Implementação de Simuladores de Robôs com o Uso da Tecnologia de Realidade Virtual

R. Redel, *DCC-CCT-UDESC* [1] e M. da S. Hounsell, *DCC-CCT-UDESC* [2]

***Resumo*—Com a popularização da Realidade Virtual (RV) e a crescente demanda por Robôs Manipuladores vê-se possibilidade da junção destas duas tecnologias. Apresenta-se neste artigo os passos sugeridos para implementar um simulador de robô utilizando tecnologia da Realidade Virtual (RV), mais especificamente Realidade Virtual Não-Imersiva (RVNI) via Internet. Um estudo de caso será apresentado com a construção de um simulador para o robô Scorbot ER-4PC, usando tecnologia VRML (Virtual Reality Modeling Language).**

***Palavras chaves* — Robótica, Robôs Manipuladores, Realidade Virtual, Simulador, Sistemas Não-Imersivos e Imersivos, VRML.**

I. INTRODUÇÃO

Nos últimos anos a tecnologia da Realidade Virtual (RV) vem ampliando seu papel na área industrial, atuando desde a apresentação personalizada de produtos até a participação em projetos que possuem alto valor agregado. Uma das áreas industriais em que a aplicação da Realidade Virtual desperta interesse refere-se à simulação de robôs manipuladores. Uma vantagem fundamental destes simuladores nas aplicações industriais é a sua utilização para o desenvolvimento da programação off-line. Utilizando estas ferramentas não seria necessário interromper o trabalho dos robôs reais na célula de trabalho, geralmente integradas em uma linha de produção. Permite também diminuir o tempo de desenvolvimento dos produtos, através da realização das tarefas de desenvolvimento dos programas para os equipamentos das células em paralelo com outras atividades de projeto. Este artigo abrangerá os conceitos básicos de robótica e de RV com a finalidade de demonstrar como esta última (RV) pode ser útil para implementar um simulador de robô, simulador este com propósitos didáticos. Devido às limitações inerentes da tecnologia de RV Imersiva (RVI), este artigo está limitado ao escopo da RV Não-Imersiva (RVNI), que será também apresentada.

II. ROBÓTICA

Automação industrial e a robótica são duas tecnologias intimamente relacionadas. A robótica é considerada uma forma de automação industrial que utiliza tecnologia de robôs na produção e controle do chão-de-fábrica. Uma definição formal seria: "A robótica é uma ciência da engenharia aplicada que é tida como uma combinação da tecnologia de máquinas operatrizes e ciência da computação" [11]. Assim, o principal instrumento utilizado na robótica é o robô.

A. Robô

Uma definição de robô é dada pela instituição americana "*Robotics Industries Association*" (RIA) anteriormente designada por "*Robotics Institute of America*" (RIA) da seguinte maneira: "Um robô é um manipulador reprogramável multifuncional projetado para manusear materiais, peças, ferramentas ou dispositivos especiais, através de movimentos programados para a realização de uma variedade de tarefas" [8]. Para que um robô possa atender às necessidades da automação industrial, dois aspectos são imprescindíveis: movimentação e programação.

1) Movimentação do Robô

Os movimentos de um robô manipulador são o resultado de uma série de movimentos elementares, independentes entre si, denominados graus de liberdade (DOF – *degrees of freedom*) de um robô [9]. Os membros de um robô são reunidos em cadeia através de ligações ou juntas permitindo um movimento relativo entre eles [3]. Quanto maior o número de juntas que um robô possuir, maior o número de DOF. Para ser possível uma movimentação adequada dos robôs, são necessários no mínimo três DOF's [9].

2) Programação de Robôs

A programação de tarefas consiste em processar uma série de pontos no espaço, definindo assim a sua trajetória. Atualmente, pode-se fazer muito mais do que isto. Os robôs de tecnologia corrente podem aceitar informações de sensores e de outros dispositivos. Eles podem enviar sinais para equipamentos que operam com eles dentro da célula de trabalho [11]. Além disto, contém estruturas de decisões próprias, comunicam-se com centrais (computadores) para obter instruções e enviar informações e problemas correntes. Para possibilitar o tratamento destas funcionalidades é necessária a programação de robôs. Existem dois métodos de programação de tarefas dos robôs industriais: diretos, que correspondem à programação por aprendizagem; e indiretos, que correspondem à utilização de uma linguagem, também chamada de "programação off-line" [3].

