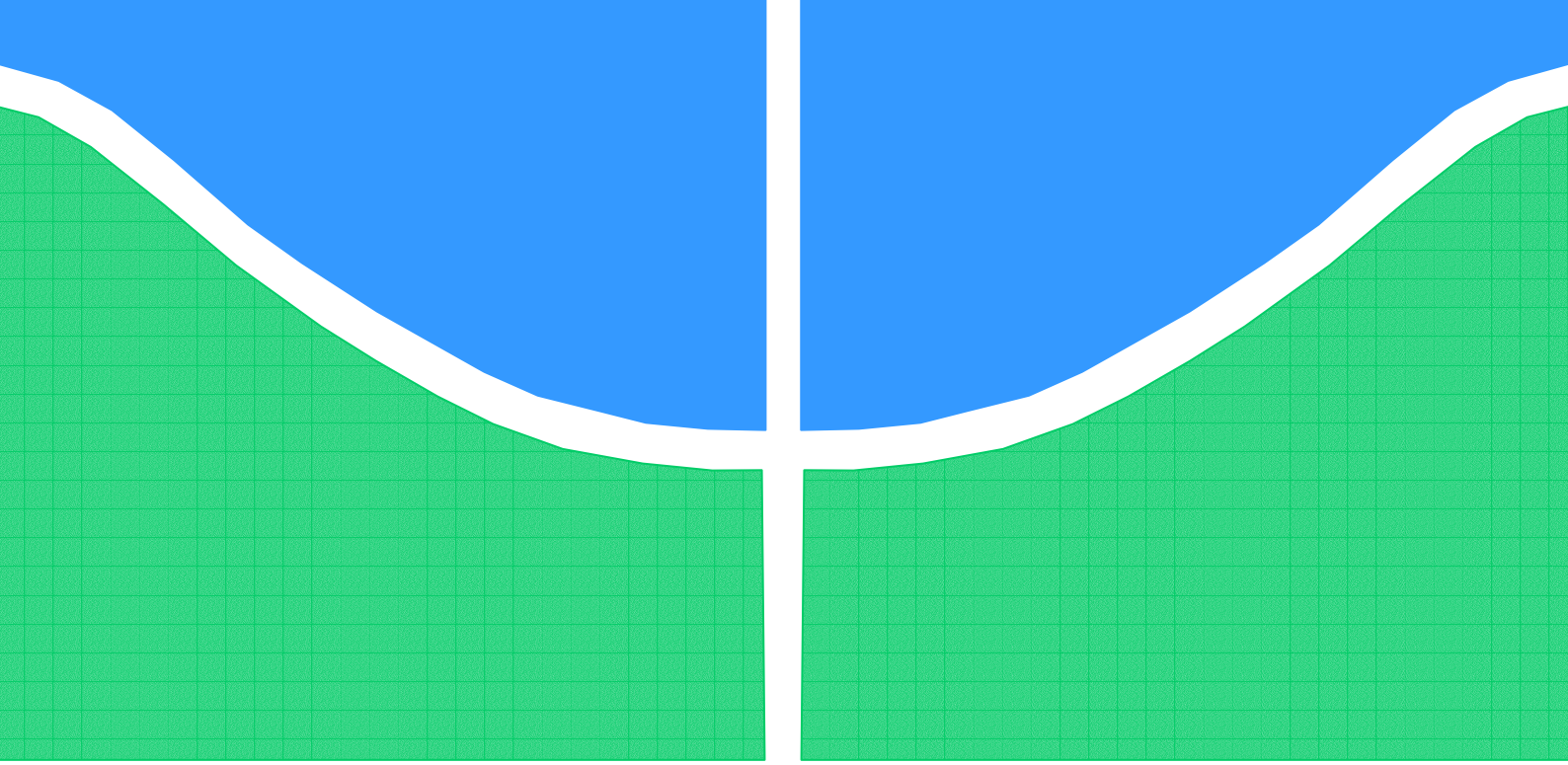


PROJETO DE GRADUAÇÃO

WebAssembly: Sistema integrado CAD/CAPP/CAA para montagem de peças LEGO via Internet

Por,

Leandro Soares Pinheiro

Brasília, 01 de Dezembro de 2006

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECANICA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

PROJETO DE GRADUAÇÃO

WebAssembly: Sistema integrado CAD/CAPP/CAA para montagem de peças LEGO via Internet

POR,

Leandro Soares Pinheiro

Relatório submetido como requisito para obtenção
do grau de Engenheiro Mecânico.

Banca Examinadora

Prof. Alberto J. Alvares, UnB/ ENM (Orientador) _____

Brasília, 01 de Dezembro de 2006

Dedicatória

Aos meus insubstituíveis pais pelo total apoio e investimento, à querida namorada pela força e companheirismo, aos meus amigos pelos bons momentos e à minha família pelo incentivo e confiança.

Leandro S. Pinheiro

RESUMO

WebAssembly é um sistema de telemanufatura operado via internet que integra as ferramentas CAD/CAPP/CAA a um processo de montagem de peças LEGO. A ferramenta de desenho auxiliado por computador (CAD) permite que o usuário construa seu modelo de montagem graficamente através de bibliotecas de *features* de forma das peças LEGO (dimensão e cor). O CAPP (*Computer-Aided Planning Process*) realiza o planejamento de montagem dos LEGOs a partir da interpretação do modelo concebido na ferramenta CAD. O CAA (*Computer-Aided Assembly*) é alimentado pelo planejamento do processo de montagem e gera seqüências de operações em nível de tarefa para o robô industrial ASEA executar a montagem.
O modelo funcional de desenvolvimento IDEF0 é usado para modelar decisões, ações, e atividades do sistema.

ABSTRACT

WebAssembly is a telemanufacturing internet based system which integrates CAD/CAPP/CAA tools for LEGO parts assembly process. The Computer-Aided Design (CAD) tool enables the user to graphically synthesize its assembly model using form-features based libraries of LEGO parts (dimension and color). The CAPP (Computer-Aided Planning Process) makes the assembly planning of LEGO´s beyond the interpretation of the CAD model. The CAA (Computer-Aided Assembly) is fed by the assembly process planning and generates task level sequences to the industrial robot ASEA execute the assembly.
The functional IDEF0 design method is used to model the decisions, actions, and activities of the system.

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Motivação 11
1.2 Objetivos 13
1.3 Organização deste documento 14
2. REVISÃO BIBLIOGRÁFICA 15
2.1 Automação na Montagem 15
2.1.1 Escolha do método de montagem 15
2.1.2 Projeto de peças para montagem 16
2.2 Robôs Industriais 19
2.2.1 Configurações de Robôs 19
2.2.3 Termos importantes na escolha de um braço manipulador 20
2.2.4 Tipos de atuadores 21
2.2.5 Tipos de programação 22
2.3 Aplicativos Java 23
2.3.1 Applets Java 23
2.3.2 Java3D 25
2.3.3 Sockets 28
2.3.4 Comunicação com paralela 30
2.3.5 XML 31
2.3.6 Arquitetura *Client-Server* 32
2.4 Interface de Comunicação 33
2.4.1 Porta Paralela 33
2.4.2 L298 35
3. METODOLOGIA DO PROJETO WEBASSEMBLY 36
3.1 Restrições de projeto 36
3.1.1 Robô ASEA IRB6 36
3.1.2 Peças LEGO 38
3.1.3 FMC e WebAssembly 39
3.2 Modelo funcional IDEF0 40
3.3 Cronograma 45
3.4 Orçamento 45
4. APLICATIVOS CAD/CAPP/CAA 46
4.1 WebAssembly – *Client Side* 46
4.1.1 Instalação 46
4.1.2 Operação 47
4.1.3 UML 49
4.1.4 Estrutura área de montagem (Java3D) 50
4.2 WebAssembly – *Server Side* 51
4.2.1 Instalação 51
4.2.2 Operação 51
4.2.3 UML 52
4.2.4 Acesso/Gravação de dados (XML) 52
4.3 Topologia da comunicação (Socket) 54
4.3 Algoritmo CAPP 55
5. PROJETO MECÂNICO 57
5.1 Mesa de Montagem 57
5.2 Finger 58
5.3 Fixadores do carro do robô 59

6. PROJETO ELÉTRICO 60
6.1 Ligação FMC x WebAssembly 60
6.1.1 Painel Robô 61
6.1.2 Painel Elétrico 62
6.2 *Inferface* de comunicação 64
7. PROGRAMAÇÃO ROBÔ 65
7.1 Topologia da programação 65
7.2 Codificações utilizadas 65
8. RESULTADOS 68
9. CONCLUSÕES E DESENVOLVIMENTOS FUTURAS 71
10. REFERÊNCIA BIBLIOGRÁFICA 72
ANEXOS 73

LISTA DE FIGURAS

Figura 1.1.1 - Integração da manufatura por computador (CIM) [2] *11*
Figura 1.1.2 – Processo de montagem manual de MP3 players [5] *12*
Figura 1.1.3 – Processo de montagem robotizada de placas PCB [5] *12*
Figura 1.1.4 – Pick-n-place de embalagens [6] *13*
Figura 1.2.5 – Arquitetura do WebAssembly *14*
Figura 2.1.1 – Exemplo do efeito do projeto orientado à montagem [4] *16*
Figura 2.1.2 – Reduzindo a resistência ao encaixe [4] *17*
Figura 2.1.3 – Peças chanfradas e guiadas para melhor encaixe [4] *17*
Figura 2.1.4 – Furo para alívio de ar na cavidade de inserção [4] *17*
Figura 2.1.5 – Melhorias de projeto no desenvolvimento de produtos [4] *18*
Figura 2.1.6 – Gráfico do tempo de encaixe para diferentes tipos de chanfros [4] *18*
Figura 2.2.1 – Robôs cartesianos [3] *19*
Figura 2.2.2 – Robôs cilíndricos [3] *19*
Figura 2.2.3 – Robôs esféricos [3] *20*
Figura 2.2.4 – Robôs de revolução [3] *20*
Figura 2.2.5 – Exemplos de fingers [9] *21*
Figura 2.2.6 – Exemplos de manipuladores a vácuo [9] *22*
Figura 2.2.7 – Atuador com complacência [10] *22*
Figura 2.3.1 – Assinatura Digital de applets [13] *24*
Figura 2.3.2 – Camadas de software no contexto de programação gráfica[17] *26*
Figura 2.3.3 – Grafo de cena em Java3D [17] *27*
Figura 2.3.4 – Hierarquia de classes em Java3D [17] *27*
Figura 2.3.5 – Conexão por socket [15] *30*
Figura 2.3.6 – Clientes conectados a um servidor em uma rede local [22] *33*
Figura 2.3.7 – Exemplos de tecnologias da arquitetura Cliente-Servidor [22] *33*
Figura 2.4.1– Pinagem do conector da porta paralela [23] *34*
Figura 2.4.2 – Diagrama esquemático interno *35*
Figura 3.1.1 – Robô industrial ASEA IRB6 [27] *36*
Figura 3.1.2 – Dimensões da garra PGN 160-1 [28] *38*
Figura 3.1.3 – Modelos de features disponíveis *38*
Figura 3.1.4 – Layout dos sistemas FMC e WebAssembly *40*
Figura 3.2.1 – Folha A0 do modelo IDEF0 - WebAssembly *41*
Figura 3.2.2 – Detalhe da folha A0 do modelo IDEF0 - WebAssembly *42*
Figura 3.2.3 – Detalhe da folha A1 (CAD) do modelo IDEF0 - WebAssembly *43*
Figura 3.2.4 – Detalhe da folha A2 (CAPP) do modelo IDEF0 - WebAssembly *44*
Figura 3.2.5 – Detalhe da folha A3 (CAA) do modelo IDEF0 - WebAssembly *44*
Figura 3.3.1 – Gantt Chart do cronograma WebAssembly *45*
Figura 4.1.1 – GUI applet client side *46*
Figura 4.1.2 – Sistemas de coordenadas do Canvas 3D *48*
Figura 4.1.3 – Diagrama de classes WebAssembly cliente *50*
Figura 4.1.4 – Estrutura dos objetos Java3D *50*
Figura 4.2.1 – Computador servidor (GRACO) *51*
Figura 4.2.2 – Diagrama de classes WebAssembly servidor *52*
Figura 5.1.1 – Vista isométrica da mesa de montagem *58*
Figura 5.2.1 – Finger manipulando os blocos LEGO *58*
Figura 5.3.1 – Desenho isométrico e fotografia dos fixadores *59*
Figura 6.1.1 – Diagrama de utilização das I/O's (WebAssembly x FMC) *60*
Figura 6.1.2 – Ligações aos relés das I/O's *61*

Figura 6.1.3 – Conexões no barramento D14 153[27] *62*
Figura 6.1.4 – Painel elétrico *62*
Figura 6.1.5 – Diagrama WebAssembly *63*
Figura 6.1.6 – Diagrama FMC *63*
Figura 6.2.1 – Interface de comunicação WebAssembly *64*
Figura 6.2.2 – PCB WebAssembly *64*
Figura 8.1 – Modelos de montagem *68*
Figura 8.2 – Montagem concluída *70*
Figura 8.3 – Segundo modelo de montagem *70*
Figura 8.4 – Resultado do segundo modelo de montagem *70*

LISTA DE TABELAS

Tabela 2.3.1 – Lista de portas de comunicação reservadas *29*
Tabela 2.4.1 – Mapa de endereços das portas paralelas [24]..... *34*
Tabela 3.1.1 – Parâmetros de fabricação da garra PGN 160-1[28] *37*
Tabela 3.1.2 – Peças LEGO disponíveis..... *39*
Tabela 3.4.1 – Orçamento projeto WebAssembly..... *45*
Tabela 4.3.1 – Lista de códigos de comunicação..... *54*
Tabela 7.2.1 – Codificações ao controlador área de montagem..... *66*
Tabela 7.2.2 – Codificações ao controlador área de estoque..... *66*
Tabela 7.2.3 – Codificações ao controlador para rotação/deslocamento..... *67*

LISTA DE SÍMBOLOS E ABREVIATURAS

Siglas

CAD	Desenvolvimento Auxiliado por Computador
CAPP	Planejamento de Processo Auxiliado por Computador
CARC	Controle Robótico Auxiliado por Computador
WWW	World Wide Web
CAQ	Qualidade Auxiliada por Computador
CAMA	Manutenção Auxiliada por Computador
CIM	Gerenciamento Integrado por Computador
VRML	Linguagem de Modelagem de Realidade Virtual
IDEF	Integrated DEFinition Methods
API	Application Programmer's Interface
UML	Unified Modeling Language
XML	eXtensible Markup Language

1 INTRODUÇÃO

1.1 Motivação

O aumento da demanda em variedade de produtos e a redução de seu ciclo de vida no mercado demandaram o auxílio do computador aplicado ao desenvolvimento de novos produtos e, em mais alto nível, ao sistema de gerenciamento da manufatura. Softwares CAD foram os pioneiros no auxílio computacional para a concepção e projeto de produto, esta ferramenta permite que projetistas reutilizem e alterem projetos anteriores com relativa facilidade, reduzindo consideravelmente *o lead time* de projeto [1]. Atualmente diversas outras ferramentas auxiliam e encurtam o caminho entre o projeto e o produto, a exemplo do CAQ (Computer-Aided Quality) e o CAMA (Computer-Aided Maintenance).

Estas e outras ferramentas em uma organização levam ao conceito de CIM (Computer Integrated Manufacturing), representado na figura 1.1.1 [2], que geram e controlam um processo de manufatura em todas suas etapas. [1]

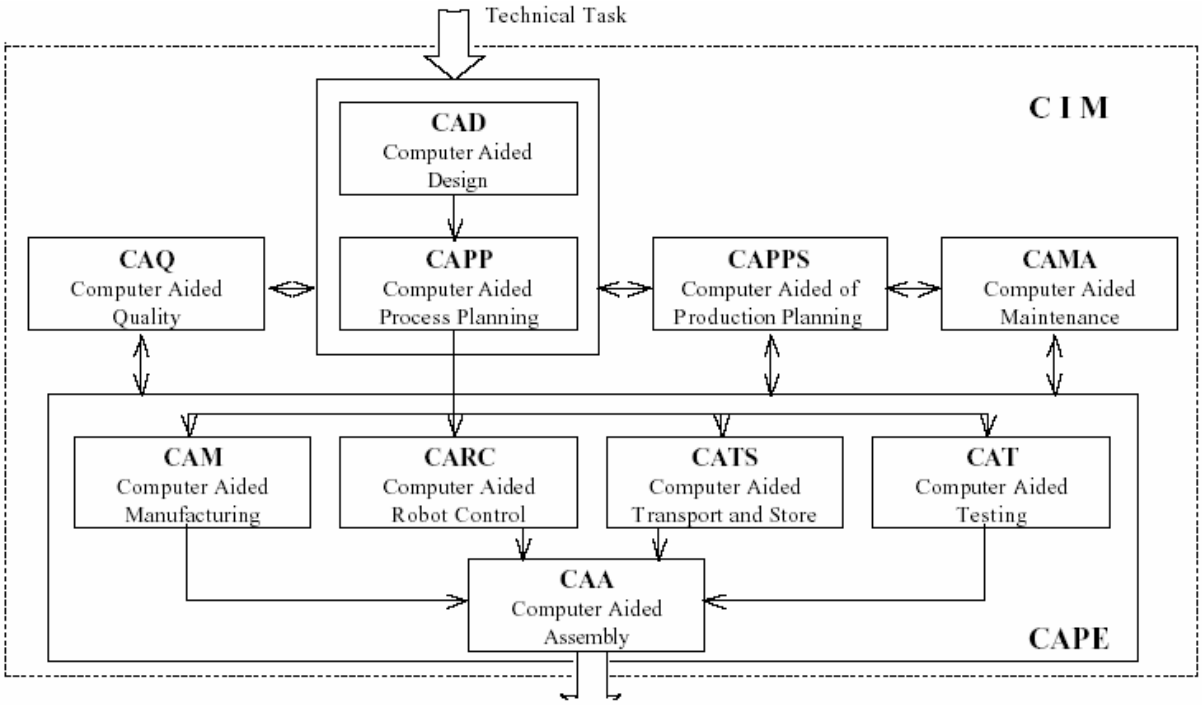


Figura 1.1.1 - Integração da manufatura por computador (CIM) [2]

Outro grande avanço computacional foi a interligação dos computadores pela grande rede mundial, a WWW (*World Wide Web*), que hoje interliga bilhões de usuários por todo o mundo. Só no Brasil, hoje, são mais de 25 milhões de usuários. O uso da Internet é mais expressivo no setor privado, estima-se que 96,29% das empresas brasileiras têm acesso à Internet [JORNAL O DIA, 2006], isso

reflete a potencialidade e a influência da Internet no setor comercial, tanto pela agilidade quanto pela confiabilidade.

Atividades de montagem compõem a etapa final do ciclo de produção e, hoje em dia, grande parte delas são realizadas manualmente, a figura 1.1.2 mostra um exemplo de um chão de fábrica na montagem de MP3 *players*. A montagem é uma atividade que agrega pouco valor ao produto e ao mesmo tempo eleva o custo de produção devido à elevada necessidade de mão-de-obra e horas de trabalho, o que cria situação favorável à automação.[3] Custos com montagens geralmente ultrapassam 50% dos custos totais de manufatura.[4] Em aplicações robotizadas, o planejamento de processo de montagem é gerado por computador diretamente através do modelo desejado, seja por métodos heurísticos ou genéticos. Robôs com maior número de graus de liberdade, programação off-line e simulação computadorizada do processo permitem que grande variedade de produtos seja ofertada e tenham curto tempo de *setup*. Neste contexto, pode-se citar montagens PCB (figura 1.1.3) e processos de *pick-n-place* de embalagens (figuras 1.1.4).

Figura 1.1.2 – Processo de montagem manual de MP3 players [5]

Figura 1.1.3 – Processo de montagem robotizada de placas PCB [5]

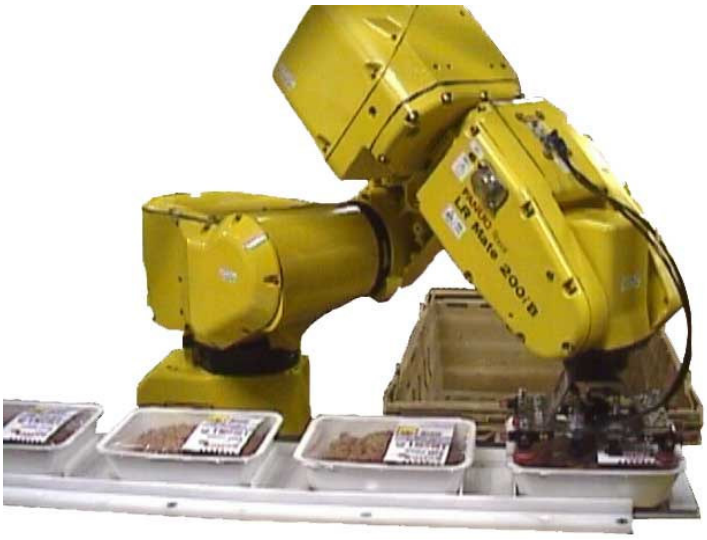


Figura 1.1.4 – Pick-n-place *de embalagens [6]*

O estudo da melhor seqüência de montagem tem despertado interesse, pois alteram tempo e custo da produção. O planejamento de montagem pode ser modelado por grafos relacionais [7] ou pela descrição de todas seqüências de montagem possíveis [8]. A seqüência com que as peças são montadas pode influenciar na quantidade de *setups*, no tipo de orientação da garra do robô e na quantidade final de operações de montagem.

A motivação deste trabalho enquadra-se nesse cenário de expansão do uso da Internet, dos softwares de auxílio à produção e à influência do planejamento de montagem no produto final. Montagens utilizando blocos LEGO evidenciam o caráter didático do projeto e abre possibilidades de testes de diferentes algoritmos, restrições e prioridades.

1.2 Objetivos

O objetivo deste trabalho é desenvolver um sistema de montagem de peças LEGO utilizando o robô industrial ASEA, situado no GRACO (Grupo de Controle e Automação), UnB. No sistema WebAssembly, um *applet* Java intuitivo dá suporte ao usuário criar sua *feature* de montagem desejada a partir das *features* de forma dos blocos LEGO. Este *applet* apresenta graficamente o modelo concebido e recebe comandos de configuração e pedido de execução, bem como apresenta o planejamento de montagem.

O *applet* cliente comunica-se por *socket* com o aplicativo Java no servidor. Este servidor, conectado ao controlador do robô, é capaz de enviar informações de posicionamento e tarefa, através da porta paralela e I/O´s do robô.

O projeto mecânico envolveu a usinagem do *finger* para o manipulador pneumático, a construção de fixadores para o carro do robô e a construção da mesa de montagem, que oferece suporte às peças em estoque e à área de montagem.

O projeto elétrico consistiu em desenvolver a interface para interligar o computador servidor às I/O´s do controlador, isolando os sinais da LPT (5V) do controlador (24V) através do componente L298 e relés.

A figura 1.2.5 mostra a visão geral da arquitetura implementada no WebAssembly.

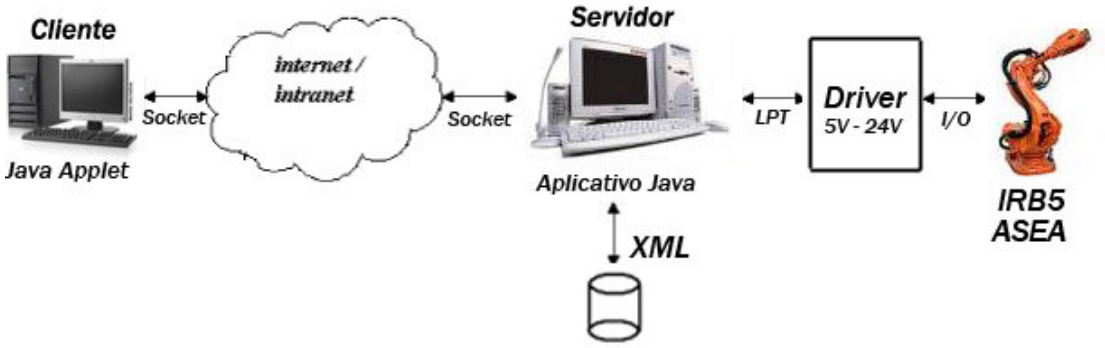


Figura 1.2.5 – Arquitetura do WebAssembly

O desenvolvimento WebAssembly considera o compartilhamento de recursos (como robô e suas I/O´s) com o projeto FMC, na célula de manufatura.

1.3 Organização deste documento

Além deste capítulo introdutório, a estrutura deste documento está dividida da seguinte forma:

Capítulo 2 traz a revisão bibliográfica dos principais tópicos utilizados no desenvolvimento do projeto.

Capítulo 3 apresenta a metodologia utilizada bem como as restrições relevantes ao projeto.

Capítulo 4 descreve os aplicativos cliente e servidor no aspecto implementação e operação.

Capítulo 5 constitui os desenvolvimentos mecânicos para a manipulação das peças LEGO e suportes do carro do robô.

Capítulo 6 desenvolve o projeto da interface de comunicação e a fiação compartilhada com o projeto FMC.

Capítulo 7 relata a metodologia e descreve o programa embarcado no controlador do robô.

Capítulo 8 apresenta os resultados obtidos e os testes realizados para diferentes montagens.

Capítulo 9 finaliza o documento com as conclusões e sugestões para trabalhos futuros.

Capítulo 10 lista as referências bibliográficas consultadas.

2. REVISÃO BIBLIOGRÁFICA

A revisão bibliográfica apresenta o fundamento teórico necessário para o desenvolvimento do projeto na forma de estudo dirigido. Os tópicos abordarão diferentes soluções para o mesmo problema e serão complementados pela escolha adotada neste projeto.

2.1 Automação na Montagem

Desde o começo do século 19, o aumento da necessidade por produtos acabados de boa qualidade e grandes quantidades, tem levado engenheiros a desenvolver novos métodos de manufatura ou produção. O resultado destes desenvolvimentos em vários processos de manufatura levou à produção em massa, produtos de alta qualidade e baixo custo. Um dos mais importantes processos de manufatura é o processo de montagem que é obrigatório quando duas ou mais partes precisam ser montadas. [4]

Este tópico abordará os fatores que afetam um processo de montagem e seus efeitos na produção.

2.1.1 Escolha do método de montagem

Resumidamente, para que seja viável automatizar um processo de montagem, este deve possuir as seguintes características: o produto deve ser projetado com vista para montagem automática, o aumento iminente da produção deve ser absorvido pelo mercado e o capital de investimento deve ser amortizado pelo ciclo de vida do produto no mercado. [4]

Processos de montagem automatizados são mais seletivos em relação à montagem manual quanto à qualidade das peças da matéria-prima e à qualidade do produto final. Na montagem manual, o colaborador pode simplesmente detectar se a peça a ser montada está defeituosa ou não e descartá-la. Na montagem automatizada, uma peça defeituosa interrompe a produção, mesmo contendo sistema de visão, o processo deverá ser reiniciado na ocorrência de falhas.

A montagem automatizada garante a qualidade do produto de saída devido à acurácia do maquinário e sua seletividade inerente em relação às peças de entrada. Caso as peças de matéria-prima não estiverem conforme, o sistema automatizado simplesmente não completará a montagem e o processo será parado. Este fato não acontece na montagem manual, pois o operador pode, de certa forma, contornar pequenos problemas de fabricação e completar a montagem.

Outra vantagem em automatizar a montagem é que força os projetistas de produtos a considerar o processo de montagem durante o desenvolvimento.

