UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

DESENVOLVIMENTO DA UNIDADE DE GERENCIAMENTO DE UMA CÉLULA FLEXÍVEL DE MANUFATURA INTEGRADA A UM SISTEMA CAD/CAPP/CAM

EVANDRO LEONARDO SILVA TEIXEIRA

ORIENTADOR: ALBERTO JOSÉ ÁLVARES

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

PUBLICAÇÃO: ENM.DM – 13A/06

BRASÍLIA/DF: DEZEMBRO – 2006

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

DESENVOLVIMENTO DA UNIDADE DE GERENCIAMENTO DE UMA CÉLULA FLEXÍVEL DE MANUFATURA INTEGRADA A UM SISTEMA CAD/CAPP/CAM

EVANDRO LEONARDO SILVA TEIXEIRA

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

APROVADA POR:

_____

Prof. Alberto José Álvares, Dr. (ENM-UnB)
(Orientador)

_____

Prof. Edson Paulo da Silva, Dr. (ENM-UnB)
(Examinador Interno)

_____

Prof. João Carlos Espíndola Ferreira, PhD. (ENM-UFSC)
(Examinador Externo)

BRASÍLIA/DF, 19 DE DEZEMBRO DE 2006.

FICHA CATALOGRÁFICA

TEIXEIRA, EVANDRO LEONARDO SILVA
Desenvolvimento da Unidade de Gerenciamento de uma Célula Flexível de Manufatura integrada a um sistema CAD/CAPP/CAM [Distrito Federal] 2006.
xvii, 178p., 210 x 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2006).
Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.
Departamento de Engenharia Mecânica.
1. FMC – Célula Flexível de Manufatura 2. Unidade de Gerenciamento
3. Redes de Petri 4. RUP – *Rational Unifield Process*
I. ENM/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

TEIXEIRA, E. L. S. (2006). Desenvolvimento da Unidade de Gerenciamento de uma Célula Flexível de Manufatura integrada a um sistema CAD/CAPP/CAM, Publicação EMN.DM – 13A/06, Departamento de Engenharia Mecânica e Mecatrônica, Universidade de Brasília, Brasília, DF, 178p.

CESSÃO DE DIREITOS

AUTOR: Evandro Leonardo Silva Teixeira.

TÍTULO: Desenvolvimento da Unidade de Gerenciamento de uma Célula Flexível de Manufatura integrada a um sistema CAD/CAPP/CAM.

GRAU: Mestre ANO: 2006

_____

Evandro Leonardo Silva Teixeira
SQN 311 Bloco H apt. 303, Asa Norte.
70.866-080 Brasília – DF – Brasil.

Dedico este trabalho em primeiro lugar a Deus, a minha namorada Isa, fonte da minha alegria, aos meus pais Evando e Terezinha pela educação exemplar que me foi concedida e em especial a minha irmã e amiga Viviane pelo apoio, carinho e sabedoria.

AGRADECIMENTOS

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concebido, através de uma bolsa, para a realização e a conclusão do curso de mestrado.

Agradeço à Fundação de Empreendimentos Científicos e Tecnológicos (FINATEC) pelo apoio financeiro concedido para a participação em um congresso internacional de notório reconhecimento.

Agradeço ao Grupo de Automação e Controle (GRACO) e ao Departamento de Engenharia Mecânica da Universidade de Brasília pelos recursos físicos fornecidos (computadores, equipamentos, etc).

Um agradecimento especial a todos os técnicos laboratoriais que contribuíram de algum modo para a realização deste trabalho.

Agradeço ao meu orientador prof. Dr. Alberto José Álvares.

Um agradecimento especial ao prof. PhD Sadek Crisóstomo Absi Alfaro em nome de todos os professores que formam o corpo docente do programa de pós-graduação em Sistemas Mecatrônicos.

E a todos os amigos e colegas que me apoiaram em mais esta conquista da minha vida.

RESUMO

Este trabalho apresenta uma metodologia para o projeto e implementação de Células Flexíveis de Manufatura (FMC's) capaz de promover a fabricação remota de peças utilizando os recursos da *World Wide Web*. Esta metodologia, denominada WebFMC, descreve um método de integração das estações de trabalho de uma FMC baseado na arquitetura de controle hierárquica modificada e uma arquitetura detalhada para a implementação do sistema de controle da célula (denominado Unidade de Gerenciamento).

A abordagem concebida poderá ser utilizada tanto na academia quanto na indústria. Na academia, esta metodologia servirá como modelo para a construção de laboratórios remotos auxiliando os cursos de engenharia no ensino a distância. Na indústria, promoverá a oportunidade de negócios às empresas que não possuem os recursos necessários para a fabricação de seus produtos.

O projeto da FMC, a qual este trabalho faz referência, foi concebido a partir de um trabalho inicial de modelagem. As técnicas utilizadas (modelagem por simulação via *workspace* e modelagem a eventos discretos via Redes de Petri) resultaram na definição do *layout* da célula e nas mensagens que devem ser trocadas entre as estações de trabalho respectivamente. Os métodos IDEF0 e IDEF1x, da metodologia IDEF, foram utilizados para projetar e documentar respectivamente o modelo funcional e de dados da Unidade de Gerenciamento.

A Unidade de Gerenciamento (MgU), como sistema computacional, foi projetada com base no Processo Unificado da Rational (RUP). Isto resultou na incorporação de elementos da tecnologia de orientação a objetos na construção de um sistema computacional reutilizável e de fácil manutenção. Utilizou-se a Linguagem de Modelagem Unificada (UML) para documentar os fluxos de trabalho do ciclo de vida da MgU.

Java é a tecnologia utilizada na implementação da MgU. As API's e tecnologias associadas fornecem o suporte necessário para que a MgU possa ser executada via internet (via JNLP - URL: *http://webfmc.graco.unb.br/mgu/mgu.jnlp*), para que tenha acesso a base de dados (via JDBC) e se comunique com as estações de trabalho (utilizando a API Java *Communication* e a interface JNI).

ABSTRACT

The aim of this work is to present a methodology to design and to implement Flexible Manufacturing Cell (FMC's) that should be used to manufacture parts remotely using the World Wide Web resources. This methodology (WebFMC) describes a method to integrate the FMC workstations based on the hierarchical modified control architecture and an architecture to implement the cell control system (Management Unit).

This methodology approach can be applied in academic centre such as in the industry. In academic centre will serve as a model to built remote laboratories to the engineering courses in the distance learning. On the industry, will promote the business opportunity to the every company that does not possess the necessary resources to manufacture their products.

The design of this FMC is based on the modeling work. The simulation modeling by workspace and the discrete event modeling by Petri Nets was used to define the cell layout and to describe the messages that should be changed among the workstations respectively. The methods IDEF0 and IDEF1x, from the IDEF methodology, were used to design and to document the functional and data modeling from the Management Unit.

The Management Unit (MgU), as a computational system, was designed based on the Rational Unified Process (RUP). Based on this approach, was possible to aggregate important elements of object oriented design to build a computational system reusable and easiest maintenance. The Unified Modeling Language (UML) was used to design and to document the Management Unit's cycle life.

Java is the technology used to implement the MgU. The API's and associated technology supply the necessary support to run the MgU by internet (via JNLP - URL: *http://webfmc.graco.unb.br/mgu/mgu.jnlp*), to access the database (via JDBC) and to establish the communication with the workstations (using the API Java Communication and the Interface JNI).

RESUMEN

Este trabajo presenta una metodología para el proyecto y la implementación de Células Flexibles de Manufactura (FMC's), utilizadas el la fabricación remota de piezas, empleando los recurso de la World Wide Web. Esta metodología, denominada WebFMC, describe un método de integración de la estaciones de trabajo de una FMC, basado en la arquitectura de control jerárquica modificada y una arquitectura para la implementación de los sistema de control de la célula (Unidad de Control).

Esta metodología podrá ser utilizada tanto en lo académico como en lo industrial. En lo académico servirá como un modelo para la implementación de laboratorios remotos; en lo industrial, promoverá la oportunidad de negócios para empresas que aún no poseen las herramientas para la manufactura del producto.

El proyecto de la FMC de este trabajo, fue concebido a partir de un trabajo inicial de modelamiento. Las técnicas utilizadas (modelamiento por simulación empleando el *workspace* y modelamiento a eventos discretos empleando Redes de Petri) resultaron en la definición del layout de la célula y los mensajes intercambiados entre las estaciones de trabajo. Los métodos IDEF0 e IDEF1x de la metodología IDEF, fueron utilizados para proyectar y documentar respectivamente el modelo funcional y de datos de la Unidad de Control.

La Unidad de Control (MgU) fue proyectada con base en el Proceso Unificado de la Rational (RUP). Esto resultó en la incorporación de elementos de tecnología orientada a objetos en la construcción de un sistema computacional reutilizable y de fácil mantenimiento. El Lenguaje Unificado de Modelaminento (UML) fue utilizado para documentar los fluxos de trabajo del ciclo de vida de la MgU.

Java es la tecnología empleada en la implementación de la MgU. Las API's y las tecnologías asociadas suministran el soporte necesario para ejecutar la MgU por la Internet (via JNLP URL: http://webfmc.graco.unb.br/mgu/mgu.jnlp), para accesar la base de datos (via JDBC) y para comunicarse con las estaciones de trabajo (API Java Communication y la interface JNI).

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

CAD - Computer Aided Design.
CAPP - Computer Aided Process Plan.
CAM - Computer Aided Manufacturing.
CIM - Computer Integrated Manufacturing.
CNC - Comando Numérico Computadorizado.
UML - Unified Modeling Language.
FOCAS1 - Fanuc OpenFactory CNC API Specification.
GUI - Graphical User Interface.
IDEF0 - Integration DEFinition for Functional Modeling.
IDEF1x - Integration DEFinition for Information Modeling.
CAD - Computer Integrated Manufacturing.
CAD - Computer Integrated Manufacturing.
AGV - Automated Guided Vehicle
MgU - Management Unit
JDBC - Java Database Connectivity
JNI - Java Native Interface
JWS - Java Web Server
JNLP - Java Network Launch Protocol
GT - Group Technology
DLL - Dynamic Linking Library
BCD - Binary Coded Decimal
FMC - Flexible Manufacturing Cell
FMS - Flexible Manufacturing System
API - Application Programming Interface
URL - Universal Resource Location
PLC - Programmable Logic Controller
VHDL - VHSIC Hardware Description Language
FPGA - Field Programmable Gate Array

1 – INTRODUÇÃO

Diante da realidade imposta pela globalização e pelo crescimento da concorrência, as empresas vêm buscando cada vez mais a otimização de seus processos produtivos. Uma das propostas atualmente estudadas (filosofia *e-manufacturing*) consiste em integrar as operações de manufatura aos objetivos funcionais da empresa, utilizando os recursos da Internet e das tecnologias associadas (Lee *et at*, 2003). Com esta integração é possível gerenciar o fluxo de informações entre os diversos setores da empresa, proporcionando visibilidade sobre todo processo produtivo e conduzindo a uma melhor definição das estratégias de negócio.

Com o suporte oferecido pela tecnologia da informação associado aos recursos disponibilizados pela Internet (comunicação instantânea, compartilhamento de informações, etc), as empresas estão rompendo não só com as barreiras físicas e filosóficas, mas também com a dificuldade em integrar o fluxo de informações da empresa em uma única rede de comunicação. Soluções comerciais como a INM (*Integrated Networked Manufacturing*) e a IA *(Integrated Architecture*) oferecidas respectivamente pelas empresas Cisco System (Cisco, 2006) e Rockwell Automation (Rockwell, 2006) fornecem o suporte necessário para unificar os sistemas de administração da produção à produção no chão-de-fábrica.

Sob este contexto, esta dissertação apresenta uma metodologia, denominada WebFMC, para o projeto e a construção de Células Flexíveis de Manufatura que utilizam os recursos da *World Wide Web* para promover a fabricação remota de peças. A FMC recebe instruções de seu sistema de controle (denominado Unidade de Gerenciamento) e as converte em operações de processamento para a fabricação do produto. Todo este processo é realizado sem a intervenção direta do operador humano.

A Unidade de Gerenciamento (*MgU*), como sistema computacional, deve fornecer suporte a programação da produção (planejamento operacional), deve obter as informações geradas por um sistema CAD/CAPP/CAM (*Webmachining*), oferecer funções que suportem a verificação da disponibilidade dos recursos produtivos; Também deve permitir o carregamento de instruções nas estações de trabalho, realizar o controle e monitoramento da célula e reagir a eventuais perturbações que possam impedir o cumprimento das atividades estabelecidas na programação da produção.

1.1 – OBJETIVOS GERAIS

Este trabalho tem como objetivo principal propor uma metodologia para o projeto e implementação de Células Flexíveis de Manufatura de peças rotacionais que possam ser operada via Internet. Esta metodologia, denominada WebFMC, apresenta um método de integração das estações de trabalho baseado na arquitetura de controle hierárquica modificada e uma arquitetura detalhada para a implementação da Unidade de Gerenciamento.

A arquitetura da *MgU* deve oferecer suporte para que a FMC trabalhe tanto como sistema integrado quanto como sistema específico. Como sistema integrado, a *MgU* deve receber informações dos níveis hierarquicamente superiores e convertê-las em instruções para as estações de trabalho. Como sistema específico deve oferecer suporte ao planejamento e escalonamento da produção, o controle e monitoramento das atividades e atuar na presença de perturbações ou condições anormais de funcionamento.

1.2 – OBJETIVOS ESPECÍFICOS

Em se tratando dos objetivos específicos, associados ao projeto e a construção da célula, estão:

- ✓ Projeto e construção de um *pallet* armazenador de peças;
- ✓ Projeto e construção de um carro posicionador;
- ✓ Projeto e construção de um suporte para o micrômetro;
- ✓ Projeto e construção de dedos para a garra do robô;
- ✓ Integração física e lógica dos controladores das estações de trabalho;
- ✓ Programação da unidade de manipulação e transporte;

Em se tratando dos objetos específicos relacionados a modelagem e a implementação da Unidade de Gerenciamento estão:

- ✓ Projeto e documentação do modelo funcional da arquitetura;
- ✓ Projeto e documentação da base de dados *WebFMC* que ofereça suporte a integração com um sistema CAD/CAPP/CAM (*Webmachining*);
- ✓ Projeto e documentação do ciclo de vida do sistema computação;
- ✓ Implementação computacional;

1.3 – METOLODOGIA

A metodologia *WebFMC* é baseada na descrição de aspectos de modelagem e de implementação. Na modelagem por simulação (via *workspace*) são levantados os requisitos funcionais para o posicionamento das estações de trabalho (definição do *layout* da célula). Na modelagem a eventos discretos (via Redes de Petri) é avaliado o comportamento dinâmico das estações de trabalho operando, com o intuito de estabelecer as mensagens a serem trocadas pelas estações de trabalho para evitar *deadlocks*.