[1] R. Redel: Departamento de Ciência da Computação, Centro de Ciências Tecnológicas, Universidade do Estado de Santa Catarina, Brasil (e-mail: dcc6rr@joinville.udesc.br).

[2] M. da S. Hounsell: Departamento de Ciência da Computação, Centro de Ciências Tecnológicas, Universidade do Estado de Santa Catarina, Brasil (e-mail: marcelo@joinville.udesc.br).

B. Programação Off-Line

Programação *off-line* consiste em elaborar todo o programa em um computador para carregamento posterior do robô, diminuindo seu tempo de ociosidade deste último. Uma forma apropriada de programar um robô de maneira *off-line* é utilizar linguagens textuais. Na prática, muitos dados relacionados as posições e orientações do robô são adquiridas *on-line*, contrariando o conceito de programação *off-line*, através de um instrumento chamado *teach-pendant*. Este instrumento bem parecido com um controle remoto com alavancas e botões movimenta o robô a uma determinada posição e captura suas coordenadas e ângulos das juntas [13]. Uma das soluções para esta dicotomia envolve utilizar um simulador de robôs a fim de capturar os dados necessários. A programação textual *off-line* tem vantagens como [3]:

1. Permite a utilização de posições determinadas analiticamente.
2. Permite utilizar dados provenientes de sistemas CAD-CAM.
3. Permite capturar informações provenientes de sensores.
4. Facilita a revisão de programas, permitindo (pequenas) alterações. Normalmente, em programação *on-line*, deve-se ensinar novamente toda a tarefa.
5. O robô pode manter-se produtivo durante a programação *off-line*.
6. A tarefa ensinada pode ser gravada em formato de arquivo e posteriormente otimizada.
7. O programa pode ser parametrizado ou repetido vários vezes com pequenas mudanças de uma iteração para outra.
8. Permite a verificação de eventuais erros (colisões com os objetos do ambiente).

Para implementar um simulador de robôs pode-se recorrer a conceitos de computação gráfica, mais especificamente, Realidade Virtual (RV).

III. Realidade Virtual (RV)

Em termos conceituais, Realidade Virtual (RV) é uma realidade que é aceita como verdadeira, embora não necessariamente exista fisicamente [7]. Em termos computacionais, a RV é considerada a forma mais avançada de interface com o computador. Permite ao usuário "realizar a imersão, navegação e interação em um ambiente sintético tridimensional (3D) gerado por computador, utilizando canais multi-sensoriais" [1]. Uma interface de RV possibilita ao usuário navegar e observar um mundo 3D, manipulando e explorando os dados da aplicação em tempo real. Deste modo, o usuário sente que faz parte do ambiente que o envolve, gerando o sentimento de imersão. Sistemas de RV são divididos em dois tipos principais: **Realidade Virtual Imersiva** (RVI) e **Realidade Virtual Não Imersiva** (RVNI) [6]. A diferença entre as duas não é realmente relatada na habilidade de promover a sensação de imersão, mas o uso de dispositivos que 'escondem'o mundo real do usuário ou não.

A. Realidade Virtual Imersiva (RVI) e Não-Imersiva (RVNI)

A RVI implica no uso de dispositivos de visualização que "escondem" o mundo real do usuário, apresentado-o a um mundo sintético tridimensional. É necessário nestes tipos de sistemas, hardware muito potente para produzir ambientes mais velozes e com melhores gráficos, para garantir a sensação de imersão no ambiente.

Por outro lado, a RVNI não necessita de hardware específico, pois os usuários podem utilizar simplesmente monitores, CPU's e mouses convencionais para acessar o mundo sintético. Em muitos casos, a RV só precisa **parecer** precisa, não necessariamente **ser** precisa [5].

Casos especiais de sistemas RVNI são aqueles que podem ser explorados através da internet através de uma linguagem chamada VRML (*Virtual Reality Modeling Language*). A RVNI, quando baseada na web, possui grandes vantagens sobre a RVI, pois a RVNI é muito mais simples e barata de se implantar num *site*.

