="1.0 1.0 0.5"/>
			</Appearance>
		<Box  size="0.3 0.6 0.02"/>
		</Shape>
	</Transform>
</Group>
```

Código 7 chavepunho.x3d

A saída da descrição do Código 7 chavepunho.x3d pode ser vista na Figura 17.

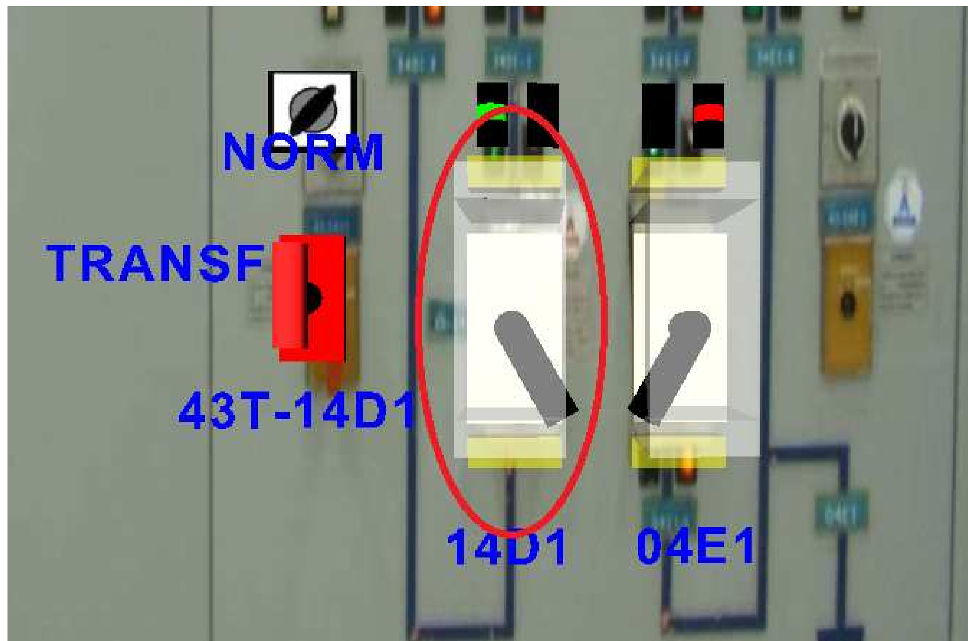


Figura 17 Chave punho no espaço 3D

- CHAVE GPG (GIRO, PRESSÃO GIRO)

A chave giro – pressão – giro modela o tipo de chave mais comum encontrado em subestações, ela pode ser associada a um disjuntor ou a uma seccionadora e esse nome vem do fato de que para ela ser operada é necessário pressionar a chave a após gira - lá no sentido que se deseja. No simulador do LIHM ela é modelada por uma esfera da cor que indica o estado da chave e um prisma onde está localizado um sensor de toque. O código referente a chave GPG pode ser visto no Código 8.

```
<Group >
    <Group>
        <Transform translation="0.00017 -0.00614 -0.24" center="0.0 0.0
0.0" rotation="0.0 0.0 1.0  1.57">
            <Shape >
                <Appearance >
                    <Material diffuseColor="0 0 0"/>
                </Appearance>
            <Box  size="0.047089 0.17798 0.025"/>
            </Shape>
        </Transform>
    <TouchSensor/>
    </Group>
    <Transform  translation="0 0 0">
        <Shape >
            <Appearance >
                <Material diffuseColor="0.0 1.0 0.0"/>
            </Appearance>
        <Sphere  radius="0.22986"/>
        </Shape>
    </Transform>
</Group>
```

Código 8 chavegpg.x3d

A saída do Código 8 pode ser visto na Figura 18.

Figura 18 Chave gpg no espaço 3D

- CHAVE DE TRANSFERÊNCIA

A chave de transferência modela o comportamento de uma chave específica encontrada em subestações onde o arranjo do pátio é do tipo barra normal e de transferência. Essa chave possui três estados o estado normal, em transferência e

transferido, atualmente no simulador do LIHM apenas dois estados são reconhecidos, o normal e o transferido.

Atualmente existem modelados três tipos de chave de transferência o tipo um é formado por um prisma que possui uma textura que é um desenho da chave que recobre esse prisma, a sua descrição pode ser vista no Código 9. A chave tipo dois também é composta por um prisma e uma textura que representa a chave e cobre o prisma,a diferença entre as chaves tipo um e tipo dois além da textura são suas dimensões. A descrição da chave tipo dois pode ser vista no Código 10.

```
<Group >
    <Transform translation="0 0 0" rotation="0.0 1.0 0.0 0">
        <Shape >
            <Appearance >
                <Material />
                    <ImageTexture
url="./Imagens/botaoGiro.jpg"/>
            </Appearance>
        <Box  size="0.38599 0.41515 0.01"/>
        </Shape>
    </Transform>
<TouchSensor />
</Group>
```

Código 9 ctf1.x3d

```
<Group >
    <Transform translation="0 0 0" rotation="0.0 1.0 0.0 0">
        <Shape >
            <Appearance >
                <Material />
                    <ImageTexture
url="./Imagens/Chave_43T_14V2.jpg"/>
            </Appearance>
        <Box size="0.27631 0.45 0.01"/>
        </Shape>
    </Transform>
<TouchSensor />
</Group>
```

Código 10 ctf2.x3d

A saída referente ao Código 9 pode ser visualizada na Figura 19

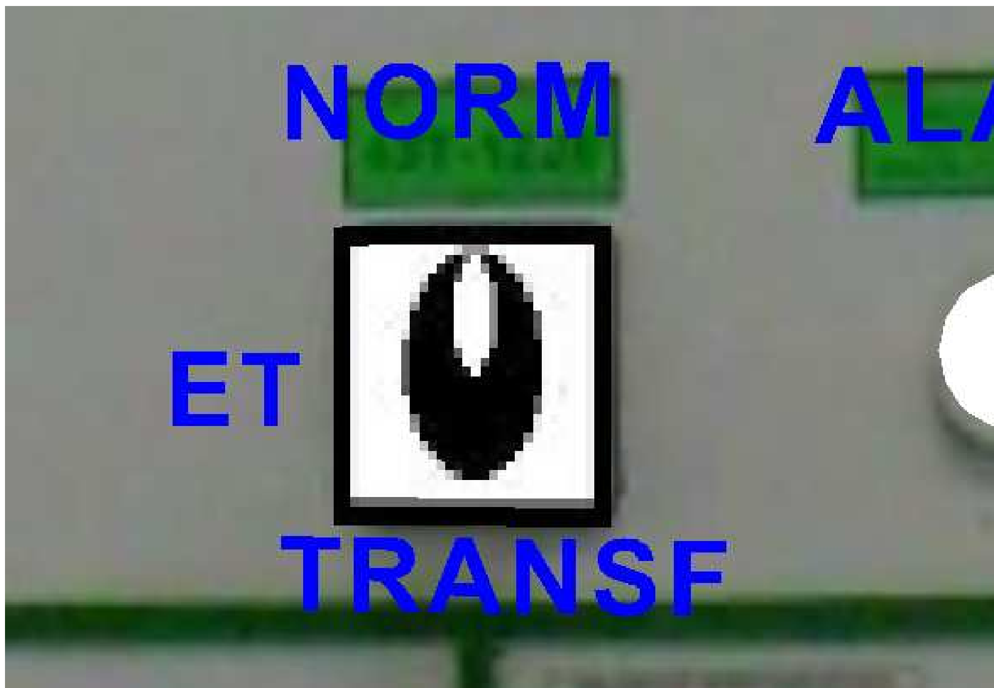


Figura 19 Ctf tipo 1 no espaço 3D

A saída referente ao Código 10 pode ser visualizada na Figura 20

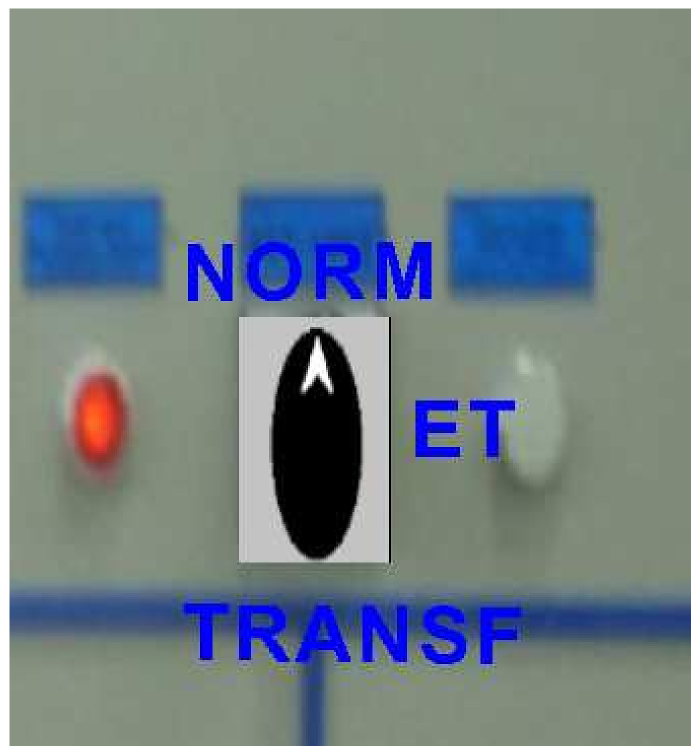


Figura 20 Ctf tipo 2 no espaço 3D

A chave de transferência do tipo três pode ser denominada também de chave de transferência tipo punho e ela se assemelha a chave tipo punho já descrita anteriormente no Código 7. Essa chave é composta de um prisma vermelho que serve de base e de

depois cilindros ortogonais entre si que compõem o punho da chave, sua descrição pode ser vista no Código 11. Nas subestações essas chaves são compostas de peças encaixáveis que formam o punho e estão localizadas no armário, quando o operador necessita operar a chave ele “monta” as peças que a compõem e opera, mas essa opção ainda não foi implementada no simulador do LIHM. Assim como a chave tipo punho as chaves de transferência possuem uma classe que realiza a comunicação com o banco de dados e a animação no momento em que se atua na chave, seu código consta nos anexos deste trabalho.

```
    <Group>
        <Group>
            <Transform translation="0 0 -0.225" center="0.0 0.0 0.0"
rotation="0.0 0.0 1.0 1.57">
                <Shape >
                    <Appearance >
                        <Material diffuseColor="1.0 0.1 0.1"/>
                    </Appearance>
                <Cylinder  height="0.3" radius="0.05"/>
                </Shape>
            </Transform>
        <TouchSensor />
        </Group>
        <Transform translation="0 0 -0.1" rotation="1.0 0.0 0.0 1.57">
            <Shape>
                <Appearance >
                    <Material  diffuseColor="1.0 0.1 0.1"/>
                </Appearance>
            <Cylinder  height="0.2" radius="0.05"/>
            </Shape>
        </Transform>
        <Transform translation="0 0 0" rotation="0.0 1.0 0.0 0">
            <Shape>
                <Appearance >
                    <Material diffuseColor="1.0 0.0 0.0"/>
                </Appearance>
            <Box  size="0.2 0.4 0.02"/>
            </Shape>
        </Transform>
    </Group>
```

Código 11 ctfpunho.x3d

A saída referente ao Código 11 pode ser visualizada na Figura 21

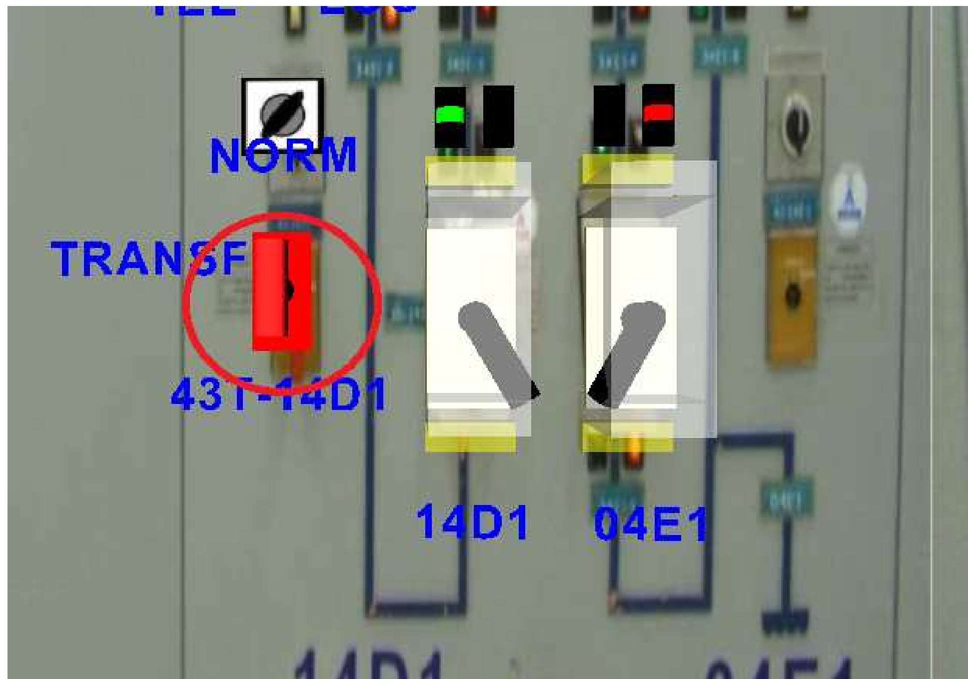


Figura 21 Ctf tipo punho no espaço 3D

- CHAVE LOCAL TELECOMANDO

A chave local telecomando, conhecida como chave loc tel, é a chave que permite a atuação sobre os equipamentos da subestação remotamente pelo centro de operação do sistema. A chave Loc Tel é modelada por um prisma coberto por uma textura que é um desenho da chave, ela possui apenas duas posições a posição Loc ou local onde a sala de comando da própria subestação atua sobre os equipamentos do pátio e a posição Tel ou telecomando onde o centro de operação do sistema é quem atua. A sua descrição pode ser visualizada segundo o Código 12.