O modelo funcional da arquitetura da *MgU* é projetado e documentado utilizando-se como ferramenta o método IDEF0 (da metodologia IDEF). Um conjunto de diagramas em uma estrutura *top-down*, documenta os relacionamentos entre os módulos que compõem a *MgU*. O modelo lógico da base de dados relacional *WebFMC* é projetado e documentado utilizando-se o método IDEF1x. Um conjunto de tabelas permite o armazenamento dos dados persistentes e ainda possibilita a integração com o sistema CAD/CAPP/CAM.

Após a modelagem da célula são discutidos os aspectos da implementação. A integração física Robô-Torno-Micrômetro, o projeto e a construção da interface de comunicação, do carro posicionador e do *pallet* estão dentre as principais atividades relacionadas a integração física da célula. Na integração lógica é apresentada a modificação do *Ladder* do Centro de Torneamento, a arquitetura CNC/PMC/Controlador do robô e os detalhes da programação do Centro de Torneamento e da Unidade de manipulação e transporte.

A implementação da Unidade de Gerenciamento é baseada no Processo Unificado da Rational para Sistemas de Engenharia (*RUP-SE*), em que se divide o ciclo de vida do aplicativo em quatro fases: contexto, análises, projeto e implementação. Utiliza-se a UML para projetar e documentar as funcionalidades oferecidas pelo sistema, bem como a modelagem estática e dinâmica da *MgU*.

A implementação computacional da *MgU* é baseada na tecnologia Java, bem como nos recursos que esta tecnologia oferece (*API's*, *Security Containers*, recursos de rede, etc). Utiliza-se a linguagem de programação C++ para desenvolver uma biblioteca (cnclib.dll) que encapsula as funções de comunicação com o Centro de Torneamento. Outros programas *opensource* também são utilizados (Apache, Tomcat e o MySQL) para oferecer o suporte necessário a *MgU*.

1.4 – ESTRUTURA

Esta dissertação está dividida em oito capítulos. O capítulo apresenta uma revisão bibliográfica com os principais tópicos relacionados ao tema. O capítulo três descreve em detalhes o projeto da célula, especificamente as técnicas de modelagem (por simulação e a eventos discretos) utilizadas para levantar os requisitos funcionais da célula, bem como a modelagem funcional e de dados da *MgU*.

O capítulo quatro descreve a montagem e implementação da célula apresentando os detalhes da integração física e lógica e de sua estrutura hierárquica. O capítulo cinco trata do ciclo de vida da *MgU* documentando os requisitos funcionais e a modelagem estática e dinâmica. No capítulo seis são apresentados os detalhes da implementação da *MgU*, as interfaces gráficas e os algoritmos concebidos.

O capítulo sete descreve um estudo de caso utilizado para validar a metodologia proposta. Neste capítulo é apresentado passo a passo o ciclo de fabricação de um produto, desde o projeto da peça ao monitoramento das estações de trabalho durante a fabricação. O capítulo oito apresenta as conclusões, contribuições e trabalhos futuros.

O apêndice A descreve a Rede de Petri detalhada da célula, bem como a validação da rede através de simulação da produção de três peças. Posteriormente o apêndice B apresenta os desenhos do carro posicionador, dos dedos da garra do robô, do suporte para o micrômetro e da unidade de armazenamento de peças (*pallet*). O apêndice C mostra os detalhes da integração Robô-Torno-Micrômetro, enquanto o apêndice D trata da aplicação dos conceitos da arquitetura MVC e dos padrões de projeto na implementação da *MgU*.

2 - REVISÃO BIBLIOGRÁFICA

2.1 – MANUFATURA INTEGRADA POR COMPUTADOR

O conceito de integração proposto pela filosofia CIM prescreve a utilização da tecnologia da informação para integrar todas as funções técnicas e operacionais da empresa relacionadas à produção (Groover, 2003). Em um ambiente de manufatura totalmente integrado, redundância de informação e a interferência do operador são reduzidas, no intuito de agilizar o processamento e aumentar a eficiência produtiva. Dentre os principais benefícios alcançados com esta filosofia estão (Bedworth *et at*, 1991):

- ✓ Melhoria dos serviços prestados ao cliente;
- ✓ Melhoria na qualidade do produto;
- ✓ Redução do lead-time produtivo;
- ✓ Redução dos níveis de estoques;
- ✓ Melhoria no desempenho dos escalonadores;
- ✓ Aumento da flexibilidade;
- ✓ Redução dos estoques intermediários (*WIP*);
- ✓ Aumento da produtividade;

Em CIM, o ciclo produtivo inicia com o pedido do cliente. Através do departamento de vendas ou de um *e-commerce* (quando a empresa oferece este serviço), o cliente fornece as especificações do produto que deseja adquirir, bem como as demais informações necessárias (como data de entrega, quantidade, etc).

As informações referentes a especificação do produto são repassadas à engenharia de projeto. O projetista avalia as especificações e, através de um sistema CAD (*Computed Aided Design*), desenvolve um modelo geométrico do produto, desenhos e a lista de materiais. Estas informações são repassadas à engenharia de fabricação.

Com o auxílio de um sistema CAPP (*Computed Aided Process Planning*), a engenharia de fabricação avalia os modelos geométricos gerados pelo sistema CAD, e elabora um plano de processo, documento com as informações tecnologias utilizadas no processo de fabricação. O plano de processo é repassado ao setor de planejamento e controle da produção. Este setor,

através de um sistema CAM (*Computed Aided Manufacturing*), verifica a disponibilidade de recursos, bem como as informações necessárias para um efetivo planejamento e gerenciamento fabricação propriamente dita.

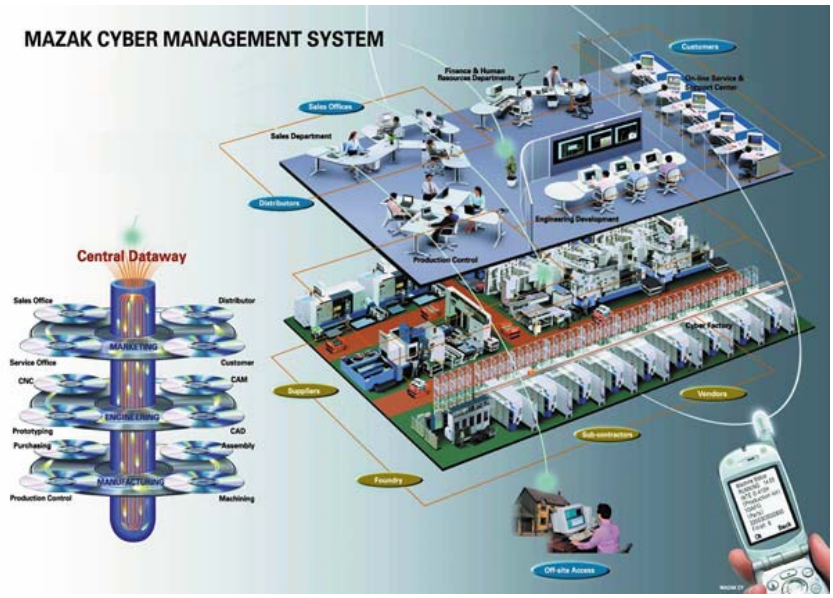

Figura 2.1 - Sistema de gerenciamento Cyber (Krar *et at*, 2003).

A figura 2.1 mostra uma solução comercial oferecida pela empresa Mazak em que se aplicam os conceitos da filosofia CIM. O sistema de gerenciamento *Cyber* possui um sistema integrado de armazenamento da informação, que inicia com o projeto do produto, continua através de todos os estágios do processo incluindo os demais serviços prestados ao cliente (Krar *et at*, 2003).

2.2 – CÉLULA FLEXÍVEL DE MANUFATURA

Uma célula flexível de manufatura pode ser definida como um grupo de recursos de processamento (PLC, máquinas CNC), interconectados por meio de uma unidade de manipulação e transporte de materiais (AGV, robôs) e de um sistema de armazenamento, sendo controlada por um sistema de controle (Quintas *et at*, 1997). Uma célula flexível de manufatura é o menor sistema de processamento em um ambiente produtivo que incorpora elementos da automação flexível e da tecnologia de grupo[1] (GT).

[1] A tecnologia de grupo (GT) é uma filosofia de gerenciamento da manufatura que enfatiza a identificação de grupos de produtos com características de projeto e processos similares.

Composta basicamente por máquinas CNC's, uma unidade de manipulação e transporte de materiais e um sistema de controle, uma FMC reúne, em um único sistema, flexibilidade, eficiência e redução de custos ao combinar médias taxas de produção a médias taxas de variabilidade do produto. Uma FMC proporciona vantagens econômicas relacionando alta taxa de produtividade e baixo volume de produção que normalmente estão associadas a elevado volumes de produção e baixa variedade de um Sistema Flexível de Manufatura (FMS) (Krar *et at* 2003).

Figura 2.2 - Célula flexível de manufatura (MEEM Laboratory, 2006).

As máquinas CNC são responsáveis pelas operações de processamento da célula. As informações tecnológicas de fabricação do produto são agrupadas em um programa NC, sendo estes programas automaticamente convertidos (pela máquina) em operações de fabricação. A unidade de manipulação e transporte realiza o transporte de materiais (matéria prima, produto em processamento e produto acabado) entre as estações de trabalho. Manipuladores robóticos são comumente empregados devido à flexibilidade oferecida.

O sistema de controle da célula é o sistema computacional que converte o plano de produção em instruções para as estações de trabalho. Desse modo, o sistema de controle da célula é responsável por programar, controlar e monitorar as atividades da célula em tempo real,

analisando os dados de produção e notificando eventuais alterações aos níveis hierárquicos superiores.

2.3 – ABORDAGEM TRADICIONAL DAS ARQUITETURAS DE CONTROLE

O avanço da tecnologia da informação aliada à velocidade de processamento e de comunicação entre sistemas computacionais resultou na mudança da concepção das arquiteturas de controle. A evolução das arquiteturas de controle é caracterizada pelo aumento da autonomia dos módulos controladores, pela redução do uso da informação agregada e pelo enfraquecimento da relação mestre-escravo (Pels *et at*, 1997).

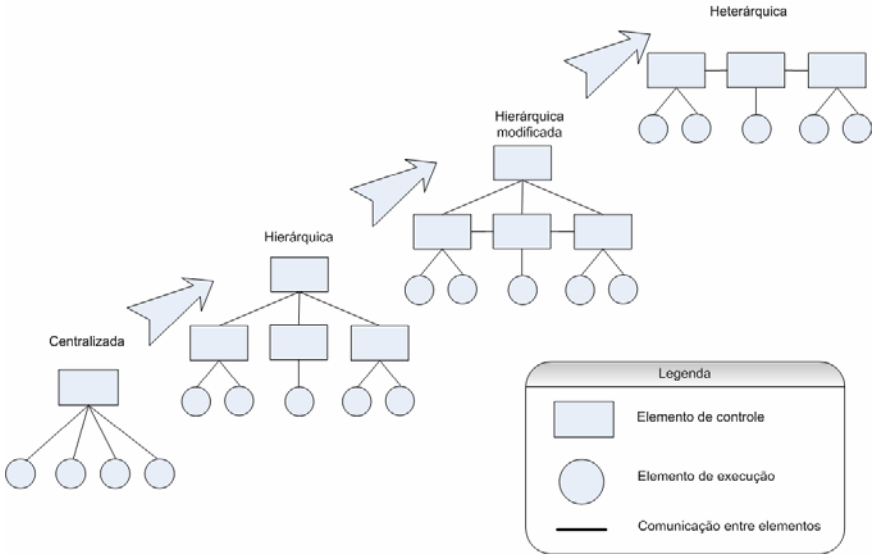

Figura 2.3 - Evolução das arquiteturas de controle (adaptado de Pels *et at*, 1997).

2.3.1 – Arquitetura centralizada

Uma das primeiras arquiteturas utilizadas no projeto de sistemas de controle é a arquitetura centralizada. Nesta arquitetura o processamento é realizado por um único módulo de controle que também concentra todo fluxo de informação. Gerenciamento e controle otimizado estão dentre as principais vantagens, por outro lado, modificações estruturais são necessárias quando uma nova estação de trabalho precisa ser adicionada (Leitão *et at*, 1999).

2.3.2 – Arquitetura hierárquica

A evolução para a arquitetura hierárquica resultou na disposição dos elementos de controle em uma estrutura piramidal e multicamadas. Com esta disposição, o processamento até então centralizado passa a ser distribuído e direcionado ao controlador responsável facilitando a tomada de decisão.

As camadas de controle são interligadas por meio de uma relação mestre-escravo, isto significa que o processo decisório inicia de cima para baixo (*top-down*) sendo o resultado repassado de baixo para cima (*botton-up*). Dentre as vantagens está o desenvolvimento modular, a redução do tempo de desenvolvimento de cada módulo e a desvantagem é a redução da eficiência na presença de perturbações.

A arquitetura hierárquica modificada é a solução para reagir aos problemas de perturbações, mantendo-se todas as características da arquitetura hierárquica e adicionando interações entre os módulos dispostos em um mesmo nível hierárquico (Leitão *et at*, 1999). Esta arquitetura é idêntica à hierárquica, todavia a conexão dos elementos controladores dispostos em uma mesma hierarquia permite a resolução de problemas simples que dispensem a intervenção do controlador principal.

2.3.3 – Arquitetura heterárquica

A descentralização do processo decisório e o aumento da autonomia são as principais vantagens da arquitetura heterárquica. A relação mestre-escravo estabelecida entre os elementos controladores é substituída por uma relação cooperativa através do mecanismo de negociação. Com a tomada de decisão local e autônoma o desempenho do sistema é otimizado, por outro lado, a descentralização dificulta a análise sobre o contexto global, já que não existe uma relação hierárquica entre os elementos de controle.

2.4 – SISTEMA DE CONTROLE/GERENCIAMENTO DE FMC's

Um dos maiores desafios em CIM consiste em integrar as atividades situadas em diferentes níveis hierárquicos (caso do planejamento e do controle da manufatura). A integração do

planejamento ao controle da manufatura possibilita a acessibilidade aos recursos produtivos, à obtenção de dados, o monitoramento e o gerenciamento do chão-de-fábrica em tempo real.

O planejamento da manufatura, atividade situada em um nível hierárquico superior, é responsável pela elaboração dos planos de produção (tanto do nível tático quanto do nível operacional) considerando o planejamento das necessidades e da capacidade produtiva. Estes documentos, em suma, fornecem informações detalhadas sobre quando e quais serão as ordens processadas, quais serão os recursos produtivos alocados, dentre outras.