2.1.2 Projeto de peças para montagem

Apesar das vantagens da automação, automatizar a montagem nem sempre significa a melhor e única solução para redução dos custos. A melhoria do projeto do produto focado em sua montagem tem muito mais potencial de reduzir custos do que simplesmente automatizar o processo. A figura 2.1.1 compara um mesmo projeto que pode ter diferentes tipos de montagem e seu custo de montagem unitário. [4]

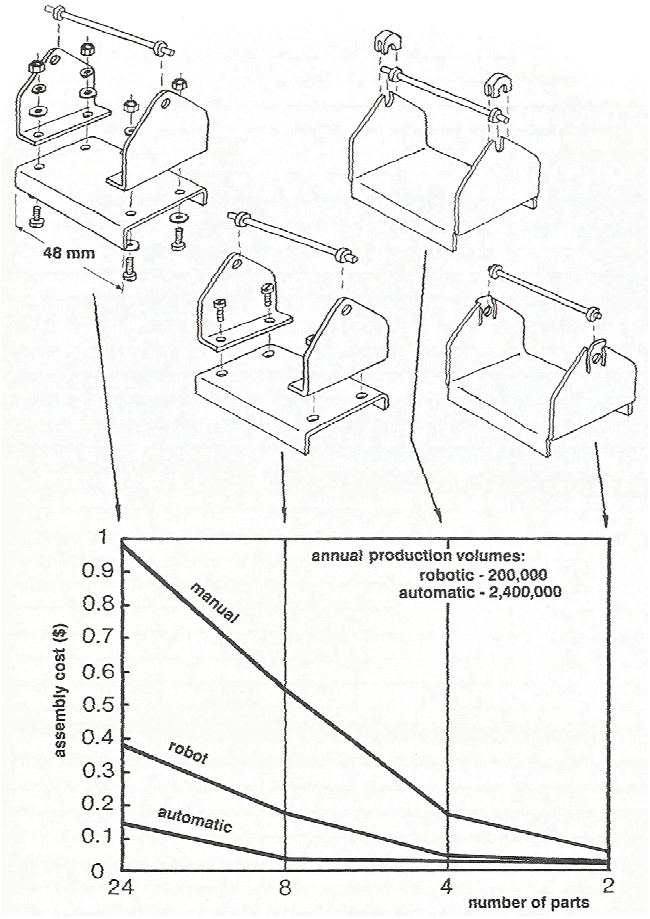


Figura 2.1.1 – Exemplo do efeito do projeto orientado à montagem [4]

A melhoria do projeto de peças para montagem pode ser dividida em dois campos: manuseio (*handling*, orientação e movimentação de peças) e inserção (encaixe de uma peça sobre a outra ou grupo de peças). Para ambas algumas regras devem ser seguidas: [4]

a) Restrições para manuseio:

- Projetar peças com o máximo de simetria possível;

- Caso as peças não sejam simétricas, adotar a assimetria ao extremo, para que não seja possível a montagem em orientação errada;

- Evite *features* que permitam uma peça travar à outra, quando armazenada;

- Evite peças escorregadias, pontiagudas, flexíveis e de difícil manuseio.

b) Restrições para encaixe:

- Projetar peças de modo que seja mínima ou ausente a resistência ao encaixe e que contenham chanfros e guias para encaixe. (figura 2.1.2 e 2.1.3)

- Em encaixes precisos, procure deixar passagem para o ar. (figura 2.1.4)

- Procure padronizar as peças utilizadas, buscando peças ou métodos comuns a produtos diferentes, a fim de permitir alto volume de produção.

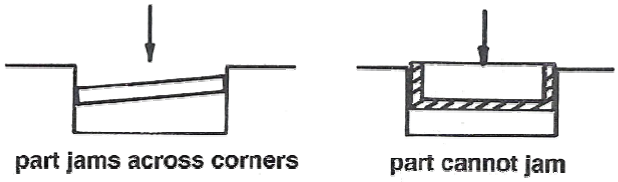


Figura 2.1.2 – Reduzindo a resistência ao encaixe [4]

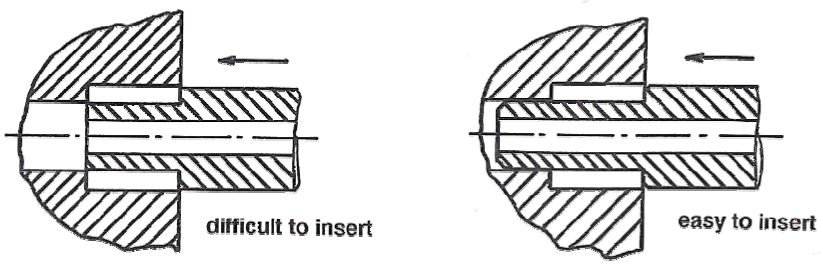


Figura 2.1.3 – Peças chanfradas e guiadas para melhor encaixe [4]

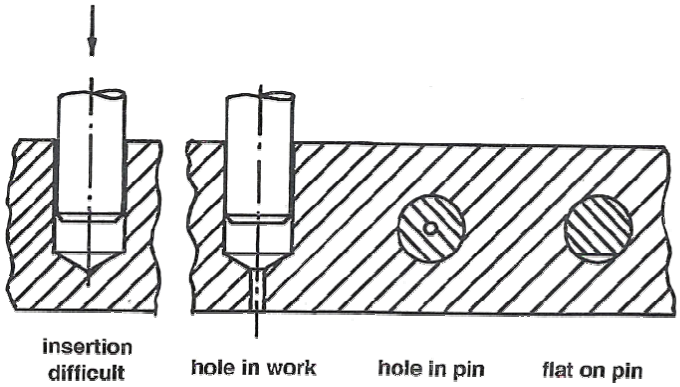


Figura 2.1.4 – Furo para alívio de ar na cavidade de inserção [4]

A figura 2.1.5 mostra um exemplo de melhoria de projeto para redução do número de peças para montagem.

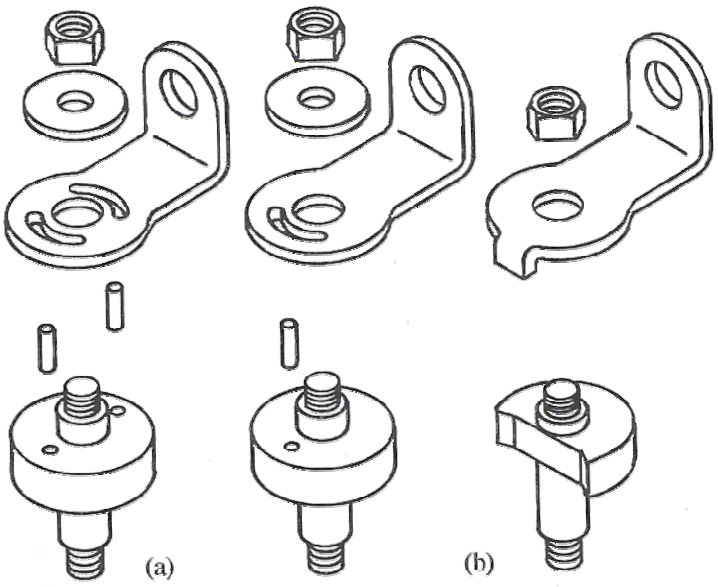


Figura 2.1.5 – Melhorias de projeto no desenvolvimento de produtos [4]

A figura 2.1.6 mostra um gráfico que compara o tempo de encaixe sob diferentes tipos de chanfros sob diferentes diâmetros de eixo.

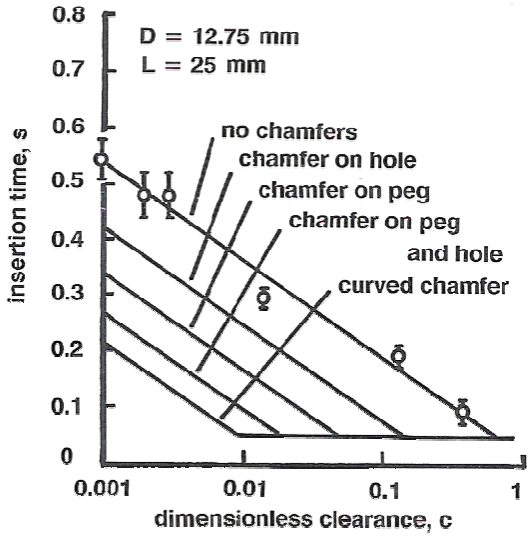


Figura 2.1.6 – Gráfico do tempo de encaixe para diferentes tipos de chanfros [4]

2.2 Robôs Industriais

"Um robô é um manipulador multifuncional, programável, desenvolvido para mover materiais, peças, ferramentas, ou dispositivos especializados, permitindo movimentos variáveis para a performance de diferentes tarefas". [*The Robot Industries Association*]

2.2.1 Configurações de Robôs

Os robôs são diferenciados pela sua geometria, que resultam em diferentes áreas de trabalho e conseqüentemente aplicações distintas. As possíveis configurações são:

1. Coordenadas cartesianas: são robôs que podem alcançar o volume retangular compreendido por todos comprimentos dos elos. (figura 2.2.1)

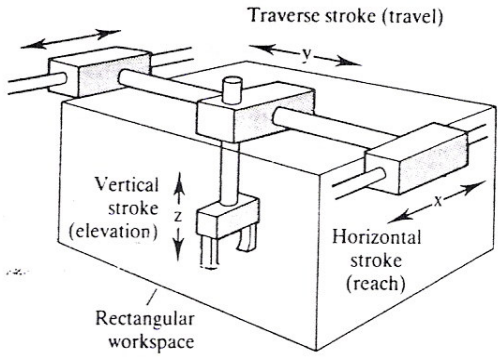


Figura 2.2.1 – Robôs cartesianos [3]

2. Coordenadas cilíndricas: são robôs que possuem uma junta de rotação principal e mais duas juntas prismáticas em torno desse eixo. Essa configuração resulta em um volume de alcance cilíndrico ao redor do robô. (figura 2.2.2)

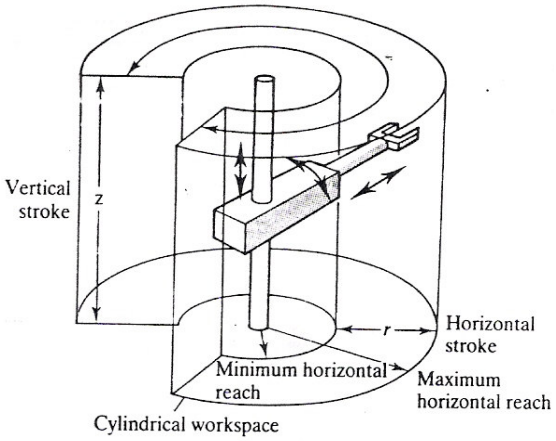


Figura 2.2.2 – Robôs cilíndricos [3]

3. Coordenadas esféricas: caracterizado por uma junta prismática acoplada a duas juntas de rotação, um na base e outro na junta superior. Essa configuração pode ser trabalhada em numa região

do tipo casca de esfera, compreendida entre o deslocamento mínimo e máximo da junta linear. (figura 2.2.3)

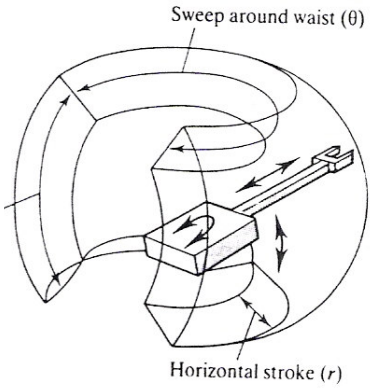


Figura 2.2.3 – Robôs esféricos [3]

4. Coordenadas de revolução: são robôs que possuem todas juntas rotacionais. Esta configuração constitui a evolução das anteriores e alcançam uma área de trabalho esférica em torno da junta da base. (figura 2.2.4)

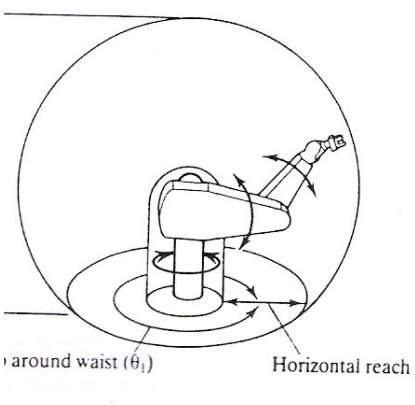


Figura 2.2.4 – Robôs de revolução [3]

Em indústrias, robôs do tipo coordenadas de revolução (braços de robô) ganharam espaço devido à simplicidade de construção e melhor vedação das juntas. Estes robôs também trabalham com amplo volume de trabalho, podendo ser utilizados em diversas aplicações sem alterar o tipo de robô.

2.2.3 Termos importantes na escolha de um braço manipulador

Diversos fabricantes oferecem configurações e capacidade de cargas diferentes, para diversos tipos de aplicações. Alguns termos são importantes na escolha do robô, como:

Exatidão: diferença entre a posição real desejada e a posição realmente alcançada.

Confiabilidade: medida do tempo em que um robô opera sem falha.

Repetibilidade: medida da diferença entre sucessivos movimentos para uma mesma posição programada.

Resolução: medida dos tamanhos dos paços entre posições (paços/mm), ou a distância mínima que o manipulador irá se mover.

Tempo de ciclo: tempo em segundos que o braço do robô, estendido em seu comprimento máximo, leva para rotacionar do extremo direito ao extremo esquerdo. [3]

2.2.4 Tipos de atuadores

Os atuadores são acoplados ao órgão terminal do robô e tem contato direto com a peça a ser manipulada. Atuadores são separados em grupos de acordo com funcionalidade e tipo de operação:

Fingers: manipulam uma ou mais peças (geralmente por atrito) e podem adquirir diversos formatos, dependendo da geometria do material. O projeto de *fingers* deve seguir as seguintes características: peças devem ser seguradas sem danificá-las; peças devem ser posicionadas firmemente e rigidamente enquanto manipuladas; em alguns casos são necessárias juntas para auto-alinhar a peça; o *finger* deve possuir no mínimo dois pontos de contato com a peça para evitar movimentos rotacionais enquanto posicionada (exemplos na figura 2.2.5). [3]

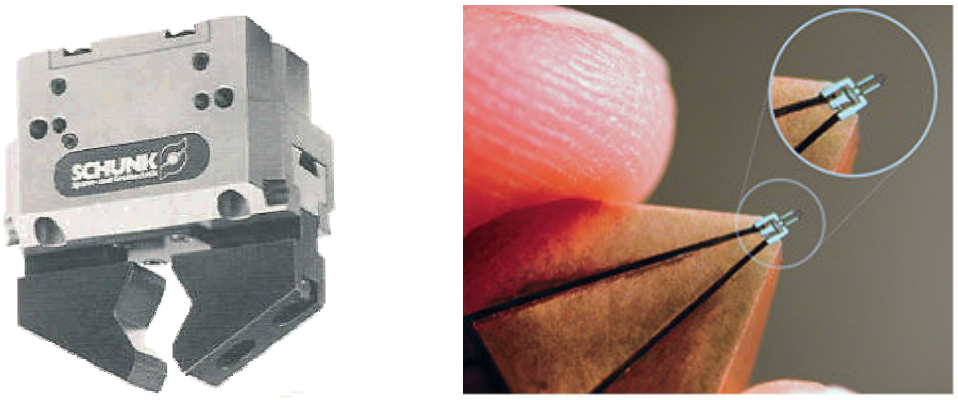


Figura 2.2.5 – Exemplos de fingers *[9]*

Atuadores a Vácuo: este tipo de atuador geralmente é utilizado para manipular folhas de metal, plásticos e papel, ou objetos com superfícies planas. Estes atuadores utilizam ventosas a vácuo, cada ventosa é conectada a uma bomba de vácuo que provoca uma diferença de pressão dentro e fora da ventosa. Essa diferença de pressão deve ser proporcional ao peso do objeto e ao número de ventosas utilizadas (figura 2.2.6).

Figura 2.2.6 – Exemplos de manipuladores a vácuo [9]

Atuadores ferramenta: uma larga variedade de ferramentas especializadas podem ser acopladas ao punho do robô, como pistola de solda, pistola de pintura, furadeiras, etc (figura 2.2.7).

Alguns atuadores são específicos para processos de montagem pois adicionam complacência ao sistema, permitindo corrigir pequenas imperfeições ou imprecisões durante a montagem, veja figura 2.2.7.

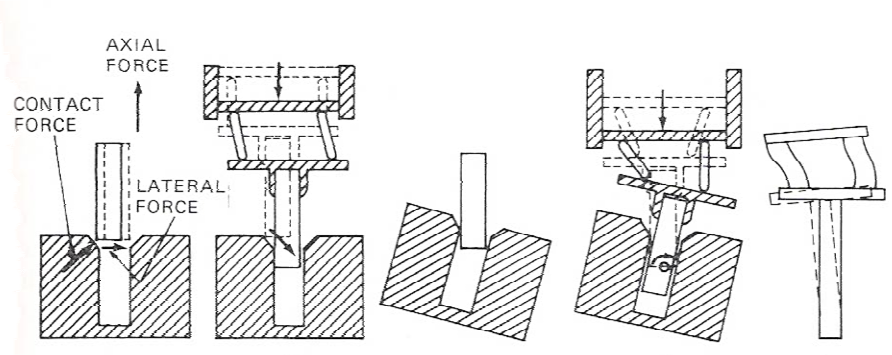


Figura 2.2.7 – Atuador com complacência [10]

2.2.5 Tipos de programação

Os tipos de programação distinguem-se pelo nível de programação e pela linguagem de programação que o controlador utiliza.

Há seis níveis de programação: Controle de Seqüências, *Teach-by-Guiding*, *Teach-by-Pendant*, Programação *Off-line*, Sistema Adaptativo e Sistemas Inteligentes. Veja:

Na programação por controle de seqüência, todo controle é realizado por dispositivos de sequenciamento, podendo ser elétrico ou mecânico. O controle de seqüência tem pouca flexibilidade já que as operações são pré-determinadas e não realmente programáveis.

Na programação *Teach-by-Guiding*, o operador move o robô manualmente e grava os pontos da trajetória. Após a programação, o robô será capaz de repetir a trajetória programada.

A programação *Tech-by-Pendant*, o programador utiliza o *pendant* (espécie de *joystick*) que controla as juntas do robô em termos de coordenadas polares ou retangulares. Assim o programador pode posicionar o robô e gravar cada ponto e determinar as trajetórias.

A programação *off-line* é concebida através de *softwares* simuladores, sem necessitar que o robô esteja ligado. Ao final da programação o programa é gravado na memória do robô e executado sucessivas vezes. Esta é uma programação que exige um modelo computacional fiel ao modelo real e *softwares* que consigam reproduzir exatamente os cálculos do controlador do robô.

Programação por sistemas adaptativos é a programação que utiliza sensoriamento para informar as variáveis do ambiente. Estes sensores podem ser de força, torque, proximidade ou sensoriamento por visão computacional. O movimento do robô dependerá das atividades programadas e dos sinais dos sensores.

A programação por sistemas inteligentes, genericamente, considera uma base de conhecimentos sobre diversos problemas e tarefas. As tarefas são passadas em alto nível ao sistema inteligente e, através de sensores e conhecimentos prévios adquiridos, o robô executa a seqüência de movimentos para realizar a tarefa. [3]

As linguagens de programação são visualizadas e editadas no *Pendant,* que recebe informações do programador e as grava na memória do controlador. Dependendo do fabricante, a programação pode oferecer comunicação com o meio externo, *setup* da velocidade de movimentação, determinação do sistema de referência, etc. Muitas linguagens foram desenvolvidas para diferentes fabricantes e tipos de robôs, algumas delas são: AR-BASIC, RAIL, GCA CLIMPLER, HAL, IBM´s AML, VAL II, assim por diante.

2.3 Aplicativos Java

Muitas características da linguagem Java não serão detalhadas, como portabilidade e desempenho, pois tratam de assuntos gerais sobre a linguagem. Por outro lado, serão abordados assuntos específicos utilizados no projeto WebAssembly, como *sockets* e arquitetura *Client-Server*.

2.3.1 Applets Java

Um *applet* Java é uma aplicação Java que executa dentro do *browser* do usuário, isto é, executa dentro de uma página HTML, chamada pelas *tags* <*applet*></*applet*>*,* ou na ferramenta para testes (*AppletView*) da *Sun*. *Applets* são aplicativos que servem da JVM (*Java Virtual Machine*) existente na máquina cliente ou embutida no próprio navegador do cliente para interpretar o seu *bytecode.* Como os *bytecodes* são independentes de plataforma, *applets* Java podem ser executadas pelo *browser* de qualquer plataforma, incluindo Windows, Unix, Mac OS e Linux.

Criados pela *Sun* em 1995, *applets* adicionam iteratividade a aplicações *web* que não podem ser geradas pelo HTML, assim como JavaScript, DHTML e ASP, porém com mais recursos gráficos e segurança. [11]

Um *applet* Java estende da classe *java.applet.Applet*, ou no caso do *Swing applet*, da classe *javax.swing.JApplet*. Java é descendente de *Panel* no qual é descendente de *Container*. [11]

Para garantir a segurança, os *applets* são executados dentro da *sandbox* (caixa de areia), impedindo-os de acessarem os dados da máquina na qual estão sendo executados. A idéia de *sandbox* foi desenvolvida no sentido de garantir que mesmo que um usuário carregue um *applet* malicioso, este não poderá causar danos ao cliente. Porém muitos aplicativos são confiáveis e precisam sair da *sandbox* e ter acesso a arquivos ou às portas de comunicação. Existem duas maneiras de o fazer: assinar o *applet* digitalmente ou modificar o "arquivo de política" chamado *java.policy* do computador.

Para assinar um *applet*, o desenvolvedor empacota todo o código Java e arquivos relacionados dentro de um arquivo JAR (*Java ARchive*), que é um formato de arquivo de compactação de propósito geral, usado para compactar os componentes de uma aplicação Java [12]. A plataforma Java assina e verifica arquivos JAR usando um par de chaves (chave pública e chave privada). A chave privada funciona como uma "caneta" eletrônica que assina o arquivo, ver Figura 2.3.1. Como o próprio nome sugere, esta chave só é conhecida pelo assinante do *applet*. O processo de verificação da assinatura pode ser feito por qualquer pessoa que possua a chave pública correspondente à chave que assinou o arquivo. [13]

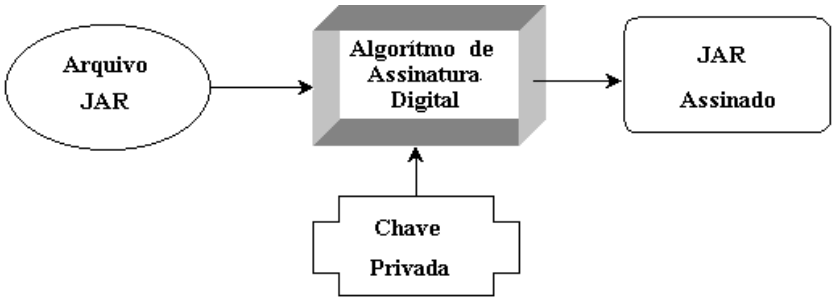


Figura 2.3.1 – Assinatura Digital de applets [13]

A chave pública é distribuída dentro de um certificado que é uma declaração assinada por uma entidade idônea, chamada Autoridade de Certificação (*Certification Authority*- CA), que confirma que a chave pública que está no mesmo é confiável. Existem várias Autoridades de Certificação, por exemplo, a *VeriSign*, *Thawte*, *Entrust* e *Certisign*. [14]

Algumas ferramentas são necessárias para assinar applets: Jar (JAR Creation Tool), Keytool (Key and Certificate Management Tool), Jarsigner (JAR Signing and Verification Tool), Jarsigner (JAR Signing and Verification Tool). [14]

Já o segundo método de modificar o arquivo *java.policy* é mais simples, porém não recomendável quando se trata de aplicativos desconhecidos e de origem duvidosa. Para fazê-lo, o usuário deve abrir o arquivo j*ava.policy* contido no diretório *\lib\security* no diretório de instalação do JRE (*Java Runtime Environment*). Este diretório é especificado pelo usuário no momento da instalação do JRE e normalmente encontra-se em *c:\Arquivos de Programas\Java*.

Após abrir o arquivo com o aplicativo *WordPad*, o usuário deve inserir o seguinte texto antes de "// default permissions granted to all domains":

```
grant    {
        permission java.security.AllPermission;
};
```

Este passo permitirá que todo aplicativo *applet* (inclusive o WebAssembly) acesse arquivos e portas de comunicação do computador (*sockets*). Sendo, portanto, altamente recomendável sua retirada após o uso.

Para executar aplicativos *applet*, o usuário deve instalar o J2RE 1.4.2 ou superior na plataforma que desejar, podendo adquirir gratuitamente em *http://java.sun.com/javase/downloads/index.jsp*.

2.3.2 Java3D

A OpenGL (*Open Graphics Library*) é a interface mais utilizada em aplicações gráficas, ela permite o controle simples e direto sobre objetos de produção de imagens [17] através de API (*Application Programmer's Interface*). A OpenGL funciona como uma máquina de estados, onde o controle de atributos é realizado através de um conjunto de variáveis de estado que inicialmente possuem valores *default*, podendo ser alterados caso seja necessário. Por exemplo, todo objeto será traçado com a mesma cor até que seja definido um novo valor para esta variável. Essa biblioteca é responsável pela criação de elementos de baixo nível de abstração destinados a renderização, e são representados por pontos, vértices e planos. Possibilitando ainda edições de iluminação, cor e textura.

O Java3D implementa o OpenGL no desenvolvimento de aplicações gráficas em Java [17], este disponibiliza objetos de alto nível de abstração e controla a biblioteca OpenGL ou Direct3D de baixo nível, dessa forma o desenvolvedor pode dedicar-se à criação e abandonar detalhes de construção, ver camadas na figura 2.3.2.

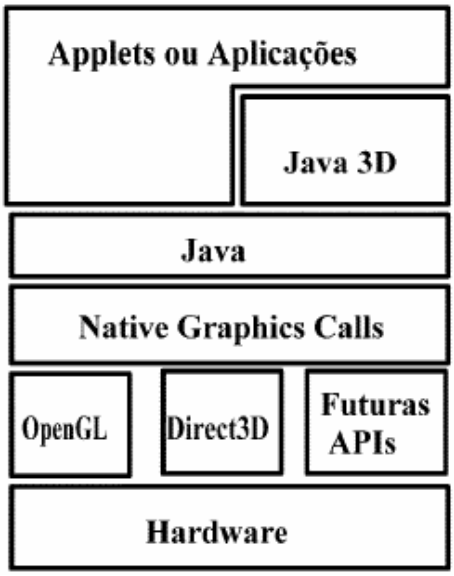


Figura 2.3.2 – Camadas de software no contexto de programação gráfica[17]

No Java3D, o grafo de cena constitui o elemento básico para a criação de cenas, é possível neste grafo controlar objetos gráficos, luz, som, objetos de interação, entre outros, que possibilitam ao programador criar mundos virtuais com personagens que interagem entre si e/ou com o usuário [Sowirzal, 1998].