B. Virtual Reality Modeling Language (VRML)

VRML (*Virtual Reality Modeling Language*) é um formato de arquivo para descrever objetos e mundos 3D interativos e foi projetado para ser veiculada via Internet, Intranet e sistemas locais em uma variedade de aplicações [15]. O VRML é um padrão (ISO 14772-1:1997) que descreve quais e como os objetos são representados. Os objetos podem ser representados através de geometrias primitivas, transformações hierárquicas, fontes de luz, pontos de visão, animações, mapeamentos de texturas, etc. O VRML é uma linguagem que colabora substancialmente para a RV atuar na internet. Ela permite a criação de mundos virtuais a partir de arquivos ".wrl" escritos em código ASCII com a possibilidade de interface com outras linguagens tais como Java, Delphi, C++, etc [14]. A fim de visualizar e interagir com ambientes e objetos virtuais, um *plug-in* necessita ser instalado. O *plug-in*, como extensão ao navegador WWW, será ativado toda vez que acessar arquivos com a extensão wrl na Internet.

A vantagem primária do VRML refere-se a independência da plataforma operacional, permitindo sua visualização em diversos sistemas operacionais desde que compatíveis com o padrão da ISO. Tratando-se de um código aberto, traz benefícios para a área educacional. Os estudantes têm livre acesso ao código fonte, podendo até mesmo modificá-lo a fim de adequar às novas situações e necessidades [10].

Uma funcionalidade bastante interessante, propiciada pelo VRML, é a sua integração com aplicações externas. Para padronizar esta integração criou-se a especificação External Application Interface (EAI) através da documentação ISO/IEC 14772-2:2004 [15]. Trata-se de uma interface projetada para que outros aplicativos acessem objetos e suas propriedades na cena VRML. A EAI possibilita a integração VRML com Applets o que combina todas as potencialidades da linguagem Java com a visualização VRML. Applet's são um tipo especial de programa Java que um navegador é capaz de executar descarregando o mesmo da Internet [2]. Para executar Applet's também é necessário uso de um *plug-in* incorporado ao *browser* (navegador da Internet).

IV. IMPLEMENTAÇÃO

A seguir são apresentados os passos necessários para implementar um simulador de robô (no caso ER4PC) usando a tecnologia RVNI (no caso VRML). Os passos, se seguidos corretamente, servirão para implementar qualquer simulador de robô manipulador em VRML. Os passos, conforme demonstrados na Fig. 2 abaixo, necessários são: descrição da geometria, especificação da estrutura e programação da interação.

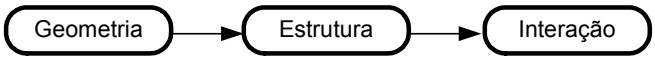

Fig 1. Passos para Implementação de um Robô Virtual

A. O Robô Scorbot ER-4PC (ER4PC)

O robô utilizado para implementar virtualmente foi o robô manipulador Scorbot ER-4PC (ER4PC) da empresa Eshed Robotec [4]. Este robô é um exemplo de robô didático, feito especialmente para o estudo da robótica. Trata-se de um robô com estrutura vertical articulada permitindo cinco DOF para manipulação (Fig. 2). A área de trabalho possui 610mm como raio máximo de alcance com 11,5 kg chegando a uma velocidade máxima de 600mm/sec.

Fig 2. Robô Scorbot ER-4PC (ER4PC)

B. Geometria

O primeiro passo para construir um robô em VRML refere-se a modelagem das suas partes individuais. Em VRML, as geometrias são definidas dentro do nó *Shape* com a propriedade *geometry*. Através desta propriedade é possível modelar a partir de geometrias primitivas (cubo, esfera, cilindro, etc.). Os componentes do robô ER4PC não possuem formatos regulares, o que impossibilita de utilizar formas primitivas. A fim de modelar geometrias complexas utiliza-se o nó *IndexedFaceSet*. A partir de uma lista de vértices o *IndexedFaceSet* constrói faces que unidas representam um objeto em 3D [12]. Na Fig. 3 mostra-se um exemplo do *IndexedFaceSet* representando uma pirâmide. A regra para o sistema de coordenadas é o da mão direita, conforme visualizado na direção dos eixos. O caracter # antecede um comentário que é ignorado pelo visualizador de VRML.