Em um nível hierárquico inferior está o controle da manufatura. Esta atividade tem como responsabilidade converter o plano de produção em tarefas a serem cumpridas pelas estações de trabalho. Cada tarefa é composta por um conjunto de programas que descrevem, através de instruções, quais são as operações que deverão ser executadas para efetuar o processamento do produto.

A integração destes níveis hierárquicos (planejamento e controle da manufatura) está sujeita aos três problemas básicos (Ou-Yang *et at*, 2000):

- ✓ Como transferir os planos de produção do nível de planejamento para as atividades de produção no chão-de-fábrica?
- ✓ Como assegurar que o plano de produção possa ser cumprido conforme o desejado?
- ✓ Como reagir a uma situação dinâmica que ocorreu no chão-de-fábrica?

A resolução destes problemas requer o desenvolvimento de funções complexas que não podem ser processadas somente utilizando os elementos da automação flexível (PLC's, CNC's, etc). Neste sentido a tecnologia da informação (representada principalmente pelo computador e as redes de comunicação) tem um importante papel já que, comparativamente, permite reduzir o tempo de processamento a fim de obter o controle em tempo real das atividades.

2.4.1 – Definição

Um sistema de controle de chão-de-fábrica relaciona gerenciamento e controle das atividades físicas no chão-de-fábrica no intuito de executar os planos de produção (Leitão, 2004). Isto significa receber as informações oriundas dos níveis hierárquicos superior, analisá-las de acordo

com a heurística estabelecida, selecionar as tarefas a serem enviadas para o chão-de-fábrica e monitorar o cumprimento das atividades. Em se tratando de um sistema de controle de chão-de-fábrica as funcionalidades podem ser subdivididas em (Wysk *et at*, 1995):

- ✓ Planejamento;
- ✓ Escalonamento;
- ✓ Execução das tarefas.

No planejamento são estabelecidas as ordens a serem produzidas e os recursos produtivos que devem ser alocados. No escalonamento é definida a seqüência (mediante o critério de desempenho adotado pelo escalonador) em que as tarefas serão processadas. A execução das tarefas trata da operação e do monitoramento, enviando instruções para o chão-de-fábrica, monitorando o cumprimento das tarefas e reagindo a eventuais perturbações. A figura 2.4 apresenta o modelo de um sistema de controle/gerenciamento de FMC's.

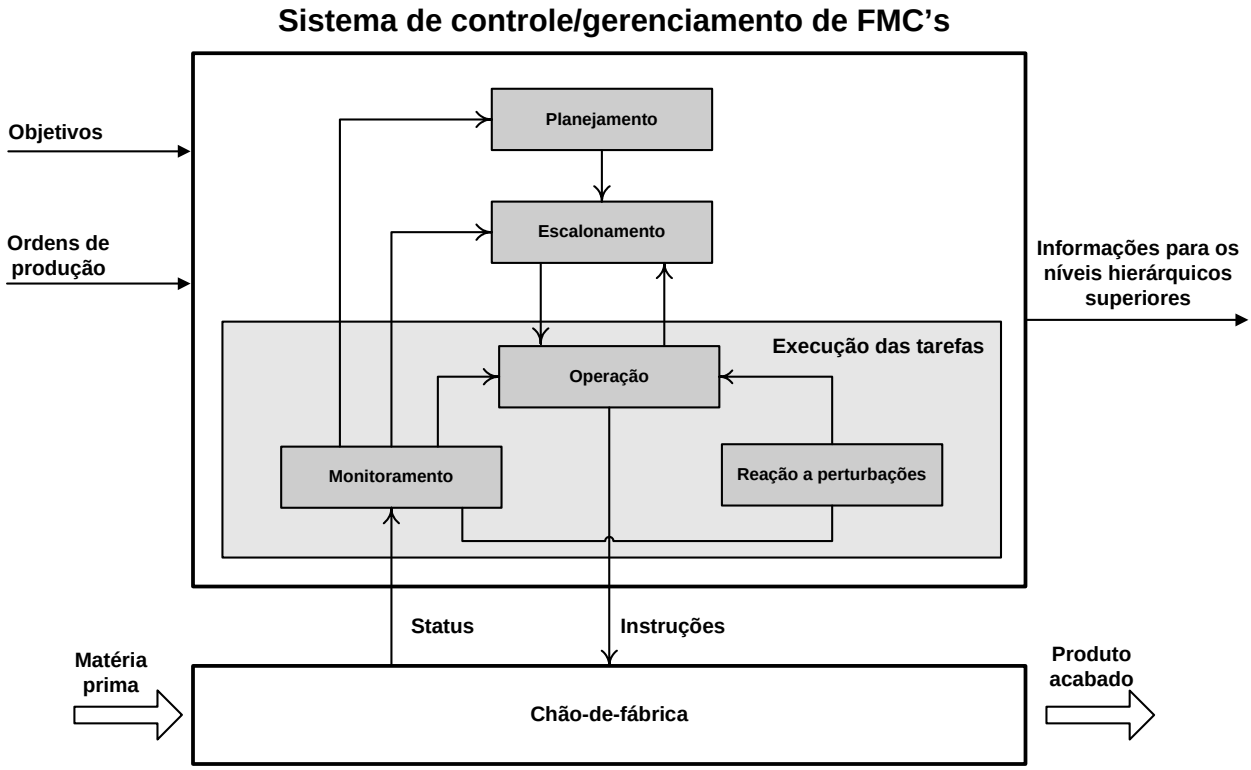

Figura 2.4 - Sistema de controle/gerenciamento de FMC's (adaptado de Leitão, 2004).

2.4.2 – Planejamento da produção

O planejamento da produção é a funcionalidade responsável pela elaboração detalhada do plano de produção. Este plano cujo horizonte de planejamento é de curta duração, descreve as tarefas a serem executadas, os recursos necessários e as restrições impostas pelo processo produtivo. As funcionalidades deste módulo são subdivididas em: Programação da produção e Formação de famílias de peças.

2.4.2.1 – Programação da produção

A programação da produção trata da efetiva conversão das ordens de produção estabelecidas no plano mestre da produção em ordens de trabalho. Uma ordem de trabalho reúne as informações detalhadas (programas NC, lista de ferramentas, dentre outras) utilizadas no carregamento de instruções nas estações de trabalho. Contudo, ao definir quais ordens de trabalho serão processadas, também são definidos quais os recursos de processamento serão utilizados.

2.4.2.2 – Formação das famílias de peças

Identificar as famílias de peças é o primeiro dentre os dois principais desafios a serem resolvidos ao se aplicar os conceitos da tecnologia de grupo em sistemas de manufatura. Peças com características geometrias ou de fabricação similares são agrupadas em famílias no intuito de obter vantagens no gerenciamento produtivo como a redução do lead-time, do tempo de *setup* e demais custos envolvidos.

Uma família de peças pode ser definida como uma coleção de peças que são similares por causa das formas geométricas ou por que requerem os mesmos passos para serem fabricadas (Groover, 2003). Peças com características similares podem ser produzidas a partir de um mesmo grupo de recursos. Isto permite que pequenos lotes de fabricação adquiram as mesmas vantagens econômicas obtidas com a produção em massa, mantendo a flexibilidade adquirida com a aplicação do método de produção *job shop* (Jeon *et at*, 2006).

Como os recursos de processamento utilizados (estações de trabalho) em uma FMC são os mesmos, a aplicação da tecnologia de grupo é baseada em similaridades de fixação, de ferramentas e de matéria prima. Peças que compartilham as mesmas características de fixação e

lista de ferramentas são agrupadas em uma mesma família. Isto significa que não há a necessidade de se efetuar um *setup* de fixação ou de ferramentas para a produção das peças que pertençam a uma mesma família, resultando em uma redução do tempo de *setup*.

2.4.3 – Escalonamento da produção

O escalonamento da produção trata da alocação temporal de tarefas para recursos limitados considerando um conjunto de restrições impostas (Sauer, 1997). Em outras palavras, o escalonamento tem por objetivo encontrar uma forma de fixar e sequenciar a utilização compartilhada dos recursos desde que as restrições sejam satisfeitas e as funções de custo reduzidas (Rodammer *et at*, 1988).

2.4.3.1 - O problema de escalonamento

O problema do escalonamento da produção surge quando um conjunto finito de recursos $\{M_1, M_2, M_3, \ldots, M_n\}$ deve ser compartilhado no processamento de um conjunto finito de operações $\{N_1, N_2, N_3, \ldots, N_n\}$ em um intervalo de tempo $\{t_i\}$. Em se tratando das restrições impostas, cada operação somente pode ser processada por um recurso no tempo, cada recurso somente processa uma operação no tempo e as operações são não-preemptivas, isto é, iniciado o processamento ele deve ser finalizado.

Definidos os recursos que serão utilizados no processamento, o problema se resume em encontrar uma seqüência de tarefas, que em termos gerais, minimize o tempo total de processamento (*makespan*) para garantir que as datas de entrega estabelecidas sejam efetivamente cumpridas.

Tradicionalmente, o escalonamento é definido como um problema de otimização, em que se busca uma seqüência ótima para as atividades. Grande parte dos problemas que requerem uma solução ótima para o escalonamento de recursos é classificada como um problema NP completo, em outras palavras, a solução ótima necessita de um tempo exponencial para ser encontrada (Sauer, 1997).

As políticas e procedimentos de sequenciamento aplicados num escalonamento consideram normalmente uma grande variedade de aspectos, mas principalmente, a disposição espacial dos

recursos de produção (Rabelo, 1997). Baseado nos recursos de produção, os problemas de escalonamento podem ser divididos em:

- ✓ Única máquina;
- ✓ Máquinas paralelas;
- ✓ *Flow shop*;
- ✓ *Job shop*;
- ✓ Em célula.

O escalonamento em uma única máquina é considerado o mais simples dos problemas. A seqüência é facilmente estabelecida, já que todas as tarefas serão processadas utilizando um único recurso de processamento. O caso de máquinas paralelas se aplica quando a mesma tarefa pode ser processada em mais de uma máquina. Neste caso, o problema consiste em certificar se é possível dividir as tarefas individuais entre dois ou mais recursos. Caso seja possível, obtém-se um melhor resultado no escalonamento, todavia a seqüência das tarefas é difícil de ser obtida.

No escalonamento *flow shop*, todas as tarefas seguem o mesmo caminho de uma máquina para a outra (Conway, 1967). A rota de processamento é fixa e o problema se resume em definir a seqüência em que as tarefas serão processadas. Este modelo é aplicado principalmente em sistema de produção seriado.

O *job shop* é considerado como o mais complexo dos problemas. As tarefas são alocadas nos recursos de produção randomicamente. Em outras palavras, um conjunto de tarefas pode ser processado em um conjunto de máquinas. A complexidade deste problema cresce exponencialmente à medida que o número de máquinas aumenta. Nestes casos, soluções ótimas são difíceis de serem alcançadas.

No caso de células as operações de cada uma das tarefas são processadas com a mesma seqüência e sobre os recursos de produção de uma dada célula. A idéia geral é de se criar ambientes de produção seriada, no intuito de facilitar a elaboração do escalonamento (Rabelo, 1997).

2.4.3.2 - Métodos de escalonamento

De acordo com Starbek (Starbek *et at*, 2003) os métodos de escalonamento podem ser divididos em dois grandes grupos:

- ✓ Métodos de otimização;
- ✓ Métodos de não-otimização.

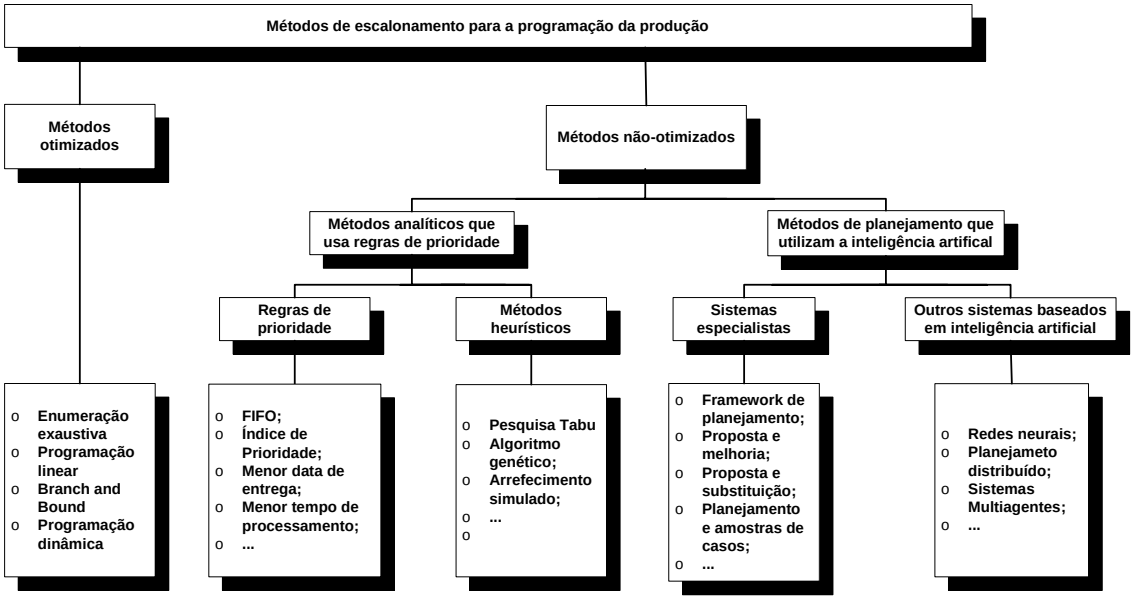

Figura 2.5 - Taxonomia dos métodos de escalonamento (adaptado de Starbek *et at*, 2003).

A figura 2.5 mostra a taxonomia dos métodos de escalonamento aplicados na programação da produção. Nos métodos de otimização, busca-se estabelecer um escalonamento ótimo. A melhor solução deve ser encontrada considerando os diversos fatores que influenciam na formação do escalonamento. A dificuldade de se aplicar este método reside no fato da grande quantidade de dados necessários e no tempo de processamento quando são muitos os recursos produtivos utilizados.

Os métodos de não-otimização surgiram à medida que a complexidade dos problemas de escalonamento tornou inviável a utilização dos métodos de otimização. Nestes casos, busca-se um meio termo entre a solução ótima e o tempo de processamento.

A abordagem baseada em regras de prioridade (também chamada de regras de sequenciamento) é provavelmente a heurística mais aplicada para resolver problemas de escalonamento devido à

simplicidade de implementação e o baixo custo computacional (Blazewicz *et at*, 1996). Nesta heurística, a seqüência de processamento das tarefas é definida de acordo com as metas de produção estabelecidas. A tabela 2.1 apresenta as principais regras de prioridade.