Para criar um *Virtual Universe* (Universo Virtual) em Java3D é necessário criar nós e ramos que definirão cenário, tela, câmera, etc, veja figura 2.3.3. Nesta figura, o ramo à esquerda do *Virtual Universe* (*content branch graphs*) descreve os objetos que serão renderizados, ou seja, especificam textura, geometria, cor, objetos de luz, etc. Já o ramo à direita (*view branch graphs*) define especificações de vista, como localização e orientação do usuário. [*17*]

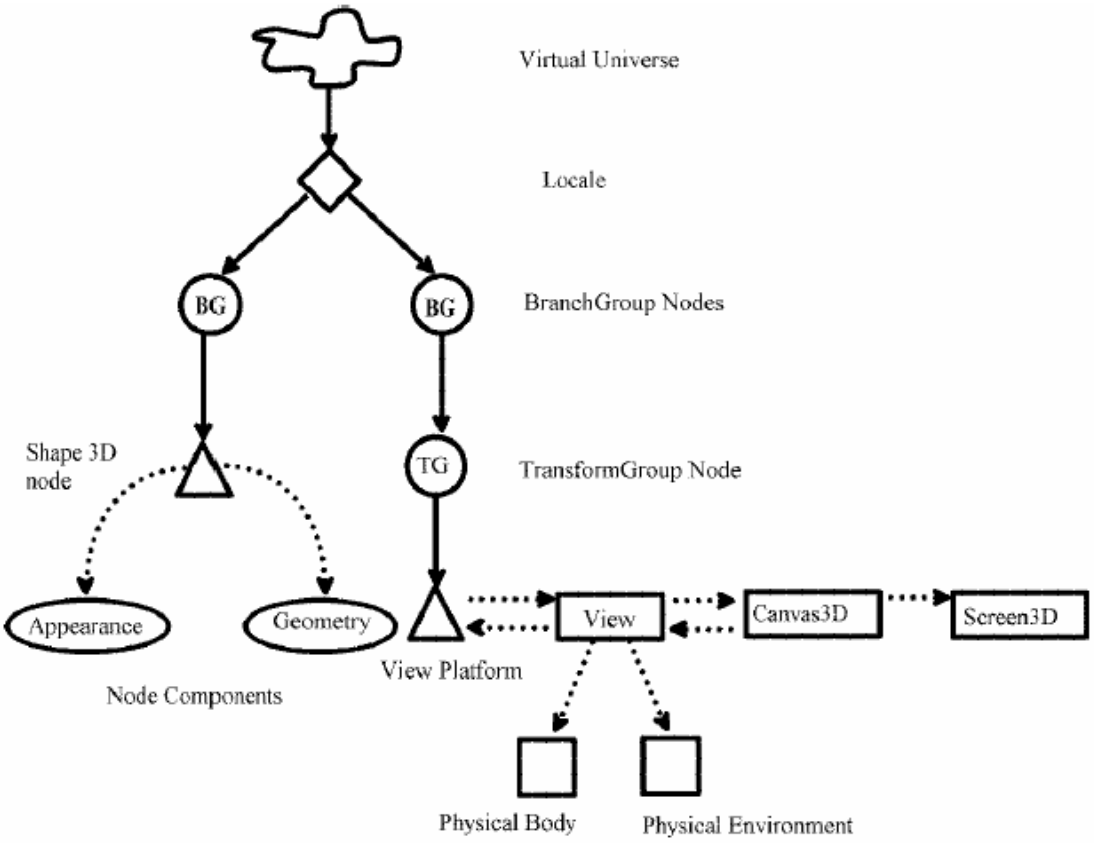


Figura 2.3.3 – Grafo de cena em Java3D [17]

A figura 2.3.4 mostra a hierarquia de classes utilizada em Java3D.

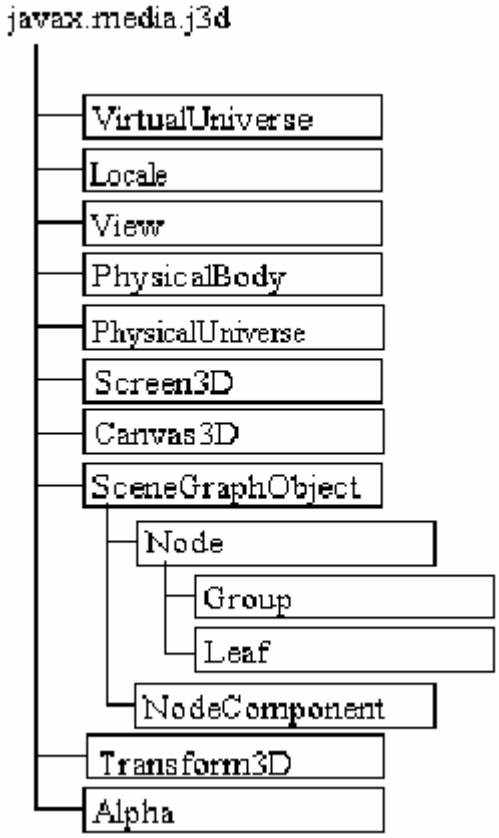


Figura 2.3.4 – Hierarquia de classes em Java3D [17]

Ao contrário das demais APIs gráficas que são *camera-based*, Java3D oferece um modelo de vista que diferencia o mundo virtual do mundo físico [Sun Microsystems, 2001].

A *Sun Microsystems* sugere um algoritmo para criação de aplicativos em Java3D, este algoritmo instancia objetos destinados a criar os ramos *view branch* e *content branch*:

1. Criar um objeto *Canvas3D*
2. Criar um objeto *VirtualUniverse*
3. Criar um objeto *Locale* e anexá-lo ao *VirtualUniverse*
4. Construir um grafo *view branch*
    (a) Criar um objeto *View*
    (b) Criar um objeto *ViewPlatform*
    (c) Criar um objeto *PhysicalBody*
    (d) Criar um objeto *PhysicalEnvironment*
    (e) Anexar os objetos criados em (b), (c) e (d) ao objeto *View*
5. Construir um ou mais grafos *content branch*
6. Compilar o(s) *branch graph(s)*
7. Inserir os subgrafos ao nó *Locale*

Os *branch graphs* comporão os objetos da cena e podem conter atributos como cor, dimensão, transformação, textura e transparência, podendo ser alterados em *runtime* pelo clique de um mouse, por exemplo.

O Java3D é definitivamente uma poderosa ferramenta para manipulação de objetos em ambientes tridimensionais. Para que *browsers* possam executar aplicativos Java3D, é necessária a instalação da versão 1.4.0 ou superior desta API, que pode ser adquirida gratuitamente em *http://java.sun.com/products/java-media/3D/*.

2.3.3 Sockets

Um *socket* é um dos pontos de um link de comunicação entre dois programas executados em rede, esta comunicação é feita através da porta pré-definida entre ambos. Classes *sockets* são usadas para representar a conexão entre um programa cliente e um programa servidor. O pacote *java.net* possui as classes: *Socket* e *ServerSocket*, que implementam o lado cliente e lado servidor da conexão, respectivamente.

Um ponto de conexão é uma combinação de um endereço IP e um número de porta ("Endereço IP":"# da Porta"). Toda conexão TCP pode ser unicamente identificada por estes dois parâmetros. Deste modo, pode-se ter múltiplas conexões entre o cliente e o servidor, utilizando *Threads*. [16]

Sockets trabalham com dois tipos de protocolos: UDP e TCP. O UDP (*User Datagram Protocol*) é um protocolo não orientado a conexão, neste caso, dados extras são mandados para identificar o cliente e o servidor, já que não há conexão estabelecida. No caso do TCP (*Transfer Control Protocol*), um método de comunicação por *Stream*, o cliente solicita a conexão para o servidor, que abre a conexão entre eles permitindo a troca de informações em ambas direções até que a conexão seja encerrada por qualquer das partes. [15]

Os *sockets* podem utilizar qualquer porta disponível menor que 65553. Muitas portas são reservadas para outros fins, como HTTP, SMTP, POP3, etc.

A tabela 2.3.1 descreve as principais portas ocupadas.

Tabela 2.3.1 – Lista de portas de comunicação reservadas

# Porta	Finalidade
80	HTTP (Hyper Text Transfer Protocol)
21	FTP (File Transfer Protocol - Control)
110	POP3 (Post Office Protocol version 3)
25	SMTP (Simple Mail Transfer Protocol)
119	NNTP (Network News Transfer Protocol)
23	telnet
194	IRC (Internet Relay Chat)
8080	Proxy
53	DNS (Domain Name Server)
20	FTP-data (File Transfer Protocol - Default Data)
70	Gopher
79	Finger
69	TFTP (Trivial File Transfer)

Diversos programas utilizam outras portas para se comunicar, porém não são reservadas pelo sistema operacional e são livres para utilização. Ficam então disponíveis para utilização.

Em um aplicativo servidor, o *ServerSocket* deve ser instanciado na porta desejada e esperar por uma solicitação de conexão por parte do cliente, como segue:

ServerSocket server = new ServerSocket (5500);

Socket connection = server.accept();

Após estabelecida e aceita a conexão, resta receber os *Streams* de entrada ou enviar os *Streams* de saída (ver figura 2.3.5), da seguinte forma:

InputStream in = connection.getInputStream();

OutputStream out = connection.getOutputStream();

out.flush();

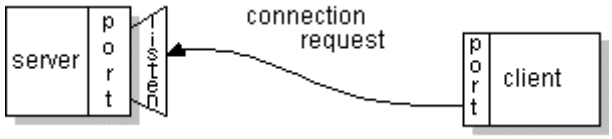


Figura 2.3.5 – Conexão por socket *[15]*

2.3.4 Comunicação com paralela

A *Sun Microsystems* disponibiliza para os desenvolvedores Java a API *Java Communications*, esta API permite a comunicação de uma aplicação com outras tecnologias, como Smart Cards, sistemas embarcados, dispositivos de impressão, fax, modems, terminais e equipamentos robóticos. Esta API (também conhecida como *javax.comm*) oferece acesso ao hardware RS-232 (porta serial) e acesso ilimitado ao IEEE-1284 (porta paralela), no modo SPP. [18]

Apesar de ser uma API completa, a *Sun* registrou um *bug* na leitura de dados pela porta paralela para o sistema operacional Windows (Nov.,2006) . Outra solução para escrita e leitura pela paralela utiliza o recurso JNI (*Java Native Interface*), que é uma ferramenta de programação que permite o código Java chamar ou ser chamado por aplicações nativas (programas específicos para hardware e sistema operacional) e bibliotecas escritas em outras linguagens, como C, C++ e assembly. [19]

JNI é muitas vezes dito como "válvula de escape" para desenvolvedores Java porque permite adicionar funcionalidades para as aplicações Java que as API´s Java não podem prover. Muitas vezes JNI também é utilizado para resolver cálculos complicados e críticos no tempo, pois os códigos nativos são potencialmente mais rápidos que código JVM (*Java Virtual Machine*). [19]

A fim de conseguir total acesso à porta paralela do PC, Juan Gabriel (*http://www.geocities.com/Juanga69/parport/*, acessado em Dez, 2006) implementou uma API chamada *ParallelPort* em que o acesso à paralela é realizada utilizando JNI e linguagem nativa C. Em C, as funções _inp() e _outp() facilmente lê e escreve, respectivamente, no endereço desejado. Em Java, os métodos correspondentes são lpt.read() e lpt.write(), chamados após instanciado *ParallelPort* lpt = new *ParallelPort*(0x378).

Os sistemas operacionais Windows NT/2000/XP limitam o uso da paralela através de um *flag* no registrador de impressão. Quando é necessário ler ou escrever na porta paralela, os *drivers* de impressão desabilitam este *flag* para ter acesso. Outros programas somente podem utilizar a paralela se desabilitar este *flag* no registro do Windows, o *driver UserPort* (pode ser obtido em *webassembly.graco.unb.br/UserPort.zip*) o faz e libera o uso da paralela para qualquer programa em execução. A instalação do *driver* é simples e está descrita em *UserPort.pdf,* dentro da pasta *UserPort*.

2.3.5 XML

XML (*eXtensible Markup Language*) é uma linguagem usada para representar dados em forma de texto, intercalados por "marcações" que descrevem as propriedades dos dados. Especificamente, a marcação fornece um mecanismo para adicionar metaconteúdo e informações sobre a estrutura de um documento.

As marcações são delimitadas por *tags*, que proporcionam um meio de inserir os dados formados por caracteres em um elemento. Um elemento é um bloco de construção, básico da XML. Eles podem conter outros elementos aninhados em seu interior, chamados de *sub-elementos*. Um documento é composto de somente um elemento externo (*root*), que contém outros elementos e/ou dados de caracteres, podendo cada *sub-elemento* conter outros *sub-elementos* intercalados por dados de caracteres e assim por diante.[20]

Documentos XML separam o conteúdo de um documento de como ele deve ser formatado, isso simplifica o desenvolvimento e a manutenção. Portanto, as descrições dos dados podem ser repetidamente revisadas quando houver alterações na área abordada. Isso torna a XML particularmente útil para áreas cujas informações sejam complexas e passem por revisões freqüentes.

A XML sozinha não tem por finalidade a apresentação ao usuário final, mas quando empregados junto com folhas de estilo, os documentos XML podem ser facilmente visualizados. Os documentos XML fornecem uma maneira de capturar os dados, e as folhas de estilo como a eXtensible Stylesheet Language (XSL) disponibilizam mecanismos de conversão da XML em HTML.[20]

Uma importante razão para se utilizar XML é que ela pode ser empregada como banco de dados. Um banco de dados é um conjunto de dados armazenados de maneira que persistam e possam ser manipulados. Por persistência entende-se que os dados permanecerão consistentes depois que o

trabalho for concluído. A maioria dos dados são persistentes, inclusive os de texto, as planilhas e as figuras. [20]

Basicamente, o acesso a dados XML pode ser realizado de duas maneiras: utilizando SGBD ou JDOM.

O SGBD é um *driver* de acesso a dados, através dele o desenvolvedor pode alterar, consultar e incluir dados aos documentos XML. Usar SGBD XML é especialmente apropriado para captura de dados de uma área com relacionamentos hierárquicos complexos, como nos bancos de dados científicos, em grandes empresas e em sistemas de produção [20].

JDOM é uma API implementada por Jason Hunter e Brett McLaughlin que, utilizando as API´s DOM (*Document Object Model*) e SAX (*Simple Api for XML*), simplificam o acesso a dados em arquivos XML através da linguagem Java [21]. Documentos XML podem ser lidos, escritos e pesquisados utilizando a API JDOM. Utilizar JDOM é simples porque podem ser carregados em arquivos JAR juntamente com a aplicação, não necessitando de instalações extras quando executado, ficando a cargo do desenvolvedor incluir as classes no *CLASSPATH*. A API JDOM está na versão 1.0 (Dez., 2006) e pode ser obtida em *www.jdom.org*, onde também há informações sobre instalação.

2.3.6 Arquitetura *Client-Server*

A arquitetura Cliente-servidor é uma arquitetura de rede que separa o cliente (geralmente uma aplicação que usa uma interface gráfica) do servidor. O cliente envia pedidos para o servidor, este processa e retorna o pedido processado. Alguns tipos de servidores são: servidores de aplicação, servidores de arquivos, servidores de banco de dados e servidores de e-mail. De todos estes tipos, a arquitetura mantém a mesma, porém com propósitos diferentes. [22]

Diversas tecnologias podem ser adotadas no uso das arquiteturas Cliente-servidor, isso dependerá da estrutura local ou do tipo de aplicação. O balanceamento de carga entre cliente e servidor determina o porte da rede utilizada e o porte do servidor. Um servidor pode prover diversos serviços, cabe ao gerente de TI decidir por qual tecnologia e balanceamento utilizar.

Em uma rede local, por exemplo, diversos computadores podem conectar-se a um único servidor que provê um ou mais serviços (figura 2.3.6).

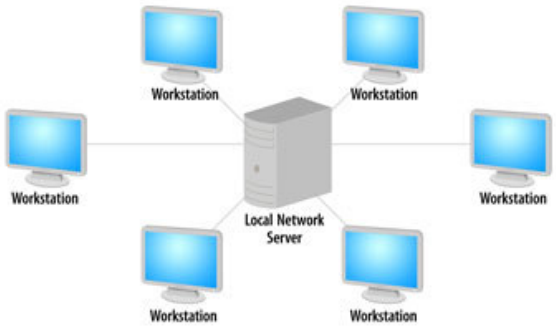
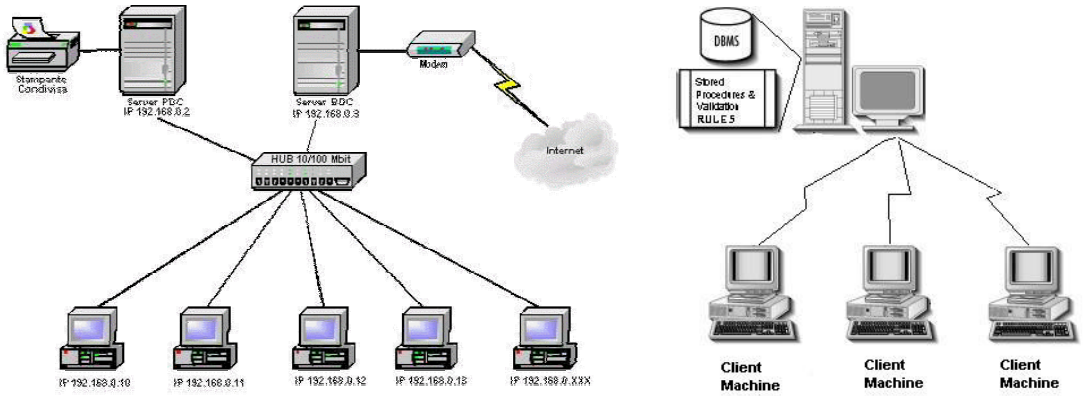


Figura 2.3.6 – Clientes conectados a um servidor em uma rede local [22]

Outro exemplo de tecnologia pode ser adotada utilizando mais de um servidor (figura 2.3.7 - a), sendo um servidor de impressão e outro de Internet. Pode-se ter ainda um servidor de múltiplos bancos de dados, gerido por um único PC (figura 2.3.7 - b).



(a) (b)

Figura 2.3.7 – Exemplos de tecnologias da arquitetura Cliente-Servidor [22]

2.4 Interface de Comunicação

2.4.1 Porta Paralela

As portas de comunicação do PC constituem em Portas Seriais (COM1, COM2 e USB) e Portas Paralelas (LPT1 e raramente LPT2 e LPT3). As portas paralelas são utilizadas geralmente para conectar impressoras e scanners e utilizam um conector DB25 fêmea localizado na parte traseira do PC. Esta porta possui 8 *bits* de dados (*Data Port*), 5 *bits* de Status (*Status*) e 4 *bits* de Controle (*Control*) (figura 2.4.1), totalizando 17 bits de comunicação.

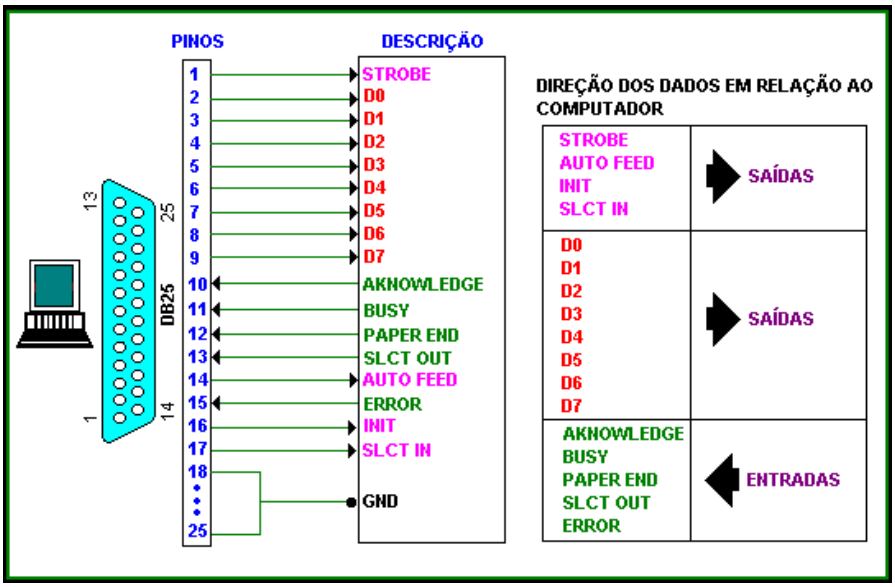


Figura 2.4.1– Pinagem do conector da porta paralela [23]

Os *bits* de dados, status e controle são acessados por endereços seqüenciais a partir do endereço dos bits de dados. A tabela 2.4.1 mostra estes endereços para as portas LPT1, 2 e 3.

Tabela 2.4.1 – Mapa de endereços das portas paralelas [24]

	Dados	Status	Controle
LPT1	0x378	0x379	0x37a
LPT2	0x3bc	0x3bd	0x3be
LPT3	0x278	0x279	0x27a

As portas paralelas utilizam o padrão TTL (*transistor-transistor logic*). Neste padrão, o nível será interpretado como alto (*high*) quando a tensão for entre 2,4 e 5 volts e baixo (*low*) quando a tensão for entre 0 e 0,8 volts [24]. Quase todas saídas na porta paralela utilizam lógica verdadeira, isto é, um valor *low* escrito na saída representará um nível baixo no pino, porém as saídas “select in”, *INIT*, *strobe* e a entrada *busy* são invertidas, o que representará nível lógico oposto ao escrito ou lido.

O projeto utiliza a porta LPT1 para comunicação com o controlador do robô e utiliza 7 de suas 8 saídas e a entrada *busy*, este último indica se o robô está no estado ocupado (em movimento) ou livre. Uma das 7 saídas é responsável por indicar que os *bits* foram escritos e que uma nova tarefa foi solicitada, o restante dos 6 indicam a próxima tarefa.

2.4.2 L298

O componente L298 constitui duas pontes completas transistorizada capaz de suprir alta corrente (1A por saída) e oferecer tensões de até 46V. Suas entradas aceitam sinais TTL e suas saídas podem ativar relés, solenóides, motores DC e de passo. Os emissores dos transistores inferiores são conectados a um resistor sensor de corrente externo. O componente possui duas entradas de alimentação, um para a tensão de saída e outra para o nível lógico (figura 2.4.2). [25]

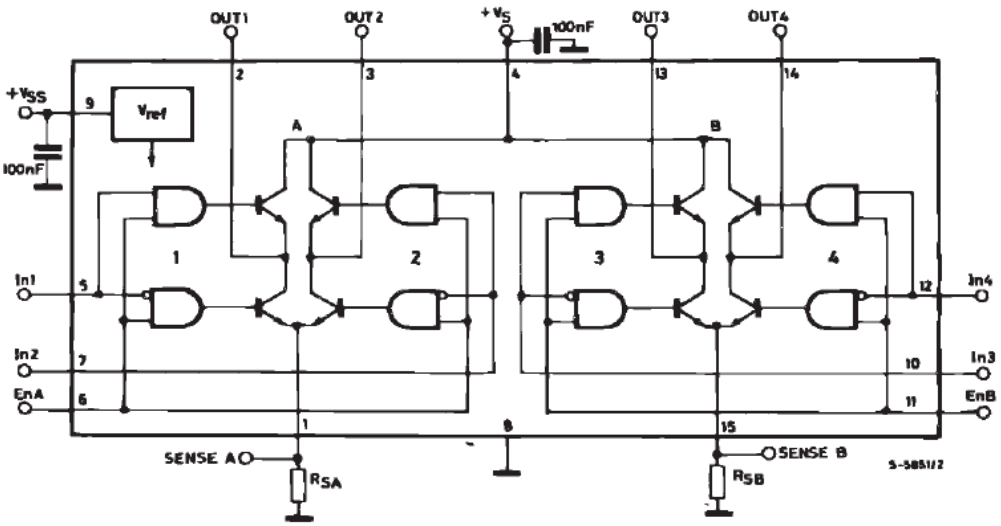


Figura 2.4.2 – Diagrama esquemático interno

Por ser um componente que utiliza chaveamento a transistor, o L298 é relativamente lento na resposta ao chaveamento (≈ 3.5 μs) [25], sendo restrito utilizações com PWM em alta freqüência. Porém, este tempo é insignificante no contexto do tempo de posicionamento do robô e orientação de tarefas.

Por possuir portas lógicas como dispositivos de entrada, estas podem ser consideradas como de alta impedância, portanto drenam corrente insignificante da porta paralela.

As saídas transistorizadas, quando operando em saturação, possuem uma queda de tensão de aproximadamente 0,2 V, o que faz garantir pouco menos de 24V (saída da interface de comunicação WebAssembly) sobre a carga.

3. METODOLOGIA DO PROJETO WEBASSEMBLY

Por ser um projeto que envolve desenvolvimento mecânico, elétrico e computacional, logicamente estas grandes áreas foram separadas em tópicos nesta documentação. Porém, o projeto, construção e implantação destas áreas foram abordados simultaneamente ou alternadamente devido à correlação entre elas e a necessidade de homologação conjugada.

3.1 Restrições de projeto

Após a definição do objetivo do projeto, a primeira abordagem do trabalho foi relacionar as restrições que eram impostas e os caminhos para contorná-las.

As restrições orçamentárias não serão discutidas por ser um projeto de baixo custo e sem direcionamento fixo de verba, porém o gasto total está listado.

3.1.1 Robô ASEA IRB6

A principal ferramenta utilizada no projeto e protagonista da motivação foi o robô industrial ASEA IRB6, localizada no GRACO – UnB (figura 3.1.1). Após *retrofitting* realizado por projetos anteriores, este antigo robô (1974) está totalmente operante e possui longo alcance (1114 mm) e boa acurácia (décimos de milímetros). Sua capacidade de carga de 6kg é totalmente suficiente para transportar blocos LEGO.

Apesar de robusto, o IRB6 é um robô leve por ter os elos fabricados em alumínio [26].

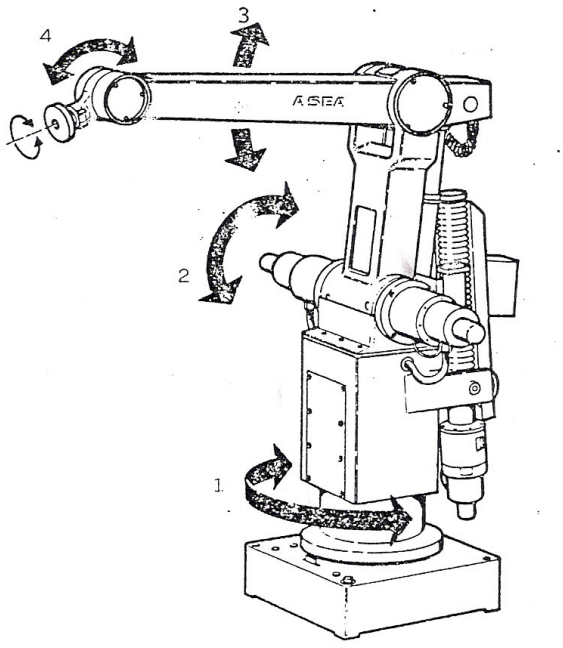


Figura 3.1.1 – Robô industrial ASEA IRB6 [27]

Apesar das vantagens, o IRB6 não possui a mobilidade *yaw* no movimento do punho (*roll*, *pitch* e *yaw*) por possuir 5 graus de liberdade, e não 6. A falta de um grau de liberdade não qualifica negativamente o robô, apenas o limita a que tipo de trabalho é destinado. O IRB6, então, é mais apropriado a operações de soldagem e funilaria do que processos de montagem, ou seja, é destinado a operações que não requer orientação na extremidade do manipulador, apenas posição. Portanto, alguns movimentos serão prejudicados na montagem.

Outra limitação do IRB6 é sua comunicação com o meio externo, pois não possui nenhum tipo de comunicação serial. Suas entradas e saídas são restritas a 8 canais digitais de saída e 7 de entrada (0-24V), sendo que 2 dos canais de saída são utilizados para acionar e desativar a garra, restando 6 canais de saída disponíveis.

Acoplado à *flange* do robô está a garra pneumático PGN 260-1, marca *SCHUNK*. Esta garra oferece um deslocamento máximo de 160mm e força de 1200 N quando sob pressão de 6 bar, a tabela 3.1.1 mostra esta e outras especificações. O acionamento da garra é realizado através de uma válvula pneumática marca *Parker,* controlada por dois solenóides (24V/3 W) sem mola. As dimensões da garra estão detalhadas na figura 3.1.2.