A propriedade *coord* contém um nó do tipo *Coordinate* que armazena um conjunto de coordenadas 3D representando os vértices do objeto. O primeiro vértice possui índice zero, o segundo possui índice um e assim sucessivamente. A propriedade *coordIndex* armazena índices com a qual é possível especificar as faces do polígono. A seqüência de vértices finalizada por –1 representa uma face. A Fig. 3 demonstra o código onde tem-se cinco faces representando as paredes e a base da pirâmide.

```
#VRML V2.0 utf8

Shape {
  geometry IndexedFaceSet  {
    coord Coordinate {
      point [
          1 0 -1, #vértice 0
         -1 0 -1, #vértice 1
         -1 0  1, #vértice 2
          1 0  1, #vértice 3
          0 2  0  #vértice 4
      ]
    }
    coordIndex [
      0 4 3 -1 #face direita
      1 4 0 -1 #face de trás
      2 4 1 -1 #face esquerda
      3 4 2 -1 #face frontal
      0 3 2 1 -1 #face inferior
    ]
  }
}
```

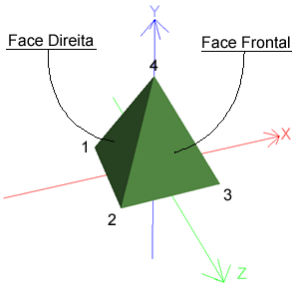

Fig. 3. Implementação da Pirâmide com *IndexedFaceSet.*

A Fig. 4 mostra o resultado do modelamento da base e do antebraço do robô ER4PC. Como as correias, engrenagens e fios do robô são geometrias complexas e não agregam cinemática de movimento, não foram modeladas.

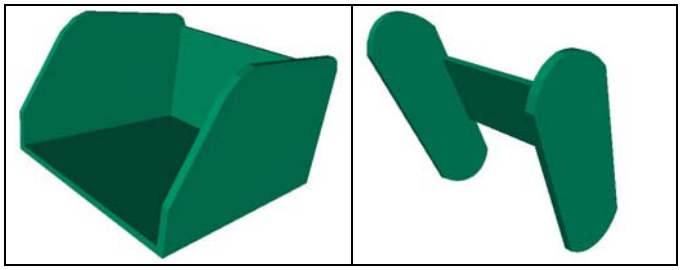

Fig. 4. Modelagem geométrica da base e do antebraço, respectivamente.

C. Estrutura

A estruturação do robô envolve posicionar os elementos geométricos da etapa anterior de maneira que mova seus componentes hierarquicamente. Por exemplo, ao rotacionar a base do robô ER4PC, deve-se rotacionar todas as estruturas ligadas à base (antebraço, braço, punho e garra). No entanto, ao movimentar o antebraço, a base não deverá sofrer alteração, mas sim todos as estruturas inferiores, a saber, braço, punho e garra.

A fim de possibilitar a hierarquia dentro do VRML, a ISO definiu o nó *Transform*. *Transform* é um nó de agrupamento que define um sistema de coordenadas para seus filhos relativo ao sistema de coordenadas de seus superiores [12]. Na prática, o nó *Transform* possibilita ao robô virtual, mover as suas juntas de forma que seus membros conectados possam sofrer as devidas alterações de posição e rotação. Para fazer isso, o nó *Transform* possui propriedades específicas: *rotation* (rotaciona o objeto), *translation* (move o objeto dentro de um sistema de coordenada), *scale* (redimensiona o objeto) e, por

fim, *children* (objetos imediatamente abaixo na hierarquia – filhos). Esta última propriedade permite adicionar geometrias como o nó *Shape*, descrito acima, representando os demais membros do robô. Todo e qualquer objeto acrescentado como filho do nó *Transform*, são afetados pelas propriedades do nó pai, a saber, rotação, translação e escala. Isto é a hierarquia.