Tabela 2.1 - Regras de prioridade

Regras de prioridade	
Regra	**Descrição**
Shortest Process Time (STP)	Maior prioridade é dada a tarefa que possui o menor tempo de processamento
Earliest Due Date	Maior prioridade a tarefa que tem a data de entregra mais próxima
Fist in First out (FIFO)	Maior prioridade é dada a primeira tarefa recebida
Longest Process Time (LPT)	Consiste em priorizar as tarefas com o maior tempo de processamento
Last in First out (LIFO)	Maior prioridade é dada a última tarefa recebida

2.4.4 – Execução das tarefas

Wysk (Wysk *et at*, 1995) define o módulo de execução das tarefas como sendo o módulo responsável por executar as tarefas escalonadas mediante interface direta com os dispositivos físicos e os sistemas externos. As funções deste módulo podem ser divididas em: despachar, monitorar e reagir a perturbações.

2.4.4.1 – Despachar

Estabelecida a seqüência em que as tarefas serão enviadas para o chão-de-fábrica, esta função é utilizada para enviar instruções às estações de trabalho. As instruções são carregadas considerando as particularidades e as restrições de cada estação de trabalho, como o protocolo de comunicação, o formato da instrução, etc. Antes das instruções serem enviadas, esta função deve certificar se a estação de trabalho está sob regime normal de funcionamento.

2.4.4.2 – Monitorar

Função que verifica o cumprimento das tarefas definidas pelo módulo de escalonamento da produção, monitorando as estações de trabalho (*status*, regime de funcionamento, ferramentas, etc), o cumprimento das tarefas enviadas para o chão-de-fábrica, dentre outros.

Os dados obtidos são utilizados nas tomadas de decisão em médio e em curto prazo. Em médio prazo são emitidos relatórios de produção para avaliar se as metas estabelecidas no plano tático

estão sendo cumpridas. Em curto prazo, o sistema de controle deve reagir a perturbações baseado no menor índice de custo produtivo.

2.4.4.3 – Reagir a perturbações

Perturbações, como quebras de maquinas e de ferramentas, são inerentes a todo sistema produtivo e, portanto não podem ser completamente evitadas. Nos sistemas de controle que incorporam o conceito de flexibilidade, funções para reagir a perturbações são desenvolvidas no intuito de reduzir o custo produtivo causado pela falha e impedir a paralisação completa do sistema.

O desenvolvimento desta função requer o levantamento das possíveis falhas que o sistema produtivo estará sujeito. Feito este levantamento, a função reagir à perturbação é desenvolvida de acordo com a solução a ser tomada. Não existe um método padrão que possa ser utilizado para desenvolver esta função, já que a tomada de decisão deve ser feita considerando as particularidades do problema ocorrido.

2.5 – CONTROLE ESTATÍSTICO DO PROCESSO

De acordo com Duncan (Duncan, 2001) a primeira ferramenta estatística utilizada no controle da qualidade foi proposta por Walter A. Shewhart. Em um livro publicado em 1931, Shewhart propôs um conjunto de padrões que poderiam ser utilizados para aplicar métodos estatísticos no controle do processo. Posteriormente, dois pesquisadores que trabalhavam no mesmo laboratório propuseram uma metodologia para aplicar a teoria da estatística na inspeção por amostragem. Estes trabalhos constituem a base teórica da estatística aplicada no controle da qualidade.

2.5.1 – A variabilidade no processo produtivo

Uma das primeiras preocupações de Shewhart, ao propor métodos estatísticos aplicados no controle da qualidade, foi classificar as variabilidades que podem ser encontradas em um processo. A variabilidade randômica é de caráter aleatório, portanto não possui uma causa precisamente conhecida. Os projetistas já consideram seus efeitos ao estabelecerem as tolerâncias de projeto.

Por outro lado, existem variações causadas por fatores (como desgaste excessivo de ferramentas, falha de máquinas, dentre outros) que não podem ser precisamente previstas durante a fase de projeto. Estas variações alteram o resultado final desejado, em alguns casos levando à perda do produto. As variações fixas são responsáveis pela alteração da variabilidade no sistema impedindo que os requisitos estabelecidos no projeto sejam alcançados.

Groover (Groover, 2003) afirma que o comportamento da variabilidade do sistema pode ser deduzido estatisticamente a partir de uma curva de distribuição normal em que as variações randômicas se distribuem aleatoriamente em torno da média, enquanto as variações fixas se afastam da mesma (figura 2.6).

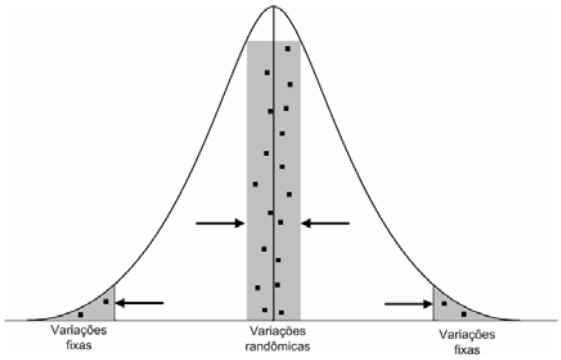

Figura 2.6 - Curva de distribuição normal.

Ao aplicar a estatística no controle do processo, o principal objetivo é identificar a presença de variações fixas, para que decisões possam ser tomadas antes mesmo da fabricação de peças não-conformes. Um processo é considerado sobre controle estatístico quando está sujeito somente a variações randômicas. Variações fixas indicam que o processo não está sob controle estatístico.

2.5.2 – Índice de capabilidade do processo

A análise da capacidade é o estudo destinado a medir a aptidão do processo em produzir de acordo com o especificado. Uma das formas de expressar a capacidade do processo é através de índices de capabilidade. Em indústrias de manufatura, estes índices têm sido propostos fornecendo medidas numéricas se um processo é capaz de reproduzir itens conforme a qualidade e a confiança requeridas pela fábrica (Pearn *et at*, 2003).

Caso as tolerâncias de projeto sejam especificadas com o mesmo valor das tolerâncias naturais, o índice de capabilidade pode ser definido como:

$$C_p = \frac{UTL - LTL}{6\sigma} \tag{2.1}$$

Em que:

Cp = Índice de capabilidade do processo;

UTL = Limite de tolerância natural superior;

LTL = Limite de tolerância natural inferior;

σ = Desvio padrão da média dos valores medidos.

2.5.3 – Cartas de controle

As cartas de controle são os resultados dos primeiros trabalhos desenvolvidos por Shewhart, na aplicação da estatística no controle da qualidade. Além de relevar a presença de variabilidade não-natural das amostras monitoradas, uma carta de controle fornece indícios das suas possíveis causa (Pacella *et at*, 2004).

Uma carta de controle é representada por uma linha central e por limites de controle. A linha central representa à média dos valores amostrados, enquanto os limites de controle delimitam a região das amostras sob controle. A figura 2.7 mostra os elementos de uma carta de controle.

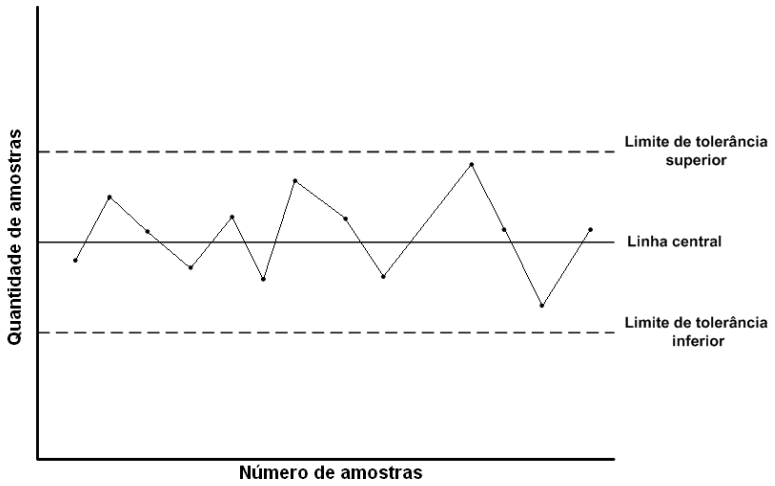

Figura 2.7 - Carta de controle.

Estando o processo operando sobre controle estatístico, uma série de amostras de pequeno tamanho é coletada e a característica de interesse é obtida. Estas amostras são representadas por pontos interligados[2] por segmentos de retas. As linhas pontilhadas representam os limites de controle. Conforme observado, todos os pontos estão na região delimitada pelos limites de controle indicando que este processo está sob controle.

2.5.4 – Pré-controle

Embora as cartas de controle seja o método mais empregado no controle estatístico do processo, as dificuldades relacionadas à elaboração e análise dos resultados fizeram com que a empresa de consultoria Rath & Strong propusessem uma técnica alternativa às cartas de controle. O pré-controle, algumas vezes chamado de gráfico de farol, foi desenvolvido para monitorar a proporção de unidades não-conformes produzidas em um processo de manufatura (Steiner, 1998).

A diferença básica dentre o pré-controle e as cartas de controle está no objetivo do método. Enquanto nas cartas de controle, o objetivo é monitorar e controlar a variabilidade do processo, no pré-controle o foco está na avaliação da capacidade do processo, a fim de evitar a produção de peças não-conformes.

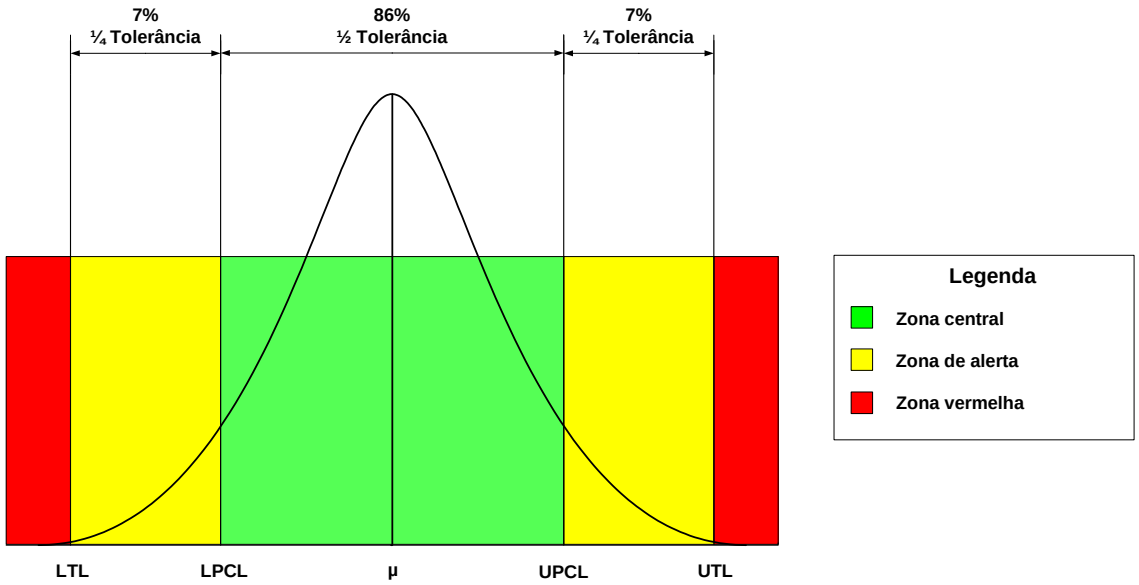

Figura 2.8 - Gráfico de pré-controle.

[2] A interligação dos pontos por segmento de retas não é necessária. Isto serve apenas para demonstrar mais facilmente o comportamento das medidas ao longo do tempo.

De acordo com Juram (Juran *et at*, 1992) o poder estatístico desde método reside na regra do produto das probabilidades independentes. Assim, a probabilidade de duas amostras consecutivas estarem na zona de alerta é de 1/196[3]. Considerando todas as quatro possíveis permutações entre duas peças consecutivas, a chance é de 4/196 (próximo a 2%), em outras palavras, o operador receberá um sinal para ajustar o processo quando na verdade, não deveria ser ajustado em 2% das vezes (Urdhwareshe, 2002).

A aplicação do pré-controle pode ser dividida em três fases:

- ✓ Qualificar a preparação;
- ✓ Operação;
- ✓ Freqüência de amostragem.

Na primeira fase, determina-se a capacidade do processo a partir da coleta de amostras. Esta coleta é feita até que cinco amostras consecutivas estejam na zona central. Sob esta condição, o processo é considerado sobre controle estatístico e passa para a fase seguinte. Caso uma amostra esteja na zona de alerta, a contagem deve ser reiniciada. Por outro lado, se duas amostras estiverem na zona de alerta ou uma na zona vermelha, o processo deve ser ajustado e a contagem reiniciada.

Durante a fase de operação, deve-se retirar uma amostra periódica de duas unidades. Caso as duas unidades estejam na zona central ou apenas uma na zona de alerta, o processo não precisa ser ajustado. Por outro lado, se as duas estiverem nas zonas de alertas, o processo deve ser ajustado, sendo a pior condição quando as duas amostras estiverem em zonas de alertas opostas ou ao menos uma estiver na zona vermelha. Nestes casos, deve-se parar a fabricação, pois as peças não-conformes já estão sendo fabricadas.

Na última fase é definida a freqüência de amostragem. Esta freqüência é determinada dividindo-se por seis a média do período entre duas paradas, isto é, entre duas unidades na zona de alerta ou uma unidade na zona vermelha. Caso esta condição seja, por exemplo, verificada no período de 6 horas, a freqüência de amostragem definida é de 1 hora.

[3] Conforme o teorema do produto das probabilidades: P(a) = 1/14, P(b) = 1/14 → P(a) x P(b) = (1/14)x(1/14) = 1/196.

Como o pré-controle tem como objetivo evitar a produção de peças não-conformes é importante salientar a necessidade de se trabalhar com altos índices de capabilidade. Desse modo, esta técnica é inadequada para processos com baixos índices de capabilidade (C_p<1), já que uma pequena alteração resulta em um incremento da variabilidade do processo (Ledolter *et at*, 1997). Todavia, a vantagem desta técnica quando se trabalha com índices de capabilidade maiores que a unidade reside no pequeno número de amostras necessárias (duas amostras) se comparado com as cartas de controle (vinte a vinte cinco amostras).

2.6 – REDES DE PETRI

2.6.1 – Definição

As Redes de Petri estão sendo usadas em todos os aspetos de projeto e operações de um FMS: modelagem e verificação, análise de desempenho, escalonamento, controle e monitoramento (Girault *et at*, 2003). Ou-Yan (Ou-Yan *et at*, 2000) propôs uma Rede de Petri Modificada para modelar e controlar o fluxo de informações das estações de trabalho de um FMS. Lee (Lee *et at*, 2004) analisa o escalonamento cíclico para determinar o tempo de ciclo ótimo e minimizar o *work-in-process* (WIP) utilizando uma Rede de Petri temporal extendida.