```
<Group>
    <Transform translation="0 0 0" rotation="0.0 1.0 0.0  0">
        <Shape>
            <Appearance >
                <Material/>
                    <ImageTexture
url="./Imagens/loctel.jpg"/>
            </Appearance>
        <Box  size="0.27631 0.25739 0.01"/>
        </Shape>
    </Transform>
<TouchSensor/>
</Group>
```

Código 12 loctel.x3d

A saída referente ao Código 12 pode ser visualizado na Figura 22

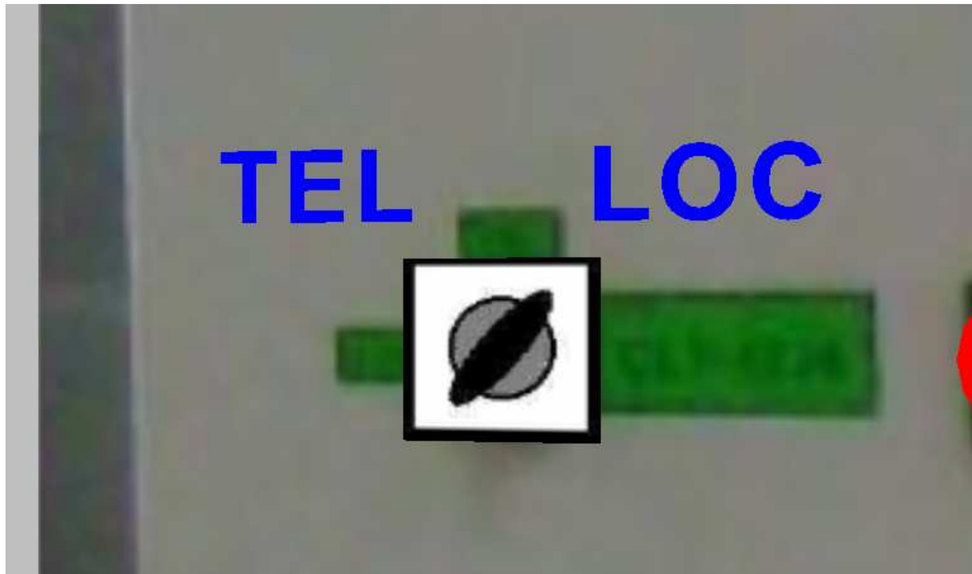


Figura 22 Chave Loc Tel no Espaço 3D

5.2.3 ACESSÓRIOS DAS CHAVES

- COPOS

O copo é a peça de acrílico utilizada para evitar que uma chave seja acionada de maneira acidental, atualmente existem dois copos, o copomaior.x3d utilizado no caso em que se precisa cobrir uma chave do tipo punho e o copo menor que protege chaves do tipo giro pressão giro, a classe conversorprotecao.java é responsável pela animação de ambos os copos independente do tamanho e se encontra no apêndice no final deste trabalho.

O copo maior é descrito segundo o Código 13.

```
<Transform translation="0 0 0" rotation="0.0 1.0 0.0 4.71" center="0 0 0">
    <TouchSensor />
    <Inline url="./Inlines/protecao.x3d"/>
</Transform>
```

Código 13 copomaior.x3d

A saída da descrição mostrada no Código 13 pode ser vista na Figura 23.

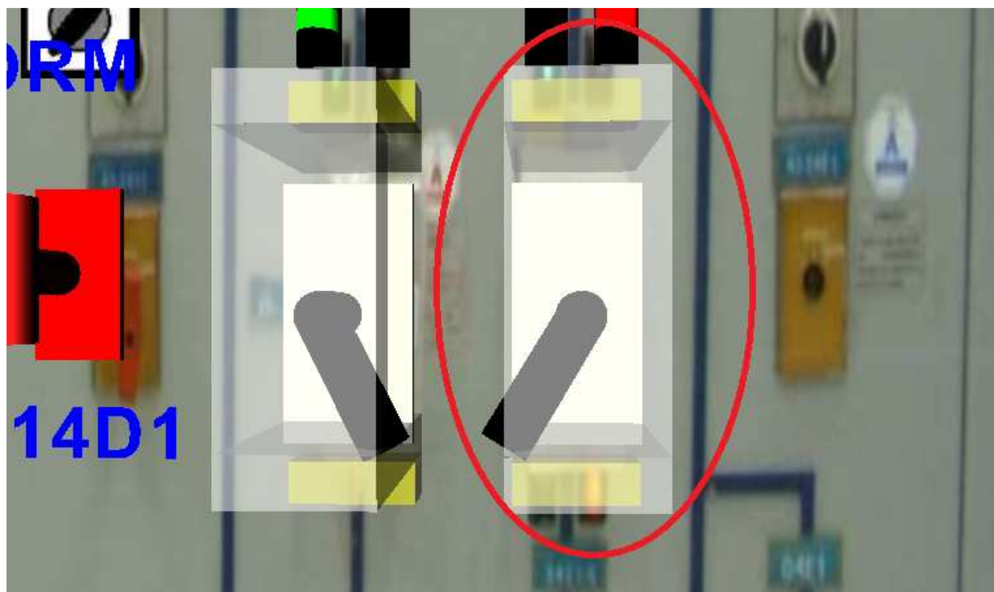


Figura 23 Copo maior no espaço 3D

Já o copo menor é descrito segundo o Código 14

```
<Transform translation="0 0 0" rotation="0.0 1.0 0.0 4.71" center="0 0 0">
    <TouchSensor />
    <Inline url="./Inlines/protecao_botao.x3d"/>
</Transform>
```

Código 14 copomenor.x3d

A saída da descrição mostrada no Código 14 pode ser vista na Figura 24.

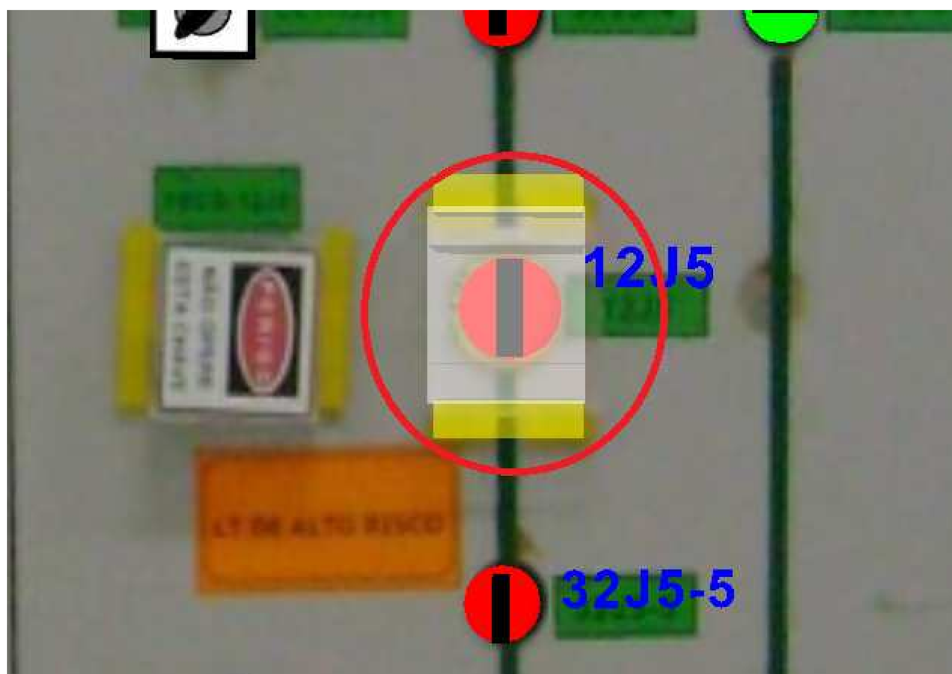


Figura 24 Copo menor no espaço 3D

Os arquivos “protecao.x3d” e “protecao_botao.x3d” são utilizados para modelar toda a estrutura física do copo maior e do copo menor respectivamente que são compostos de cinco prismas de tamanhos diferentes que estão dispostos compondo o copo de proteção, além do *inline* temos um nó sensor para capturar eventos sobre o copo.

5.2.4 STRINGS

- STRINGS DE TEXTO

As strings são as etiquetas colocadas perto dos componentes dos painéis para indicar o estado em que a chave está ou o nome daquele componente.

```
<Transform translation="0 0 0" rotation="0.0 0.0 0.0  3.14">
        <Shape>
                <Appearance>
                        <Material  diffuseColor="0.0 0.0 1.0"/>
                </Appearance>
                <Text  string="32J5-7">
                <FontStyle style="BOLD" family="Arial" size="0.18"/>
                </Text>
        </Shape>
</Transform>
```

Código 15 string.x3d

A saída do Código 15 pode ser visualizada na Figura 25

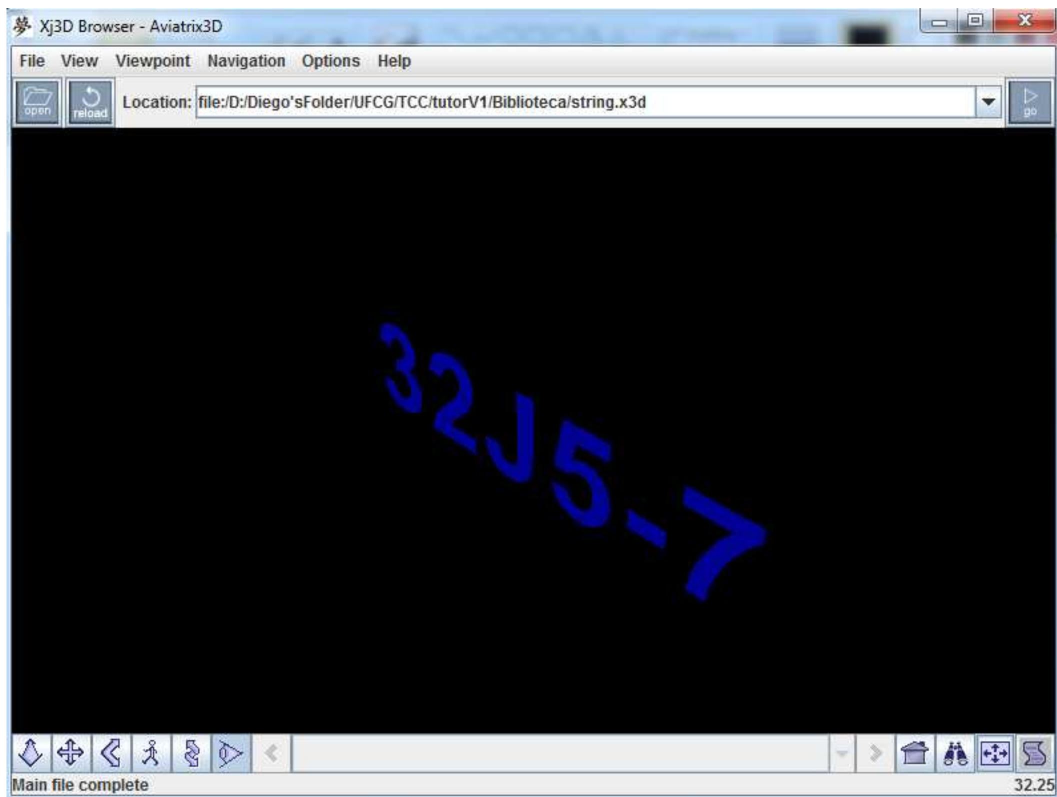


Figura 25 String de texto no espaço 3D

5.2.5 Objetos que compõem a sala de comando

- Armário

O armário é a representação dos painéis encontrados na sala de operação de subestações, esse objeto é composto de dois prismas onde um é imerso no outro, o prisma menor que fica localizado no interior do maior é revestido com a textura referente ao painel que se quer modelar. No Código 16 vemos como isso é feito, a textura foi definida com fins de demonstração na prática ela será colocada pelo usuário utilizando a ferramenta de criação de mundo e cenários do LIHM.