Tabela 3.1.1 – Parâmetros de fabricação da garra PGN 160-1[28]

Type		**PGN 160-1**
Id.-No.		**370 104**
Stroke per finger	mm [inch]	16 [0.630]
Gripping force at 6 bar [87 psi]*	N [lbf]	1200 [270]
Workpiece weight (recommended) **	kg [lbs]	6 [13.228]
Air consumption per double stroke	cm^3 [$inch^3$]	140 [8.540]
Opening time		0.10 s
Closing time		0.10 s
Mass	kg [lbs]	2.4 [5.291]
Own mass moment of inertia I_y	kg cm^2 [lb-$inch^2$]	64 [21.875]
Repeatability ***	mm [inch]	0.02 [0.001]
Max. finger length	mm [inch]	160 [6.299]

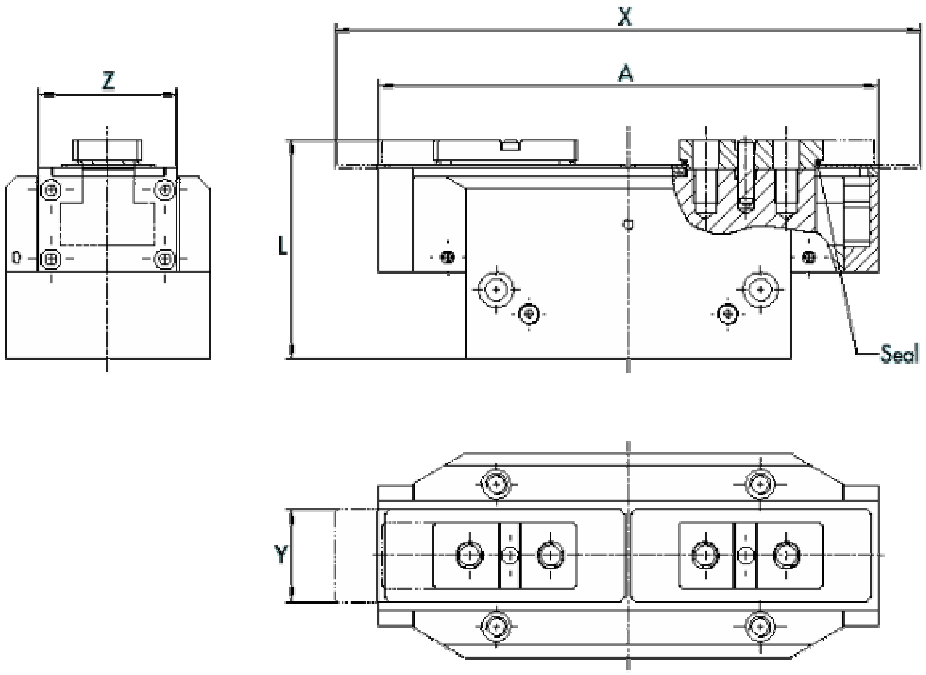


Figura 3.1.2 – Dimensões da garra PGN 160-1 [28]

A fim de determinar a posição da garra, este tem acoplado um sensor de posição magnético (FPS-F5), também fabricando por SCHUNK. Pode-se neste sensor configurar 5 posições diferentes no curso da garra, incluindo as posições de garra aberta e garra fechada. Este projeto não utilizará este sensor pois não é necessário reconhecer o estado garra nem a dimensão do objeto manipulado.

3.1.2 Peças LEGO

Diversos são os tipos de produtos que necessitam de montagem e utilizam diferentes tipos de peças. As peças LEGO foram escolhidas pois são facilmente encontradas no mercado, são de baixo custo e, principalmente, são peças desenvolvidas para montagens. Por possuírem chanfros nos encaixes, os blocos LEGO facilitam a montagem; por serem fabricados de plástico, adicionam complacência ao sistema e permitem que testes sejam feitos sem danificar os blocos, o robô e o *finger* (figura 3.1.3).

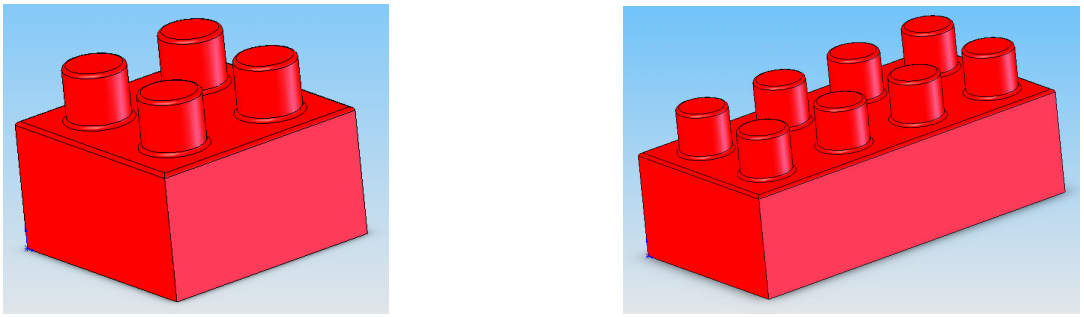


Figura 3.1.3 – Modelos de features *disponíveis*

Os blocos utilizados são peças de um *kit* desenvolvido para crianças de 0 a 4 anos, são as maiores peças LEGO de plástico do mercado e possuem dimensões manipuláveis pelo robô. As *features* de tamanho, as *features* de cor e a quantidade de cada tipo de peça são mostradas na tabela 3.1.2.

Tabela 3.1.2 – Peças LEGO disponíveis

Feature de cor	*Feature* de dimensão	Quantidade
Amarelo	60 mm x 60 mm	4
Azul	60 mm x 60 mm	4
Rosa	60 mm x 60 mm	4
Azul	60 mm x 120 mm	2
Rosa	60 mm x 120 mm	1

Apesar de terem sido favoráveis ao projeto por todos os aspectos citados, especificamente as peças maiores (fabricante *TANDY*) apresentaram inconformidades na dimensão. Portanto, algumas peças sofrem interferência ao montar, prejudicando a montagem robotizada. O problema na qualidade das peças foi citado na seção 2.1.1 *(Escolha do método de montagem)* e a literatura cita como peças que devem ser rejeitadas.

3.1.3 FMC e WebAssembly

O sistema FMC e o WebAssembly compartilham o robô IRB6 (braço e I/O´s); estes sistemas estão localizados em extremidades opostas no trilho que suporta e oferece um grau de mobilidade prismática ao robô (ver figura 3.1.4). Para que o carro do robô pudesse ser fixado em uma extremidade ou outra do trilho, foram construídos fixadores desparafusáveis que permitem fácil *setup* e rigidez.

Figura 3.1.4 – Layout dos sistemas FMC e WebAssembly

A extremidade FMC é composta do robô, torno CNC, micrômetro laser e paleteira. Esse projeto aborda questões de automação da manufatura e Manufatura Assistida por Computador (CAM).

A extremidade WebAssembly é composto da mesa de montagem e o robô com o devido *finger* acoplado à flange.

Além de compartilhar o meio físico e a ferramenta robô, o sistema FMC e WebAssembly compartilham a fiação que dá acesso às I/O´s do controlador. Este detalhamento segue nos próximos capítulos.

3.2 Modelo funcional IDEF0

IDEF0 (*Integrated DEFinition Methods*) é um método que descreve decisões, ações e atividades de um sistema. Modelos de IDEF0 apóiam a comunicação entre analista de projeto e cliente, aumentando o grau de envolvimento da equipe através de representações simples de decisões. Como ferramenta de análise, IDEF0 apóia o desenvolvedor em identificar funções, o que é preciso executar para estas funções e ajuda a identificar atividades certas e erradas. IDEF0 então deve ser o primeiro passo para o desenvolvimento do sistema [29].

Graficamente, o método IDEF0 representa funções como “caixas” e as interfaces de entrada e saída como “setas” que entram ou saem das caixas. Para representar funções, caixas interagem simultaneamente com outras caixas através das setas, que identificam a interface.

O modelo IDEF0 faz uma abordagem *top-down* do problema e visa acompanhar a modelagem desde a concepção inicial até o detalhamento completo. A figura 3.2.1 e 3.2.2 representam o modelo geral criado no software *AI0 WIN* (disponível em *www.idef.com*).

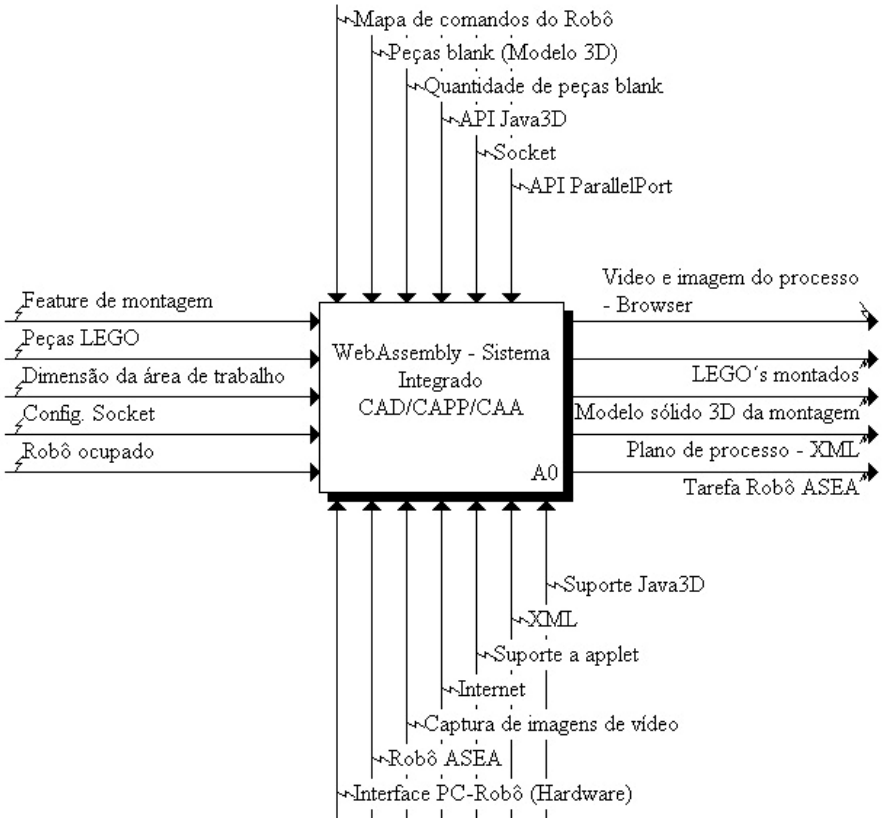


Figura 3.2.1 – Folha A0 do modelo IDEF0 - WebAssembly

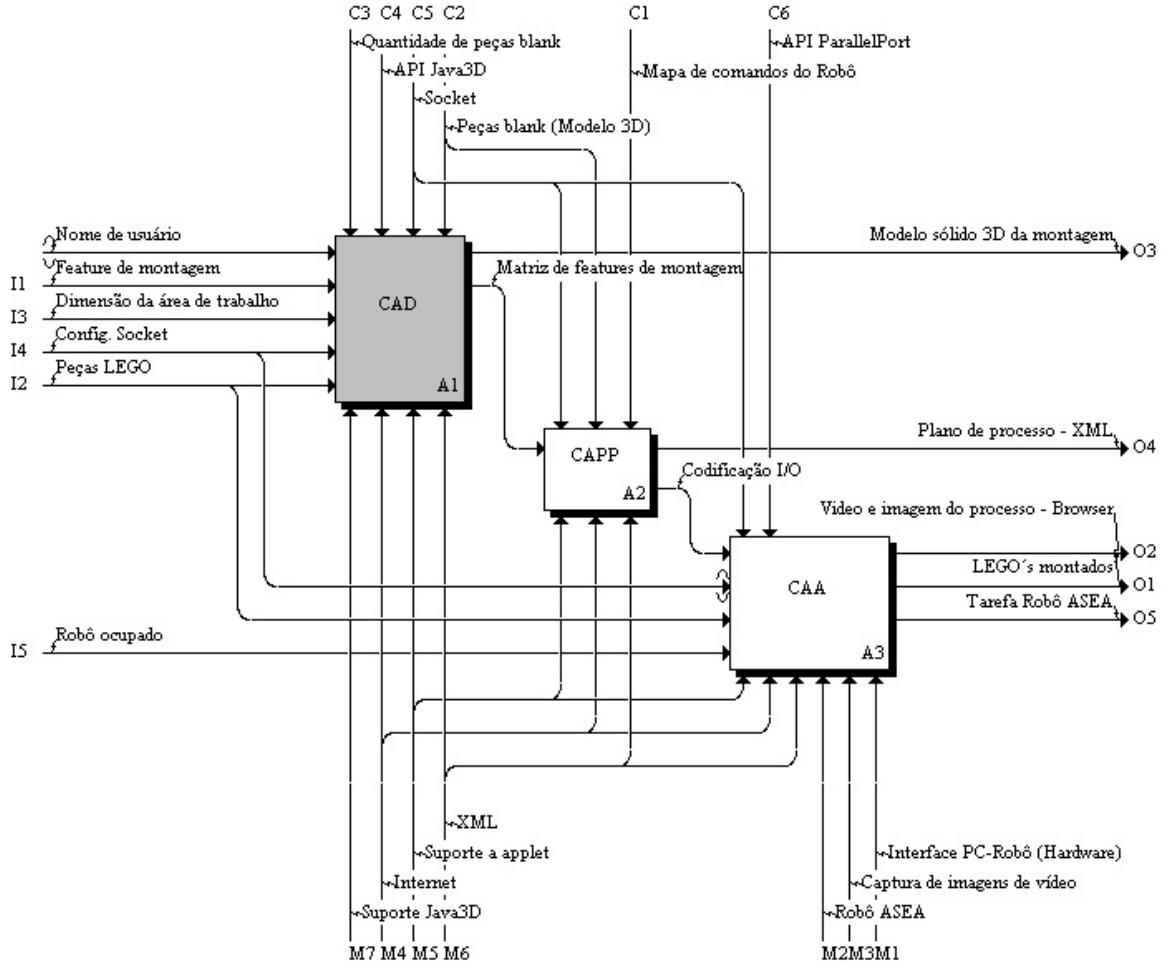


Figura 3.2.2 – Detalhe da folha A0 do modelo IDEF0 - WebAssembly

O modelo IDEF0 deve ser detalhado quanto necessário para que o modelo seja esclarecedor ao cliente e desenvolvedor. As figuras 3.2.3, 3.2.4 e 3.2.5 constituem o detalhamento das folhas A1, A2 e A3, respectivamente.

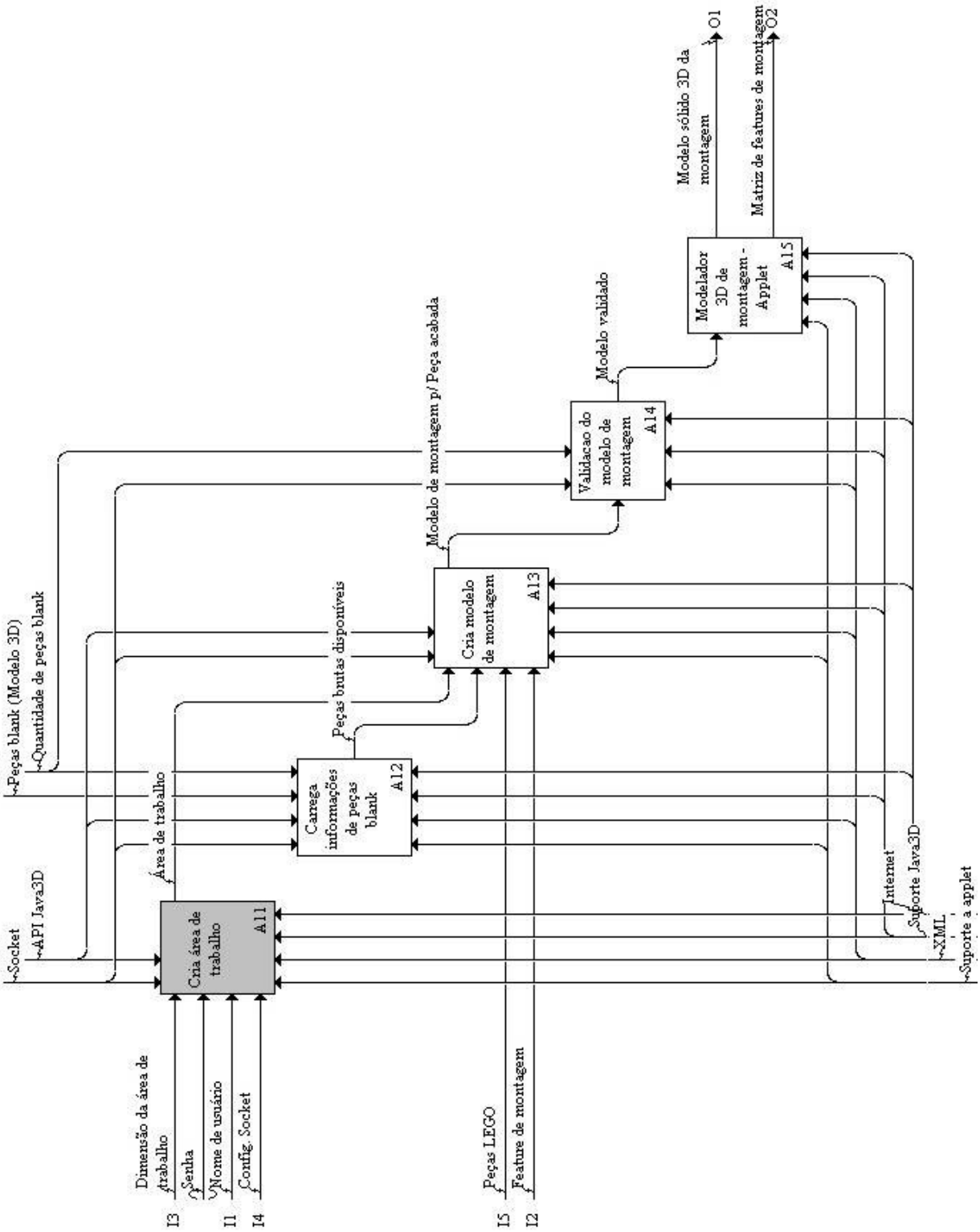


Figura 3.2.3 – Detalhe da folha A1 (CAD) do modelo IDEF0 - WebAssembly

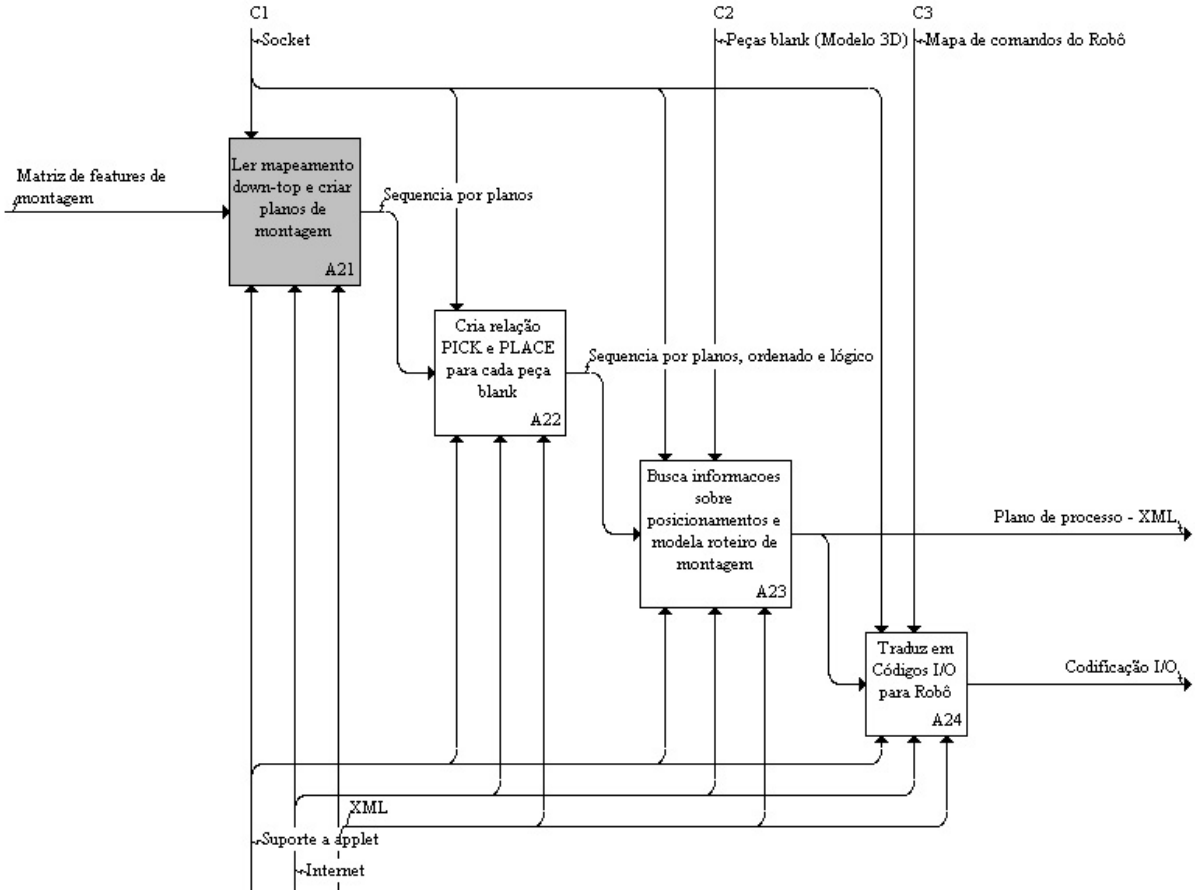


Figura 3.2.4 – Detalhe da folha A2 (CAPP) do modelo IDEF0 - WebAssembly

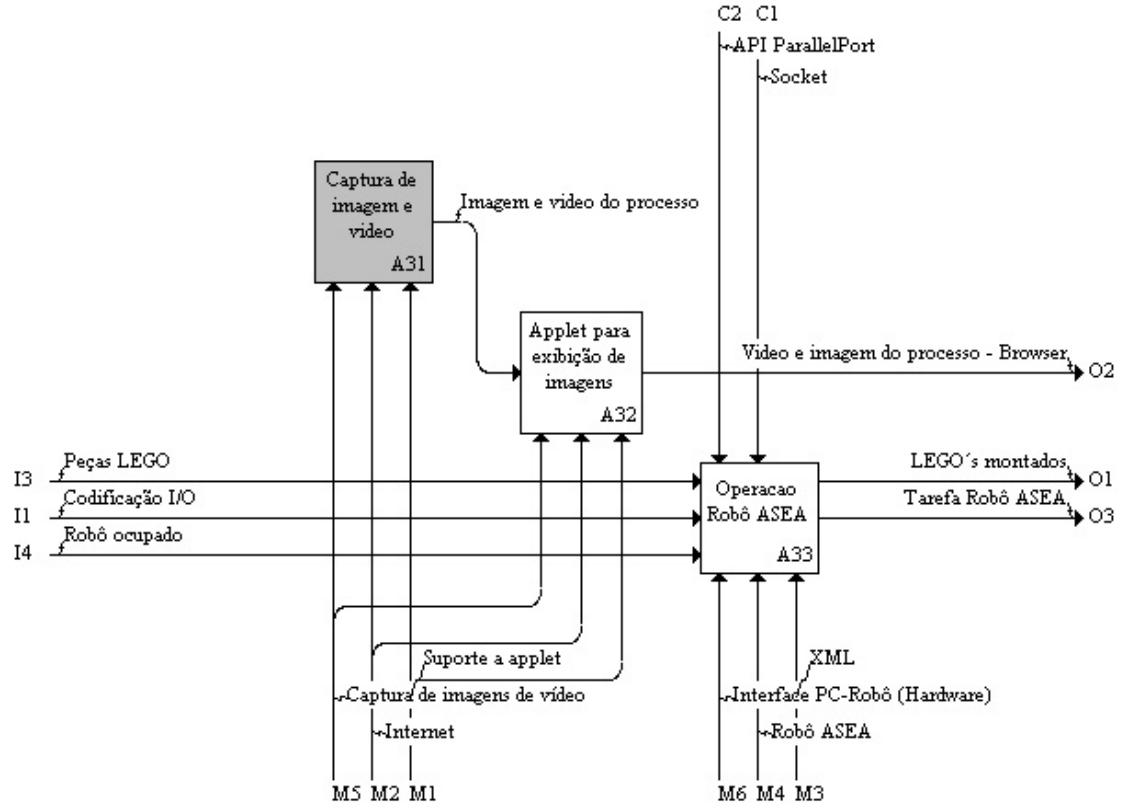


Figura 3.2.5 – Detalhe da folha A3 (CAA) do modelo IDEF0 - WebAssembly

3.3 Cronograma

O cronograma idealizado no início do projeto teve uma modificação em virtude de uma atividade extra e teve como ponto de partida as implementações realizadas no projeto 1. O cronograma apresentou variação final de 3 dias e abordou o projeto e construção das vertentes elétrica, mecânica e computacional. A figura 3.3.1 representa o *Gantt Chart* simplificado, sem especificar os *stakeholders*.

Figura 3.3.1 – Gantt Chart do cronograma WebAssembly

3.4 Orçamento

Alguns itens e custos, com mão-de-obra, não estão relacionados pois foram disponibilizados pelos laboratórios SG-9 e SG-11 ou executados pelos técnicos do SG-9. Itens adicionais seriam barras trefiladas perfil quadrado, chapas de aço, dobradiças, barras chatas e computador para servidor. A tabela 3.4.1 consta, portanto, os gastos referentes ao reembolso direto pela FINATEC e não ao gasto total real do projeto.

Tabela 3.4.1 – Orçamento projeto WebAssembly

Projeto Elétrico

Item	Descrição	Qnt	Valor unit.	Valor total
1	Relés T3RC Metaltex 24V c/ soquete	4	R$ 52.00	R$ 208.00
2	L298	2	R$ 12.00	R$ 24.00
3	Cabo/Conectores	1	R$ 10.00	R$ 10.00
4	Componentes diversos	1	R$ 15.00	R$ 25.00

Projeto Mecânico

Item	Descrição	Qnt	Valor unit.	Valor total
1	Peças LEGO	1	R$ 52.00	R$ 52.00
2	Barra tecnil (1500 mm x 16 mm)	1	R$ 20.00	R$ 20.00
3	Tarugo alumínio 120 mm (*finger*)	1	R$ 93.00	R$ 93.00
4	Parafusos, porcas, arruelas	22	R$ 2.50	R$ 55.00

Total: R$ 487.00

4. APLICATIVOS CAD/CAPP/CAA

Os programas desenvolvidos foram implementados na arquitetura cliente-servidor, sendo que o cliente executa a aplicação *applet* via internet enquanto a máquina servidor executa seu aplicativo no PC conectado ao controlador do robô e faz o acesso aos dados em XML. O *applet* cliente disponibiliza a ferramenta CAD e o aplicativo servidor disponibiliza as ferramentas CAPP e CAA.

A comunicação cliente-servidor é realizada através de *sockets*, sobre o protocolo TCP/IP.

4.1 WebAssembly – *Client Side*

A aplicação *applet*, executada no *browser* do usuário via internet (*webassembly.graco.unb.br*), oferece recursos de configuração do *socket*, edição das dimensões da área de montagem, edição de sua montagem através de uma lista de peças disponíveis e visualização do plano de processo de montagem. A figura 4.1.1 mostra a GUI da aplicação *applet client side*.

A ferramenta CAD é gerada por um *Canvas* 3D, utilizando API Java3D, que cria um modelo matricial de 3 dimensões, ao qual o usuário insere as peças camada por camada nas células disponíveis (visualizada pela cor cinza).

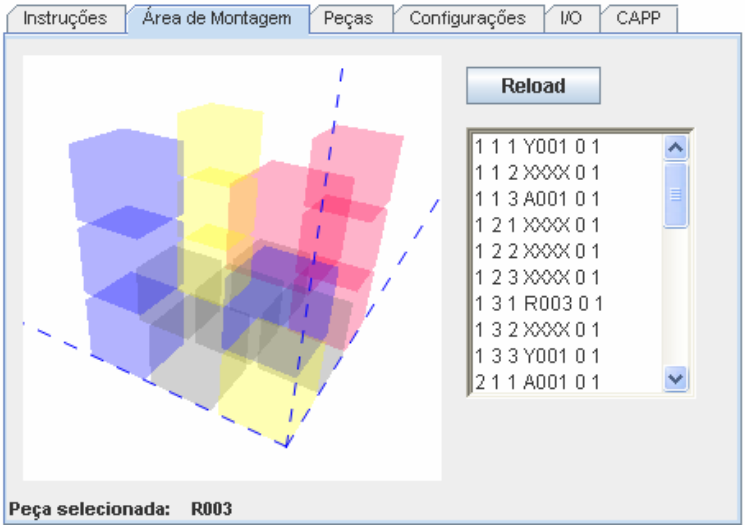


Figura 4.1.1 – GUI applet client side

4.1.1 Instalação

Para executar o WebAssembly cliente - *applet,* é necessário possuir os seguintes quesitos Java:

- instalar o J2RE 1.4.2 ou superior (http://java.sun.com/javase/downloads/index.jsp).