De acordo com Girault (Girault *et at*, 2003) uma Rede de Petri Ordinária pode ser definida como uma tripla:

$$N = (P, T, Ppre, Ppost) \tag{2.2}$$

Em que:

$P =$ É o conjunto finito de lugares (o conjunto de lugares da rede N);
$T =$ É o conjunto finito de transições (o conjunto de transições de N);
$Ppre, Ppost \in IN^{|P|\times|T|} =$ São matrizes (matrizes de pré e pós-incidência da rede).

2.6.2 – Propriedades de uma Rede de Petri

Através do estudo das propriedades de uma Rede de Petri (RdP) é possível analisar as principais características do sistema que está sendo modelado. Ao interpretar as propriedades no contexto do sistema, o projetista pode identificar a presença ou ausência de propriedades funcionais

específicas do sistema que está sendo projetado. Dentre as propriedades que dependem do estado inicial ou da marcação da rede estão:

- ✓ Alcançabilidade;
- ✓ Vivacidade;
- ✓ Reversibilidade;
- ✓ Conflito.

O estudo da alcançabilidade da rede demonstra a capacidade do sistema em atingir um determinado estado. Isto permite descrever o comportamento dinâmico do sistema através de um mapa de estados alcançáveis ou mapa de alcançabilidade. A vivacidade verifica a potencialidade da rede em disparar em todas as marcas alcançáveis. Uma rede viva é uma rede que não possui *deadlocks*.

A terceira propriedade listada, a reversibilidade está relacionada com a capacidade da rede em retornar a marcação inicial (M_0) a partir de qualquer marcação (M_i). Com isso é possível verificar a capacidade do sistema em retornar ao estado inicial após ocorrer uma falha. A última propriedade listada sugere a presença de indeterminação do sistema, o que deve ser considerado durante a fase de modelagem.

2.6.3 – Modelagem de sistemas a eventos discretos por Redes de Petri interpretadas

As Redes de Petri têm se mostrado uma importante ferramenta para análise e modelagem de sistemas a eventos discretos (SED). Com esta ferramenta é possível descrever a estrutura lógica e comportamental do sistema modelado (Gu *et at*, 2000). Os passos para a modelagem utilizando Redes de Petri interpretadas podem ser divididos em (Cardoso *et at*, 1997):

- ✓ Definição da estrutura da rede;
- ✓ Analisar e verificar as propriedades da rede (garantindo a ausência de *deadlocks*);
- ✓ Simular a rede interpretada para tentar obter o comportamento dinâmico do sistema.

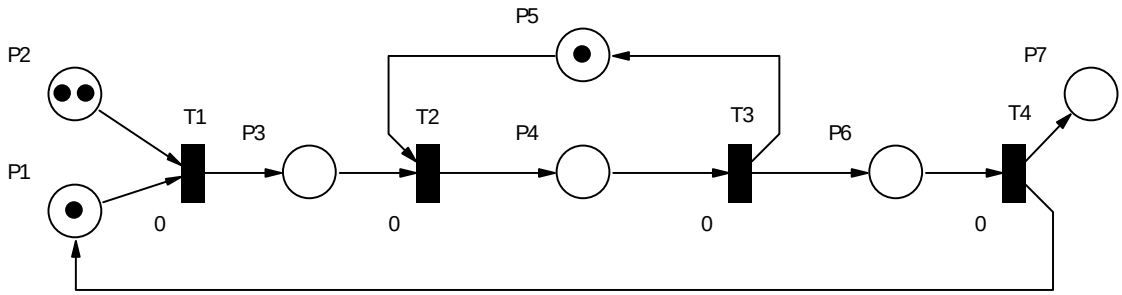

Figura 2.9 - Rede de Petri – Estrutura 1.

A figura 2.9 apresenta a estrutura de uma RdP. Esta rede representa apenas um modelo gráfico/matemático do sistema a ser modelado. Definida a estrutura da rede, o passo seguinte consiste em atribuir um sentido concreto ao modelo matemático, isto é, interpretar a rede. Em se tratando de sistemas flexíveis de manufatura, os elementos da Rede de Petri (lugares, transições arcos e marcas) possuem diferentes significados quando o sistema está sendo interpretado:

- ✓ Lugares representam o estado dos recursos;
- ✓ Transições modelam a seqüência de eventos que modifica o estado dos recursos;
- ✓ Os arcos representam o fluxo de processamento;
- ✓ As marcas a disponibilidade de recursos (matéria prima ou produto acabado).

A figura 2.10 apresenta a Rede de Petri Interpretada. Nesta rede, utiliza-se como recurso de produção, um manipulador robótico, um *pallet* de peças e um Torno CNC. Os lugares P1, P3 e P6 correspondem aos estados assumidos pelo manipulador (robô livre, robô carrega tarugo, robô descarregando peça), enquanto que os lugares P4 e P5 correspondem estados assumidos pelo Torno CNC (torno usinando, torno livre).

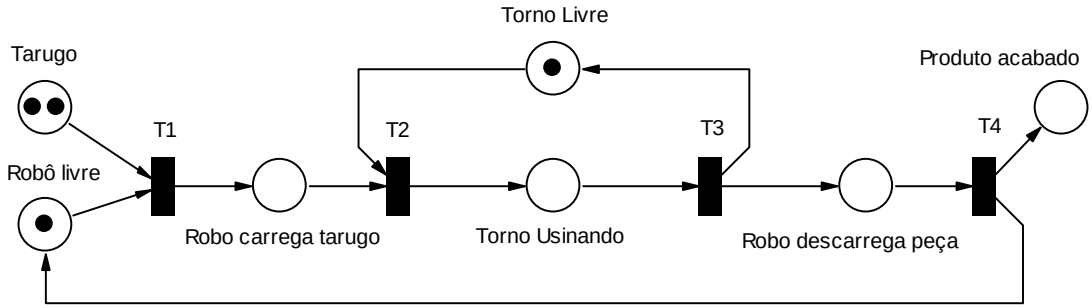

Figura 2.10 - Rede de Petri interpretada – Estrutura 2.

As marcas nos lugares Robô Livre, Tarugo e Torno Livre são necessárias para que o sistema possa operar. Estas marcas foram definidas no segundo passo da modelagem, portanto fazem parte da estrutura da rede, já que a ausência dessas marcas levaria a rede a um estado de *deadlock* (as transições T1, T2 e T3 não poderiam ser sensibilizadas).

Nos sistemas reais, o disparo de uma transição não está associado somente à disponibilidade de um recurso produtivo (transição sensibilizada), mas precisamente ao tempo em que o recurso leva para estar disponível. Desse modo, para finalizar a interpretação do modelo é necessário associar expressões booleanas às transições. Estas expressões representam o estado de uma transição que são modificados com a ocorrência de um evento externo.

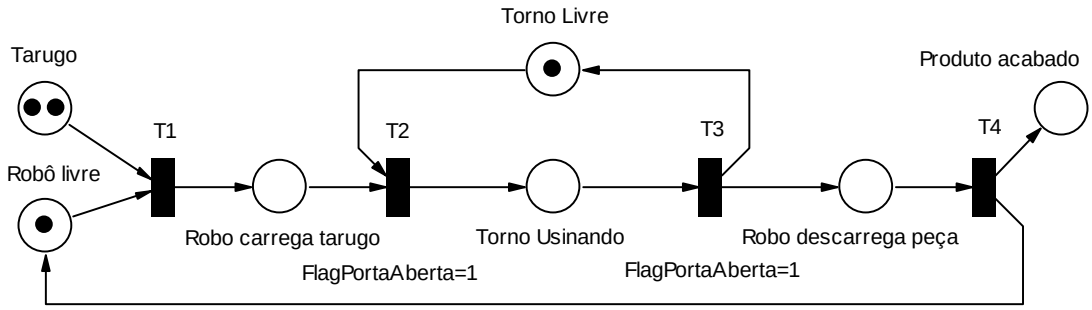

Figura 2.11 - Rede de Petri interpretada – Estrutura 3.

A figura 2.11 mostra a rede final que descreve a modelagem do sistema. Observe que apesar do torno estar disponível é necessário que a condição booleana Porta Aberta seja satisfeita para que as transições T2 e T3 sejam disparadas. Em outras palavras, é necessário que o torno envie um sinal para robô indicando que a porta está aberta.

2.6.4 – Métodos de implementação

Implementar uma RdP consiste em converter o modelo gráfico/matemático obtido com a modelagem em um sistema que possa ser utilizado na prática. Redes de Petri têm sido implementadas tanto em hardware quanto em software. Sotto (Sotto *et at*, 2001) discute a implementação de Redes de Petri em FPGA's utilizando como linguagem descritiva o VHDL. Minas *et at* (Minas *et at*, 2002) propõem uma ferramenta de programação de PLC's baseada em Redes de Petri sinal interpretadas.

Independente da forma como a rede será implementada, os métodos de implementação de uma rede baseada em regras podem ser divididos em (Cardoso *et at*, 1997):

- ✓ Abordagem procedimental;
- ✓ Abordagem não procedimental.

No primeiro e mais específico método, a rede é implementada como um componente conservativo que contém apenas uma ficha (a rede pertence a uma subclasse de máquinas de estados). No segundo método, a rede é convertida em um banco de regras em que cada transição é considerada como uma regra de transformação. A rede possui transições internas e externas. As transições internas estão associadas a eventos de transformação interna, enquanto as transições externas são associadas a eventos externos para permitir que a rede possa interagir com a planta.

3 – WEBFMC: MODELAGEM DA CÉLULA E DA UNIDADE DE GERENCIAMENTO

Neste capítulo é apresentado o projeto e a modelagem da FMC e da Unidade de Gerenciamento. A modelagem da célula teve como principal objetivo definir o posicionamento das estações de trabalho (*layout* da célula), bem como especificar as mensagens que devem ser trocadas entre elas. No que se refere a modelagem da MgU, os métodos IDEF0 e IDEF1x foram utilizados para especificar o modelo funcional e de dados respectivamente.

3.1 – DESCRIÇÃO DAS ESTAÇÕES DE TRABALHO

A Célula Flexível de Manufatura a qual esta dissertação faz referência é composta por uma unidade de processamento (Centro de Torneamento), uma unidade de manipulação e transporte de materiais (manipulador robótico), uma unidade de inspeção (micrômetro laser), uma unidade de armazenamento de peças (*pallet*), um AGV e por um sistema de controle[1] (Unidade de Gerenciamento). A figura 3.1 mostra as estações de trabalho que compõem a FMC.

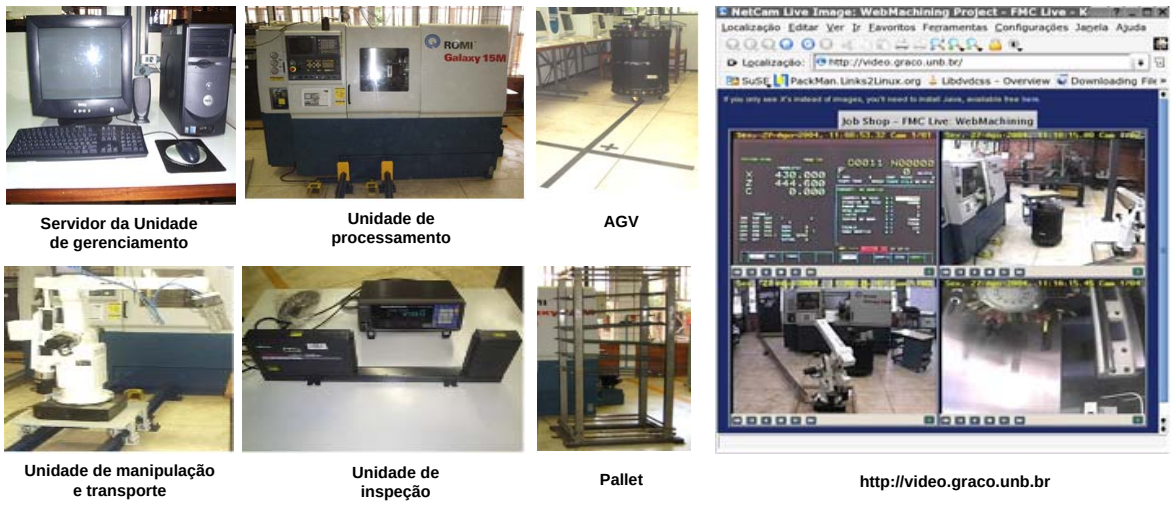

Figura 3.1 - Estações de trabalho da célula.

3.1.1 – Unidade de processamento – Centro de Torneamento

O processamento da matéria – prima é realizada por um Centro de Torneamento Romi Galaxy 15M CNC Fanuc (18i-ta) com interface de comunicação *Ethernet*. A torre contendo 12 porta –

[1] A descrição da Unidade de Gerenciamento será realizada ao longo deste trabalho.

ferramentas possibilita a acoplagem de ferramentas acionadas, o que permite a execução das principais operações de usinagem (torneamento, fresamento e furação).

3.1.2 – Unidade de manipulação e transporte – robô ASEA IRB6

O transporte e manipulação de peças entre as estações de trabalho são feitos por meio de um manipulador robótico ASEA IRB6, com 5 graus de liberdade, alcançabilidade de 1.140 mm e capacidade máxima de manipulação de 6Kg. O manipulador possui uma garra pneumática *Schunk* dotada de um sensor de posicionamento programável que permite verificar a abertura e fechamento da garra em cinco posições diferentes (*Open*, A, B,C e *Close*).

3.1.3 – Unidade de inspeção – Micrômetro laser

Um Micrometro laser Mitutoyo é utilizado como unidade de inspeção da célula. Esta estação de trabalho é composta por uma unidade de medição (modelo LSM 512H, faixa de medição de 1 a 120 mm, resolução de 0.1 μM e faixa de temperatura de 0 a 40ºC) e uma unidade de processamento (modelo LSM 6100) dotada de uma interface RS232C e um conjunto de funções utilizadas na programação *on-line* desta estação de trabalho.

3.1.4 – Unidade de transporte de ferramentas - AGV

O transporte de ferramentas é realizado por um robô móvel (*Nomad* XR4000) que trabalha como um AGV na célula. O *Nomad* XR4000 é um sistema avançado de robótica móvel, que incorpora sistema de controle, sensores, gerenciamento de energia, programação e comunicação e tecnologias de desenvolvimento de software (Nomadic Tecnologies, 1999). Com dimensões de 620mm x 850mm, aceleração translacional máxima de $5m/s^2$, rotacional de $2rad/s^2$ e com alimentação através de baterias, este robô possibilita o desenvolvimento de pesquisas em robótica móvel, navegação robótica e visão computacional.