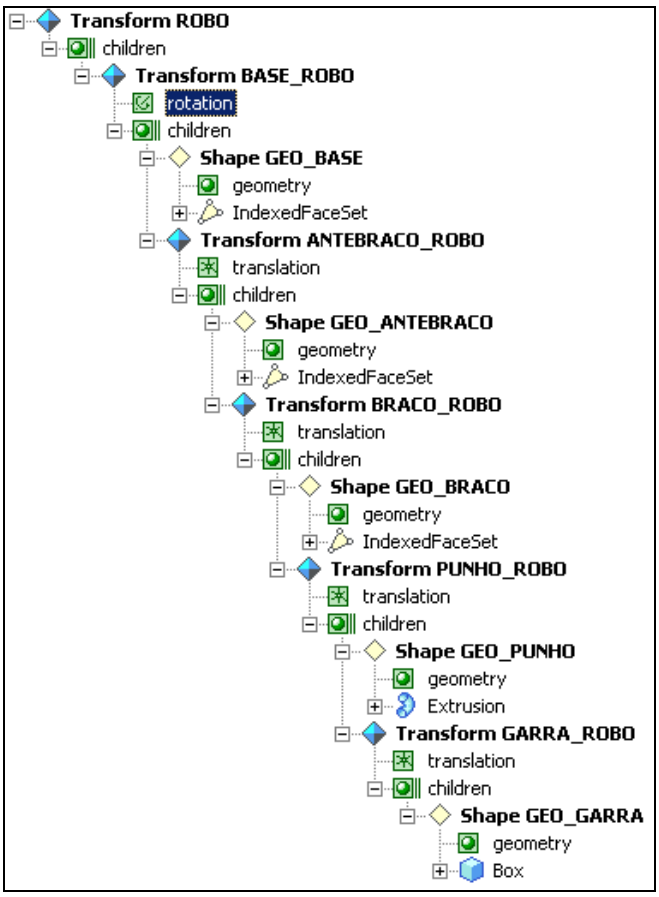

Fig. 5. Hierarquia do Robô Virtual ER4PC

A Fig. 5 demonstra a árvore de hierarquia do robô virtual ER4PC. Vê-se que foi definido um agrupamento chamado ROBO possuindo um sistema de coordenadas local (sem nenhuma alteração de *translation*, *rotation* ou *scale*). Este aglomera filhos que são a estrutura da base (GEO_BASE) e um agrupamento chamado ANTEBRACO_ROBO. Este novo agrupamento, e sucessivamente daí em diante, terá também seu próprio sistema de coordenadas local (desta vez com uma adaptação de translação, pois o antebraço se localiza ligeiramente acima da base), sua própria geometria (GEO_ANTEBRACO) e uma sub-hierarquia dos demais membros do ROBO. Esta estruturação é assim para todos os robôs manipuladores de 5 DOF, mas pode-se acrescentar/diminuir níveis conforme se aumenta/diminua o número de DOF's.

D. Interação

Até o presente momento, a estrutura e a hierarquia do robô estão definidos. No entanto, está faltando uma peça fundamental, a interação do robô por parte do usuário. A linguagem VRML permite que a interatividade de objetos por parte do usuário seja satisfeita através de "sensores" e "rotas".

Para tornar possível a interação, a linguagem VRML utiliza o modelo de eventos de saída e de entrada. A conexão entre o nó que gera um evento e o nó que o recebe é chamada de *rota*. Um evento de um nó pode ser roteado para um evento do mesmo tipo de outro nó de acordo com sintaxe mostrada na Fig. 6.

```
ROUTE NodeName1.eventOutName_changed TO
      NodeName2.set_eventInName
```

Fig. 6. Sintaxe para Rotas em VRML

Onde *NodeName1* é o nome do nó que envia o evento, *eventOutName* é o nome do evento de saída, *NodeName2* é o nome do nó que recebe o evento e *eventInName* é o nome do evento de entrada. Quando um evento é roteado para outro, eles não precisam ter o mesmo nome, mas têm que ser, necessariamente, do mesmo tipo de dados.

Os eventos são, geralmente, acionados por sensores. Sensores são nós que têm a capacidade de gerar eventos respondendo à ações do usuário (*ProximitySensor*, *VisibilitySensor*, *TouchSensor*, *CylinderSensor*, *PlaneSensor*, *SphereSensor* e o nó de agrupamento *Collision*) ou ao passar do tempo (*TimeSensor*).

Nós sensores são nós que esperam um evento para realizar alguma ação em resposta. Um destes nós que se utilizou para implementar o ER4PC foi o *CylinderSensor*. Trata-se de um nó abstrato, não visível ao usuário, que cria na sua área de atuação um cilindro invisível. Através da movimentação do dispositivo apontador (*mouse*), este sensor possibilita que todos os componentes que estão definidos na sua área de atuação rotacionem no eixo y, por exemplo. Uma grande utilidade do *CylinderSensor* é o fato de este definir limites mínimos e máximos do ângulos de rotação, em radianos, conforme mostra a Fig. 7.

```
CylinderSensor
{
  minAngle -2.6179 #ângulo mínimo = -150°
  maxAngle 2.6179 #ângulo máximo = 150°
}
```

Fig. 7. Sintaxe do nó *CylinderSensor*

Uma vez que um sensor ou um script tenha gerado um evento inicial, o evento é propagado através de quaisquer rotas para outros nós. Estes nós podem responder gerando eventos adicionais, e assim sucessivamente. Este processo é chamado de *cascata de eventos*. Todos os eventos gerados durante uma cascata de eventos tem o mesmo *timestamp* (tempo de execução) que o evento inicial (considera-se que todos ocorrem instantaneamente).