Obs.: muitos computadores já possuem *Java Runtime Environment* instalado, é necessário que o usuário verifique a versão;

- instalar a API Java3D 1.4.0 ou superior (*http://java.sun.com/products/java-media/3D/*).

Obs.: esta API será responsável por carregar o C*anvas* 3D da área de montagem;

- substituir o arquivo *java.policy* pelo disponível em *webassembly.graco.unb.br/java.policy* ou alterar o *java.policy* de acordo com a seção 2.3.1 (*Applets Java*). Esses passos darão acesso à comunicação por *socket* com o servidor. Sem esta substituição o *applet* não tem permissão sobre a porta de comunicação.

Obs.: é aconselhável retornar a configuração anterior a fim de manter o nível de segurança estabelecida pela *Sun Microsystems*.

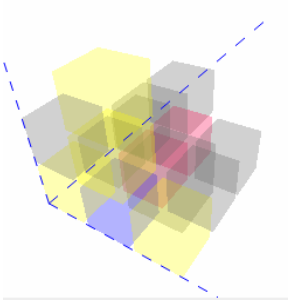
Na aba I/O encontra-se o campo de edição URL, este campo contém o endereço da máquina servidor, a priori este endereço deve ser http://164.41.17.7:5500 (# IP:# porta), sendo que o valor 5500 corresponde à porta de comunicação configurada no *socket* servidor e não deve ser alterada.

4.1.2 Operação

Após instalado os requisitos Java, a página apresentará o *applet* WebAssembly e o *applet* do servidor de vídeo do GRACO.

O *applet* WebAssembly iniciará na aba Créditos, que apresenta os créditos de implementação do *applet*. As abas seguintes terão as seguintes funcionalidades:

- Área de montagem:



Canvas 3D

Canvas 3D: neste local o usuário inserirá as peças que comporão a montagem.

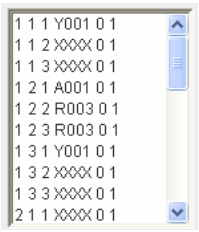
Os cubos representam peças colocadas (cubos na cor da peça) ou espaços que podem adicionar peças (cubos cinzas).

As peças LEGO de dimensão 60mm x 60mm ocupam uma célula dessa matriz tri-dimensional. As peças 60mm x 120mm ocupam duas células.

Ao clicar com o mouse em algum cubo cinza (vazio), após selecionar alguma peça, o sistema pede confirmação ao usuário, insere a peça e libera a posição correspondente na camada imediatamente superior.

* Ao confirmar a inserção da peça, o sistema decrementará uma unidade do palete.

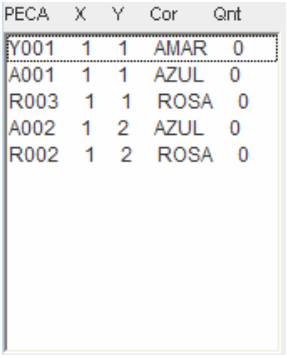
Caso o usuário necessite de outro ângulo para melhor visualizar seu modelo de montagem, pode-se clicar, segurar e arrastar o botão esquerdo do mouse dentro da área *Canvas*. O botão direito pode ser utilizado para mover a câmera para esquerda ou direita. O *scroll* é utilizado para *zoom*.



Este é um objeto *list* representando o espelho do arquivo XML (*areaMontagem.xml* – no servidor) que armazena o local onde o usuário inseriu as peças, não é editável. Corresponde à área de montagem interpretável pelo aplicativo.

O botão *Reload* recarrega o modelo 3D.

- Peças:



Este *list* lista as peças que estão disponíveis para montagem nos paletes e suas quantidades. Representam o espelho do arquivo XML *pecas.xml* localizado no servidor.

Da esquerda para direita teremos: Código da peça, largura da peça, comprimento da peça, cor da peça e quantidade disponível da peça.

Para inserir uma peça, o usuário deve selecionar uma peça no *list*, clicar no botão inserir e depois selecionar a célula desejada na aba Área de Montagem.

- Configurações:

Esta aba configura a dimensão da área de montagem. O valor máximo para cada parâmetro é 3 devido às limitações físicas na mesa de montagem.

Cada parâmetro está relacionado pelo sistema de coordenadas na figura 4.1.2.

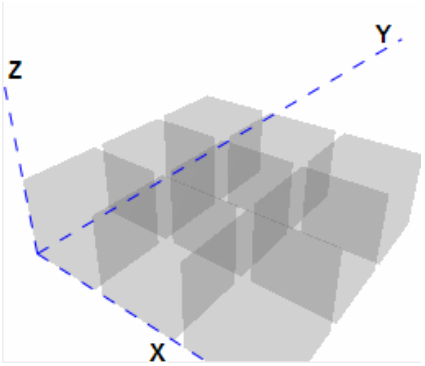
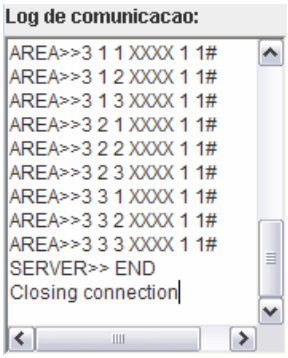


Figura 4.1.2 – Sistemas de coordenadas do Canvas 3D

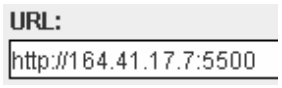
- I/O:



Os objetos nesta aba são referentes à comunicação com o servidor: O *list* mostra o *log* de comunicação entre cliente e servidor.

O campo URL permite alterar o endereço e porta de comunicação. *Default*: http://164.41.17.7:5500

O botão *Desconectar* desconecta o servidor e o torna inativo a receber outras comunicações, ou seja, força o servidor a fechar o *socket*. * O restabelecimento da comunicação somente poderá acontecer com o pedido de conexão no servidor.



O botão *Habilitar* força o servidor a alterar o indicador de habilitação. Este indicador é reiniciado quando uma montagem é concluída pelo usuário e objetiva assegurar que nenhuma montagem ocorrerá após a outra. Este procedimento ainda restabelece 4 unidades para cada peça de dimensão 60x60 e nenhuma unidade para as peças de dimensão 60x120.

- CAPP:

A aba CAPP mostra a seqüência de montagem criada pelo algoritmo no servidor.

Para executar a montagem, o usuário deve clicar no botão *CAPP!!*, que dispara o algoritmo CAPP. O algoritmo lê os arquivos XML *areaMontagem.xml* e *pecas.xml* e cria o arquivo *CAPP.xml*, que corresponde à seqüência de operações (tarefas) ao robô. A primeira coluna corresponde ao número da operação, a segunda descreve a operação e a terceira representa o código a ser escrito na porta paralela.

O botão *Execute!!* envia o comando para o servidor proceder com a escrita na porta paralela.

4.1.3 UML

A linguagem de modelagem (UML) foi utilizada para auxiliar a visualização do modelo e comunicação entre os objetos. A figura 4.1.3 mostra o diagrama de classes do WebAssembly *applet*. Os demais diagramas não foram especificados pois são específicos à engenharia de software e necessário apenas a *softwares* de maior complexidade.

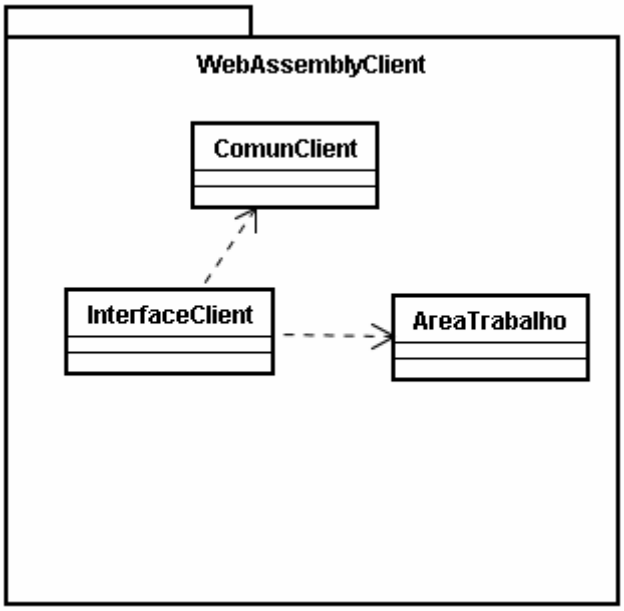


Figura 4.1.3 – Diagrama de classes WebAssembly cliente

4.1.4 Estrutura área de montagem (Java3D)

A estrutura Java3D foi montada seguindo o algoritmo fornecido pela *Sun* [15] resultando na estrutura da figura 4.1.4.

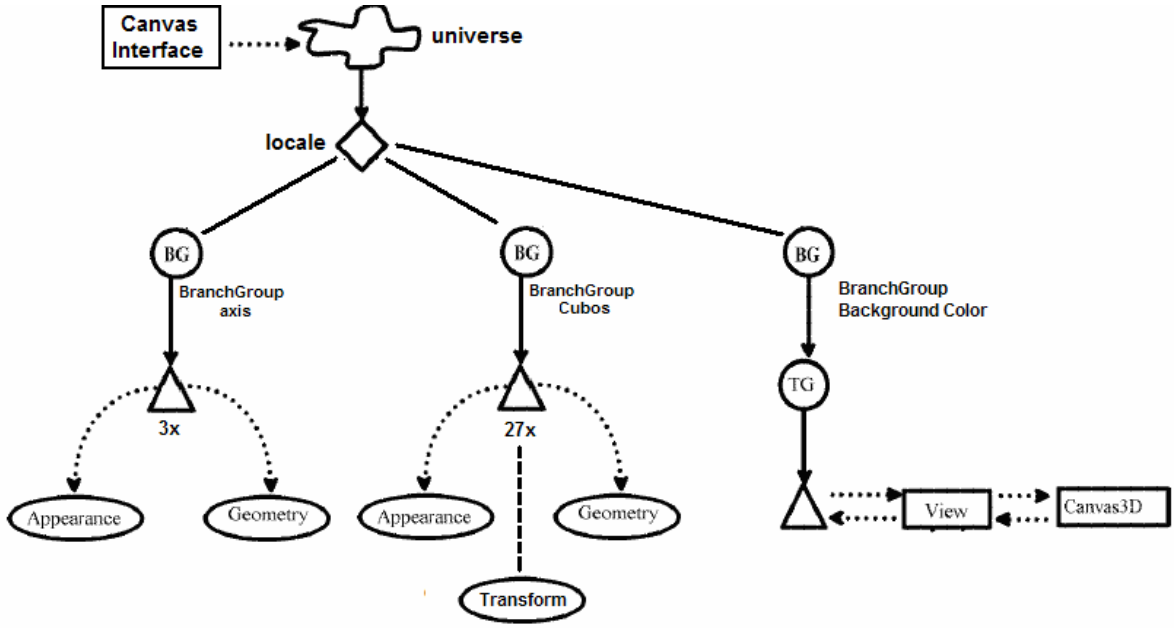


Figura 4.1.4 – Estrutura dos objetos Java3D

Os *BranchGroup Cubos* são objetos que representam os cubos na área de montagem, cada um *TransformGroup* posiciona os cubos na posição correta no sistema de coordenadas. O método

Appearance permite modificar a transparência e cor dos objetos, e podem ser realizadas em *Runtime*. Essa propriedade permite que os objetos mudem de cor ao serem clicados pelo mouse.

4.2 WebAssembly – *Server Side*

A aplicação *server side* roda no computador servidor localizado no GRACO (figura 4.2.1). Sua GUI é simples porque o usuário não a opera diretamente. Sua tela é constituída por um *log* de comunicação com o cliente, um botão de conexão e um botão que permite escrita na paralela.

Figura 4.2.1 – Computador servidor (GRACO)

Este aplicativo abre a porta 5500 para comunicação com a internet, processa as solicitações do cliente, executa os algoritmos e faz o acesso aos arquivos XML (*areaMontagem.xml*, *pecas.xml* e *CAPP.xml*).

4.2.1 Instalação

Para executar o WebAssembly servidor é necessário ter instalado o J2RE 1.4.2 ou superior (*http://java.sun.com/javase/downloads/index.jsp*).

É também preciso instalar os aplicativos que liberam a porta paralela para escrita e leitura, para isso deve-se baixar o driver *UserPort* em *webassembly.graco.unb.br/UserPort.zip*. O arquivo *UserPort.pdf* explana a instalação.

Ao ser solicitada uma conexão, o aplicativo servidor verifica se o usuário é o mesmo da solicitação anterior (através do IP), caso seja, o servidor mantém a montagem anterior, caso não, as dimensões da área de montagem são setadas para Z=2, Y=2 e X=2, criando novo projeto de montagem.

4.2.2 Operação

A interface WebAssembly servidor é constituída por três objetos:

- *Log* de comunicação: Faz os registros dos endereços e solicitações dos clientes que comunicaram com o servidor, durante o período de execução;

- Botão de conexão: utilizado para abrir a porta de comunicação 5500 e aguardar por uma solicitação de conexão por algum cliente.

- Botão manual: utilizado para escrever manualmente na porta paralela, este botão é utilizado para depurar as atividades de tarefa do robô e verificar se a escrita nas I/O's do controlados está correta.

4.2.3 UML

A figura 4.2.2 especifica o diagrama de classes para o aplicativo WebAssembly servidor.

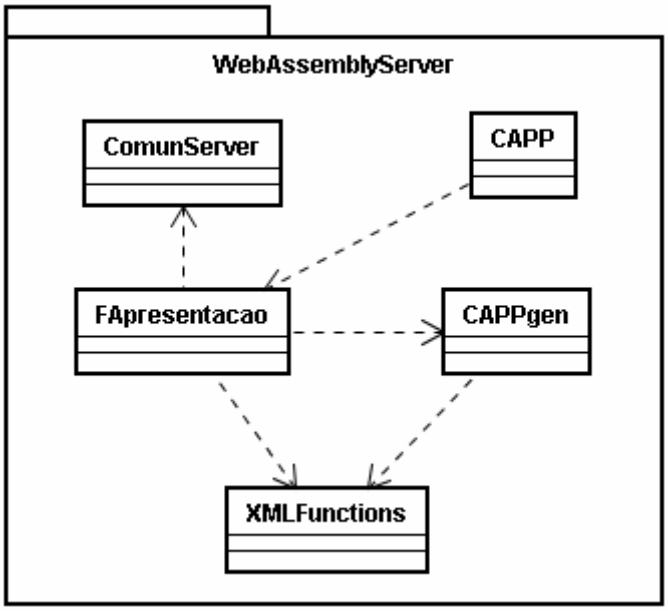


Figura 4.2.2 – Diagrama de classes WebAssembly servidor

4.2.4 Acesso/Gravação de dados (XML)

O acesso aos arquivos XML foi realizado utilizando a API JDOM 1.0 (*www.jdom.org*) e pode ser realizado pelo seguinte pseudo-código:

```
SAXBuilder builder = new SAXBuilder();

doc = builder.build("pecas.xml");

Element raiz = doc.getRootElement();

List lista = raiz.getChildren();

for (Iterator i=lista.iterator();i.hasNext();) {     Element a = (Element) i.next();   }

XMLOutputter outputter = new XMLOutputter();

FileWriter f = new FileWriter(new File("pecas.xml"));
```

outputter.output(doc, f);

O arquivo *pecas.xml* contém as informações das peças disponíveis nos paletes de estoque e deve seguir a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8"?>
<PECAS>
    <Y001 TAMX="1" TAMY="1" COR="AMAR" QNT="6" CodPos="29" />
    .....
</PECAS>
```

Esta estrutura cria o elemento *root* (PECAS) e seus sub-elementos. A seqüência dos sub-elementos representa: nome da peça, comprimento, largura, cor, quantidade disponível e código da posição. Esse código da posição será o código a ser enviado pela porta paralela e representará a orientação e a posição (tarefa) que o robô buscará a peça. As demais informações serão passadas ao *applet* cliente quando solicitado o *Reload* das peças.

O arquivo a*reaMontagem.xml* guarda as informações das células da área de montagem, como posição e bloco inserido, segue sua estrutura:

```
<?xml version="1.0" encoding="UTF-8"?>
<AREA_MONTAGEM>
    <POS1 Z="1" X="1" Y="1" PECANOME="A001" REF="0" sit="1" at="0" />
    <POS2 Z="1" X="1" Y="2" PECANOME="XXXX" REF="0" sit="1" at="0" />
    .....
</AREA_MONTAGEM>
```

Os argumentos representam:

Z	Camada da área de montagem
X	Orientação X na matriz de montagem
Y	Orientação Y na matriz de montagem
PECANOME	Nome da peça inserida na área de montagem
REF	Recebe os valores 0 ou 1: 0 – Peça com dimensão 60x60 1 – Peça com dimensão 60x120

Esse argumento é utilizado na geração do CAPP, indicando que o algoritmo deve procurar pela orientação com que a peça se encontra.

sit e at Juntos esses dois argumentos indicam:

sit	at	Interpretação
0	0	Não usado
0	1	Não usado
1	0	Posição livre para inserção (bloco cor cinza) ou com algum bloco inserido, dependerá do argumento PECANOME
1	1	Posição não disponível (nenhum bloco é mostrado)

4.3 Topologia da comunicação (Socket)

O *socket* localizado no servidor fica em espera por um pedido de conexão. O cliente, após requerer a conexão, fica apto a enviar/receber códigos de comunicação para o/do servidor. Esses códigos devem obedecer ao formato "MENSAGEM">>"PARÂMETROS"# (salvo exceções).

A mensagem será interpretável se constar na tabela 4.3.1.

Tabela 4.3.1 – Lista de códigos de comunicação

Interpretável pelo servidor	
CONFIG>>	Cria um novo *setup* para área de trabalho, de acordo com as dimensões fornecidas pelo usuário
RELOAD>> PECAS	Solicita leitura do arquivo *pecas.xml*
RELOAD>> AREA	Solicita leitura do arquivo *areaMontagem.xml*
HAB>>	Cria nova área de montagem (Z2,X2,Y2) e atualiza *pecas.xml* com 4 unidades para as peças de 60x60 e nenhuma unidade para as pecas 60x120. *Condição inicial
DES>>	Desabilita servidor. Utilizado quando uma montagem é concluída.
CAPP>>	Solicita rodar CAPP

EXEC>>	Solicita executar montagem criada no CAPP
INSERE>>	Solicita inserir peça nas coordenadas passadas pelos argumentos
CLIENT>> Disconnect Server	Desconectar servidor
Interpretável pelo cliente	
PECA>>	Informações das peças em estoque
AREA>>	Informações da área de montagem
OPER>>	Informações do CAPP gerado
MSGE>>	Mensagem de erro enviada pelo servidor ao cliente
SERVER>> END	Fim do envio de informações

4.3 Algoritmo CAPP

O arquivo *CAPP.xml* somente é gerado após a solicitação pelo cliente (botão CAPP!!, aba CAPP). Este é obtido através do algoritmo que lê o arquivo *areaMontagem.xml* e cria a seqüência de tarefa para o robô executar. O arquivo *CAPP.xml* possui a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8"?>
    <CAPP>
    <OP10 NOMEOP="Posiciona Estoque " CODIGO="29" />
    <OP20 NOMEOP="Pega peça Y001 " CODIGO="67" />
    .....
</CAPP>
```

Os argumentos representam a descrição da operação e o código a ser enviado pela porta paralela.

A leitura do arquivo *areaMontagem.xml* é realizada de forma seqüencial., o que resulta em uma leitura camada por camada e célula por célula do modelo tridimensional. Quando a rotina de leitura encontra uma peça reconhecível no argumento PECANOME no arquivo *areaMontagem.xml*, este verifica se o argumento REF (indica peças 60x120) está *setado* ou não. Caso REF seja 0, o algoritmo insere as operações de Posiciona estoque, Pega peça, Posiciona área e Solta peça, o que indica as tarefas de posicionar o manipulador no estoque, pegar peça, posicionar o manipulador na área de montagem e encaixar peça, respectivamente. Caso o argumento REF seja 1, o algoritmo criará as operações Posiciona estoque, Pega peça, Posiciona área e Rotaciona 90º e Solta peça (ou apenas Solta

peça). A decisão por apenas soltar peça ou rotacionar e soltar peça é feita através do resultado da busca de orientação com que o usuário escolheu para inserir a peça. Caso o algoritmo encontre a peça na direção X, ele deve rotacionar e posteriormente encaixar a peça, caso esteja na orientação Y, este deve apenas encaixar a peça.

Os códigos 67 e 35 são correspondentes a pegar e soltar peça, predefinido. Já o código de posicionar no estoque e posicionar na área são dependentes da localização da peça (obtido no arquivo *pecas.xml*, argumento CodPeca) e do local de destino na área de montagem (obtido do arquivo *areaMontagem.xml,* nos argumentos Z, X e Y) .

A execução da montagem será feita através da leitura do arquivo *CAPP.xml*, e consta em escrever o valor do argumento CODIGO na porta paralela na forma seqüencial. O processo de escrita na porta paralela consta em ler a entrada *busy* desta porta e verificar se está em nível alto, caso esteja, o programa procede com a escrita, aguarda 500ms e escreve 0 na LPT. Caso o *busy* da porta esteja em nível baixo, o programa faz 15 tentativas intercaladas de 500ms, se o estado da entrada não mudar, um *timeout* é disparado e a operação é cancelada, identificando algum erro de comunicação entre controlador e servidor.

5. PROJETO MECÂNICO

O projeto mecânico abordou a construção da mesa de montagem, construção do *finger* (específico para manipular as peças LEGO) e a construção dos fixadores do carro do robô que permitem cambiar do sistema FMC ao WebAssembly.

5.1 Mesa de Montagem

A mesa de montagem foi projetada a partir dos seguintes requisitos:

- Comportar todas as variações de *features* das peças LEGO. Característica que levou à implantação de 5 paletes para armazenagem das peças;

- Construir e posicionar a mesa dentro do espaço de trabalho do robô. A mesa não poderia ter altura inferior a 700mm, foi então escolhido 900mm como altura de segurança;

- Implantar uma área de montagem que fosse condizente com a quantidade de peças em estoque. A base da área de montagem possibilita 3 LEGOs na vertical e 3 LEGOs na horizontal, totalizando 9 LEGOs por camada.

- A estrutura deve ser suficientemente rígida e suportar eventuais esforços mecânicos do robô. Foram utilizadas barras de aço trefiladas de perfil quadrado com reforço na mediana da estrutura dos pés de apoio;

- Especificamente a área de montagem deve possuir complacência para facilitar o encaixe das peças. Foram utilizadas borrachas na área de montagem a fim de permitir movimentos nas direções X, Y e Z.

A partir dos quesitos citados, a mesa de montagem foi projetada em *SolidWorks* e está representada na figura 5.1.1. Anexo A representa o projeto mecânico da mesa de montagem.

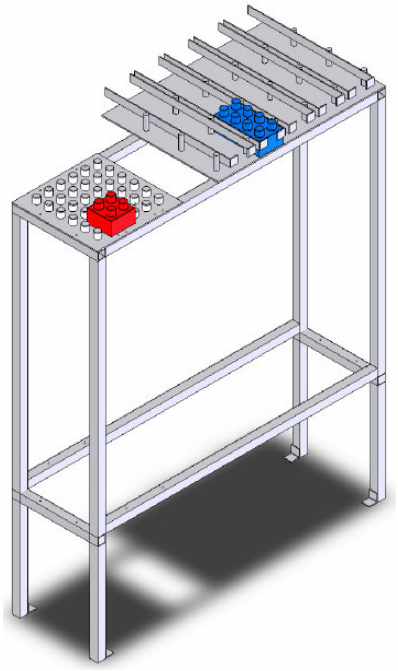
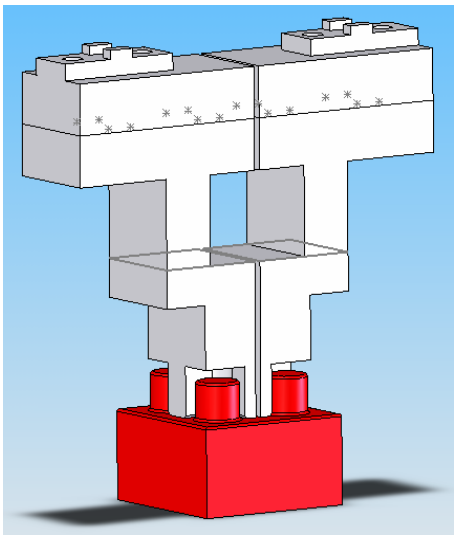


Figura 5.1.1 – Vista isométrica da mesa de montagem

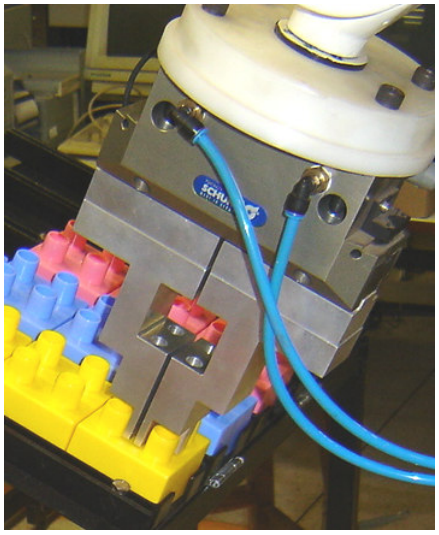
5.2 Finger

Devido à proximidade de uma peça à outra durante a montagem, o *finger* foi projetado para trabalhar no espaço interno aos LEGOs. Esta característica permitiu que os blocos fossem montados sem interferir nos blocos adjacentes já montados.

Por trabalhar no espaço interno dos blocos, a lógica de manipulação foi invertida, ou seja, a peça é manipulada quando a garra estiver aberta. A figura 5.2.1 (a) e (b) mostra o desenho isométrico e a fotografia do *finger* manipulando uma peça LEGO. O Anexo B mostra as folhas de projeto.



(a)

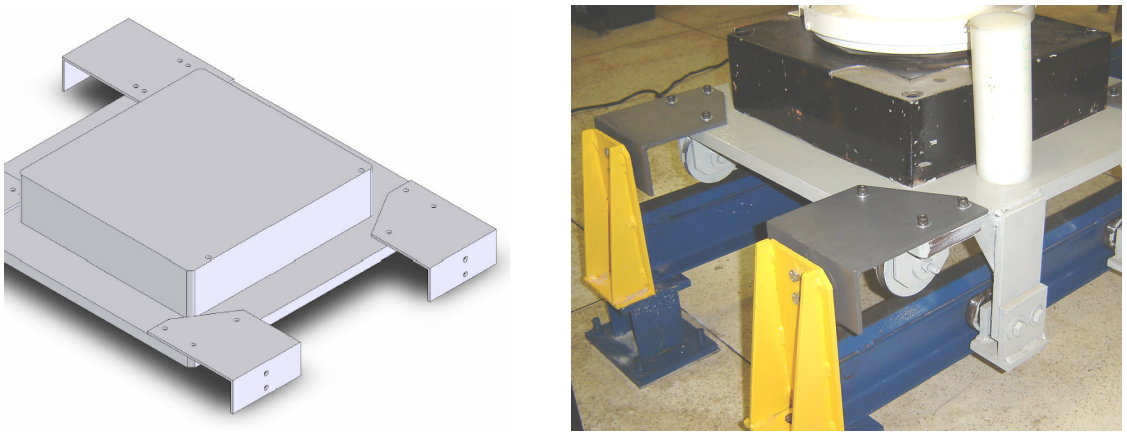


(b)

Figura 5.2.1 – Finger *manipulando os blocos LEGO*

5.3 Fixadores do carro do robô

Os fixadores permitem que o carro do robô seja posicionado na extremidade WebAssembly ou FMC. Os parafusos fixados em cada extremidade garantem que o carro retornará à mesma posição quando modificado o *setup*. A figura 5.3.1 (a) e (b) mostra o desenho isométrico e a utilização dos fixadores no carro do robô.



(a) (b)

Figura 5.3.1 – Desenho isométrico e fotografia dos fixadores

O Anexo C representa as folhas de projeto dos fixadores em detalhe.