3.1.5 – Unidade de armazenamento – *Pallet*

Esta estação de trabalho é responsável pelo posicionamento e armazenamento da matéria – prima e dos produtos acabados. Com dimensões de 2000 mm (altura) por 660 mm (largura), o

pallet possui 6 unidades armazenadoras com dimensões variáveis (100 a 500 mm) que garantem maior flexibilidade na acomodação das peças.

3.2 – MODELAGEM DA CÉLULA

Uma técnica de modelagem pode ser utilizada para analisar e identificar as especificações que possivelmente limitam o funcionamento do sistema (Teixeira *et at*, 2005). Através da modelagem é possível visualizar o sistema como um todo, especificar seu comportamento e documentar as tomadas de decisões. Duas são as técnicas de modelagem utilizadas: modelagem por simulação e modelagem a eventos discretos.

3.2.1 – Modelagem por simulação (Workspace)

Para definir a disposição física das estações de trabalho em relação ao *layout* da célula é necessário avaliar a movimentação da unidade de manipulação e transporte de materiais, bem como a trajetória do AGV. A partir desta análise é possível definir a área de trabalho destas unidades e ainda detectar a possíveis rotas de colisão entre elas. As figuras 3.2 e 3.3 demonstram a modelagem desenvolvida utilizando o software *Workspace*^{TM}.

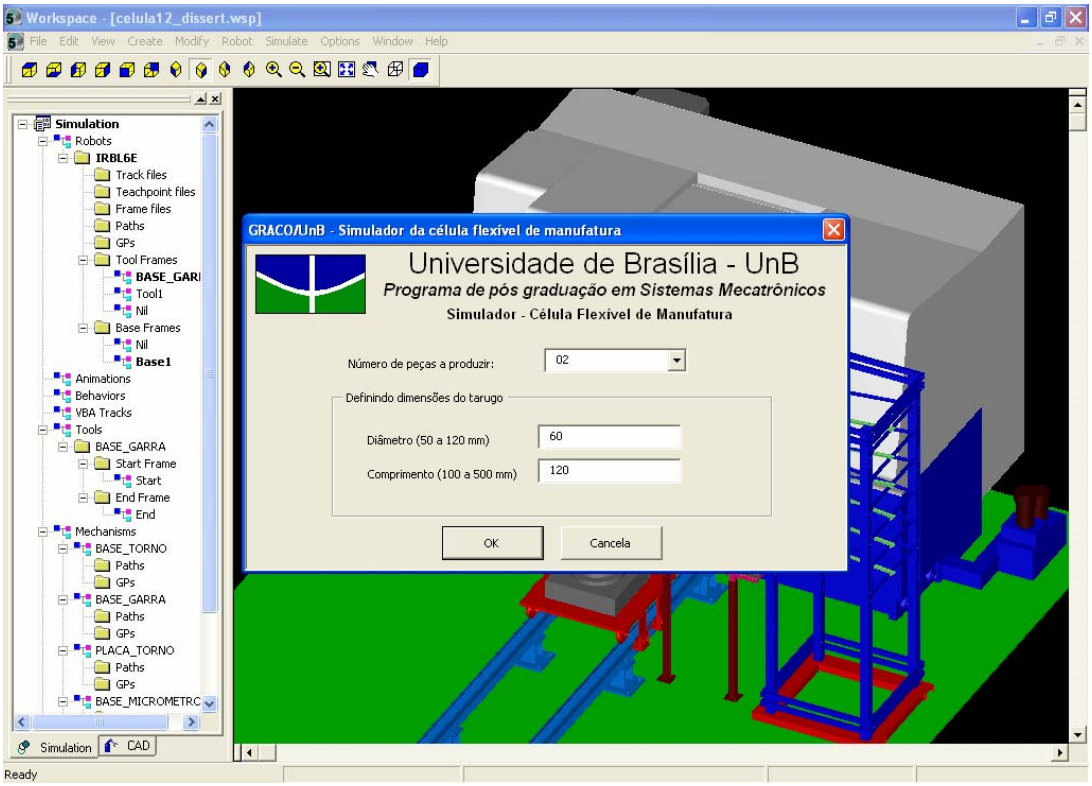

Figura 3.2 - Modelagem da célula via *workspace* – Simulador da célula.

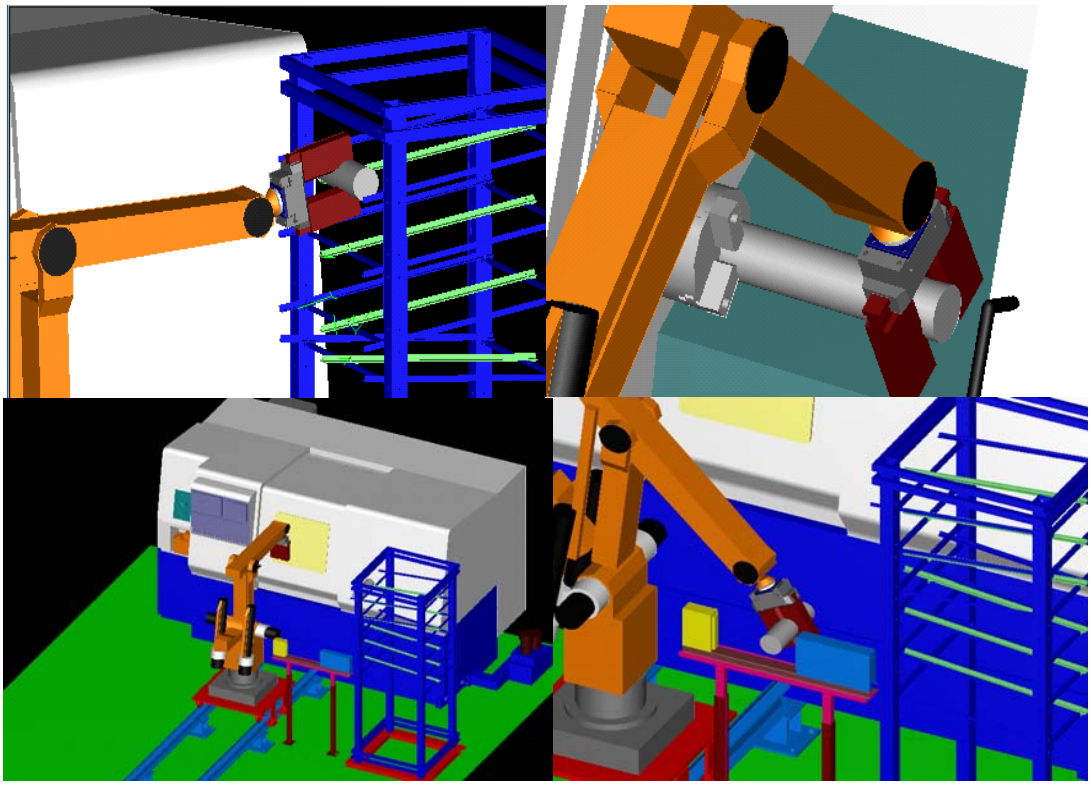

Figura 3.3 - Aspectos da modelagem via *workspace*.

O *Workspace* é um sofisticado sistema de simulação projetado para a indústria robótica, embora ofereça aplicações em outras áreas de mecanismos ou simulação de objetos (Workspace, 2006). Com este sistema é possível importar projetos CAD de outros sistemas CAD, modelar células de trabalho a partir de objetos CAD, definir a trajetória automaticamente com alto nível de processamento, além de permitir a programação *off-line* e *download* de programas sem a necessidade de pós-processamento.

O resultado da modelagem por simulação (via *workspace*) demonstrou a necessidade em posicionar a unidade manipulação e transporte cerca de 1000 mm da base do Centro de Torneamento e a 380 mm do solo e ainda a área de trabalho do AGV. O posicionamento no manipulador robótico nas coordenadas estabelecidas é fundamental para que a matéria-prima possa ser inserida na placa do torno pelo manipulador.

No entanto, posicionar o manipulador nas coordenadas estabelecidas impede que o operador tenha acesso à torre porta-ferramenta, o que impossibilita o *setup* de ferramentas. A solução adotada resultou no projeto e na construção de um carro posicionador para adicionar mais um grau de liberdade ao manipulador, permitindo que o *setup* de ferramentas seja realizado.

3.2.2 – Modelagem a eventos discretos (Redes de Petri)

As Redes de Petri estão sendo usadas em todos os aspetos de projeto e operações de um FMS: modelagem e verificação, análise de desempenho, escalonamento, controle e monitoramento (Girault *et at*, 2002). Na modelagem da célula, as RdP estão sendo utilizadas como ferramenta de análise e levantamento da árvore de alcançabilidade a fim de evitar os indesejáveis *deadlocks*.

A figura 3.4 apresenta a Rede de Petri (simplificada) utilizada para analisar o funcionamento da célula. A presença de marcas nos lugares representa a disponibilidade dos recursos produtivos. Para as transições T1, T2 e T4 serem disparadas é necessário respectivamente que o manipulador esteja disponível, o *pallet* com matéria-prima, o torno esteja operando e o micrômetro livre. Estas condições são necessárias para iniciar a fabricação, portanto devem ser verificadas antes mesmo da produção ser iniciada.

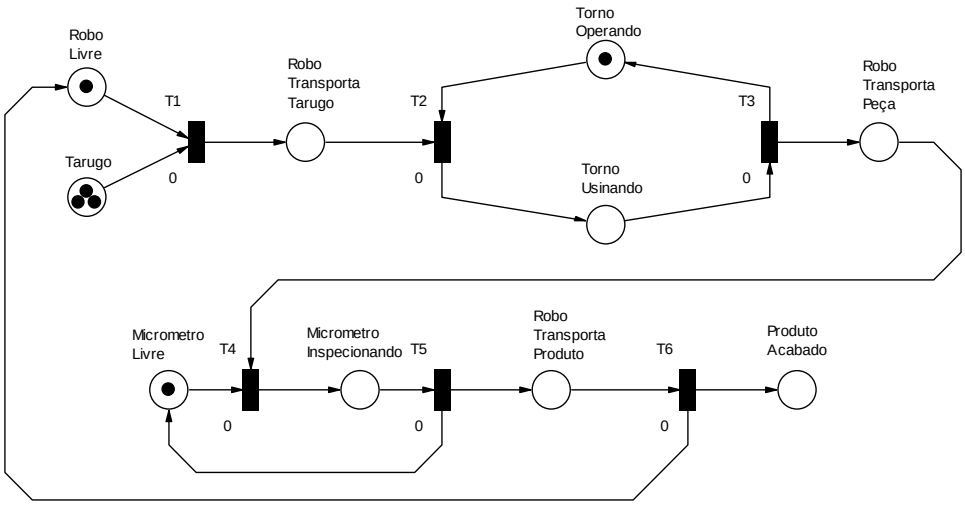

Figura 3.4 - RdP simplificada da FMC.

As figuras 3.5 e 3.6 mostram respectivamente a RdP detalhada da célula e a simulação para a produção de três pecas. A rede detalhada foi projetada utilizando a ferramenta ARP versão 2.4 desenvolvida pelo laboratório de controle e microinformática da Universidade Federal de Santa Catarina (LCMI/DEFL/UFSC) (ARP2-4, 2006). Maiores detalhes sobre a estrutura da rede e a simulação podem ser obtidos no apêndice A.

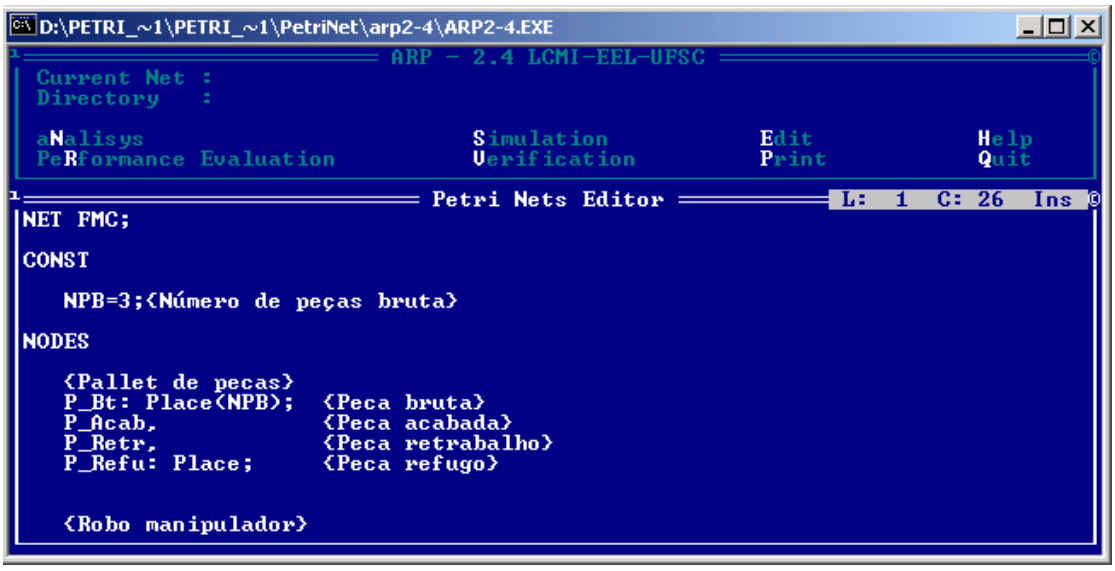

Figura 3.5 - RdP detalhada da célula.

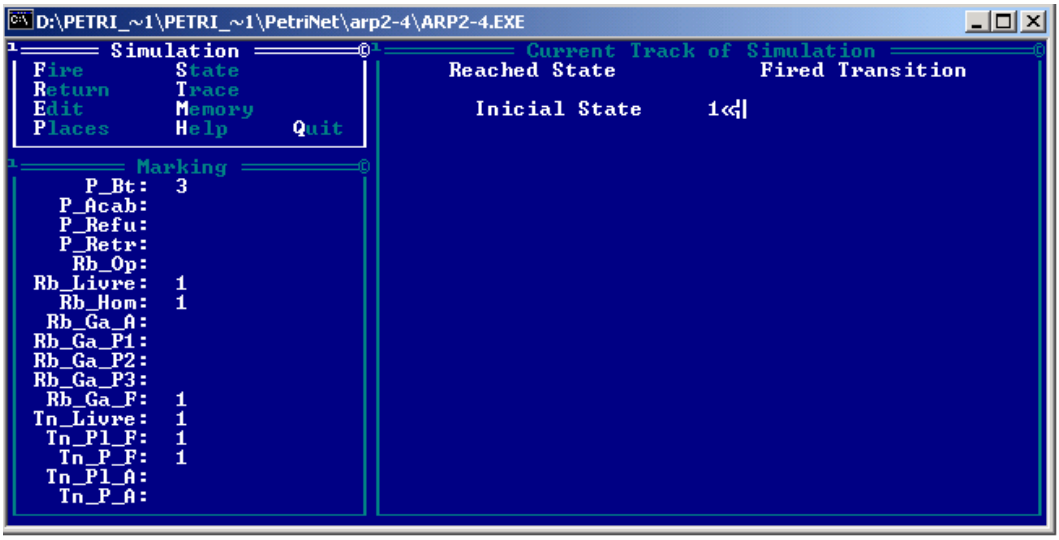

Figura 3.6 - Simulação da RdP detalhada.