As animações em VRML consistem em enviar eventos seguindo uma seqüência de passos, embora a maior parte das animações possa omitir um ou mais passos. A Fig. 8 mostra um diagrama do fluxo de eventos completo em uma animação em VRML.

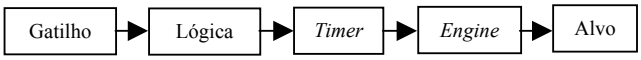

Fig. 8. Diagrama do Fluxo de Eventos

Os passos ilustrados na Fig. 8 têm as funções descritas a seguir:

Gatilho/Trigger: Algo que dá inicio à animação. Geralmente, um gatilho é um sensor que envia um evento quando o usuário executa uma ação em particular;

Lógica: Este passo é geralmente desnecessário para animações simples. Ele envolve a realização de algum processamento sobre o evento enviado pelo gatilho antes de iniciar o *Timer*. Para iniciar o estágio de lógica, o *browser* passa o evento enviado pelo gatilho para um nó *Script*, e chama a rotina apropriada dento do programa especificado naquele nó. O *Script* processa o evento e, geralmente, envia um evento de saída para iniciar o *Timer*;

Timer (Relógio): O *browser* repassa os eventos de saída do gatilho ou do estágio de lógica para o campo *starTime* de um nó *TimeSensor*. Este sensor então gera eventos de tempo por um período determinado ou continuamente até que seja explicitamente interrompido;

Engine (*Máquina de execução*): O *browser* envia os eventos de saída do sensor de tempo para um nó ou conjunto de nós que determinam os parâmetros da animação naquele dado momento e geram mais eventos de acordo com esses parâmetros. Um *Engine*, consiste, tipicamente, de um nó interpolador, mas pode incluir um nó *Script* para realizar qualquer tipo de tarefa que seja necessária;

Alvo: O *browser* envia a saída da máquina de execução para um nó relevante na hierarquia da cena, cujos campos possam ser modificados de acordo com os valores da saída produzida, alterando o seu estado.

O esquema demonstrado na Fig. 8 refere-se ao funcionamento interno do processamento dos movimentos. Por parte do usuário, apenas os sensores, a hierarquia e os roteamentos precisam ser devidamente definidos para a geração de simulador de robô.

Observe que devido ao *CylinderSensor*, ao clicar na área de atuação e arrastar com o *mouse*, um evento de saída chamado *rotation_changed* é gerado. Trata-se de um evento de saída que determina a quantidade de rotação que o objeto receptor terá. O objeto receptor neste caso será o nó *Transform* definido por CONJ_BASE. Assim, o evento do sensor alterará diretamente a propriedade *rotation*, através do identificador *set_rotation* que é o evento de entrada.

A quantidade de sensores utilizados para implementar um robô virtual depende do número de DOF do mesmo. No caso do robô ER4PC, por ser um robô de cinco DOF, foram necessários cinco sensores de rotação, um para cada junta do robô: base, antebraço, braço, punho e garra. Como cada junta possui limites de movimentação, as propriedades *maxAngle* e *minAngle* do *CylinderSensor* foram úteis determinando o ângulo máximo e o ângulo mínimo respectivamente.