6. PROJETO ELÉTRICO

O projeto elétrico foi o último projeto a ser concluído e abordou a interligação do computador servidor às I/O's do controlador do robô, garantindo também a integridade do sistema elétrico FMC.

6.1 Ligação FMC x WebAssembly

O sistema WebAssembly compartilha o cabo RA-COM com o sistema FMC. Este cabo conecta as 7 inputs e 1 output do controlador do robô ao sistema WebAssembly e ao mesmo tempo disponibiliza um fio para fornecer 24V do robô à entrada do relé do torno CNC. Para tanto, alguns cuidados na instalação e utilização devem ser tomadas a fim de evitar que não se tenha os dois sistemas conectados.

O cabeamento elétrico compartilhado foi realizado de acordo com o diagrama da figura 6.1.1. A seleção das conexões é feita no painel elétrico localizado próximo ao computador servidor.

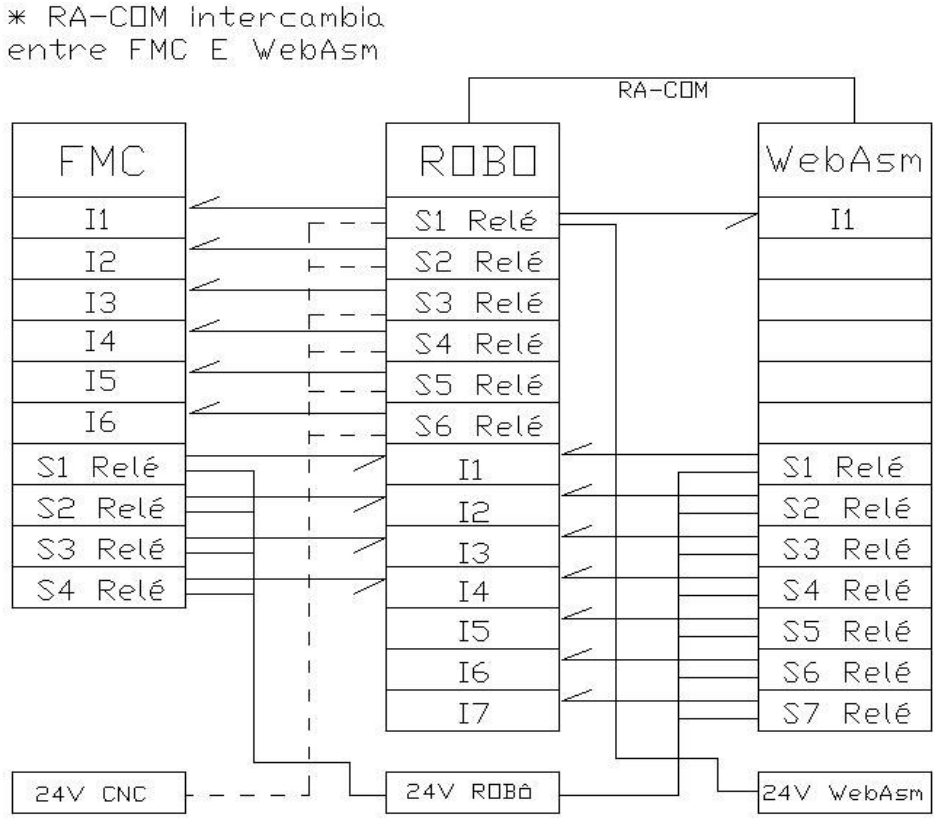


Figura 6.1.1 – Diagrama de utilização das I/O's (WebAssembly x FMC)

Para ligar o WebAssembly, deve-se proceder da seguinte forma:

- Desconectar todos os conectores do painel e do torno CNC (J70 e J80);

- Mudar a posição da chave que encontra-se na parte frontal do controlador do robô para ASM;

- Fazer as conexões do diagrama WebAssembly (figura 6.1.4);

- Ligar o computador servidor;

- Conectar o *plug* de energia na tomada e ligar a placa do WebAssembly (PCB_ASSEM.).

6.1.1 Painel Robô

Ao painel do robô estão instalados os trilhos *din* que dão suporte aos relés de entrada e saída (figura 6.1.2). Estes relés correspondem às 7 entradas (relés superiores), 6 saídas (relés inferiores esquerdos) e 2 sinais das garras (inferior direito).

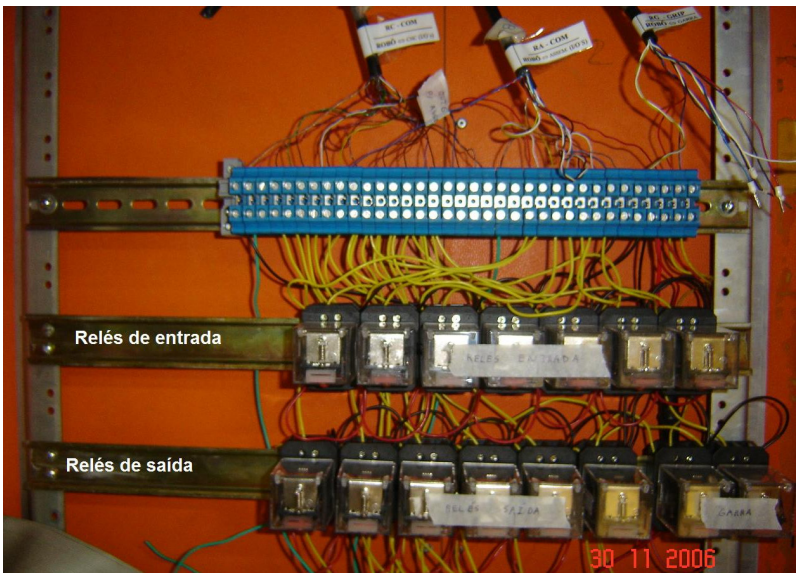
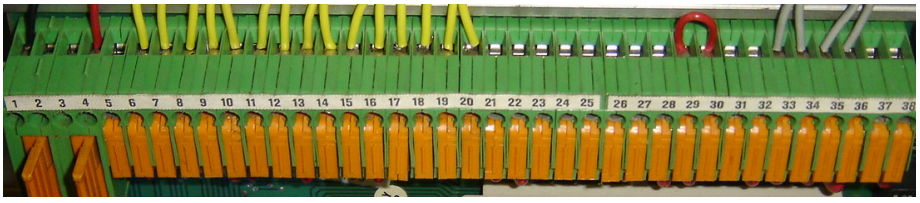


Figura 6.1.2 – Ligações aos relés das I/O's

Os relés da figura 6.1.2 são ativados pelos sinais da placa D14 153. A figura 6.1.3 (a) e (b) mostra as conexões e a pinagem do correspondente barramento.

(a)



(b)

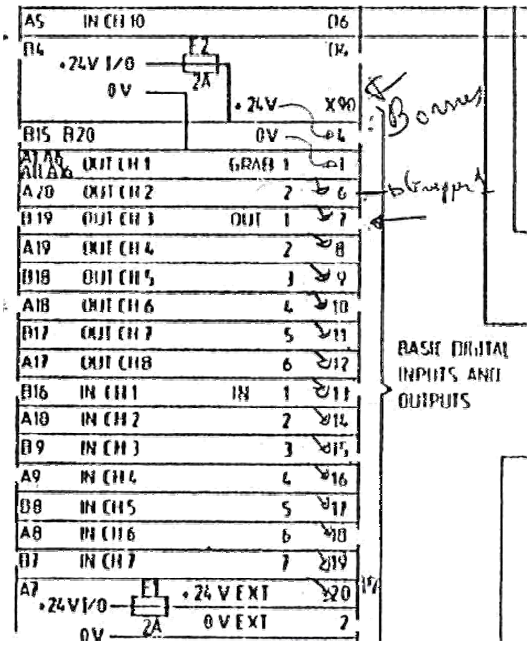


Figura 6.1.3 – Conexões no barramento D14 153[27]

6.1.2 Painel Elétrico

O painel elétrico contém as placas utilizadas pelos projetos WebAssembly e FMC e cada uma delas possui suas funções específicas e independentes. Além das placas, estão instalados nesse painel o transformador da placa WebAssembly, os cabos de conexão e a válvula pneumática (ver figura 6.1.4).

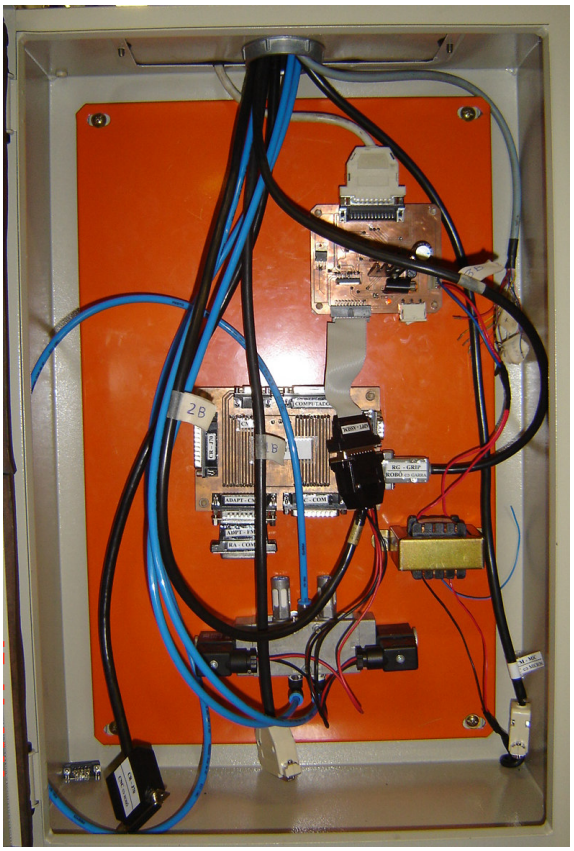


Figura 6.1.4 – Painel elétrico

Os cabos foram marcados com os seguintes *labels*: RG-GRIP (Garra), RC-COM (I/O's para FMC), RA-COM (I/O's para WebAssembly e 24V robô para FMC), CR-J70 (Torno FMC), CM-MIC (Torno CNC) e cabo proveniente da porta paralela.

A figura 6.1.5 representa o diagrama de conexão para o sistema WebAssembly e a figura 6.1.6 a conexão para o sistema FMC. As placas estão marcadas com as mesmas etiquetas mostradas nestas figuras.

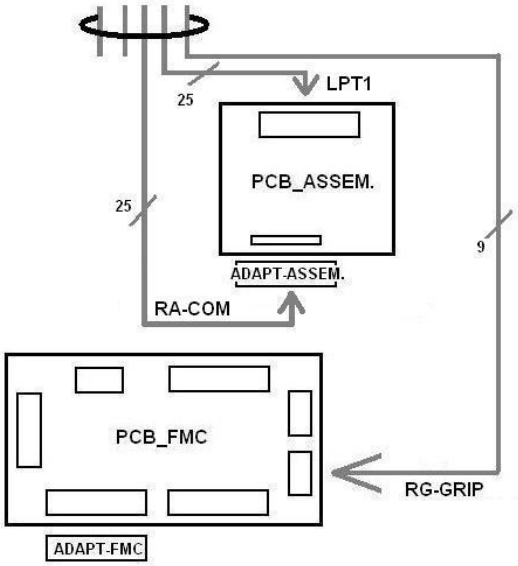


Figura 6.1.5 – Diagrama WebAssembly

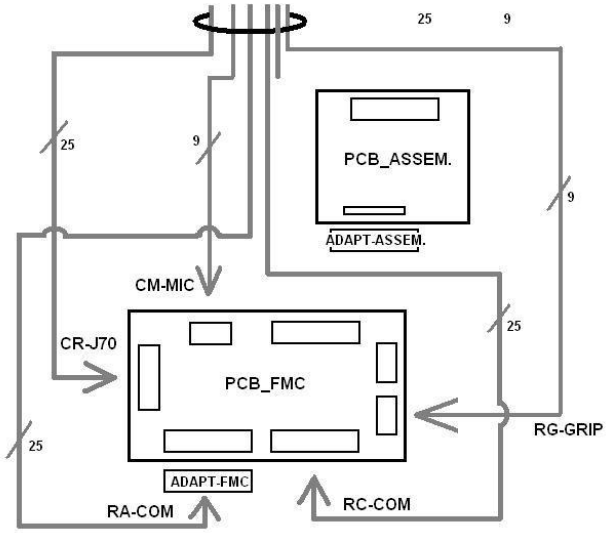


Figura 6.1.6 – Diagrama FMC

6.2 *Inferface* de comunicação

A interface de comunicação é composta de uma pequena fonte retificadora e reguladora de tensão (5V e 24V), dois L298 e um 7805, ver figura 6.2.1.

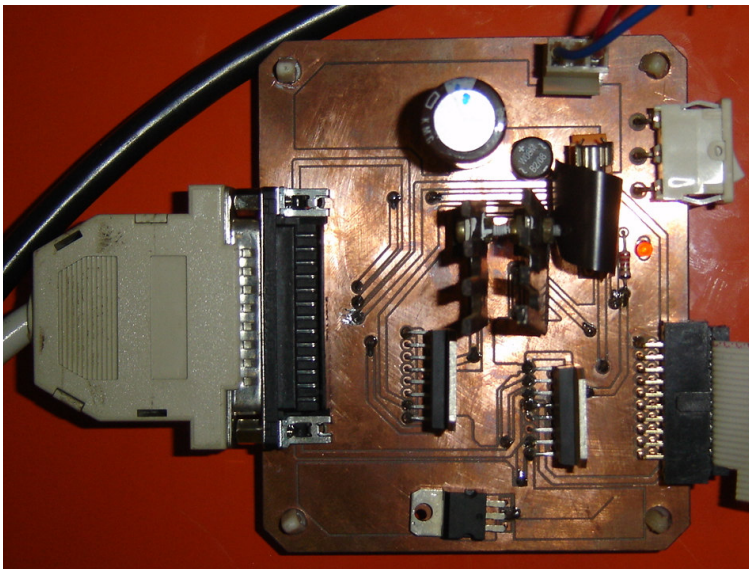


Figura 6.2.1 – Interface de comunicação WebAssembly

O componente L298 foi utilizado para converter os 5V provenientes da saída da porta paralela para 24V, sinal para ativar os relés de entrada do robô.

O regulador de tensão 7805 foi utilizado para reduzir o sinal de saída do controlador de 24V para 5V. Estes 5V está conectado à entrada *busy* da porta paralela. A figura 6.2.2 mostra o PCB da placa, projetada no Protel 99 e confeccionada no SG-11.

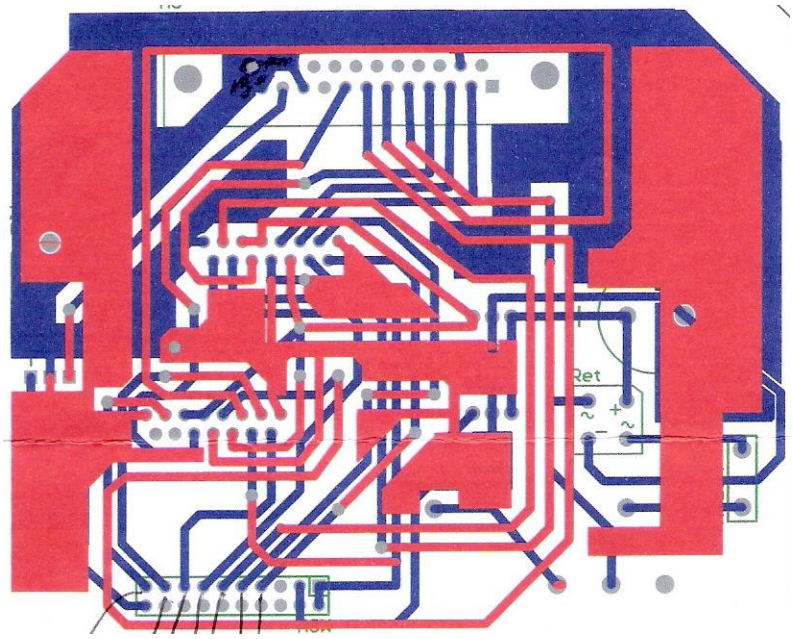


Figura 6.2.2 – PCB WebAssembly

7. PROGRAMAÇÃO ROBÔ

Como o IRB6 não possui unidade de gravação de disco, não há cópias do programa que executa o WebAssembly. Para evitar reconstrução da lógica da programação, o Anexo D traz o programa completo utilizado no controlador, gravado no programa de número 10, no controlador.

7.1 Topologia da programação

O *loop* principal do programa constitui na leitura da *Input* 7, caso o programa leia nível lógico 1 nesta entrada, o programa sai do laço infinito e segue a leitura das demais entradas. Cada entrada recebe um peso (1, 2, 4, 8, 16 e 32) e adiciona esse peso ao registrador 2 do controlador. Esse peso cria uma conversão das entradas para um número decimal, gravado no REG 2.

O número decimal gravado no registrador 2 corresponde a que tarefa o robô deve executar. As tarefas podem ser: posição absoluta, posição incremental, abrir garra, fechar garra ou uma seqüência destas.

Cada posição na área de montagem e área de estoque foi gravada no robô seguindo a topologia:

- Na área de montagem, a tarefa de posicionar em uma determinada célula é sempre realizada 100 mm acima do exato local de encaixe. Portanto, nessa área são 27 pontos gravados, 9 para cada uma das 3 camadas.

- Na área de estoque, a posição de pegar uma peça localiza o manipulador entre os encaixes da peça. Portanto, existem 5 posições gravadas;

- A tarefa de pegar a peça constitui em abrir a garra, mover o manipulador para fora do estoque em movimento relativo, levar a peça até a estrutura a fim de alinhar a peça ao plano de montagem e retornar a uma posição que não interfira na montagem;

- A tarefa de colocar a peça constitui em abaixar 100 mm em Z como um movimento relativo (V=5%), fechar a garra e subir 100 mm em Z, também em movimento relativo.

Para toda programação foi estabelecido o TCP (*Tool Center Point*) como 1 na programação, caso não tivesse sido estipulado, o controlador setaria o TCP como sendo no centro da flange de conexão com a garra [27].

7.2 Codificações utilizadas

As codificações que resultarão em tarefas de robô estão listadas nas tabelas 7.2.1 (área de montagem), 7.2.2 (área de estoque) e 7.2.3 (rotação/deslocamento). Estas tabelas também relacionam os valores que o aplicativo servidor escreve na porta paralela.

Tabela 7.2.1 – Codificações ao controlador área de montagem

			COMPUTADOR		ROBÔ	
Z	**Y**	**X**	**COD**	**Valor**	**COD**	**Valor**
1	1	1	1000001	65	0000001	1
1	1	2	0100001	33	0000010	2
1	1	3	1100001	97	0000011	3
1	2	1	0010001	17	0000100	4
1	2	2	1010001	81	0000101	5
1	2	3	0110001	49	0000110	6
1	3	1	1110001	113	0000111	7
1	3	2	0001001	9	0001000	8
1	3	3	1001001	73	0001001	9
2	1	1	0101001	41	0001010	10
2	1	2	1101001	105	0001011	11
2	1	3	0011001	25	0001100	12
2	2	1	1011001	89	0001101	13
2	2	2	0111001	57	0001110	14
2	2	3	1111001	121	0001111	15
2	3	1	0000101	5	0010000	16
2	3	2	1000101	69	0010001	17
2	3	3	0100101	37	0010010	18
3	1	1	1100101	101	0010011	19
3	1	2	0010101	21	0010100	20
3	1	3	1010101	85	0010101	21
3	2	1	0110101	53	0010110	22
3	2	2	1110101	117	0010111	23
3	2	3	0001101	13	0011000	24
3	3	1	1001101	77	0011001	25
3	3	2	0101101	45	0011010	26
3	3	3	1101101	109	0011011	27

Tabela 7.2.2 – Codificações ao controlador área de estoque

			COMPUTADOR		ROBÔ		
Nome	**X**	**Y**	**COD**	**Valor**	**COD**	**Valor**	**Cor**
Y001	1	1	0011101	29	0011100	28	AMAR
A001	1	1	1011101	93	0011101	29	AZUL
R001	1	1	0111101	61	0011110	30	ROSA
A002	1	2	1111101	125	0011111	31	AZUL
R002	1	2	0000011	3	0100000	32	ROSA

Tabela 7.2.3 – Codificações ao controlador para rotação/deslocamento

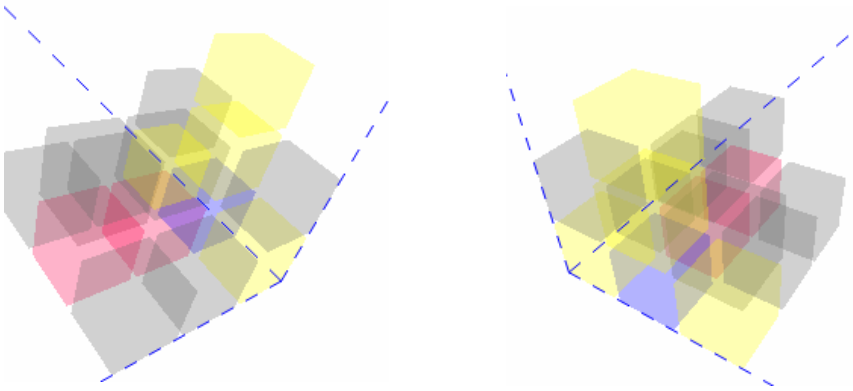
	COMPUTADOR		ROBÔ	
Nome	**COD**	**Valor**	**COD**	**Valor**
Pega Peça	1000011	67	0100001	33
Monta Peça	0100011	35	0100010	34
Rotaciona 90º horário	1100011	99	0100011	35
Rotaciona 90º anti-horário	0010011	19	0100100	36

* MSB confirma recebimento dos dados no prog. do robô e deve ser 1

8. RESULTADOS

O projeto foi validado com a programação de quatro montagens via internet, comandadas pelo servidor e executadas pelo IRB6. Não foi utilizada as peças de dimensão 60x120 pois estas foram fabricadas com dimensões pouco além das especificadas, provocando interferência para montagens em algumas posições. A pressão da garra foi ajustada para 3psi. Este documento listará duas das quatro montagens realizadas utilizando peças de dimensão 60x60.

A figura 8.1 (a) e (b) mostra um dos modelos de montagens concebidos no *applet* cliente, sob diferentes ângulos.



(a) (b)

Figura 8.1 – Modelos de montagem

O arquivo *areaMontagem.xml* para a referida montagem teve o código:

```
<?xml version="1.0" encoding="UTF-8" ?>
<AREA_MONTAGEM>
  <POS1 Z="1" X="1" Y="1" PECANOME="Y001" REF="0" sit="1" at="0" />
  <POS2 Z="1" X="1" Y="2" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS3 Z="1" X="1" Y="3" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS4 Z="1" X="2" Y="1" PECANOME="A001" REF="0" sit="1" at="0" />
  <POS5 Z="1" X="2" Y="2" PECANOME="R003" REF="0" sit="1" at="0" />
  <POS6 Z="1" X="2" Y="3" PECANOME="R003" REF="0" sit="1" at="0" />
  <POS7 Z="1" X="3" Y="1" PECANOME="Y001" REF="0" sit="1" at="0" />
  <POS8 Z="1" X="3" Y="2" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS9 Z="1" X="3" Y="3" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS10 Z="2" X="1" Y="1" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS11 Z="2" X="1" Y="2" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS12 Z="2" X="1" Y="3" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS13 Z="2" X="2" Y="1" PECANOME="Y001" REF="0" sit="1" at="0" />
  <POS14 Z="2" X="2" Y="2" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS15 Z="2" X="2" Y="3" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS16 Z="2" X="3" Y="1" PECANOME="XXXX" REF="0" sit="1" at="0" />
  <POS17 Z="2" X="3" Y="2" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS18 Z="2" X="3" Y="3" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS19 Z="3" X="1" Y="1" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS20 Z="3" X="1" Y="2" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS21 Z="3" X="1" Y="3" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS22 Z="3" X="2" Y="1" PECANOME="Y001" REF="0" sit="1" at="0" />
  <POS23 Z="3" X="2" Y="2" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS24 Z="3" X="2" Y="3" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS25 Z="3" X="3" Y="1" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS26 Z="3" X="3" Y="2" PECANOME="XXXX" REF="0" sit="1" at="1" />
  <POS27 Z="3" X="3" Y="3" PECANOME="XXXX" REF="0" sit="1" at="1" />
</AREA_MONTAGEM>
```

O arquivo *CAPP.xml* foi construído para o modelo concebido, resultando em:

```
<?xml version="1.0" encoding="UTF-8"?>
<CAPP>
        <OP10 NOMEOP="Posiciona Estoque " CODIGO="29" />
        <OP20 NOMEOP="Pega peça Y001 " CODIGO="67" />
        <OP30 NOMEOP="Posiciona Área " CODIGO="65" />
        <OP40 NOMEOP="Solta peça Y001 " CODIGO="35" />
        <OP50 NOMEOP="Posiciona Estoque " CODIGO="93" />
        <OP60 NOMEOP="Pega peça A001 " CODIGO="67" />
        <OP70 NOMEOP="Posiciona Área " CODIGO="33" />
        <OP80 NOMEOP="Solta peça A001 " CODIGO="35" />
        <OP90 NOMEOP="Posiciona Estoque " CODIGO="61" />
        <OP100 NOMEOP="Pega peça R003 " CODIGO="67" />
        <OP110 NOMEOP="Posiciona Área " CODIGO="81" />
        <OP120 NOMEOP="Solta peça R003 " CODIGO="35" />
        <OP130 NOMEOP="Posiciona Estoque " CODIGO="61" />
        <OP140 NOMEOP="Pega peça R003 " CODIGO="67" />
        <OP150 NOMEOP="Posiciona Área " CODIGO="9" />
        <OP160 NOMEOP="Solta peça R003 " CODIGO="35" />
        <OP170 NOMEOP="Posiciona Estoque " CODIGO="29" />
        <OP180 NOMEOP="Pega peça Y001 " CODIGO="67" />
        <OP190 NOMEOP="Posiciona Área " CODIGO="97" />
        <OP200 NOMEOP="Solta peça Y001 " CODIGO="35" />
        <OP210 NOMEOP="Posiciona Estoque " CODIGO="29" />
        <OP220 NOMEOP="Pega peça Y001 " CODIGO="67" />
        <OP230 NOMEOP="Posiciona Área " CODIGO="105" />
        <OP240 NOMEOP="Solta peça Y001 " CODIGO="35" />
        <OP250 NOMEOP="Posiciona Estoque " CODIGO="29" />
        <OP260 NOMEOP="Pega peça Y001 " CODIGO="67" />
        <OP270 NOMEOP="Posiciona Área " CODIGO="21" />
        <OP280 NOMEOP="Solta peça Y001 " CODIGO="35" />
</CAPP>
```

Os resultados da montagem são mostrados pelas fotografias na figura 8.2.

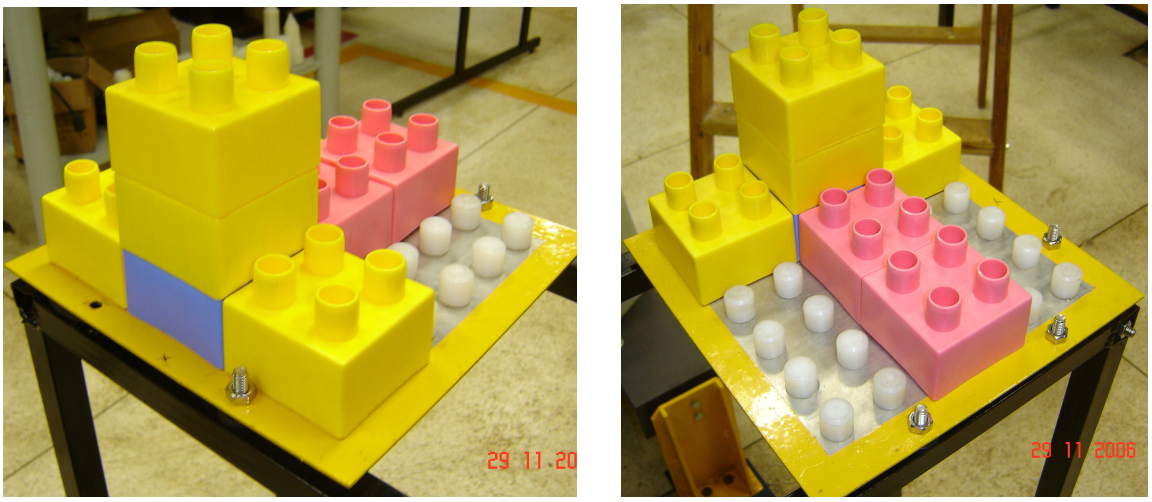


Figura 8.2 – Montagem concluída

A segunda montagem consistiu no modelo concebido na figura 8.3. A figura 8.4 mostra a s fotografias do modelo montado pelo robô.