O estudo do comportamento dinâmico das estações de trabalho, baseado na modelagem por Redes de Petri, resultou na identificação das mensagens que devem ser trocadas entre as estações de trabalho de acordo com a evolução dos eventos. Estas trocas de mensagens são necessárias para garantir o sincronismo das operações e evitar a presença de *deadlocks* (robô tenta posicionar a peça na placa do torno estando o mesmo com a porta fechada).

Os resultados da modelagem por Redes de Petri descritos no apêndice A foram repassados ao setor de engenharia da Romi (fabricante do Centro de Torneamento). A Romi, por sua vez, utilizou-se dos resultados desta simulação para projetar a integração física e lógica das estações de trabalho (alterações no *ladder* do Centro de Torneamento, etc).

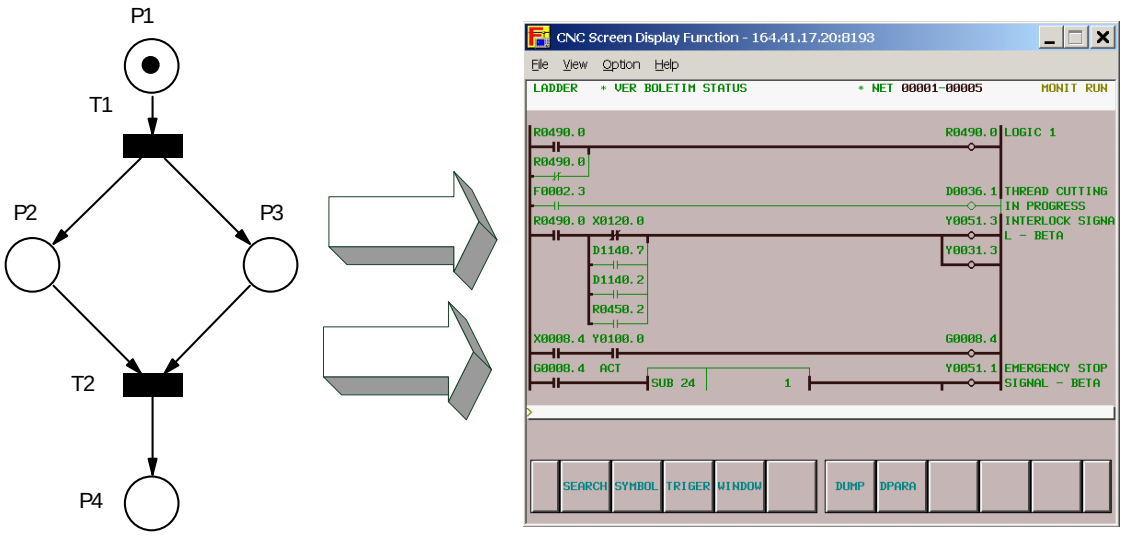

Figura 3.7 - Alteração do *Ladder* do Centro de Torneamento.

3.3 – MODELAGEM DA UNIDADE DE GERENCIAMENTO

3.3.1 – Modelagem Funcional (IDEF0)

O método IDEF0 (modelagem funcional) pode ser utilizado para modelar as decisões, ações e atividades de uma organização de manufatura ou de um sistema em uma forma gráfica estruturada (Kim *et at*, 2002). A abordagem adotada neste método descreve cada processo (ou atividade) como uma combinação de processos, entradas, controles e mecanismos em um modelo hierárquico (Ryan *et at*, 2006).

O método IDEF0 foi utilizado para especificar e documentar o modelo funcional da Unidade de Gerenciamento. Além de apresentar uma breve descrição especificando o propósito do modelo, este método também especifica o ponto de vista que ajuda a guiar e a restringir a criação do mesmo (IDEF0, 1993). A figura 3.8 apresenta o diagrama A0 com as respectivas entradas, mecanismos, controles e saídas.

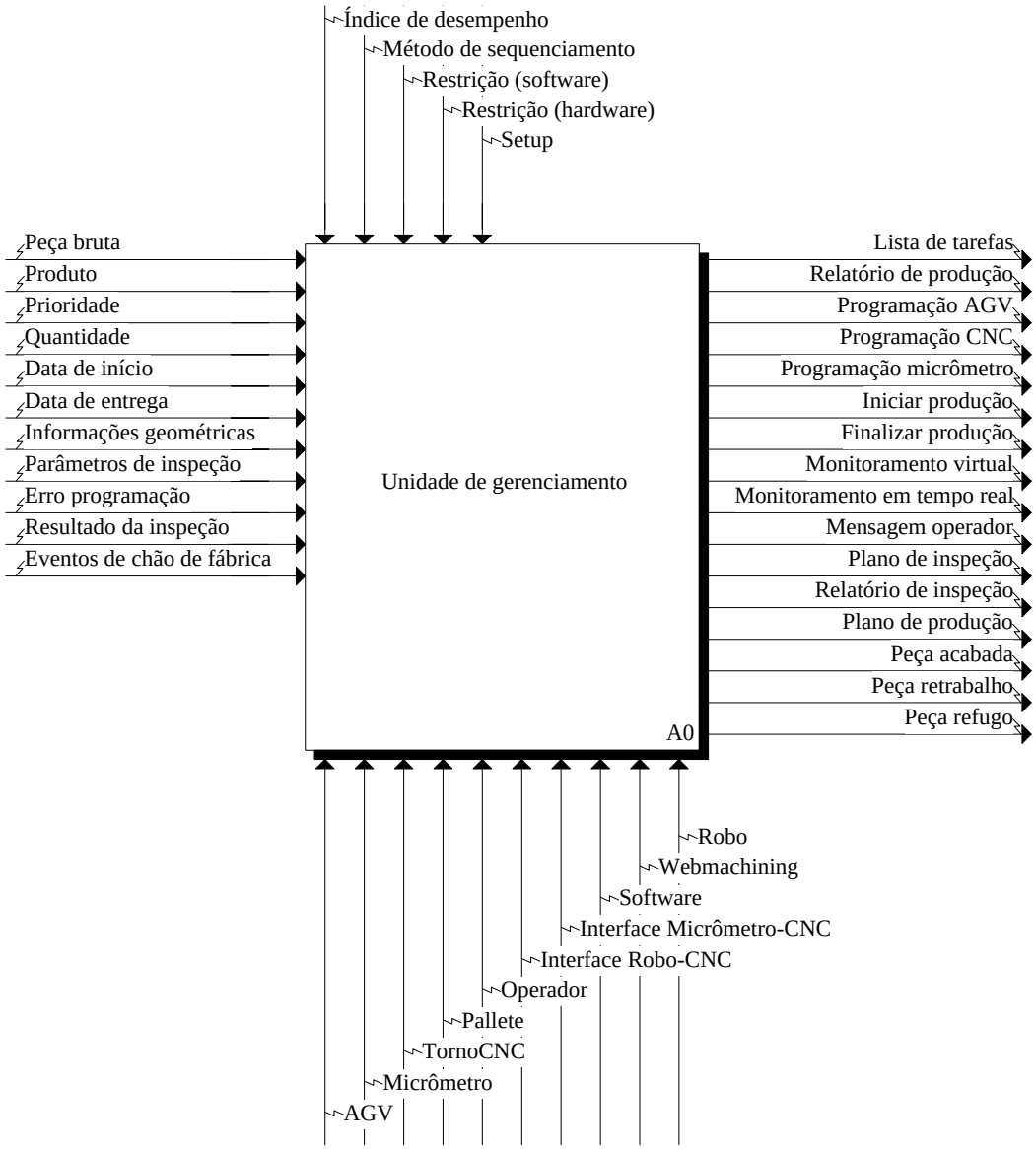

Figura 3.8 - Diagrama A0 da Unidade de Gerenciamento.

Uma importante vantagem em se utilizar este método reside no fato da modelagem ser estritamente *top-down*. Contudo, para especificar com maior nível de detalhamento o modelo funcional da Unidade de Gerenciamento, a função A0 é decomposta em outras três subfunções: *Scheduler* (A1), *Dispatcher* (A2) e *Monitor* (A3). A figura 3.9 apresenta o diagrama filho da função A0.

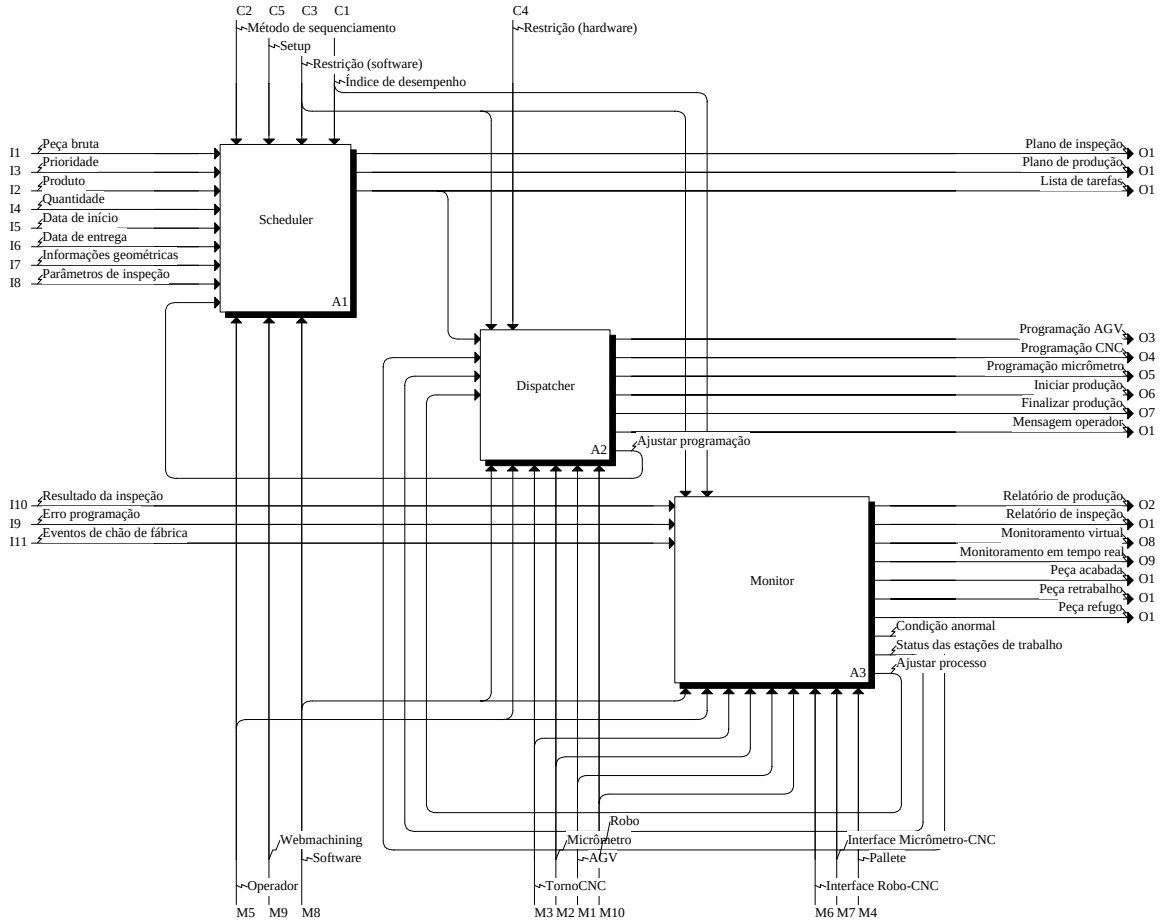

Figura 3.9 Diagrama filho da função A0.

3.3.1.1 – Scheduler

A função *Scheduler* consiste na programação da inspeção (A11) e da produção (A12) (figura 3.10). A programação da inspeção agrupa os programas de inspeção que serão utilizados. Cada programa armazena os atributos necessários para realizar a inspeção da peça, isto é, informações geométricas (diâmetro e tolerâncias) e as condições em que a medição deverá ser efetuada (escala, referência, segmentação, etc).

A programação da produção (A12) é subdividida em outras três sub-funções: Plano mestre da produção (A121), Programar a produção (A122) e Escalonar a produção (A123) (figura 3.11). A função Plano mestre da produção reúne as ordens de trabalho cadastradas na base de dados, além de oferecer suporte a inserção, edição ou até mesmo a edição de uma ordem de trabalho.

Programar a produção (A122) consiste em selecionar as ordens de trabalho (A1221) que serão enviadas para o chão-de-fábrica e formar as famílias de peças (A1222). A formação das famílias de peças está condicionada as restrições de *setup* (especificamente a lista de ferramentas) impostas pelo plano de processo gerado pelo sistema CAD/CAPP/CAM (*Webmachining*). Após a formação das famílias de peças, o escalonamento da produção (A123) deve ser executado.