A Fig. 9 demonstra o código enxuto da implementação da base e do antebraço do ER4PC, com os sensores e rotas necessárias exemplificados. Observa-se que para a junta da base tem-se o *CylinderSensor* denominado ROTACAO_BASE com seus limites definidos, sua geometria, denominada GEO_BASE, e os roteamentos necessários. Os detalhes da geometria e topologia dos membros foram omitidos (identificado por **[...]**) por questões de espaço bem como estes mesmos detalhes para os demais membros. Os nós destacados em negrito são alguns elementos do VRML explicados neste artigo.

```
#VRML V2.0 utf8

DEF ROBO Transform {
  translation 0 -0.45 0
  children [
    DEF CONJ_BASE Transform {
     children [
        DEF ROTACAO_BASE CylinderSensor
        {
          minAngle -2.6179
          maxAngle 2.6179
        }
        Transform {
          children [
            DEF GEO_BASE Shape {
              geometry IndexedFaceSet {
                coord Coordinate {
                  point [...]
                }
                coordIndex [...]
              }
            }
            DEF CONJ_ANTEBRACO Transform {
            children [
                DEF ROTACAO_ANTEBRACO
                  CylinderSensor
                {
                  minAngle -0.6108
                  maxAngle 2.2689
                }
                Transform {
                  children [
                    DEF GEO_ANTEBRACO Shape {
                      geometry IndexedFaceSet {
                        coord Coordinate {
                          point [...]
                        }
                        coordIndex [...]
                      }
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
ROUTE ROTACAO_BASE.rotation_changed TO CONJ_BASE.rotation
ROUTE ROTACAO_ANTEBRACO.rotation_changed TO
  CONJ_ANTEBRACO.rotation
```
Fig. 9. Implementação da Base e do Antebraço do ER4PC

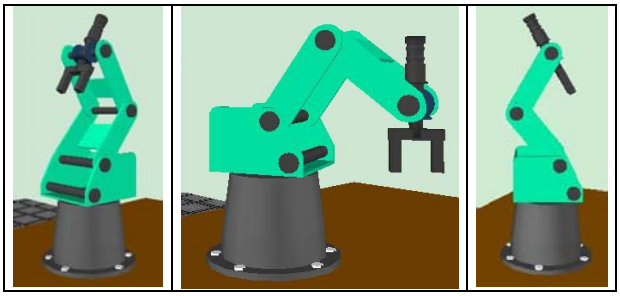
Fig. 10. Robô ER4PC Virtual

A Fig. 10 mostra o robô virtual ER4PC implementado com tecnologia RVNI, utilizando os recursos e conceitos aqui detalhados.

V. DISCUSSÃO

Utilizar o VRML como recurso para construção de um simples simulador de robô se mostra uma alternativa interessante sob vários aspectos:

Baixo Custo. Tanto o *plug-in* no *browser* quanto um editor ASCII para gerar o arquivo com o código VRML, são ferramentas gratuitas. A relação custo-benefício é bastante favorável se comparada com vários outros simuladores.

Código Aberto. O arquivo .wrl resultante traz toda a descrição de goemetria, topologia, estrutura, sensores, etc. que pode ser melhorada, estendida ou alterada conforme a necessidade do usuário. Pode inclusive ser alvo de uma tarefa de um curso de robótica com finalidade de alterar a estrutura para contemplar outros tipos de robôs manipuladores com outras geometrias e/ou número de DOF's diferentes.

Produtividade. Comparado a um conjunto de ferramentas como C++, Delphi, Java3D, ou outra, o esforço na programação é bastante minimizado se incluído o resultado visual obtido como critério.

Acesso Facilitado. Por ser uma ferramenta para Web, pode-se inclusive disponibilizar o simulador em uma página Web, deixando então disponível a qualquer pessoa que interesse.

Uma desvantagem é que para altas performances visuais é necessário um bom computador e ter uma boa qualidade do *rendering*.

O ambiente ainda prevê, para futuros trabalhos, aspectos de programação *off-line*, ou seja, o simulador poderá ter um compilador que lerá um arquivo texto gerado pelo simulador e gerará um código que o robô real interpretará e realizará as operações programadas e testadas pelo simulador, ou seja, o usuário irá fazer uma certa instrução e simulará os movimentos do robô real através do robô virtual e, no momento que se tem certeza da segurança da operação, ele enviará a instrução para o robô real realizar.

Outras funcionalidades que poderão ser implementadas no futuro são a criação de objetos que serão manipulados pelo robô, a disponibilidade do usuário trocar de garra e até mesmo de robô, ou seja, criar uma biblioteca de juntas, tanto rotacionais quanto translacionais e garras, deixando assim a possibilidade de criação de robôs específicos pelos usuários possibilitando um estudo específico dentro das questões que envolvem a robótica, como a cinemática e dinâmica dos robôs.