```
<Group>
        <Group>
                <Transform translation="0 0 -0.2" rotation="0.0 1.0 0.0  3.14">
                        <Shape>
                                <Appearance>
                                        <Material />
                                        <ImageTexture  url="./Imagens/LT12T5.JPG"/>
                                </Appearance>
                                <Box  size="2.7 14.0 3.0"/>
                        </Shape>
                </Transform>
                <Transform translation="0 0 0" rotation="0.0 1.0 0.0  3.14">
                        <Shape >
                        <Appearance>
                                <Material  diffuseColor="0.5 0.5 0.5"/>
                        </Appearance>
                                <Box  size="3.1 14.4 3.2"/>
                        </Shape>
                </Transform>
        </Group>
</Group>
```

Código 16 armário.x3d

A saída da descrição do Código 16 armário.x3d) pode ser vista na Figura 26

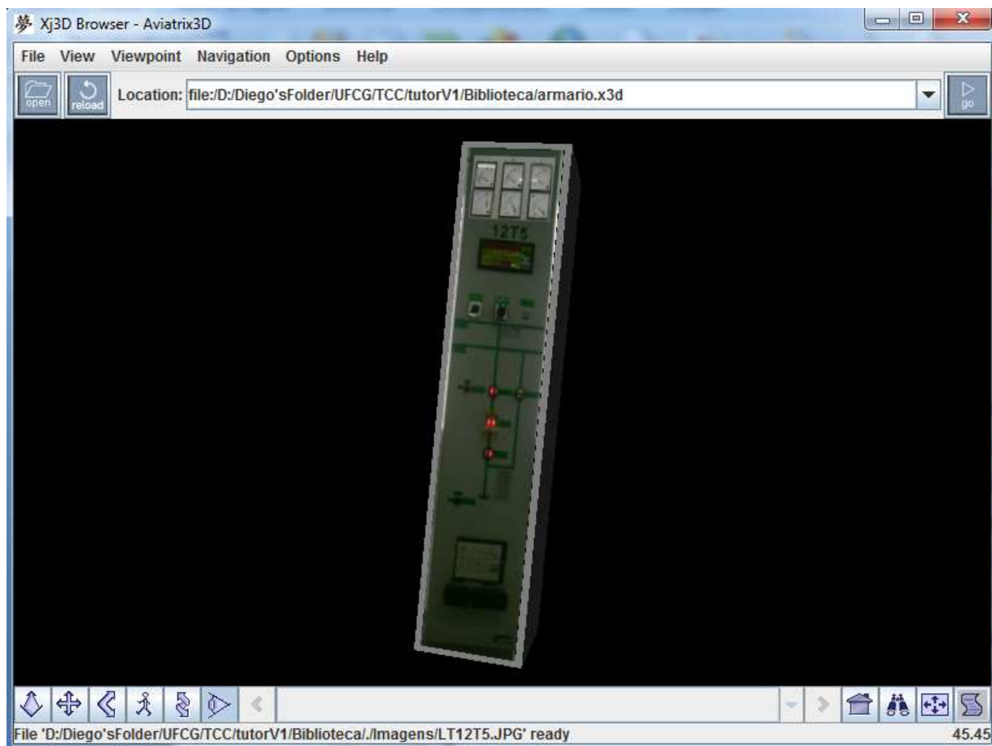


Figura 26 Armário no espaço 3D

- MONITOR

O monitor é o objeto onde é mostrada a tela do supervisório na sala de operações, basiamente é o mesmo encontrado em computadores pessoal, o código que modela o monitor pode ser encontrado na Código 17 e sua saída na.

```
<Group>
        <Transform DEF="Monitor1" scale="0.5 0.5 0.5" translation="-3 1.63 -1.65">
                <Inline  url="monitor.x3d"/>
        </Transform>
</Group>
```

Código 17 Código que modela o monitor

Figura 27 Monitor no espaço 3D

- TECLADO

Assim como o monitor o teclado é o mesmo encontrado em computadores pessoais e sua descrição pode ser vista no Código 18, o arquivo “teclado.x3d” localiza-se na mesma pasta que o simulador por isso a url só contem o nome do arquivo.

```
<Group>
<Transform DEF="Teclado" scale="2 2 2" rotation="0 1 0 4.71" translation="-2 0.1 -1.3">
        <Inline  url="teclado.x3d"/>
</Transform>
</Group>
```

Código 18 Código que modela o Teclado

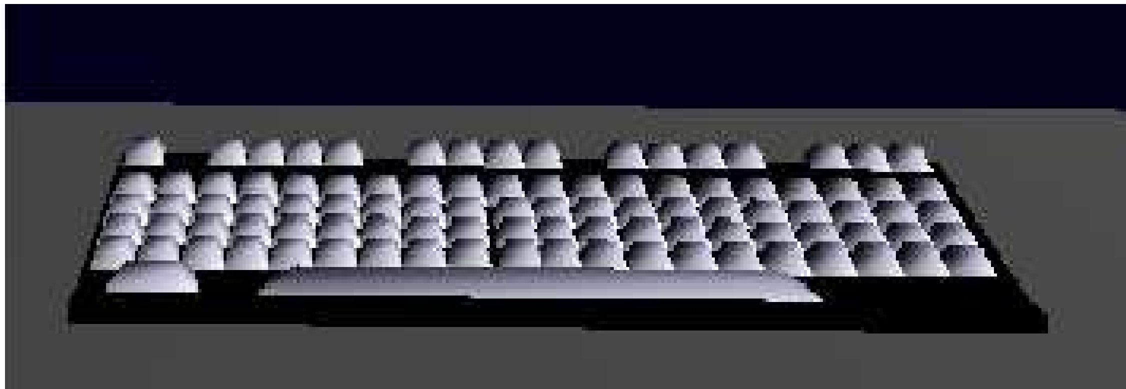


Figura 28 Teclado no espaço 3D

- CADEIRA

A cadeira é a modelagem de uma poltrona encontrada na sala de comando de subestações e ambientes de escritório

```
<Group>
<Transform DEF="cadeira" translation = "-2.7 -2 2">
                <Inline url="cadeira.x3d"/>
</Transform>
</Group>
```

Código 19descrição da cadeira em X3D

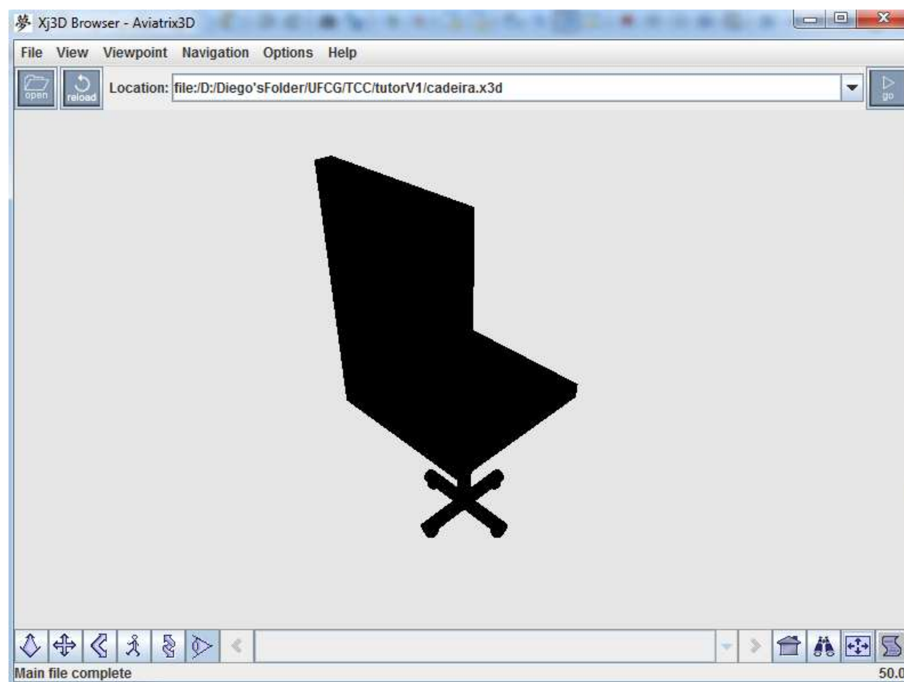


Figura 29 Cadeira no espaço 3D

Na seção seguinte será especificada a ferramenta de edição de cenários e de mundos virtuais a qual terá como base a biblioteca de objetos estruturada neste trabalho.

5.3 OBJETOS – REPRESENTAÇÃO COMPORTAMENTAL

Além da representação visual existe também a representação comportamental que consiste nas alterações realizadas em um determinado objeto quando há algum tipo de interação com o mesmo, por exemplo, o cursor está sobre ele ou quando o objeto recebe um clique do *mouse*. A representação comportamental é realizada utilizando a linguagem de programação Java, cada classe modela o comportamento de um objeto ou de uma série de objetos que são modelados fisicamente de maneiras diferentes, mas que possuem o mesmo comportamento.

- CONVERSORCTF.JAVA

A classe Convesorctf é responsável pelo comportamento das chaves de transferência, sejam elas do tipo um, tipo dois ou tipo punho, nelas são definidos os estados FECHADO_V e ABERTO_V, FECHADOS_V2 e ABERTO_V2 devido a posição inicial da chave. O método convesorCTF é chamado no momento em que um sensor de uma chave de transferência atua por intermédio do usuário, o nó referente a chave que se deseja atuar é procurado, caso ele seja encontrado e esteja em um das posições definidas sua posição é alterada mudando a posição da chave no mundo virtual. Ao final desse processo as ações do usuário são registradas no bando de dados.

```
package AV;

import comunicacao.Servidor;
import comunicacao.ServidorBD;
import java.util.Arrays;
import org.web3d.x3d.sai.*;

/**
 *
 * @author Diego
 */
public class ConversorCTF {
    float[] FECHADO_V = {0, 0, 1, 0};
    float[] ABERTO_V = {1, 0, 0, 1.57f};
    float[] FECHADO_V2 = {0.70711f, 0,0.70711f, 3.14f};
    float[] ABERTO_V2 = {0, 1, 0, 1.57f};


    String idInterface = "1";
    String estadoAV = "";
    String estadoCampoS = "";

     public String conversorCTF(String idUsuario, String dispositivo, String nomeRede,
X3DScene  mainScene, Servidor conector, ServidorBD conectorBD){
           float[] estadoChave = new float[4];
           X3DNode  tipoChave= mainScene.getNamedNode(dispositivo);
           if (tipoChave == null) {
                 System.out.println("Nó: "+dispositivo+" não encontrado CTF.");
                 return "";
           }
           else{
                 SFRotation pos = (SFRotation) tipoChave.getField("rotation");
                 //Salva a posicao de rotacao do objeto e armazena em estadoChave
                 pos.getValue(estadoChave);

                 if(Arrays.equals(estadoChave, ABERTO_V)){
//                                            Muda o valor de rotacao
                       pos.setValue(FECHADO_V);
                       estadoAV = "Normal";
                 }
                //Compara o valor atual de rotation (estadoChave) com valor da constante
ABERTO/FECHADO
                 else if (Arrays.equals(estadoChave, FECHADO_V)){
                       pos.setValue(ABERTO_V);
                       estadoAV = "Transf";
                 }

                 else if (Arrays.equals(estadoChave, ABERTO_V2)){
                       pos.setValue(FECHADO_V2);
                       estadoAV = "Transf";
                 }
                 else if (Arrays.equals(estadoChave, FECHADO_V2)){
                       pos.setValue(ABERTO_V2);
                       estadoAV = "Normal";
                 }
                 if(idUsuario.equals("1")){
//                        Envia o nome do painel e chave para a rede de paineis do CPNTools
                       conector.envia(nomeRede);
                 }
```

```
                //Salvando no banco as acoes do usuario
                conectorBD.inserirAcaoLog(idUsuario, idInterface, dispositivo, estadoAV,
estadoCampoS, estadoCampoS);
            }
            return estadoAV;