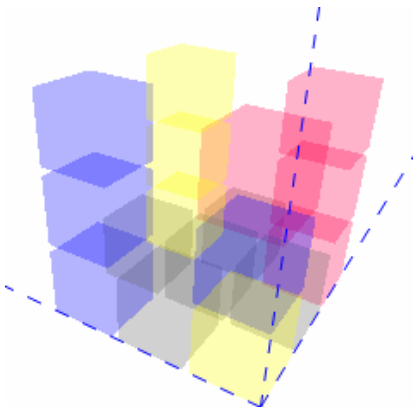


Figura 8.3 – Segundo modelo de montagem

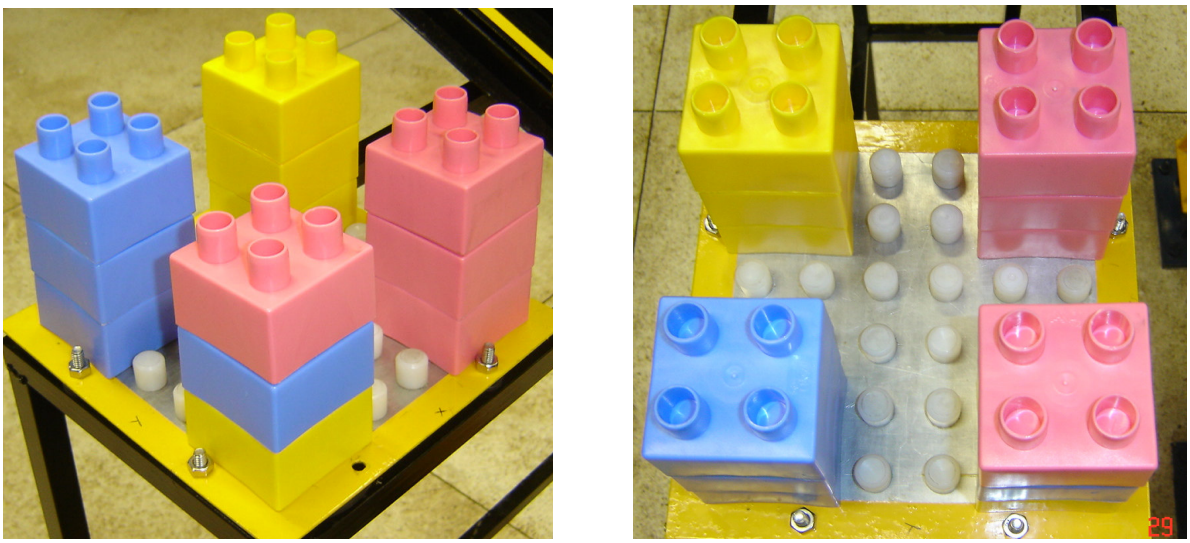


Figura 8.4 – Resultado do segundo modelo de montagem

9. CONCLUSÕES E DESENVOLVIMENTOS FUTURAS

O projeto desenvolveu um sistema de montagens de peças LEGO e implementou as funcionalidades CAD/CAPP/CAA em aplicativos que poderiam ser utilizados e visualizados pela Internet.

O *applet* atende o usuário pela simplicidade de uso por permitir criar um modelo visual 3D do modelo desejado e por executar em *browser*. O aplicativo Java garantiu a integridade dos dados gravados em XML e respondeu com velocidade aos comandos do usuário. O controle de acesso feito por IP faz com que o usuário não herde o plano de montagem do usuário anterior, o que cria um ciclo de produção.

A mesa de montagem atende simultaneamente as necessidades de complacência à área de montagem (minimizam os problemas de tolerância) e a rigidez como estrutura mecânica.

A placa de interface entre a porta paralela do computador e o controlador do robô foi totalmente satisfatória, não apresentou qualquer problema de operação.

Por fim, o projeto foi concluído através de diferentes modelos de montagem realizados pela Internet e efetuados pelo robô, o que valida o projeto e os componentes que o integram.

Algumas sugestões para projetos futuros:

- melhor manipulação das peças com dimensões de 60x120, oferecendo rotinas especiais para inserir estas peças;

- maior número de mesas de montagem a fim de expandir o número de montagens sem que um humano interfira no sistema;

- disponibilizar outras *features* de forma para as peças a serem montadas;

- desenvolvimentos de outros algoritmos de planejamento de processo, o que poderia resultar em diferentes tempos de montagem.

10. REFERÊNCIA BIBLIOGRÁFICA

[1] NOVÁK-MARCINCIN, J., "Computer Aided Production Engineering (CAPE) as Subsystem of CIM", Department of Production Technologies, Faculty of Production Technologies of TU of Košice, Prešov, Slovakia, 2002.

[2] ABRANTES, M. J. and HILL, S. D., "Computer-Aided Planning of Mechanical Assembly Sequences" , Departament of Robotics and Digital Technology, Clayton, Australia, 1996.

[3] CRITCHLOW, Arthur J., "Introduction to Robotics", Macmillan, USA, 1985.

[4] BOOTHROYD, Geoffrey, "Assembly automation and product design", USA, 1992.

[5] China Manufacturers. Disponível em: http://www.madeinchina.com. Acesso em: 05 de ago. 2006.

[6] FANUC Robotics. Disponível em: http://www.fanucrobotics.com. Acesso em: 15 de set. 2006.

[7] KANAI, S., "ASPEN: Computer-Aided Assembly Sequence Planning and Evaluation System Based on Predetermined Time Standard", Hokkaido University, Japan, 1996.

[8] L. Homem de Mello and A. Sanderson, "A correct and complete algorithm for the generation of mechanical assembly sequences", IEEE Transactions on Robotics and Automation, April 1991.

[9] Industrial Robotix. Disponível em: http://www.industrialrobotix.com. Acesso em: 02 de dez. 2006.

[10] HOLZBOCK, Werner G., "Robotic Technology", p.p. ix, USA, 1990.

[11] Sun Microsystems Inc., Applet tutorial. Disponível em http://java.sun.com/docs/books /tutorial/deployment/applet/index.html. Acesso em: 05 de des. 2005

[12] Sun Microsystems Inc., JAR Guide. Disponível em http://java.sun.com/products/jdk/1.2/docs /guide/jar/jarGuide.html. Acesso em: 05 de des. 2005

[13] OAKS, Scott, Segurança de dados em JAVA, Editora Ciência Moderna Ltda, Rio de Janeiro, 1999.

[14] Assinatura Digita. Disponível em: http://www.das.ufsc.br/jacoweb/restrito/documentos /assinatura/index.htm. Universicade de Santa Catarina. Acesso em: 02 de dez. 2006.

[15] Bicho, A., Silveira Jr., L. G., da Cruz, A. J. A. e Raposo, A. B., "Programação Gráfica 3D com OpenGL, Open Inventor e Java 3D", Revista Eletrônica de Iniciação Científica, vol. II, n. I, 2002.

[16] Sockets em Java. Disponível em: http://java.sun.com/docs/books/tutorial /networking/sockets/index.html. Sun Microsystems. Acesso em: 05 de dez. 2006.

[17] Sockets em Java. Disponível em: http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html?page=2. Java World. Acesso em: 02 de dez. 2006.

[18] Javacomm. Disponível em: http://java.sun.com/products/javacomm/. Sun Microsystems. Acesso em: 02 de dez. 2006.

[19] Java Native Interface. Disponível em: http://java.sun.com/j2se/1.4.2/docs/guide /jni/spec/intro.html#wp725. Sun Microsystems. Acesso em: 02 de dez. 2006.

[20] GRAVES, Mark, "Projeto de Banco de dados com XML", Ed. Makron, São Paulo, 2003.

[21] JDOM. Disponível em: http://www.jdom.org/. Acesso em: 02 de dez. 2006.

[22] VASKEVITCH, David. "Estratégia cliente/servidor". Sãoo Paulo: Berkeley Brasil, 1995. 506 p. ISBN 85-7251-270

[23] Porta paralela. Disponível em: http://www.mrshp.hpg.ig.com.br/rob/paralela.htm. Acesso em: 02 de dez. 2006.

[24] MELO, Bruno S. V., OLIVEIRA, Luis A. B., "Desenvolvimento de uma mesa posicionadora X-Y para aplicação em bordado", Universidade de Brasília, 1999.

[25] STMicroeletrônics, "L298, Dual full-bridge driver", Italy, 2000.

[26] Industrial Robotix. Disponível em: www.industrialrobotix.com/IRB6_2.htm. Acesso em: 05 de dez. 2006.

[27] COMAU BRASIL, "Manual Asea Vol. 2, serviço de manutenção". SITO ROBÓTICA.

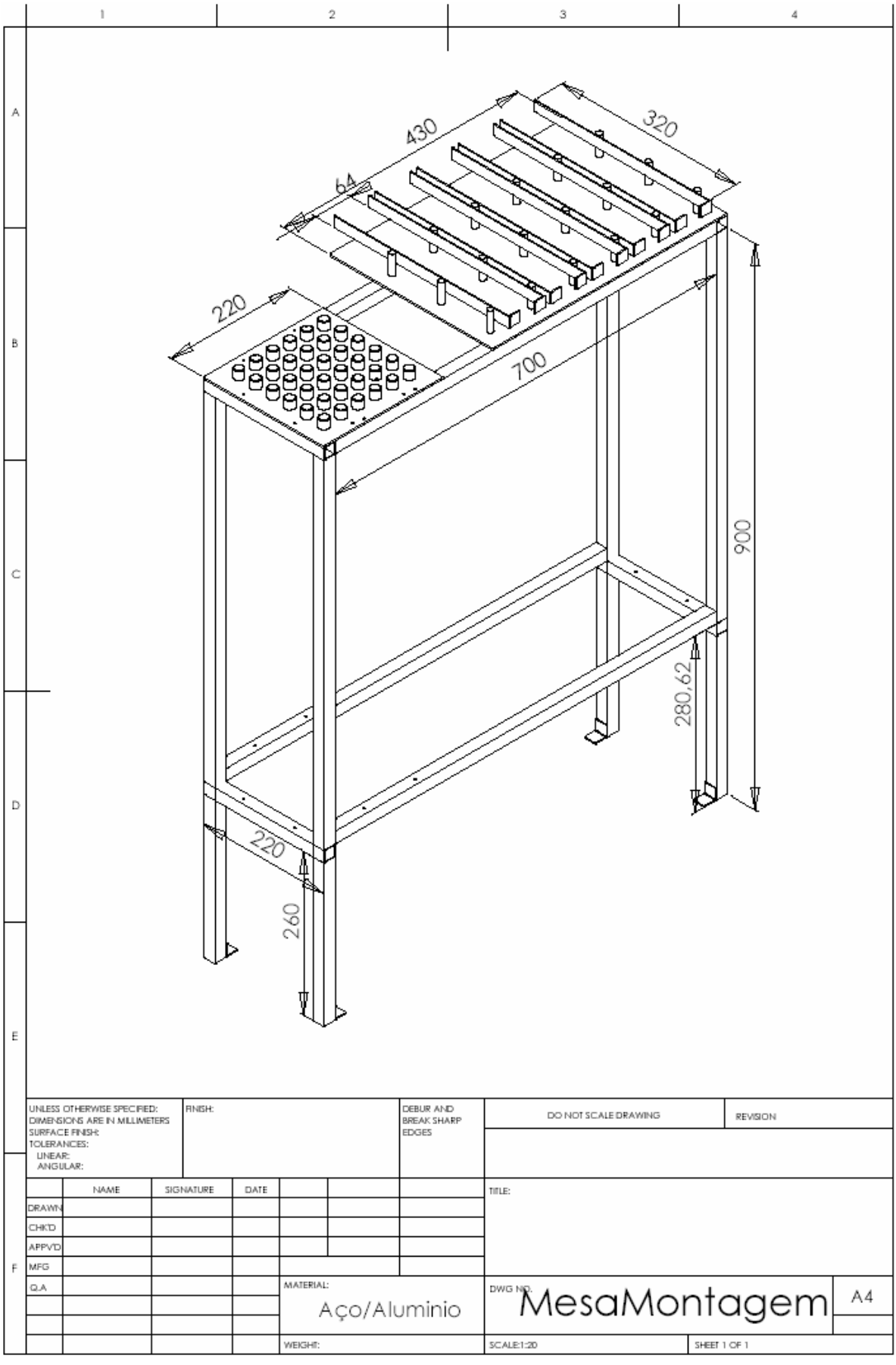
[28] SCHUNK, "1-Finger-Parallel-Gripper Type: PGN", Alemanha, 2004.

[29] Integrated DEFinition Methods - IDEF. A Overview. Disponível em: http://www.idef.com/idef0.html. Acesso em: 27 de jul. 2006.

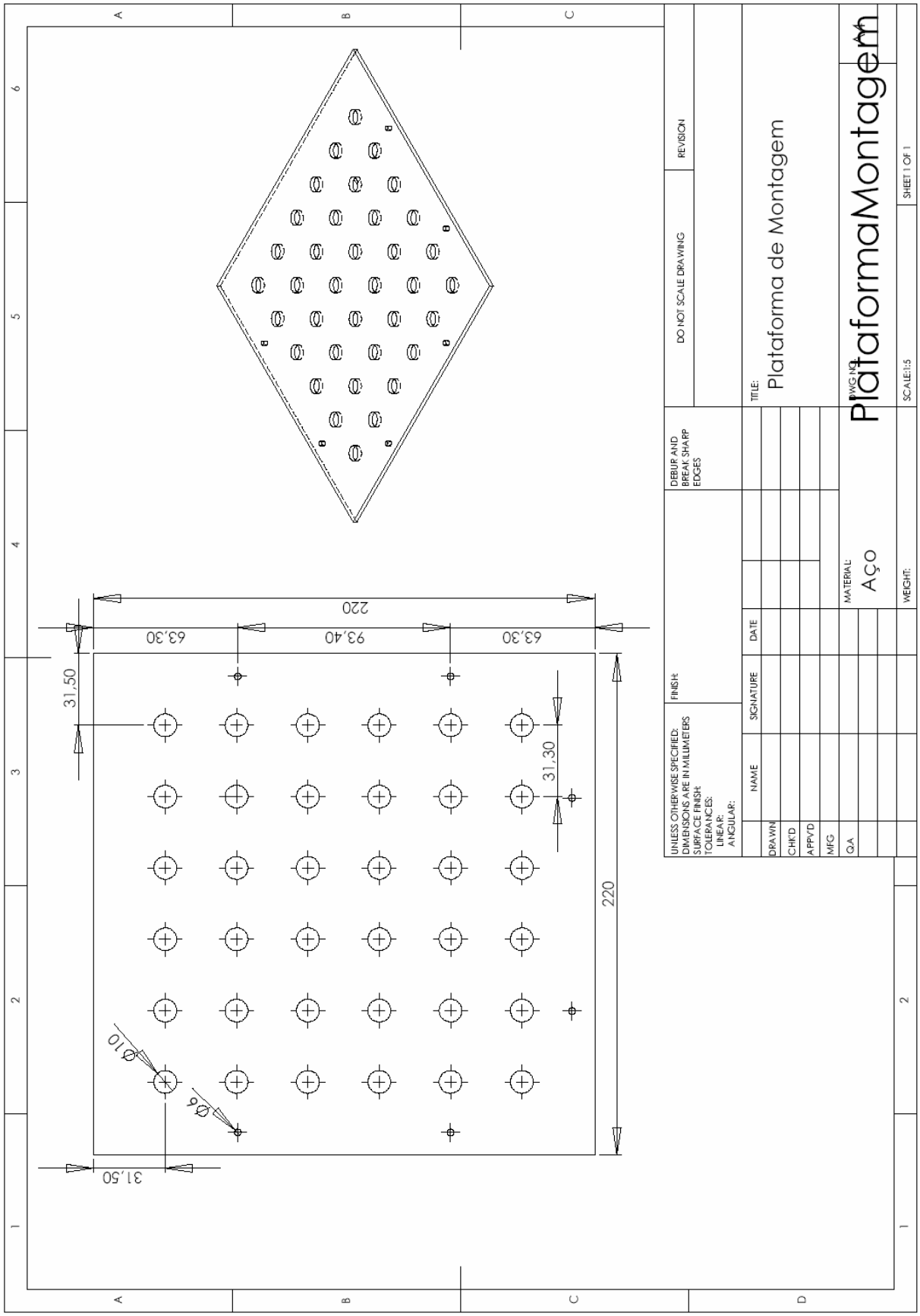
ANEXOS

		Pág.
ANEXO A	Projeto da mesa de montagem (Folha 1/5)	72
ANEXO A	Projeto da mesa de montagem (Folha 2/5)	73
ANEXO A	Projeto da mesa de montagem (Folha 3/5)	74
ANEXO A	Projeto da mesa de montagem (Folha 4/5)	75
ANEXO A	Projeto da mesa de montagem (Folha 5/5)	76
ANEXO B	Projeto do *finger* (Folha 1/2)	77
ANEXO B	Projeto do *finger* (Folha 2/2)	78
ANEXO C	Projeto dos fixadores (folha 1/4)	79
ANEXO C	Projeto dos fixadores (folha 2/4)	80
ANEXO C	Projeto dos fixadores (folha 3/4)	81
ANEXO C	Projeto dos fixadores (folha 4/4)	82
ANEXO D	Fonte do programa no controlador	83 a 86

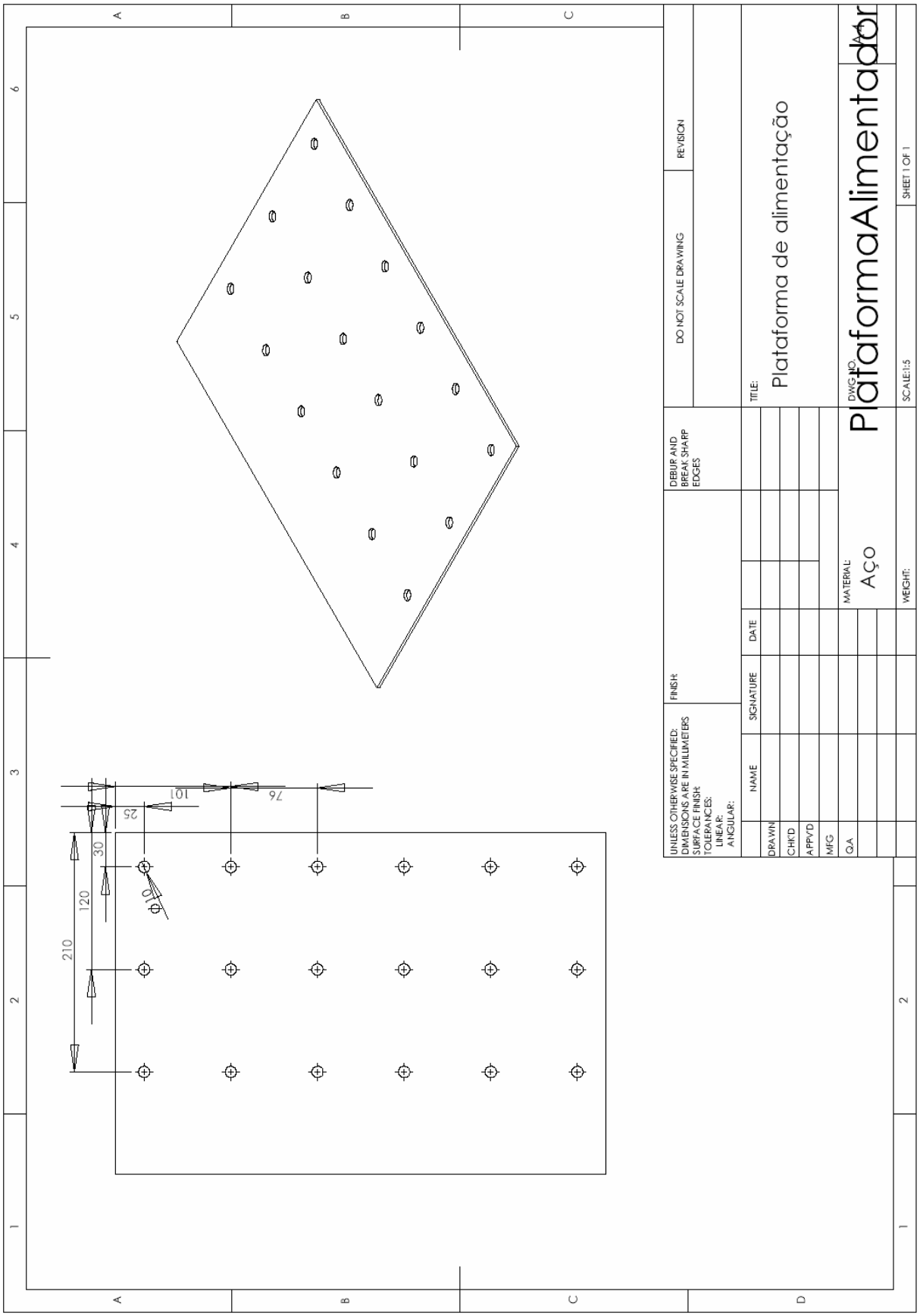
ANEXO A: Projeto da mesa de montagem (Folha 1/5)



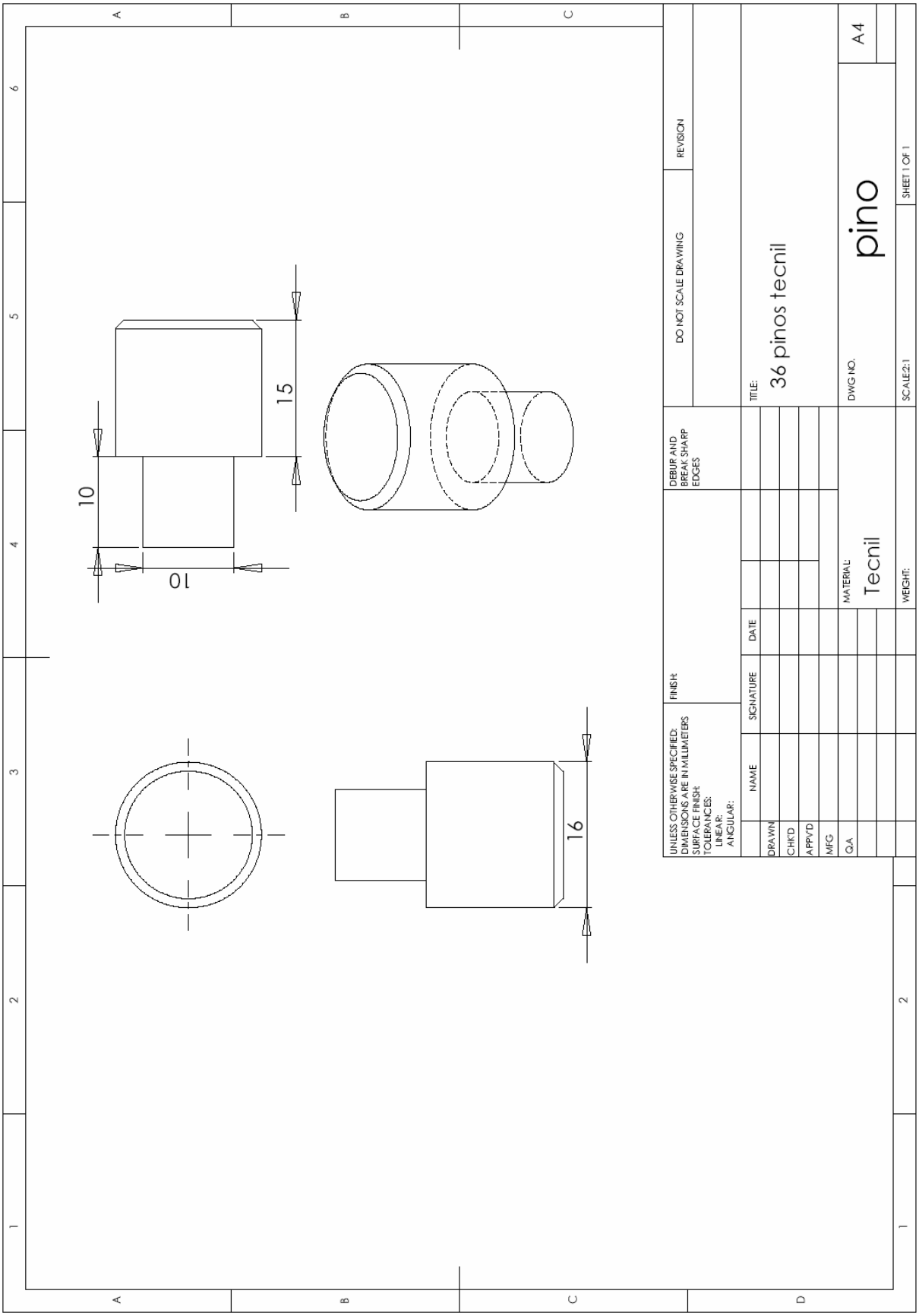
ANEXO A: Projeto da mesa de montagem (Folha 2/5)



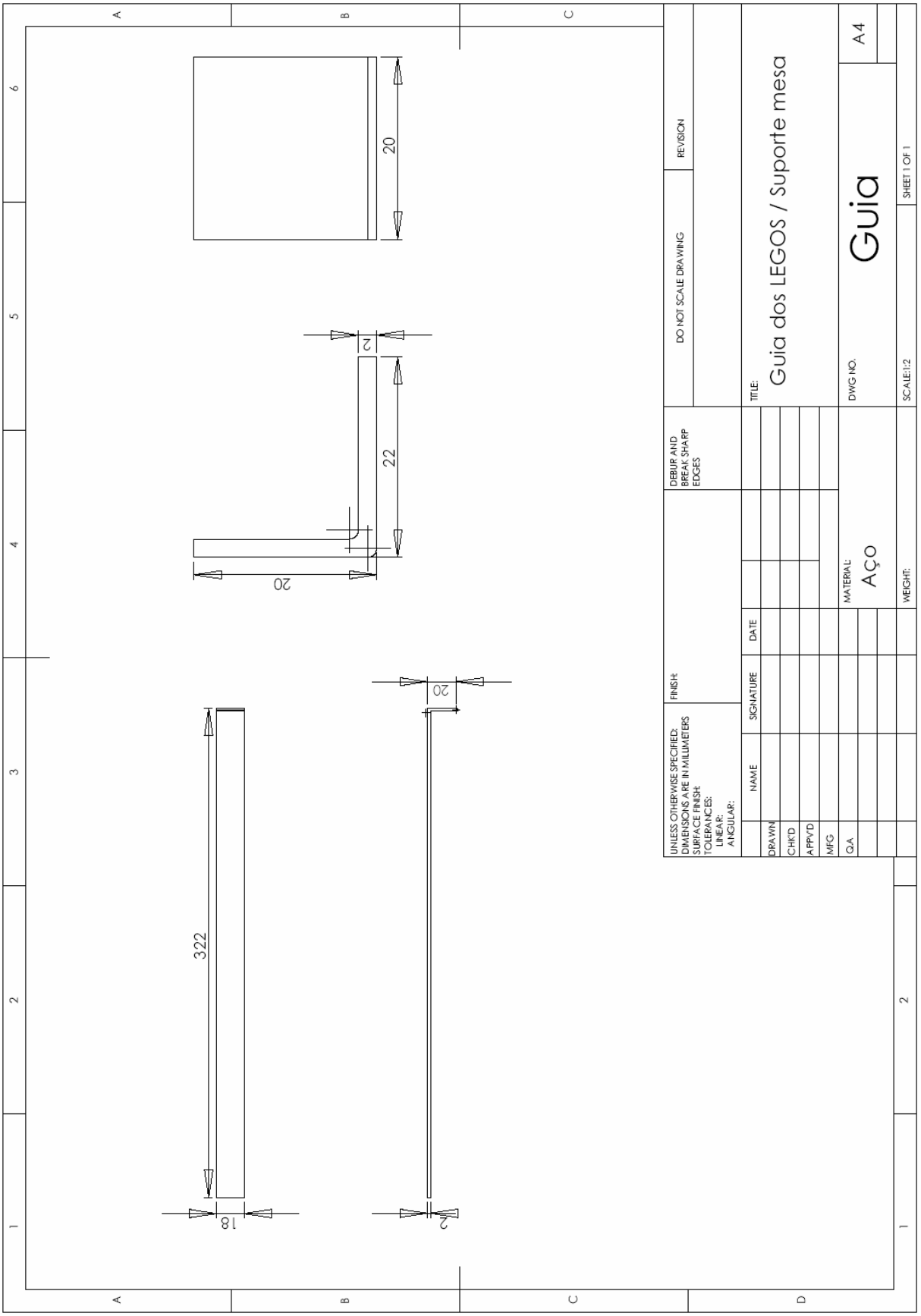
ANEXO A: Projeto da mesa de montagem (Folha 3/5)