VI. CONCLUSÃO

Este artigo mostrou a grande importância que a robótica vem adquirindo na automação industrial e também como a Realidade Virtual pode ser usada para simular um robô manipulador. Passo a passo foi formalizado oe descrito como desenvolver um simulador de robô usando-se recursos da linguagem VRML (Virtual Reality Modeling Language), que é uma linguagem destinada ao desenvolvimento de aplicações de RVNI (Realidade Virtual Não Imersiva). Da forma como descrita e devido ao resultado simples e aberto do código esta abordagem torna-se um bom recurso educacional para o ensino de manipuladores. O ambiente gerado é atrativo (tanto visual quanto funcionalmente), iterativo e de fácil utilização e extensão.

VII. REFERENCES

[1] A. V. Netto, L. dos S. Machado e M. C. F. de Oliveira, *Realidade Virtual: Fundamentos e Aplicações*, Ed. Visual Books, 2002.

[2] C. S. Horstmann e G. Cornell. *Core Java 2: Fundamentos*, São Paulo: Makron Books, 2001.

[3] E. de P. Ferreira, *Robótica Básica,* Rio de Janeiro: [s.n.], 1991.

[4] E. Robotec, "Scorbot ER-4PC: User's Manual", Rosh Ha'ayin: Israel, 1982.

[5] G. A. Francis, H.S Tan. "Virtual Reality as a Training Instrument". *The Temasek Journal*, Vol. 7. pp. 4 –15, 1999

[6] G. Taxen, A. Naeve, "A System for Explorating Open Issues in VR-based Education". Computer and Graphics, 26, pp. 593-598. 2002.

[7] J. Vince, *Essential Virtual Reality Fast: How to Understand the Techniques and Potential of Virtual Reality*, Berlin: Springer, 1998, pp. 1-26.

[8] L.-W. Tsai, Robot analysis: the mechanics of serial and parallel manipulators, 1 ed. New York: John Wiley & Sons, 1999.

[9] M. Armada, P. González e M. A. Jiménez, "Arquitectura Mecánica y Modelos Cinemáticos", in *II Jornadas Iberoamericanas de Robótica*, Antigua, Guatemala, 1998.

[10] M. da S. Hounsell e A. Pimentel, "On the Use of Virtual Reality to Teach Robotics", in *3rd International Conference on Engineering and Computer Education*, São Paulo, 2003.

[11] M. P. Groover, *Robótica: tecnologia e programação,* 1 ed. Sao Paulo: McGraw-Hill, 1989.

[12] R. Carey e G. Bell. *The annotated VRML 2.0 reference manual,* Mass.: Addison-Wesley Developers Press, 1997.

[13] U. Rembold e W. Epple, "Present State and Future Trends in the Development of Programming Languages for Manufacturing", in *Computer-Aided Design and Manufacturing Methods and Tools*, 2nd ed., Springer-Verlag, 1986, pp. 279-321.

[14] V. S. Pantelidis, "Virtual Reality in the Classroom". Educacional Tecnology. V. 33. Abr, 1993. P. 23-27.

[15] VRML97 Functional specification and VRML97 External Authoring Interface (EAI), International Standard ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2004.

Marcelo da Silva Hounsell nasceu em Belém do Pará, Brasil em 31 de março de 1967. Ele é Engenheiro Eletricista pela UFPA – Universidade Federal do Pará em 1989, Mestre em Engenharia na área de Computação e Automação Industrial pela UNICAMP em 1992 e PhD. pela Universidade de Loughborough, Inglaterra em 1998. É professor efetivo do Departamento de Ciência da Computação da UDESC-Joinville desde 1990, onde também é líder do grupo de pesquisa LARVA – LAboratório de Realidade Virtual Aplicada. Suas áreas de interesse são: Computação Gráfica, Realidade Virtual, Robótica e Modelagem Avançada de Produtos.

Rubens Redel nasceu em Joaçaba, Santa Catarina, Brasil em 15 de junho de 1982. Está Bacharelando em Ciência da Computação pela UDESC – Universidade do Estado de Santa Catarina. É técnico em Programação Java Básico e Avançado pelo ISPA – Internet/Intranet Systems Programmer and Analyst. Atualmente, é bolsista de Iniciação Científica pelo Departamento de Ciência da Computação desde 2002, onde também é atuante no grupo de pesquisa LARVA – LAboratório de Realidade Virtual Aplicada. Suas áreas de interesse são: Computação Gráfica e Realidade Virtual.