        }

}
```

Código 20 Código da classe Conversorctf.java

Para que o sensor atue é necessário que o mesmo seja inserido no vetor de chaves localizados no método *public static ArrayList<String> adicionarChave()* do jfTutor e jfTreinando assim como na função *public void tratarEventosSensores(Object touchAtivado, String idUsuario)*, que trata as mensagens dos nós sensores. Todos os sensores de todos os objetos, que possuam sensores, devem estar presentes nesse array assim como a chamada da função referente ao tratamento das mensagens de cada nó sensor deve está presente na função *tratarEventosSensores.*

- CONVERSORALARME.JAVA

Esta classe é responsável pelo comportamento do alarmes, nela existem os estados WHITE e YELLOW que representam as posições de operação do alarme, na posição WHITE o alarme está desligado e sua cor é a cor branca já na posição YELLOW o alarme está disparado e a cor do alarme varia entre branco e amarelo além de um ruído sonoro ser emitido até o usuário atue no sensor do alarme.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package AV;

import org.web3d.x3d.sai.*;
import java.util.*;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.util.Arrays;

import javazoom.jl.player.Player;
/**
 *
 * @author LIHM
 */
public class ConversorAlarme {
    float[] WHITE = {1.0f,1.0f,1.0f};
    float[] YELLOW = {1.0f, 1.0f,0};


    private static ArrayList<String> alarmesAtivos;
    X3DScene myMainScene;
Thread alarmeLuminosoThread;
    Thread alarmeSonoroThread;

    public void inicializaAlarme(X3DScene mainScene){
        alarmeLuminosoThread = new alarmeLuminoso();
        alarmeSonoroThread = new alarmeSonoro();
        myMainScene = mainScene;
        alarmeLuminosoThread.start();
        alarmeSonoroThread.start();
        alarmesAtivos= new ArrayList<String>();
    }
    public void adicionaAlarme(String alarme){
        if (!alarmesAtivos.contains(alarme)) {
            alarmesAtivos.add(alarme);
        } else {
            System.out.println("Alarme já inserido na lista");
        }
    }
    public void retiraAlarme(String alarme) {
        float[] corDoObjeto = new float[3];
        X3DNode mat = myMainScene.getNamedNode("SphereColor_" + alarme);
        if (mat == null) {
            System.out.println("Couldn't find material named:");
            return;
        } else {
            SFColor color = (SFColor) mat.getField("diffuseColor");
            color.getValue(corDoObjeto);
            if (Arrays.equals(corDoObjeto, YELLOW)) {
                color.setValue(WHITE);
            }
        }
         alarmesAtivos.remove(alarme);
    }
```

```
class alarmeLuminoso extends Thread {
    @Override
    public void run() {
        try {
            do {
                int counter = 0;
                float[] corDoObjeto = new float[3];
                for (counter = 0; counter < alarmesAtivos.size(); counter++) {
                    X3DNode mat =
myMainScene.getNamedNode("SphereColor_"+alarmesAtivos.get(counter));
                    if (mat == null) {
                        System.out.println("Couldn't find material named:");
                        return;
                    } else {
                        SFColor color = (SFColor) mat.getField("diffuseColor");
                        color.getValue(corDoObjeto);
                        if (Arrays.equals(corDoObjeto, WHITE)) {
                            color.setValue(YELLOW);
                        } else if (Arrays.equals(corDoObjeto, YELLOW)) {
                            color.setValue(WHITE);
                        }
                    }
                }
                Thread.sleep(500);
            } while (true);
        } catch (Exception e) {
            System.out.println("Thread do alarme luminoso interrompida.");
        }
         System.out.println("teste");
    }
} class alarmeSonoro extends Thread {
           String path = "C:\\Audio.mp3";
    private File Alarme_mp3;
    private Player Alarme_player;
    private ConversorBotao novaChave;

    public void LoadAlarme() {

        try {
            this.Alarme_mp3 = new File(path);
            FileInputStream fis = new FileInputStream(Alarme_mp3);
            BufferedInputStream bis = new BufferedInputStream(fis);
            this.Alarme_player = new Player(bis);

        } catch (Exception e) {
            System.out.println("Problema ao carregar " + Alarme_mp3);
            e.printStackTrace();
        }

    }

    private void tocar() {
        try {
            this.Alarme_player.play();
        } catch (Exception e) {
            System.out.println("Problema ao tocar " + Alarme_mp3);
            e.printStackTrace();
        }
    }
```

```
        @Override
        public void run() {
            try {
                do {
                    while(!alarmesAtivos.isEmpty()){
                        LoadAlarme();
                        tocar();
                    }
                    Thread.sleep(1200);
                } while (true);

            } catch (Exception e) {
                System.out.println("Thread do Alarme Sonoro interrompida.");
            }
        }
    }
}
```

Código 21 Código da classe Conversoralarme.java

No momento em que o sensor do alarme atua é chamado o método *adicionaAlarme()* que adiciona o identificador do alarme atuado na lista de alarmes ativos que é verificado pela thread e enquanto houver alarmes inseridos nessa lista o ruído sonoro não cessará, no momento que se atua novamente no alarme ele chama a função *retiraAlarme()* que remove o identificador do alarme da lista, fazendo com que este alarme pare de piscar e se ele for o ultimo da lista o ruído cessará.

- CONVERSOLOCTEL.JAVA

A classe conversorloctel.java modela o comportamento das chaves local/telecomando possui os estados FECHADO_V, ABERTO_V, FECHADO_V2 e ABERTO_V2 devido a posição inicial da chave. O principio de funcionamento é o mesmo da chave de transferência.

```
package AV;

import comunicacao.Servidor;
import comunicacao.ServidorBD;
import java.util.Arrays;
import org.web3d.x3d.sai.*;

/**
 *
 * @author LIHM
 */
```

```
public class ConversorLoctel {
   float[] FECHADO_V = {0, 1, 0, 0};
   float[] ABERTO_V = {0, 0, 1, 1.57f};
   float[] FECHADO_V2 = {0, 0, 1, 0};
   float[] ABERTO_V2 = {1, 0, 0, 1.57f};

   //Dados para inserir na tabela de log
   String estadoAV = "";
   String estadoCampoS = "";
   String idInterface = "1";
/**
 * Conversor para a Chave do tipo Local/Telecomando
 * @param dispositivo Nome da chave loctel
 * @param nomeRede Nome da Ficha na rede de petri
 */
 public String conversorLoctel(String idUsuario, String dispositivo, String nomeRede,
X3DScene  mainScene, Servidor conector, ServidorBD conectorBD){

        float[] estadoChave = new float[4];

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

            if(Arrays.equals(estadoChave, ABERTO_V)){
            //Muda o valor de rotacao
                pos.setValue(FECHADO_V);
                estadoAV = "Loc";
            }
            else if (Arrays.equals(estadoChave, FECHADO_V)){
                pos.setValue(ABERTO_V);
                estadoAV = "Tel";
            }
            if(Arrays.equals(estadoChave, ABERTO_V2)){
//                                Muda o valor de rotacao
                pos.setValue(FECHADO_V2);
                estadoAV = "Loc";
            }

else if (Arrays.equals(estadoChave, FECHADO_V2)){
                pos.setValue(ABERTO_V2);
                estadoAV = "Tel";
            }

            if(idUsuario.equals("1")){
            //Envia o nome do painel e chave para a rede de paineis do CPNTools
              conector.envia(nomeRede);
            }
            //Salvando no banco as acoes do usuario
            conectorBD.inserirAcaoLog(idUsuario, idInterface, dispositivo, estadoAV,
estadoCampoS, estadoCampoS);
```

```
                conectorBD.updateMonitoramentoAV(dispositivo, estadoAV);
            }
            return estadoAV;

        }
}
```

Código 22 Código da classe Conversorloctel.java

- CONVERSORPROTECAO.JAVA

A classe conversor proteção modela o comportamento dos copos de proteção das chaves e dos botões. Existem os estados FECHADO_P, ABERTO_P, FECHADO_P2, ABERTO_P2, FECHADO_P3 e ABERTO_P3 e haverá a necessidade de no futuro existir os estados FECHADO_P4 e ABERTO_P4 já que a atuação sobre os copos de proteção ocorrem devido ao modo como eles estão dispostos no mundo virtual e em uma das áreas do mundo virtual modelado pelo simulador ainda não foi inserida nenhuma proteção. Basicamente o funcionamento ocorre da mesma forma dos objetos já mostrados, no momento em que o sensor referente ao copo é acionado ele chama o método *conversorProtecao()* que será responsável por encontrar no código x3d o copo referente ao sensor que foi acionado, verifica qual a posição em que esse copo se encontra e a modifica.

```
package AV;
import comunicacao.Servidor;
import comunicacao.ServidorBD;
import java.util.Arrays;
import org.web3d.x3d.sai.*;
/**
 *
 * @author LIHM
 */
public class ConversorProtecao {
    float[] FECHADO_P = {0, 1, 0, 4.71f};
    float[] ABERTO_P = {0.57735f, -0.57735f, -0.57735f, 2.094f};
    float[] FECHADO_P2 = {0, 1, 0, 3.14f};
    float[] ABERTO_P2 = {0.71702f, -0.70711f, 0, 3.14f};
    float[] FECHADO_P3 = {0, 1, 0, 0};
    float[] ABERTO_P3 = {0, 0, -1, 1.57f};

    String estadoAV = "";
    String estadoCampoS = "";
    String idInterface = "1";
    public String conversorProtecao(String idUsuario, String dispositivo, String nomeRede,
X3DScene mainScene, Servidor conector, ServidorBD conectorBD) {

        float[] estadoChave = new float[4];
```

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

           if (Arrays.equals(estadoChave, FECHADO_P)) {
              pos.setValue(ABERTO_P);
              estadoAV = "Aberto";
           }

            } else if (Arrays.equals(estadoChave, ABERTO_P)) {
              pos.setValue(FECHADO_P);
              estadoAV = "Fechado";
           } else if (Arrays.equals(estadoChave, ABERTO_P2)) {
              pos.setValue(FECHADO_P2);
              estadoAV = "Fechado";

           } else if (Arrays.equals(estadoChave, FECHADO_P2)) {
              pos.setValue(ABERTO_P2);
              estadoAV = "Aberto";
           } else if (Arrays.equals(estadoChave, ABERTO_P3)) {
              pos.setValue(FECHADO_P3);
              estadoAV = "Fechado";

           } else if (Arrays.equals(estadoChave, FECHADO_P3)) {
              pos.setValue(ABERTO_P3);
              estadoAV = "Aberto";
           }

           //Salvando no banco as acoes do usuario
           conectorBD.inserirAcaoLog(idUsuario, idInterface, dispositivo, estadoAV,
estadoCampoS, estadoCampoS);
           conectorBD.updateMonitoramentoAV(dispositivo, estadoAV);
        }
        return estadoAV;
    }
}
```

Código 23 Código da classe Conversorprotecao.java

- PARAMETRIZADOR.JAVA

A classe parametrizadora serve para converter os valores reais de potência, corrente ou tensão definidos no JfTutor e jfTreinando em rotação dos ponteiros, para serem utilizados nos mostradores analógicos.

```
package AV;

/**
 *
 * @author Erick Lucena
 * @version 1.0
*/
public class Parametrizador {

	private float graus;
	private int numeroMaximoDaUnidade;
	private int numeroMinimoDaUnidade;
	private int delta;
	public Parametrizador(float graus,int numeroMinimoDaUnidade, int
numeroMaximoDaUnidade) {
		this.graus = graus;
		this.numeroMinimoDaUnidade = numeroMinimoDaUnidade;
		this.numeroMaximoDaUnidade = numeroMaximoDaUnidade;
		delta = numeroMaximoDaUnidade - numeroMinimoDaUnidade;


	}
	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;
	}
}
```

Código 24 Código da classe parametrizador.java

- MOSTRADOR.JAVA

A classe mostrador.java é uma das classes que modela o comportamento dos mostradores de sete segmentos, modificando a cor dos segmentos para a representação de um numero, e define a quantidade de algarismos de um determinado mostrador no momento de sua construção.

```
package AV;
import org.web3d.x3d.sai.*;
/**
 * 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;
   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());
        }
   }
}
```

Código 25 Código da classe mostrador.java

- MOTORDEDISPLAY.JAVA

O motordedisplay.java juntamente com a classe mostrador.java modelam o comportamento do mostrador de sete segmentos, porém a classe motordedisplay.java gerencia se é possível inserir determinado numero em um display de sete segmentos e é a classe que o mostrador.java utiliza para modificar a cor do segmento para representação de um numero.