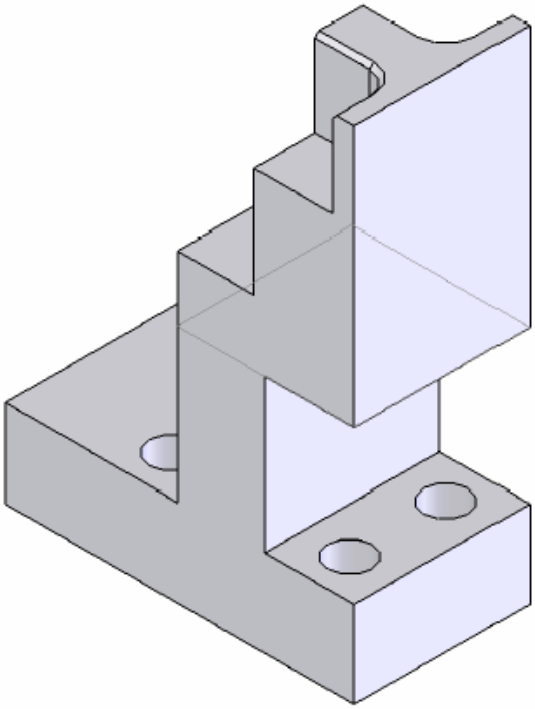
ANEXO A: Projeto da mesa de montagem (Folha 4/5)



ANEXO A: Projeto da mesa de montagem (Folha 5/5)

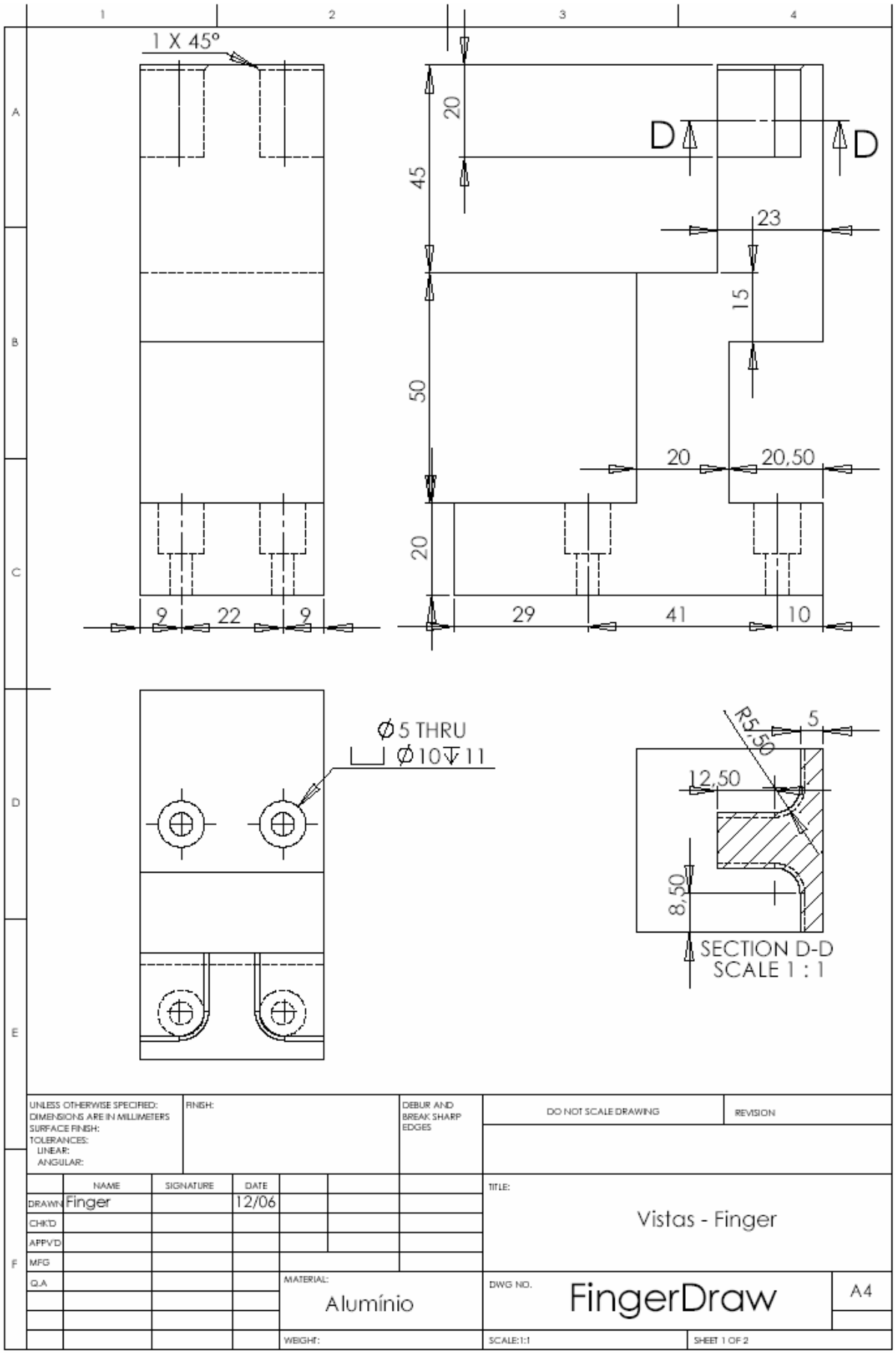


ANEXO B: Projeto do *finger* (Folha 1/2)

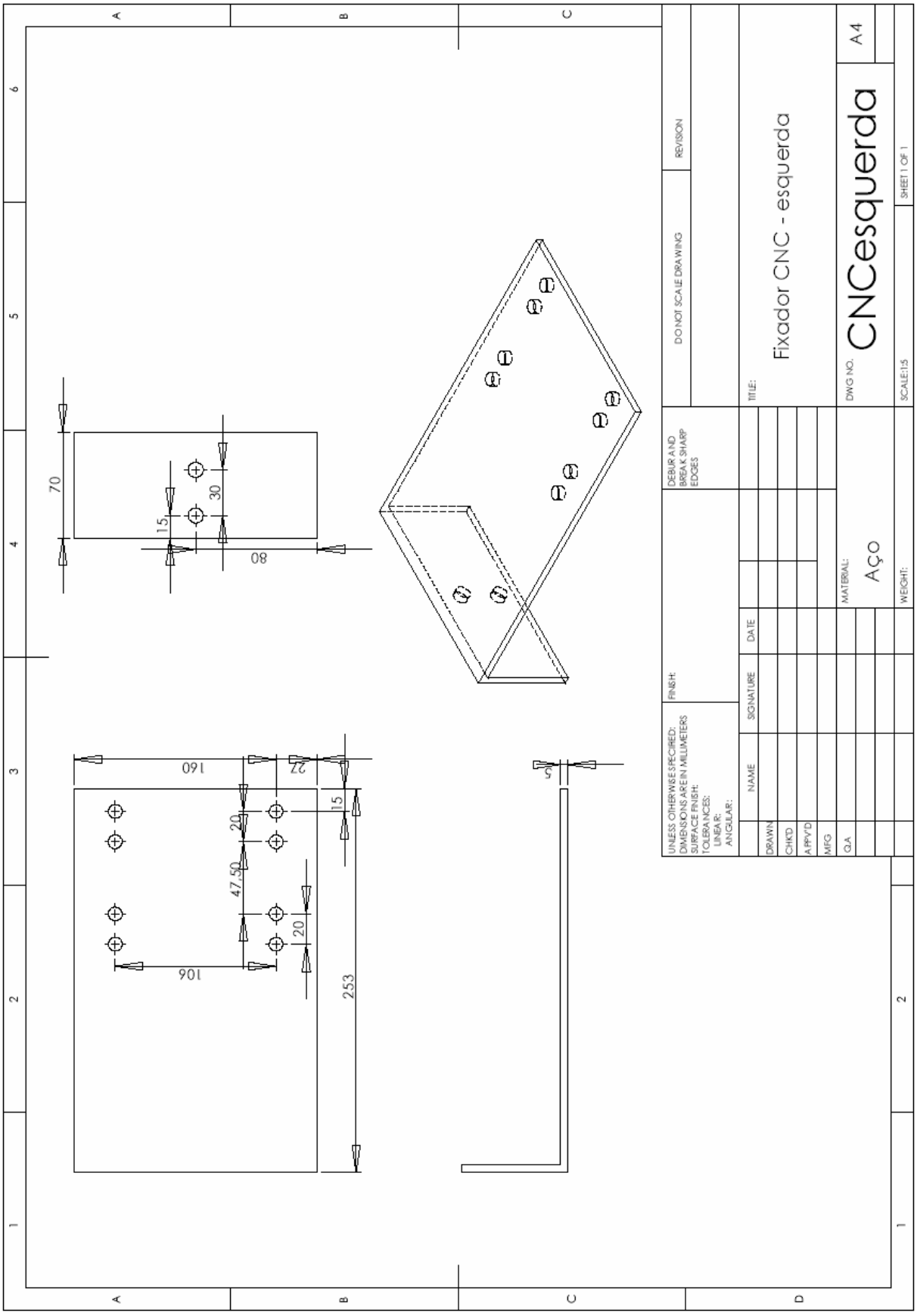


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:			FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE: Vista Isométrica - Finger	
DRAWN	Finger		12/06					
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL: Alumínio			DWG NO. FingerDraw	A4
				WEIGHT:			SCALE:1:1	SHEET 2 OF 2

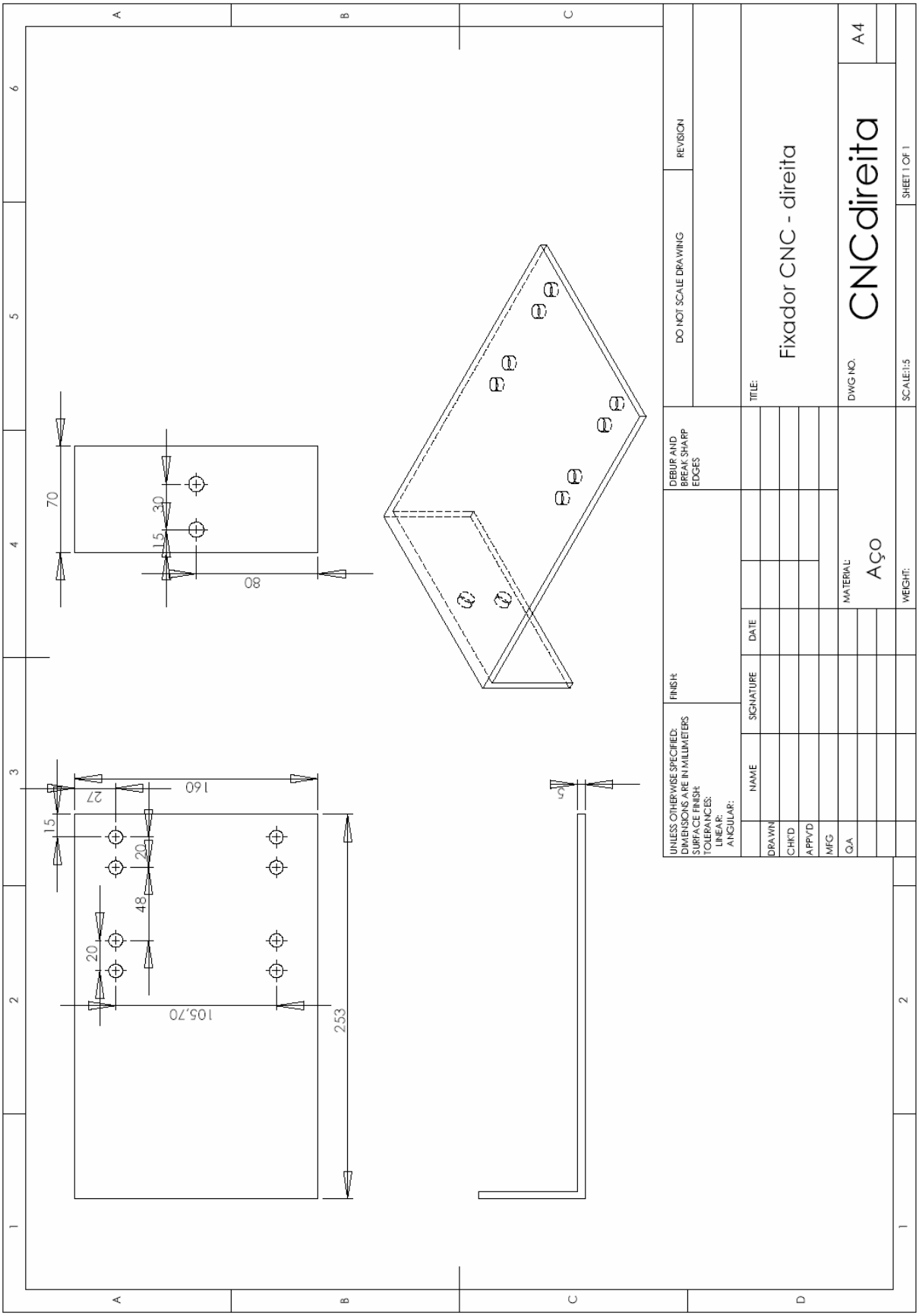
ANEXO B: Projeto do *finger* (Folha 2/2)



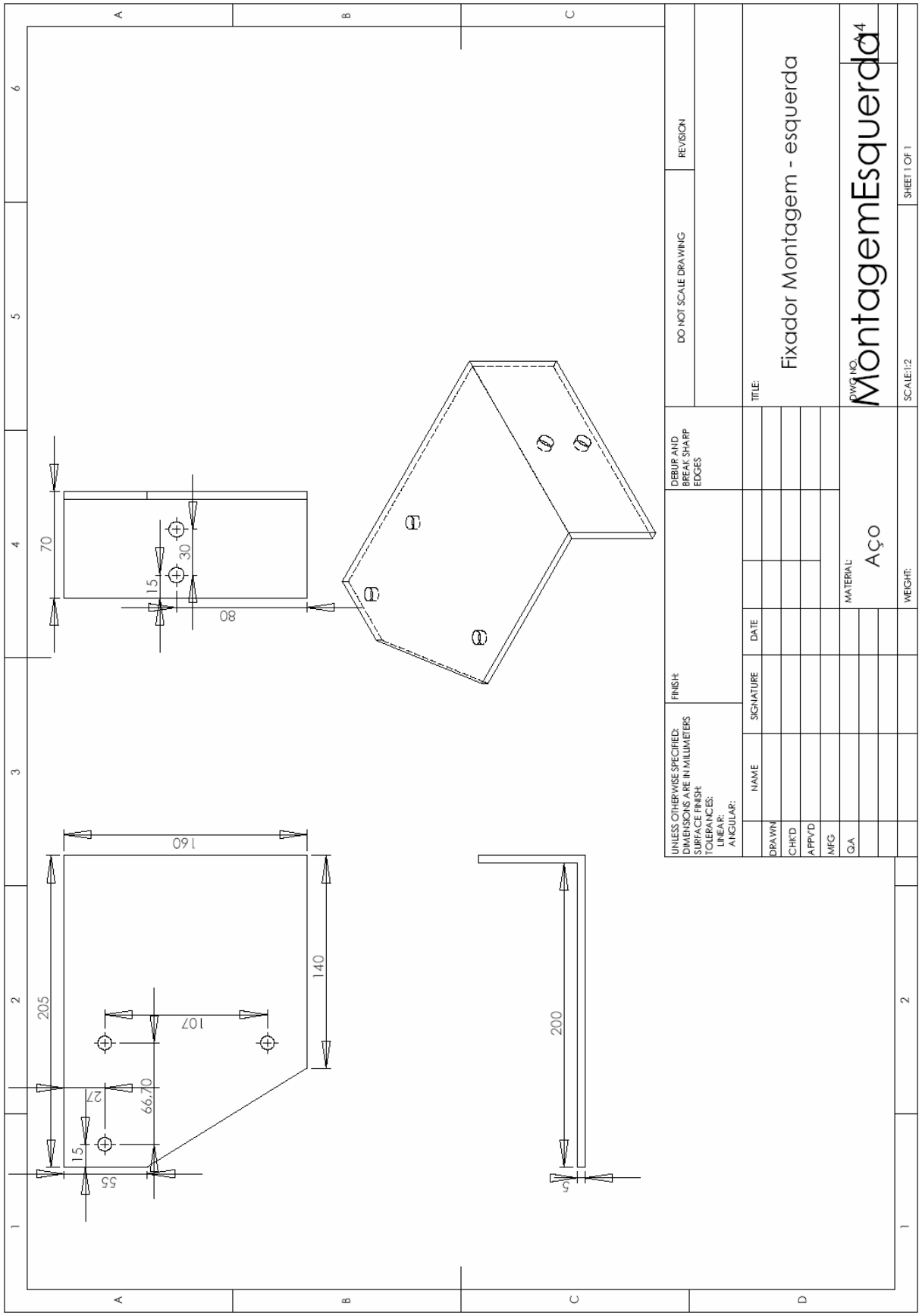
ANEXO C: Projeto dos fixadores (folha 1/4)



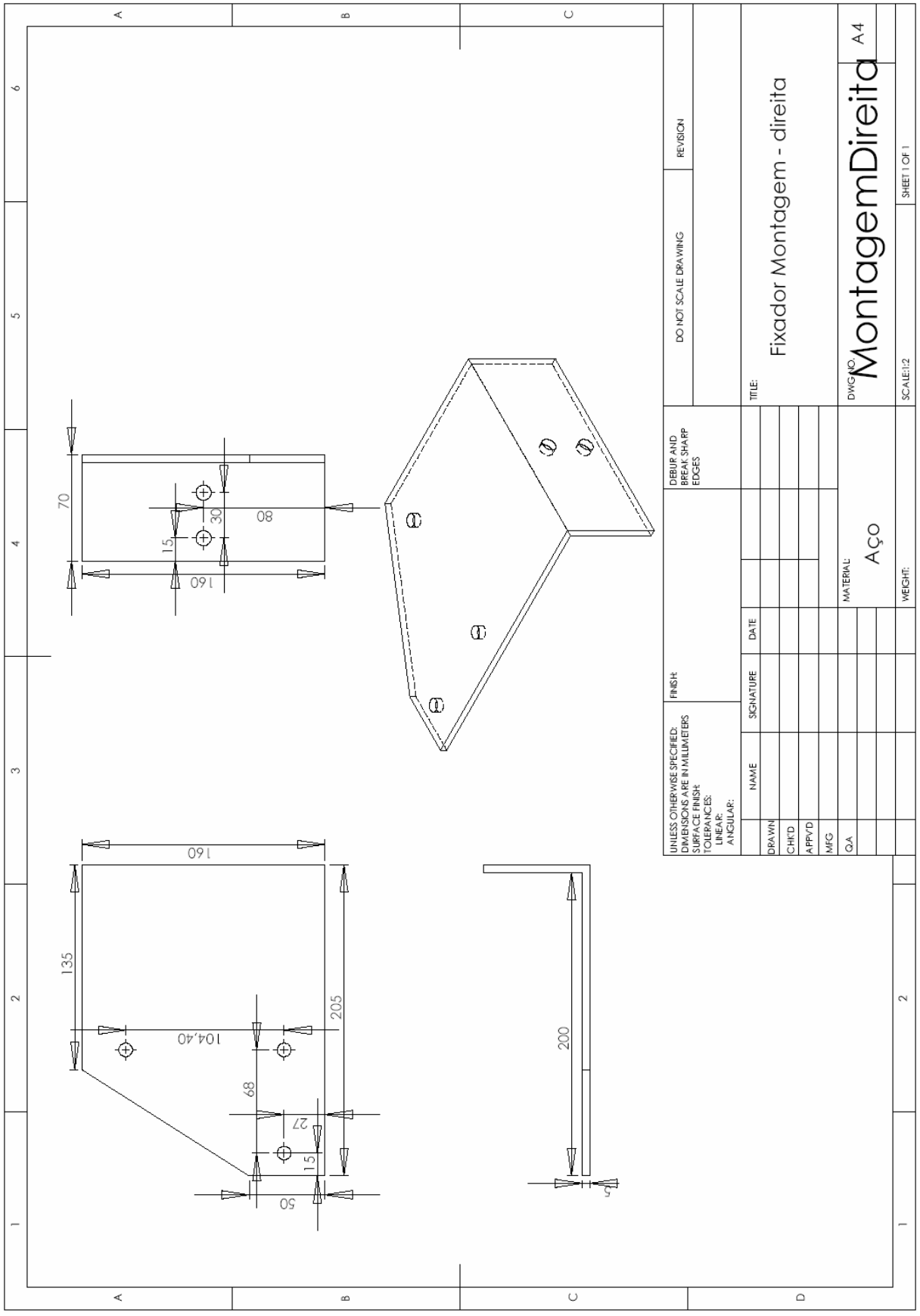
ANEXO C: Projeto dos fixadores (folha 2/4)



ANEXO C: Projeto dos fixadores (folha 3/4)



ANEXO C: Projeto dos fixadores (folha 4/4)



```
 10   V=1000        MM/S      MAX=2000    MM/S
 20   TCP           1
 21   WAIT          0.4S
 22   RESET         OUTP6

 30   SET           R2=0
// Lê as entradas
 39   WAIT          0.2 S
 40   JUMP          TO        39          IF  INP7=0
 41   JUMP          TO        120         IF  INP1=1
 42   JUMP          TO        122         IF  INP2=1
 43   JUMP          TO        124         IF  INP3=1
 44   JUMP          TO        126         IF  INP4=1
 45   JUMP          TO        128         IF  INP5=1
 46   JUMP          TO        130         IF  INP6=1
 47   JUMP          TO        40          IF  R2=0
 48   SET           OUTP6

// Redireciona para posição. Área de montagem
 50   JUMP          TO        100         IF  R2<28
 52   JUMP          TO        140         IF  R2=1
 53   JUMP          TO        142         IF  R2=2
 54   JUMP          TO        144         IF  R2=3
 55   JUMP          TO        146         IF  R2=4
 56   JUMP          TO        148         IF  R2=5
 57   JUMP          TO        150         IF  R2=6
 58   JUMP          TO        152         IF  R2=7
 59   JUMP          TO        154         IF  R2=8
 60   JUMP          TO        156         IF  R2=9
 61   JUMP          TO        158         IF  R2=10
 62   JUMP          TO        160         IF  R2=11
 63   JUMP          TO        162         IF  R2=12
 64   JUMP          TO        164         IF  R2=13
 65   JUMP          TO        166         IF  R2=14
 66   JUMP          TO        168         IF  R2=15
 67   JUMP          TO        170         IF  R2=16
 68   JUMP          TO        172         IF  R2=17
 69   JUMP          TO        174         IF  R2=18
 70   JUMP          TO        176         IF  R2=19
 71   JUMP          TO        178         IF  R2=20
 72   JUMP          TO        180         IF  R2=21
 73   JUMP          TO        182         IF  R2=22
 74   JUMP          TO        184         IF  R2=23
 75   JUMP          TO        186         IF  R2=24
 76   JUMP          TO        188         IF  R2=25
 77   JUMP          TO        190         IF  R2=26
 78   JUMP          TO        192         IF  R2=27
 79   JUMP          TO        30          IF R2<28

// Redireciona para posição. Estoque
 80   JUMP          TO        110         IF  R2<33
```

```
 82   JUMP          TO          200        IF  R2=28
 83   JUMP          TO          202        IF  R2=29
 84   JUMP          TO          205        IF  R2=30
 85   JUMP          TO          250        IF  R2=31
 86   JUMP          TO          253        IF  R2=32

// Funções de pegar/colocar/rotacionar horario e anti
 90   JUMP          TO          210        IF  R2=33
 91   JUMP          TO          220        IF  R2=34
 92   JUMP          TO          230        IF  R2=35
 93   JUMP          TO          240        IF  R2=36
 94   JUMP          TO           30        IF R2>36

// Posiciona na posição default na área de montagem (1ª peça)
100       POS       V=50%
101       JUMP      TO          52

// Posiciona na posição default no estoque (1ª peça)
110       POS       V=50%
111       JUMP      TO          82

// Adiciona os valores para montar valor final de entrada
120       ADD       1           TO        R2
121                             JUMP      42
122       ADD       2           TO        R2
123                             JUMP      43
124       ADD       4           TO        R2
125                             JUMP      44
126       ADD       8           TO        R2
127                             JUMP      45
128       ADD       16          TO        R2
129                             JUMP      46
130       ADD       32          TO        R2
131                             JUMP      47

// Movimentos relativos na área de montagem
140       POS       V=30%
141                             RETURN
142       POS       V=30%
143                             RETURN
144       POS       V=30%
145                             RETURN
146       POS       V=30%
147                             RETURN
148       POS       V=30%
149                             RETURN
150       POS       V=30%
151                             RETURN
152       POS       V=30%
153                             RETURN
154       POS       V=30%
155                             RETURN
156       POS       V=30%
157                             RETURN
158       POS       V=30%
```

```
159                                 RETURN
160       POS       V=30%
161                                 RETURN
162       POS       V=30%
163                                 RETURN
164       POS       V=30%
165                                 RETURN
166       POS       V=30%
167                                 RETURN
168       POS       V=30%
169                                 RETURN
170       POS       V=30%
171                                 RETURN
172       POS       V=30%
173                                 RETURN
174       POS       V=30%
175                                 RETURN
176       POS       V=30%
177                                 RETURN
178       POS       V=30%
179                                 RETURN
180       POS       V=30%
181                                 RETURN
182       POS       V=30%
183                                 RETURN
184       POS       V=30%
185                                 RETURN
186       POS       V=30%
187                                 RETURN
188       POS       V=30%
189                                 RETURN
190       POS       V=30%
191                                 RETURN

// Movimentos relativos no estoque
200       POS       V=40%
201                                 RETURN
202       POS       V=40%
203       POS       V=10%
204                                 RETURN
205       POS       V=40%
206       POS       V=10%
207                                 RETURN

// Pega peça do estoque
210       POS       V=20%       STORE       LOCATION1
211    GRIPPER                   OPEN        0.2 S
212    GRIPPER                   DES.        0.2 S
213       POS       V=10%    LOCATION1
214       POS       V=10%
215       POS       V=10%
216       POS       V=10%
217                                 RETURN
// Coloca peça na área de montagem
 220       POS       V=20%       STORE       LOCATION1
```

```
221     POS       V=5%     LOCATION1   OFFSET Z=-100
222   GRIPPER               CLOSE       0.2 S
223   GRIPPER               DES.        0.2 S
224     POS       V=20%    STORE       LOCATION2
225     POS       V=5%     LOCATION2   OFFSET Z=50
226     POS       V=20%    STORE       LOCATION2
227     POS       V=20%    LOCATION2   OFFSET Z=50
228                        RETURN
```

```
// Movimentos relativos no estoque
250     POS       V=40%
251     POS       V=10%
252                        RETURN
253     POS       V=40%
254     POS       V=10%
255                        RETURN
```