```
package AV;

import org.web3d.x3d.sai.*;

/**
 * Descricao: Classe responsavel pela atualizacao de um numero para um display.
 * @author Erick
 * @version 1.0
 */
public class MotorDeDisplay {
	public Object[] arraysDoDisplay;
	public int tamanhoDoDisplay;
	public MotorDeDisplay(int tamanhoDoDisplay) {
		this.tamanhoDoDisplay = tamanhoDoDisplay;
	}

	public void modificaSegmento(int numero, SFColor[][] segmento) throws
Exception {
		boolean negativo = (numero < 0);
		if(negativo) {
			numero = -1 *numero;
		}
		String auxiliar = Integer.toString(numero);

		if (((auxiliar.length() >= tamanhoDoDisplay) && (!(negativo))) ||
				((auxiliar.length() > tamanhoDoDisplay-1) &&
(negativo))) {
			Exception e = new Exception("Número não suportado pelo
display.");
			throw e;
		} else {
			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);
			}
		}
	}

}
```

Código 26 Código da classe motordedisplay.java

- MOTORDEPONTEIRO.JAVA

A classe motordeponteiro.java é responsável pela animação dos ponteiros dos mostradores analógicos, os estados FECHADO_V, ABERTO_V, FECHADO_H, ABERTO_H, FECHADO_V2, ABERTO_V2, FECHADO_H2 e ABERTO_H2 são necessários para a verificação da posição da chave ,normalmente um disjuntor, que está associado ao mostrador analógico. Caso a chave esteja aberta os valores mostrados serão iguais a zero. Através dos valores parametrizados com o uso da classe parametrizador.java o campo keyvalues presente nos ponteiros é alterado ocorrendo assim a sua animação.

```
package AV;

import java.util.*;
import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.grouping.Transform;
import org.web3d.x3d.sai.interpolation.OrientationInterpolator;

/**
 *
 * @author Ademar
 */
public class MotorDoPonteiro {
    private boolean estadoDoPonteiro;
    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[] ABERTO_H2 = {1, 0, 0, 1.57f};
    float[] FECHADO_H2 = {0, 1, 0, 0};
    float[] valorTranslacao = new float[4];
    //Funcao de movimentacao dos ponteiros
    public void motorDoPonteiro(String nomeDoTouch, String dispositivo, String
posicaoChave, String nomeDoTime, String positionInterpolator, String nomeDoPonteiro,
float[] valoresParametrizados, X3DScene mainScene) {
        float[] keyvalues = new float[20];
        float[] estadoChave = new float[4];
        X3DNode TCHS = mainScene.getNamedNode(nomeDoTouch);
        X3DNode TS = mainScene.getNamedNode(nomeDoTime);
        X3DNode PI = mainScene.getNamedNode(positionInterpolator);
        ((OrientationInterpolator) PI).getKeyValue(keyvalues);

                }else if(Arrays.equals(estadoChave, FECHADO_V)){
                keyvalues[3] = valoresParametrizados[3];
                keyvalues[7] = valoresParametrizados[2];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[0];
                keyvalues[19] = 0;
                }
```

```
            }else if(posicaoChave.equals("horizontal")){
              if(Arrays.equals(estadoChave, FECHADO_H)){
                keyvalues[7] = valoresParametrizados[0];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[2];
                keyvalues[19] = valoresParametrizados[3];
              }else if(Arrays.equals(estadoChave, ABERTO_H)){
                keyvalues[3] = valoresParametrizados[3];
                keyvalues[7] = valoresParametrizados[2];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[0];
                keyvalues[19] = 0;
              }
            }
          }else if((valorTranslacao[0] == -12.62f) || (valorTranslacao[0] == 64.475f) ||
(valorTranslacao[1] == 0.6f)){
            if(posicaoChave.equals("vertical")){
              if(Arrays.equals(estadoChave, FECHADO_V2)){
                keyvalues[3] = 0;
                keyvalues[7] = valoresParametrizados[0];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[2];
                keyvalues[19] = valoresParametrizados[3];
                }else if(Arrays.equals(estadoChave, ABERTO_V2)){
                keyvalues[3] = valoresParametrizados[3];
                keyvalues[7] = valoresParametrizados[2];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[0];
                keyvalues[19] = 0;
                }
            }else if(posicaoChave.equals("horizontal")){
              if(Arrays.equals(estadoChave, FECHADO_H2)){
                keyvalues[7] = valoresParametrizados[0];
                keyvalues[11] = valoresParametrizados[1];
                keyvalues[15] = valoresParametrizados[2];
                keyvalues[19] = valoresParametrizados[3];

              }else if(Arrays.equals(estadoChave, ABERTO_H2)){
                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");
    }
  }
}
```

Código 27 Código da classe motordeponteiro.java

- CONVERSORBOTAO.JAVA

A classe conversorbotao.java juntamente com a classe indicadorcampo.java modelam o comportamento da chave giro – pressão – giro com sinaleiros incorporados e da chave tipo punho associado a LED's de sinalização de estados. A classe utiliza a posição (translação e rotação) da chave GPG para decidir como deve ser realizada a animação, altera o estado da chave e muda sua cor para branco, envia uma mensagens para as Redes de Petri Coloridas indicando o novo estado da chave e a chave que foi alterada e por fim armazena as alterações realizadas pelo usuário no banco de dados.

```
package AV;
import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.grouping.Transform;
import java.util.Arrays;
//Pacotes internos
import comunicacao.Servidor;
import comunicacao.ServidorBD;
/**
 *
 * @author LIHM
 */
public class ConversorBotao {

    float[] GREEN = {0, 1.0f, 0};
    float[] RED = {1.0f, 0, 0};
    float[] YELLOW = {1, 1, 0};
    float[] WHITE ={1.0f, 1, 1};
    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 = {1, 0, 0, 1.57f};
    float[] ABERTO_V2 = {0, 1, 0, 0};
    float[] ABERTO_H2 = {1, 0, 0, 1.57f};
    float[] FECHADO_H2 = {0, 1, 0, 0};
    float[] ABERTO_P = {0.53501f,-0.26238f,-0.80307f, 3.01f};
    float[] FECHADO_P = {-0.95748f, 0.17268f, -0.23112f, 2.664f};
    float[] DESLIGADO_A = {0, 1, 0, 0};
    float[] LIGADO_A = {0, 0, 1, 1.57f};

    float[] valorTranslacao = new float[4];
    String idInterface = "1";
    String estadoAV = "";
    String estadoCampoS = "";

    public String conversorBotao(String idUsuario, String dispositivo, String nomeRede,
String posicaoChave, X3DScene mainScene, Servidor conector, ServidorBD conectorBD
){
        float[] estadoChave = new float[4];
        X3DNode mat = mainScene.getNamedNode("SphereColor_"+dispositivo);
        SFColor color = (SFColor) mat.getField("diffuseColor");
```

```
        X3DNode  tipoChave= mainScene.getNamedNode(dispositivo);
        if (tipoChave != null) {
           SFRotation pos = (SFRotation) tipoChave.getField("rotation");
           //Salva a posicao de rotacao do objeto e armazena em estadoChave
           pos.getValue(estadoChave);
           X3DNode DJ = mainScene.getNamedNode(dispositivo);
           ((Transform) DJ).getTranslation(valorTranslacao);

         if(valorTranslacao[2]==37.66f){
              if(posicaoChave.equals("vertical")){
                 if(Arrays.equals(estadoChave, ABERTO_V)){
                    pos.setValue(FECHADO_V);
                    estadoAV = "Fechado";
                 }else if(Arrays.equals(estadoChave, FECHADO_V)){
                    pos.setValue(ABERTO_V);
                    estadoAV = "Aberto";
                 }

            }else if(posicaoChave.equals("horizontal")){
                 if(Arrays.equals(estadoChave, ABERTO_H)){
                    pos.setValue(FECHADO_H);
                    estadoAV = "Fechado";
                 }else if(Arrays.equals(estadoChave, FECHADO_H)){
                    pos.setValue(ABERTO_H);
                    estadoAV = "Aberto";
                 }
              }
           }
           else if((valorTranslacao[0] == -12.62f)|| (valorTranslacao[0] == 64.475f) ||
(valorTranslacao[1] == 0.6f))
           {
              if(posicaoChave.equals("vertical")){
                 if(Arrays.equals(estadoChave, ABERTO_V2)){
                    pos.setValue(FECHADO_V2);
                    estadoAV = "Fechado";
                 }else if(Arrays.equals(estadoChave, FECHADO_V2)){
                    pos.setValue(ABERTO_V2);
                    estadoAV = "Aberto";
                 }

              }
              else if(posicaoChave.equals("horizontal")){
                 if(Arrays.equals(estadoChave, ABERTO_H2)){
                    pos.setValue(FECHADO_H2);
                    estadoAV = "Fechado";
                 }else if(Arrays.equals(estadoChave, FECHADO_H2)){
                    pos.setValue(ABERTO_H2);
                    estadoAV = "Aberto";
                 }
              }
           }
           color.setValue(WHITE);
          if(idUsuario.equals("1")){
              conector.envia(nomeRede);
           }

           //Salvando no banco as acoes do usuario
           conectorBD.inserirAcaoLog(idUsuario, idInterface, dispositivo, estadoAV,
estadoCampoS, estadoCampoS);
           conectorBD.updateMonitoramentoAV(dispositivo, estadoAV);
        }
        return estadoAV;
    }
```

```
        public String conversorBotaoPunho(String idUsuario, String dispositivo, String
nomeRede, X3DScene mainScene, Servidor conector, ServidorBD conectorBD ){
    float[] estadoChave = new float[4];
    X3DNode  tipoChave = mainScene.getNamedNode(dispositivo);
    X3DNode matG = mainScene.getNamedNode("LED_"+dispositivo+"_G_Material");
    SFColor colorG = (SFColor) matG.getField("diffuseColor");
    X3DNode matR = mainScene.getNamedNode("LED_"+dispositivo+"_R_Material");
    SFColor colorR = (SFColor) matR.getField("diffuseColor");
    if(colorG!=null){
      colorG.setValue(WHITE);
      colorR.setValue(WHITE);

    }else{
      System.out.println("erro");
      return estadoAV;
    }

    if(idUsuario.equals("1")){
      Servidor conector;
      conector = supervisorio.getConector();
      conector.envia("LT02J5,DJ12J5,Aberto");
    }
      estadoAV = "Fechado";
      estadoCampoS = "Aberto";
      conectorBD.updateAlarmes("DJ12J6");
      conectorBD.inserirAcaoLog(idUsuario, idInterface, "DJ12J6", estadoAV,
estadoCampoS, estadoCampoS);
      conectorBD.updateMonitoramentoAV("DJ12J6", estadoAV);
  }
}
```

Código 28 Código da classe conversorbotao.java

- INDICADORCAMPO.JAVA

A classe indicadorcampo.java é responsável por receber a mensagem vinda das Redes de Petri Coloridas que representam o campo, caso o nome estado enviado utilizando a classe conversorbotao,java esteja de acordo com o estado no campo a chave muda de cor para verde ou vermelho dependendo da situação, caso o estado enviado utilizando a classe conversorbotao.java esteja em discordância com o indicado no campo a chave se torna amarela, e caso algum defeito ocorra a chave se torna preta.

```
package AV;
import comunicacao.Servidor;
import comunicacao.ServidorBD;
import java.util.Arrays;
import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.grouping.Transform;
/**
 *
 * @author Ademar
 */
public class IndicadorCampo {
    float[] cor = {0, 0, 0};
    float[] corR = {0, 0, 0};
    float[] corG = {0, 0, 0};
    float[] valorTranslacao = new float[4];
    float[] GREEN = {0, 1.0f, 0};
    float[] RED = {1.0f, 0, 0};
    float[] YELLOW = {1.0f, 1.0f, 0};
    float[] BLACK = {0, 0, 0};
    float[] ABERTO_P = {0.53501f, -0.26238f, -0.80307f, 3.01f};
    float[] FECHADO_P = {-0.95748f, 0.17268f, -0.23112f, 2.664f};

    //Dados para inserir na tabela de log
    String estadoAV = "";
    String estadoCampoS = "";
    String idInterface = "0";
    String idUsuario = "0";

    public float[] indicadorCampo(String dispositivo, String posicaoChave, String
nomeRede, String msgRecCampo, X3DScene mainScene, Servidor conector, ServidorBD
conectorBD) {
        float[] estadoChave = new float[4];
        X3DNode tipoChave = mainScene.getNamedNode(dispositivo);
        X3DNode DJ = mainScene.getNamedNode(dispositivo);
        ((Transform) DJ).getTranslation(valorTranslacao);

        X3DNode matG = mainScene.getNamedNode("LED_" + dispositivo +
"_G_Material");
        SFColor colorG = (SFColor) matG.getField("diffuseColor");
        X3DNode matR = mainScene.getNamedNode("LED_" + dispositivo +
"_R_Material");
        SFColor colorR = (SFColor) matR.getField("diffuseColor");
        if (tipoChave == null) {
            System.out.println("Nó: " + dispositivo + " não encontrado.");
            return cor;
        } else {
            SFRotation pos = (SFRotation) tipoChave.getField("rotation");
            //Salva a posicao de rotacao do objeto e armazena em estadoChave
            pos.getValue(estadoChave);
            //System.out.println(estadoChave[0] + " " + estadoChave[1] + " " + estadoChave[2]
+ " " + estadoChave[3]);
            if (posicaoChave.equals("posicao1")) {
                if (Arrays.equals(estadoChave, ABERTO_P)) {
                    estadoAV = "ABERTO";
                    if (msgRecCampo.equalsIgnoreCase(nomeRede + ",Aberto")) {
                        corG = GREEN;
                        corR = BLACK;
                        cor = GREEN;
                        estadoCampoS = "ABERTO";
```

```
                } else {
                    corG = BLACK;
                    corR = BLACK;
                    cor = YELLOW;
                    estadoCampoS = "FECHADO";
                }
            } else if (Arrays.equals(estadoChave, FECHADO_P)) {
                estadoAV = "FECHADO";
                if (msgRecCampo.equalsIgnoreCase(nomeRede + ",Fechado")) {
                    corG = BLACK;
                    corR = RED;
                    cor = RED;
                    estadoCampoS = "FECHADO";
                } else {
                    corG = BLACK;
                    corR = BLACK;
                    cor = YELLOW;
                    estadoCampoS = "ABERTO";
                }
            }
        } else if (posicaoChave.equals("posicao0")) {

            if (msgRecCampo.equalsIgnoreCase(nomeRede + ",Aberto")) {
                corG = GREEN;
                corR = BLACK;
                cor = GREEN;
                estadoCampoS = "ABERTO";
            }
            if (msgRecCampo.equalsIgnoreCase(nomeRede + ",Fechado")) {
                corG = BLACK;
                corR = RED;
                cor = RED;
                estadoCampoS = "FECHADO";
            }
        }
        //Salvando no banco as acoes do usuario
        conectorBD.inserirAcaoLog(idUsuario, idInterface, dispositivo, estadoAV,
estadoCampoS, estadoCampoS);

    }
    colorG.setValue(corG);
    colorR.setValue(corR);
    return cor;
  }
}
```

Código 29 Código da classe indicadorcampo.java

- CONVERSORSINÓTICO.JAVA

O conversorsinotico.java modela o comportamento do quadro sinótico e funciona de maneira semelhante ao conversoralarme.java possuindo uma lista dos alarmes que estão ativos no quadro. Existe uma thread que monitora o anunciador e no momento em que um alarme entra na lista ele começa piscar no mundo virtual entre a

cor ON e OFF até que um comando seja executado para que o alarme seja retirado la lista e a alternância de cores cesse. Cada quadro anunciador possui um sensor para que quando o usuário clique ele exiba este quadro anunciador de perto.

```
package AV;
import java.util.Arrays;
import org.web3d.x3d.sai.*;
import java.util.ArrayList;
import org.web3d.x3d.sai.grouping.Transform;
import org.web3d.x3d.sai.navigation.NavigationInfo;
import org.web3d.x3d.sai.navigation.Viewpoint;
/*
 * @author LIHM
 */
public class ConversorSinotico {
   float[] GREEN_ON = {0, 1.0f, 0};
   float[] RED_ON = {1.0f, 0, 0};
   float[] YELLOW_ON = {1.0f, 1.0f, 0};
   float[] GREEN_OFF ={0, 0.3f, 0};
   float[] RED_OFF = {0.3f, 0, 0};
   float[] YELLOW_OFF = {0.3f, 0.3f,0};
   float[] SIN12J5VIEW = {-7.67f, 3.5f, 36.2f};
   float[] posicaoAnterior = new float[3];
   int flag = 0;
   private static ArrayList<String> alarmesAtivos;
   X3DScene myMainScene;
   Thread sinoticoThread;
   public void inicializaSinotico(X3DScene mainScene){
      sinoticoThread = new alarmeSinotico();
      myMainScene = mainScene;
      sinoticoThread.start();
      alarmesAtivos= new ArrayList<String>();
   }
   public void adicionaAlarme(String alarme) {
      X3DNode mat2 = myMainScene.getNamedNode(alarme + "Color");
      if (mat2 == null) {
         System.out.println("Couldn't find material named:");
         return;
      } else {
         if (!alarmesAtivos.contains(alarme)) {
            if (alarmesAtivos.isEmpty()) {alarmesAtivos.add(alarme);
            } else {
               float[] corDoObjeto = new float[3];
               X3DNode mat =
myMainScene.getNamedNode(alarmesAtivos.get(alarmesAtivos.size() - 1) + "Color");
               if (mat == null) {
                  System.out.println("Couldn't find material named:");
                  return;
               } else {
                  SFColor color = (SFColor) mat.getField("diffuseColor");
                  SFColor color2 = (SFColor) mat2.getField("diffuseColor");
                           color.getValue(corDoObjeto);
```

```
                if (Arrays.equals(corDoObjeto, GREEN_ON) || Arrays.equals(corDoObjeto,
RED_ON) || Arrays.equals(corDoObjeto, YELLOW_ON)) {
                    if (Arrays.equals(corDoObjeto, GREEN_OFF)) {
                        color2.setValue(GREEN_ON);
                    } else if (Arrays.equals(corDoObjeto, RED_OFF)) {
                        color2.setValue(RED_ON);
                    } else if (Arrays.equals(corDoObjeto, YELLOW_OFF)) {
                        color2.setValue(YELLOW_ON);
                    }
                }
else if (Arrays.equals(corDoObjeto, GREEN_OFF) || Arrays.equals(corDoObjeto,
RED_OFF) || Arrays.equals(corDoObjeto, YELLOW_OFF)) {
                    if (Arrays.equals(corDoObjeto, GREEN_ON)) {
                        color2.setValue(GREEN_OFF);
                    } else if (Arrays.equals(corDoObjeto, RED_ON)) {
                        color2.setValue(RED_OFF);
                    } else if (Arrays.equals(corDoObjeto, YELLOW_ON)) {
                        color2.setValue(YELLOW_OFF);
                    }
                }
            }
            alarmesAtivos.add(alarme);
        }
    } else {
        System.out.println("Alarme já inserido na lista");
    }
  }
}
public void retiraAlarme(String alarme) {
    if (alarmesAtivos.contains(alarme)){
    System.out.println(alarmesAtivos.size());
    float[] corDoObjeto = new float[3];
    X3DNode mat = myMainScene.getNamedNode(alarme + "Color");
    if (mat == null) {
        System.out.println("Couldn't find material named:");
        return;
    } else {
        SFColor color = (SFColor) mat.getField("diffuseColor");
        color.getValue(corDoObjeto);
        if (Arrays.equals(corDoObjeto, GREEN_ON)) {
            color.setValue(GREEN_OFF);
        } else if (Arrays.equals(corDoObjeto, RED_ON)) {
            color.setValue(RED_OFF);
        } else if (Arrays.equals(corDoObjeto, YELLOW_ON)) {
            color.setValue(YELLOW_OFF);
        }
    }
    alarmesAtivos.remove(alarme);
    }else{
        System.out.println("Alarme ainda não foi inserido na lista");
    }
}
mainScene.getNamedNode("myNavigation");
    newView.setBind(true);
    if (dispositivo.equals("S12J5")) {
        if (flag == 0) {
            ((Transform) avatar).getTranslation(posicaoAnterior);
            newNavigation.setType(navigationModeNone);
            newView.setPosition(SIN12J5VIEW);
            flag = 1;
        }
```

```
else if(flag == 1) {
            newNavigation.setType(navigationModeWalk);
            newView.setPosition(posicaoAnterior);
            flag = 0;
        }
    }

}
  class alarmeSinotico extends Thread {

    @Override
    public void run() {
      try {
        do {
          int counter = 0;
          float[] corDoObjeto = new float[3];
          for (counter = 0; counter < alarmesAtivos.size(); counter++) {
            X3DNode mat = myMainScene.getNamedNode(alarmesAtivos.get(counter)
+ "Color");
            if (mat == null) {
              System.out.println("Couldn't find material named:");
              return;
            } else {
              SFColor color = (SFColor) mat.getField("diffuseColor");
              color.getValue(corDoObjeto);
              if (Arrays.equals(corDoObjeto, GREEN_ON)) {
                color.setValue(GREEN_OFF);
              } else if (Arrays.equals(corDoObjeto, GREEN_OFF)) {
                color.setValue(GREEN_ON);
              } else if (Arrays.equals(corDoObjeto, RED_ON)) {
                color.setValue(RED_OFF);
              } else if (Arrays.equals(corDoObjeto, RED_OFF)) {
                color.setValue(RED_ON);
              } else if (Arrays.equals(corDoObjeto, YELLOW_ON)) {
                color.setValue(YELLOW_OFF);
              } else if (Arrays.equals(corDoObjeto, YELLOW_OFF)) {
                color.setValue(YELLOW_ON);
              }
            }
          }

          Thread.sleep(1000);
        } while (true);

      } catch (Exception e) {
        System.out.println("Thread Sinótico interrompida.");
      } System.out.println("teste");
    }
  }
}
```

Código 30 Código da classe conversorsinotico.java

Do ponto de vista da representação visual alguns objetos já estavam presentes no simulador antes do desenvolvimento deste trabalho, são eles o mostrador analógico,

mostrador de sete segmentos, a chave GPG a chave de transferência tipo um, a chave local telecomando, as strings de texto e o armário. Os alarmes, LED com a base preta, sinótico, chave tipo punho, chave de transferência tipo dois e tipo punho, copos, monitor, teclado e cadeira foram desenvolvidos durante este trabalho.

Do ponto de vista comportamental as classes que já estavam implementadas antes do desenvolvimento deste trabalho eram conversorctf.java, conversorloctel.java, parametrizador.java, mostrador.java, motordedisplay.java, motordeponteiro.java, conversorbotao.java e indicadorcampo.java. Já as que foram desenvolvidas no andamento deste trabalho foram conversoralarme.java, conversorprotecao.java e o conversorsinótico.java além de alterações nas classes conversorctf.java e conversorbotao.java.

5.4 EXEMPLO DE COMO INSTANCIAR UM OBJETO

Como exemplo mostraremos como se insere uma chave Local/Telecomando tanto no ambiente X3D quanto o que deve ser feito para inserir essa chave no código java.

Primeiramente inserimos a chave no ambiente X3D para isso faremos uso do código presente na biblioteca mostrado no Código 12 loctel.x3d) e juntamente com o que foi apresentado na seção 5.1.1 sobre o objeto genérico. Verificamos os campos presentes no código da chave Local/Telecomando que podem ser editados, começaremos definindo um nome para a chave fazendo <*Group DEF = "ChaveLocTelExemplo"*>, após definir o nome da chave definiremos o nome que o SAI utilizará como variável para manipular a posição da chave da seguinte maneira <*Transform DEF="CLTExemplo"*>, onde CLTExemplo é o nome da variável, após isso deve ser definidas as posições para localização da chave CLT e definiremos a posição como sendo na origem do sistema de coordenadas e a rotação da seguinte maneira <*Transform DEF = "CLTExemplo" translation = " 0 0 0" rotation = "0 1 0 0"*>. A textura que representa a chave local/telecomando pode ser modificada caso seja interessante, no nosso manteremos a textura utilizada pelo SimuLIHM e por fim definimos um nó sensor para a chave da seguinte maneira <*TouchSensor DEF = "CLTExemplo_Sensor"/*>. Após essas alterações vemos que o código da chave Local/Telecomando será como o exibido no Código 31 .

```
<Group DEF = "ChaveLocTelExemplo">
        <Transform DEF = "CLTExemplo" translation="0 0 0" rotation="0 1 0 0">
                <Shape>
                        <Appearance >
                                <Material/>
                                        <ImageTexture  url="./Imagens/loctel.jpg"/>
                        </Appearance>
                <Box  size="0.27631 0.25739 0.01"/>
                </Shape>
        </Transform>
<TouchSensor DEF = "CLTExemplo_Sensor" />
</Group>
```

Código 31 Chave Loc/Tel exemplo

Após ter inserido a chave local/telecomando no mundo virtual devemos informar para ao código java que esta chave existe, primeiramente devemos ir nas classes jfTutor e jfTreinando no escopo do método *public static ArrayList<String> adicionarChave()* e inserir o nome do nó sensor na lista de sensores existentes, da seguinte maneira *nome.add("CLTExemplo_Sensor");*. Ao termino desse comando deve-se inserir o código que tratará esse evento no momento que este *TouchSensor* for acionado na função *public void tratarEventosSensores(Object touchAtivado, String idUsuario)* , no escopo da função deve ser inserido o seguinte comando:

```
else if (touchAtivado.equals("CLTExemplo_Sensor")) {
        estadoAV = loctel.conversorLoctel(idUsuario, "CLTExemplo", "Painel,LocTel", mainScene, conector, conectorBD);
}
```

Após inseridos estes códigos a chave Local/Telecomando estará pronta para ser utilizada no ambiente virtual.

6 FERRAMENTA DE EDIÇÃO DE MUNDOS VIRTUAIS

A ferramenta de edição de mundos virtuais e cenários de treinamento tem como objetivo facilitar a descrição de um mundo virtual tanto nos aspectos visuais dos objetos ali representados utilizando o X3D, objetivo deste estudo e apresentados neste documento, quanto nos aspectos comportamentais dos objetos a partir da descrição de modelos CPN e construção de código Java.

Essa ferramenta apoiará a modelagem de um novo mundo ou a edição de um mundo existente, a partir de uma interface gráfica, sem a necessidade de que o usuário tenha conhecimento sobre a sintaxe do X3D, Java ou de Redes de Petri Coloridas nem de suas ferramentas de utilização. Ela também poderá ser utilizada por usuários que já tem experiência com essas tecnologias, mas querem acelerar a geração de código.

6.1 NÍVEIS DE ACESSOS À FERRAMENTA

Na ferramenta estarão disponíveis dois níveis de acesso para seus usuários. No nível mais baixo, acessa o projetista, usuário capaz de:

- Inserir novos objetos na biblioteca desde que ele o especifique como mostrado na seção de um objeto genérico e determine quais os parâmetros editáveis, além de fornecer uma classe Java que modele seu comportamento.
- Editando conteúdo de objetos já existentes, gerando objetos derivados daqueles já presentes na biblioteca sem a necessidade de um tratamento comportamental diferente mas apenas com uma modelagem física que difere da já presente na biblioteca.
- Excluindo objetos.
- Renomeando objetos que consiste em apenas mudar o nome caso o projetista ache que um determinado nome seja mais adequado para um determinado objeto.

- Agrupando objetos simples em criando um novo objeto, por exemplo confeccionar um armário completo com todas as chaves e alarmes já incorporados.

O diagrama de caso de uso da Figura 30 mostra as atividades que podem ser realizadas pelo usuário projetista.

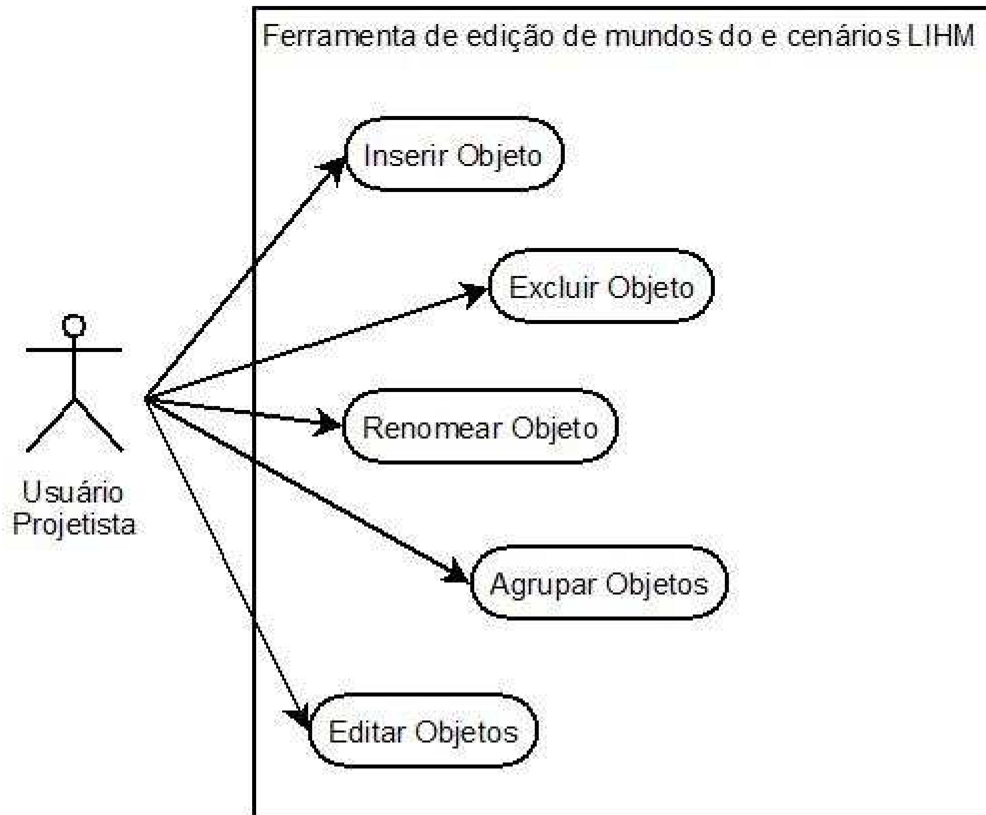


Figura 30 Diagrama de casos de uso do usuário projetista

No nível mais alto, o usuário típico seria o tutor, utilizando a ferramenta para a construção de um novo mundo virtual ou de um novo cenário. Esse usuário não manipula a biblioteca de objetos, ele apenas a utiliza.

As atividades que esse tipo de usuário pode realizar com a ferramenta são:

- A criação de um mundo virtual que consiste na elaboração de um novo ambiente modelado tanto do ponto de vista físico quanto comportamental.
- A escolha de um objeto presente na biblioteca para ser inserido nesse mundo através da inserção de parâmetros simples como posição, rotação, cor dentre outros, sem a necessidade de um conhecimento mais aprofundado sobre x3d ou Java,
- Salvar as alterações realizadas no mundo virtual.

- Programar os objetos escolhendo seu estado inicial, tal como definir: cor, posição, estado de uma chave, dentre outros.

O diagrama de casos de uso do usuário tutor é ilustrado na Figura 31.

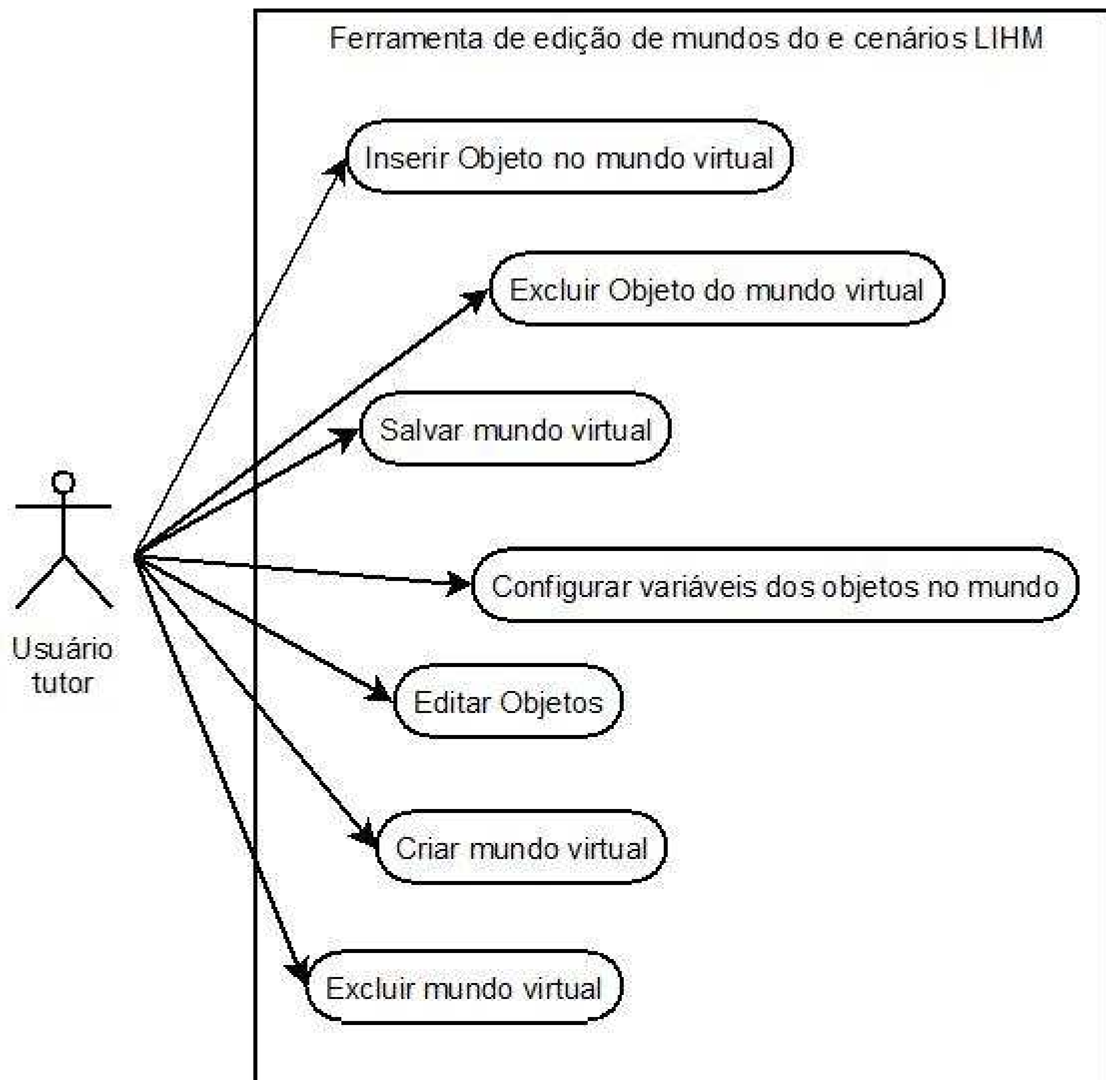


Figura 31 Diagrama de casos de uso do usuário comum

6.2 ARQUITETURA DE CLASSES DA FERRAMENTA

Basicamente quatro classes principais compõem a ferramenta, é possível que ao longo do seu desenvolvimento essas classes sejam substituídas, ou que surjam outras classes que se adéquem as necessidades encontradas pelo programador, mas a função exercida por cada classe não deve deixar de existir. Na Figura 32 encontra-se o diagrama de classes proposto neste documento.

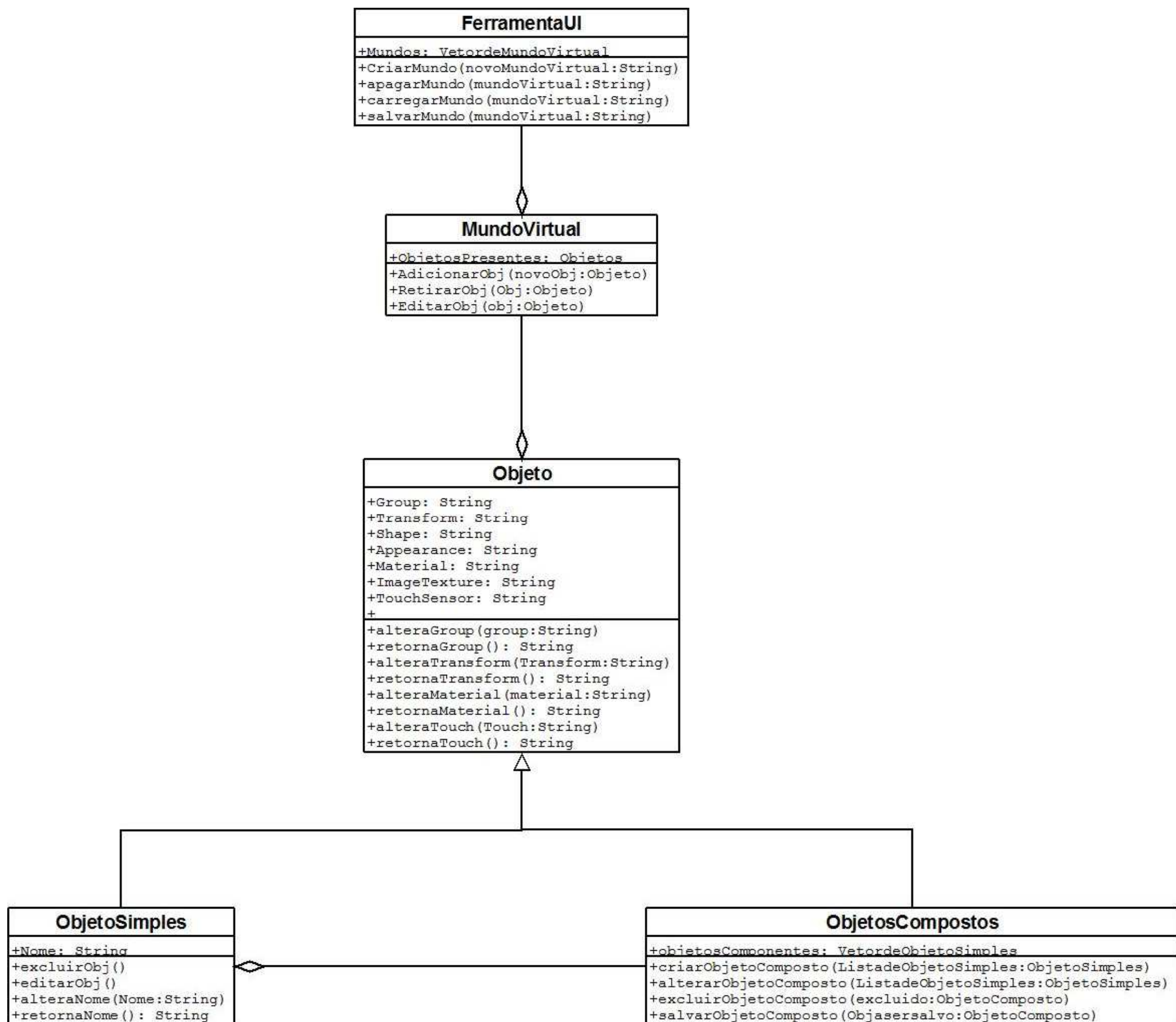


Figura 32 Diagrama de classes proposto

Basicamente o diagrama de classes é composto por cinco classes que modelam o funcionamento da ferramenta de construção de mundos e cenários do SimuLIHM, primeiramente temos a classe FerramentaUI que implementa a interface com o usuário, nessa classe é que o usuário poderá criar um novo mundo ou cenário, salvar um novo mundo que está sendo editado, carregar um mundo para editá-lo, essa classe é composta de uma lista de objetos da classe mundoVirtual.

A classe mundo virtual é formada por uma lista de objetos que compõem o ambiente que está sendo modelado e através desta classe, pode-se inserir objetos no mundo que se está trabalhando, excluir objetos ou editar um objeto configurando um determinado cenário de operação.

Cada objeto possui a estrutura do objeto genérico mostrado na seção 5.1.1 deste documento que descreve um objeto genérico e funções para verificar e alterar os nomes

dados a cada tag de cada objeto. Os objetos podem ser de dois tipos os objetos simples, ou seja, primitivos já modelados na biblioteca e os objetos compostos que são agregações dos objetos simples para compor um objeto mais elaborado, tanto novos objetos simples como objetos compostos podem ser salvos na biblioteca, tanto a classe objetoSimples quanto a classe objetoCompostos herdam da classe objeto e possuem seus atributos.

6.3 INTERFACE DA FERRAMENTA

Um exemplo de como poderia ser implementada a interface da ferramenta é ilustrado na Figura 33. Nela, os objetos presentes na biblioteca estão listados no canto esquerdo da tela ao lado da área de trabalho. Nesta área, será representado o mundo que está sendo editado. Para inserir um objeto no mundo virtual deve-se selecioná-lo e arrastar a tag correspondente a este objeto para a área de trabalho, como ilustrado na Figura 33.

Após inserir o objeto no mundo virtual deve ser possível alterar suas características, a partir de um duplo clique sobre o objeto , que provocará a exibição de uma lista com os atributos do objeto e seu estado atual ou default, a qual poderá ser editada pelo usuário.

A aba “Biblioteca” ao lado da aba “Editar” no canto superior esquerdo devem ser utilizadas pelos usuários projetistas, para selecionar uma das opções possíveis, tais como: inserir novo objeto na biblioteca, agrupar objetos e todas as demais funcionalidades previstas para a edição com a ferramenta.

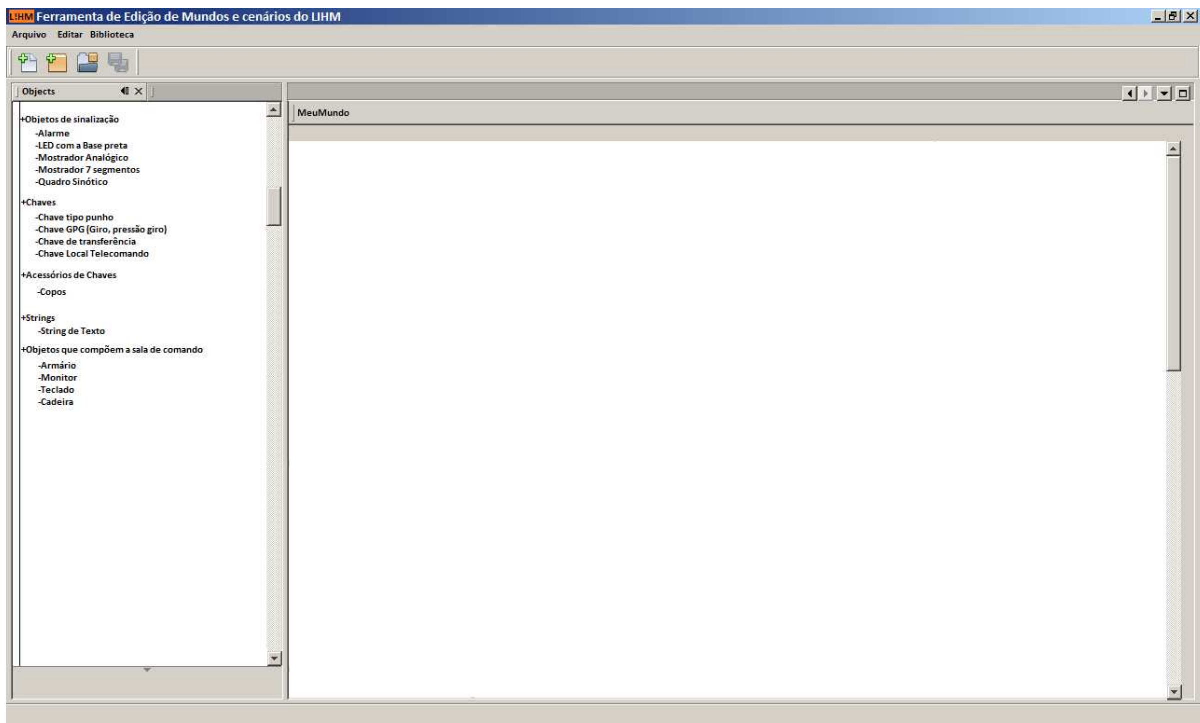


Figura 33 Interface da ferramenta de desenvolvimento de mundos do LIHM

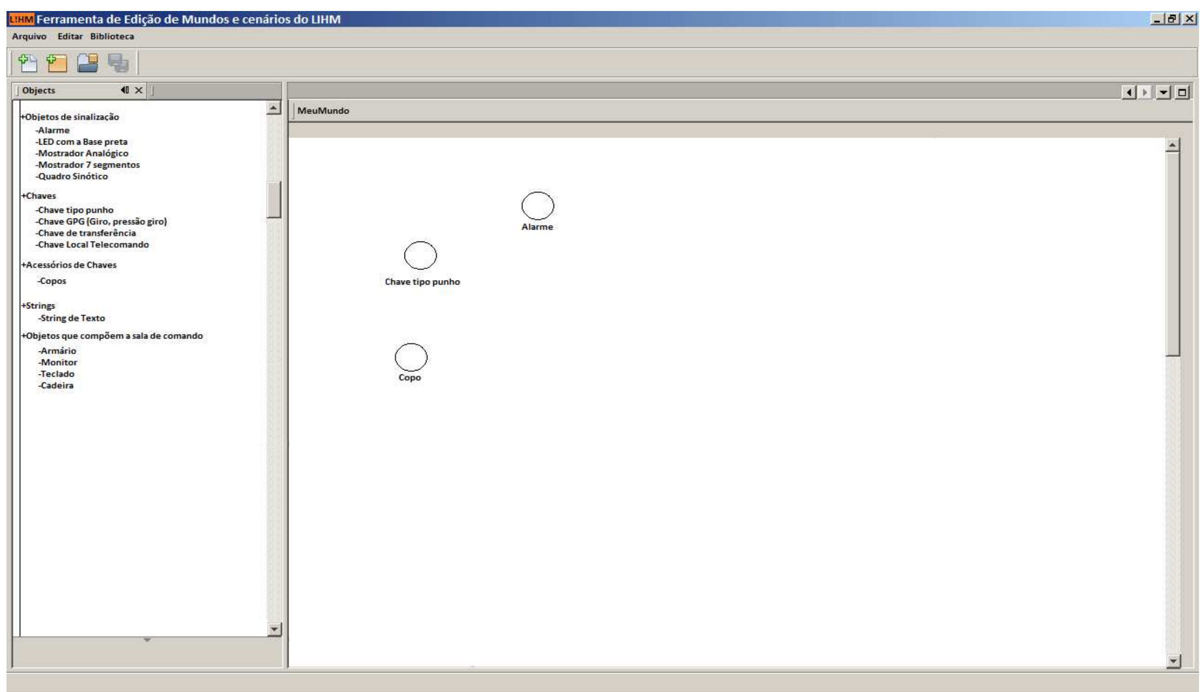


Figura 34 Objetos sendo inseridos em um mundo

É recomendável que a implementação seja realizada utilizando a linguagem de programação Java. Além de ser a linguagem utilizada pelo SimuLIHM existem vantagens de certas API's e do SAI que facilitariam o desenvolvimento da ferramenta. É interessante que o responsável pela implementação da ferramenta se familiarize com as tags utilizadas nos arquivos x3d.

Está em elaboração no LIHM um documento que define os valores padrões dos objetos que estão presentes na biblioteca e o que é necessário para que a ferramenta se torne “inteligente” a ponto de conseguir montar um objeto não só na camada de visualização, como está proposto neste trabalho, mas nos níveis de comportamento e animação (Java,CPN).

7 CONCLUSÃO

Neste documento foi abordado o desenvolvimento e a organização da biblioteca de objetos do mundo virtual do SimuLIHM. A descrição consiste de extratos de código e comentários.

Neste trabalho foi também especificada uma ferramenta de edição para facilitar o uso da biblioteca apresentada; permitindo ao usuário criar mundos virtuais e cenários em X3D de subestações elétricas sem a necessidade de conhecimento desta linguagem.

Tanto a Biblioteca quanto a ferramenta podem evoluir e abranger mais objetos a partir do surgimento de novos trabalhos. Como trabalhos futuros podemos ter a implementação da ferramenta especificada neste trabalho além da elaboração de novos objetos para compor a biblioteca.

REFERÊNCIAS

PASQUALOTTI, A., **Introdução a Realidade Virtual.** Disponível em: http://usuarios.upf.br/~pasqualotti/ccc053/intr_rv/Acessado em 30 de Out. 2011.

BRUTZMAN, D; DALY, L. X3D: **Extensible 3D Graphics for Web Authors**, editor Morgan Kaufmann, 2007, p 425.

CASSANDRAS, G. C; LAFORTUNE, S. **Introduction to Discrete Event System**, editor Springer 2 Edição, 2008, p 771.

BARROSO,G. C; SAMPAIO,R. F; SOARES, J. M; BEZERRA, H; LEÃO,R. P. S; MEDEIROS, E. B. Medeiros. **Desenvolvimento de Sistema de Treinamento em Proteção de Sistemas Elétricos em Ambiente Virtual de Aprendizagem**.2008, P 9.

JENSEN, K; KRISTENSEN L.M. **Coloured Petri Nets Modeling and Validation of Concurrent Systems**, Springer, 2009, P. 381.

MACEDO, R.R. **Aplicativo para treinamento dos operadores do sistema**. Plena Transmissoras, 2010, P 12.

Documento sobre X3D. Disponível em **http://www.web3d.org/x3d/content/README.X3D-Edit.html**. Acesso em 30 de Outubro de 2011.

Documento sobre o Notepad++. Disponível em **http://notepad-plus-plus.org/**. Acesso em 30 de Outubro de 2011.

Documento sobre o Octaga Layer. Disponível em **http://www.softpedia.com/get/Multimedia/Graphic/Graphic-Others/Octaga-Player.shtml**. Acesso em 30 de Outubro de 2011.

Documento sobre o Vivaty Studio Player.]Disponível em: **http://mediamachines.com/downloads.php**. Acesso em 30 de Outubro de 2011.

Documento sobre o Xj3D. Disponível em **http://www.xj3d.org/**. Acesso em 30 de Outubro de 2011.

Documento sobre o NetBeans. Disponível em **http://netbeans.org/**. Acesso em 30 de Outubro de 2011.

Documento sobre o CPN Tools. Disponível em **http://cpntools.org/**. Acesso em 30 de Outubro de 2011.

Documento sobre o Java. Disponível em **http://java.com/pt_BR/**. Acesso em 30 de Outubro de 2011