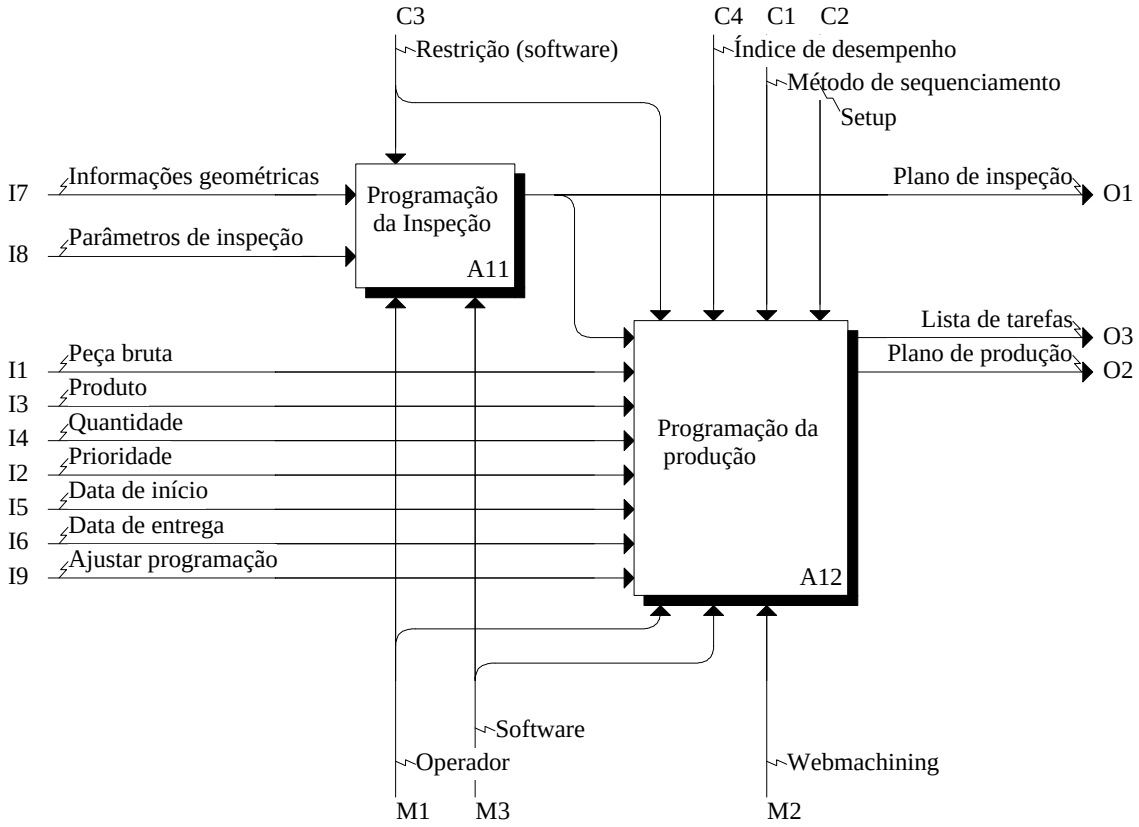

Figura 3.10 - Diagrama filho da função A1.

O escalonamento da produção pode ser executado em modo manual ou automático. No modo manual o operador é quem define a seqüência em que as ordens serão produzidas. No modo automático, o escalonamento é feito pela MgU baseado no método e no critério de sequenciamento selecionado pelo operador.

No método *forward*, as ordens de trabalho são alocadas o mais cedo possível, considerando a data de início e a disponibilidade de recursos. No método *backward*, a última operação é a primeira a ser escalonada a partir da data de entrega. Os critérios de sequenciamento disponíveis são baseados em regras de prioridade. Quatro são as regras de prioridade que podem ser utilizadas: *Priority*, *Earliest Due Date*, *First In First Out* e *Shortest Processing Time*.

O resultado obtido após a execução do escalonamento é apresentado em um gráfico de Gantt. Neste gráfico as operações de cada tarefa são alocadas ao longo do tempo de acordo com a seqüência estabelecida na execução do escalonamento. Esta função Escalonar a produção (A123) também fornece a possibilidade de salvar a lista de tarefas a ser posteriormente carregada.

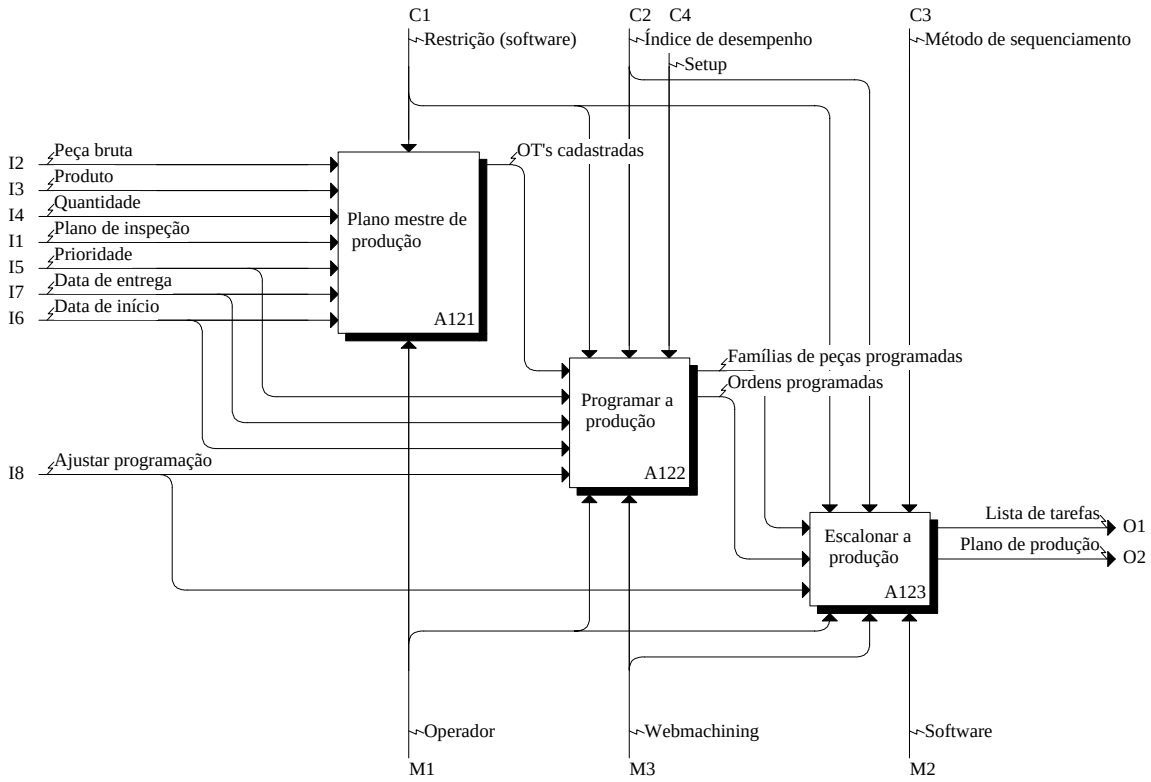

Figura 3.11 - Diagrama filho da função A12.

3.3.1.2 – Dispatcher

Esta função inicializa as estações de trabalho alocando as tarefas e executando o controle em tempo real das atividades. A função *Dispatcher* é decomposta em outras três: ExecutarSetupFMC (A21), ExecutarOperaçãoFMC (A22) e EmergênciaFMC (A23) (figura 3.12).

ExecutarSetupFMC (A21) encapsula o processo de *setup* célula recebendo como entrada a lista de tarefas e fornecendo como saída o despacho de instruções para as estações de trabalho, bem como as condições necessárias para iniciar a produção. Esta função é subdivida em outras seis:

ExecutarSetupCNC (A211), ExecutarSetupMicrometro (A212), ExecutarSetupAGV (A213), ExecutarSetupRobo (A214), ExecutarTryOut (A215), Iniciar a produção (A216) (figura 3.13).

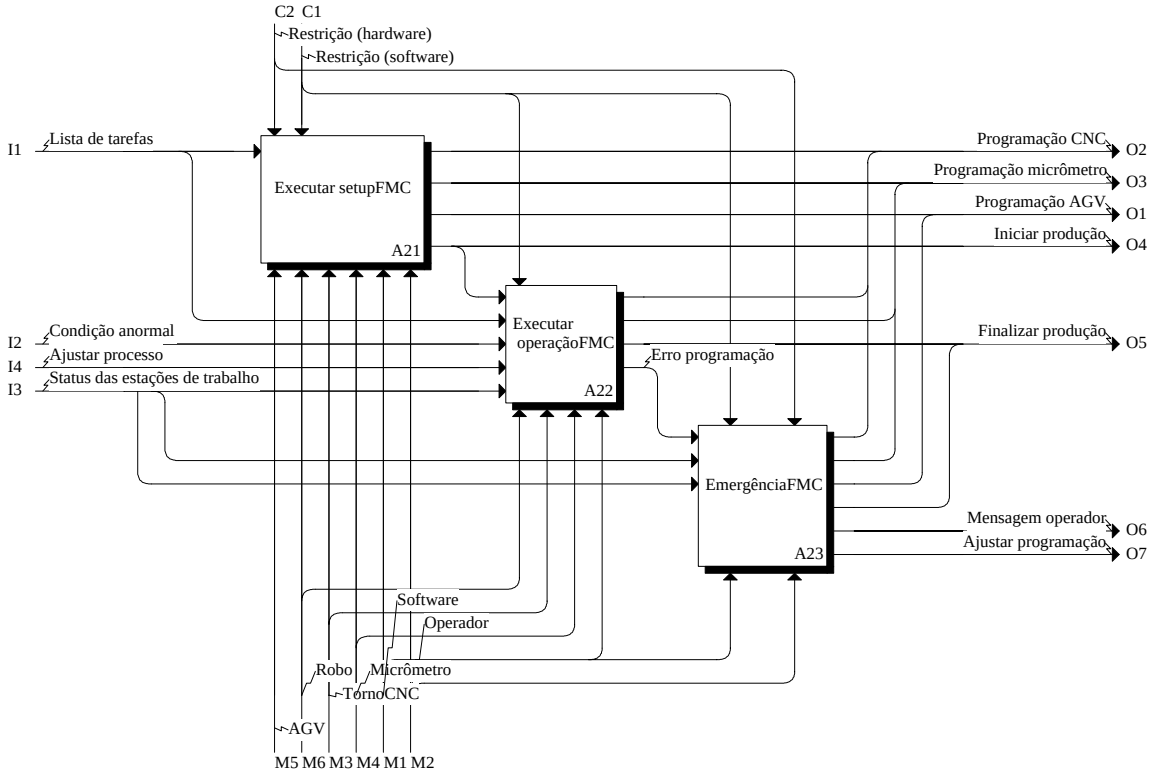

Figura 3.12 - Diagrama filho da função A2.

As três primeiras funções são executadas automaticamente pela MgU através do envio do programa NC para o Centro de Torneamento, do programa de inspeção para o micrômetro e da lista de ferramentas para o AGV. Por outro lado, a função ExecutarSetupRobo (A214) é realizada pelo operador já que o manipulador não possui um sistema de programação *off-line*. Por este motivo esta função recebe como entrada uma seta denominada Comandos do pendante com a extremidade tunelada.

Definida a programação das estações de trabalho, a função de *try-out* é executada. O *try-out* consiste inicialmente em operar o Centro de Torneamento em modo teste, sem a movimentação dos eixos, para a simulação do programa no próprio CNC (Álvares, 2005). Posteriormente, executa-se o programa NC, bloco a bloco, em modo *dry-run*. O *try-out* finaliza com a usinagem da peça a partir da execução de um ciclo de fabricação em que todas as estações de trabalho estejam operando.

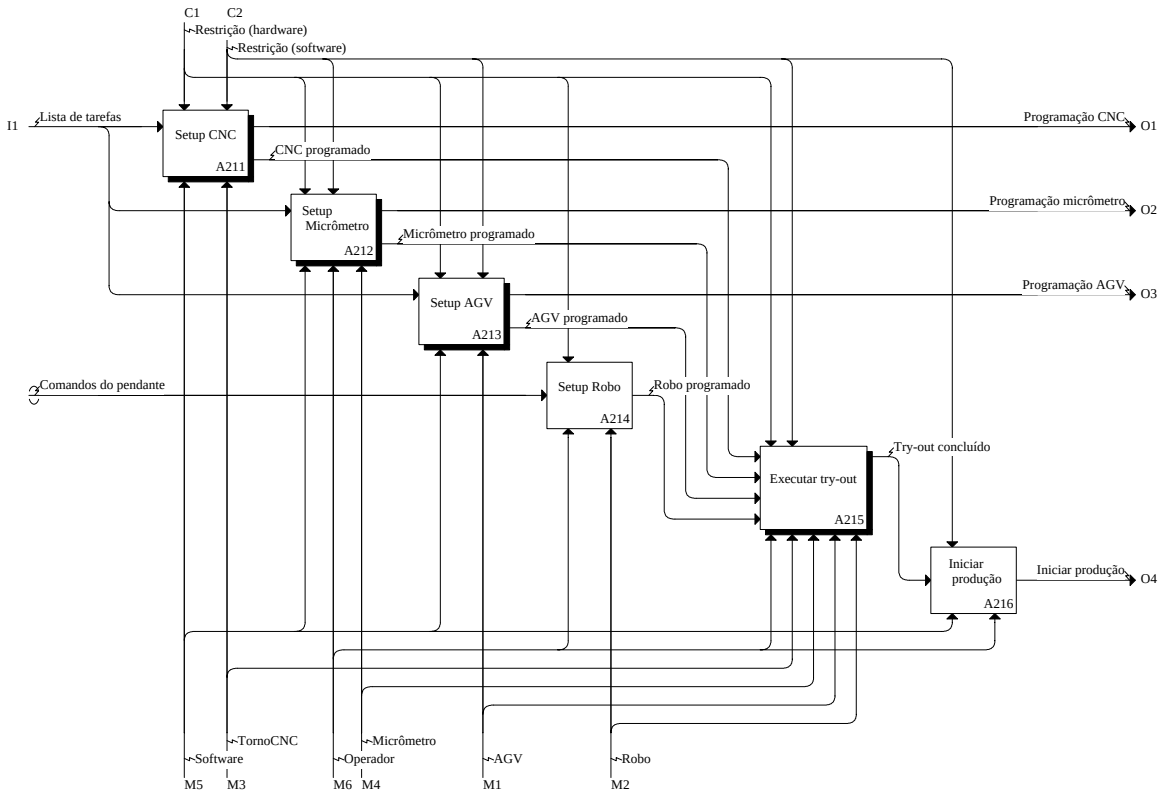

Figura 3.13 - Diagrama filho da função A21.

Iniciada a produção, a função ExecutarOperaçãoFMC (A22) é executada. Esta função é responsável pelo controle em tempo real das atividades e subdividida em: Controlador a eventos discretos (A221) e Controlador de operações (A222) (figura 3.14). O controlador a eventos discretos tem por finalidade certificar que a evolução dos eventos no chão-de-fábrica está de acordo com a programação estabelecida, identificando a transição disparada (A2211), o estado anterior (A2212), o estado posterior (A2213), comparando o estado posterior com o obtido (A2214) e executando a ação de controle (A2214).

O controle de operações (A222) tem como responsabilidade efetuar a re-programação das estações de trabalho para que um novo produto (da família de peças) possa ser produzido, ou até mesmo solicitar um novo *setup* quando uma nova família de peças será produzida. Esta função é subdividida em: Carregar próxima tarefa (A2221), Despachar programa NC (A2222) e Despachar programa de inspeção (A2223). A função carregar próxima tarefa verifica, dentro da fila de produtos a serem produzidos, qual é o de maior prioridade.

Verificado qual dentre os produtos é o de maior prioridade, o programa NC é carregado na função Despachar programa NC (A2222), bem como o programa de inspeção na função

Despachar programa de inspeção (A2223). Caso seja verificado que o tarugo do próximo produto a ser produzido pertença à outra família, a função SetupFMC (A21) é notificada. Caso as dimensões não correspondam a de um produto de outra família, um sinal de erro de programação é enviado a função EmergênciaFMC(A23).

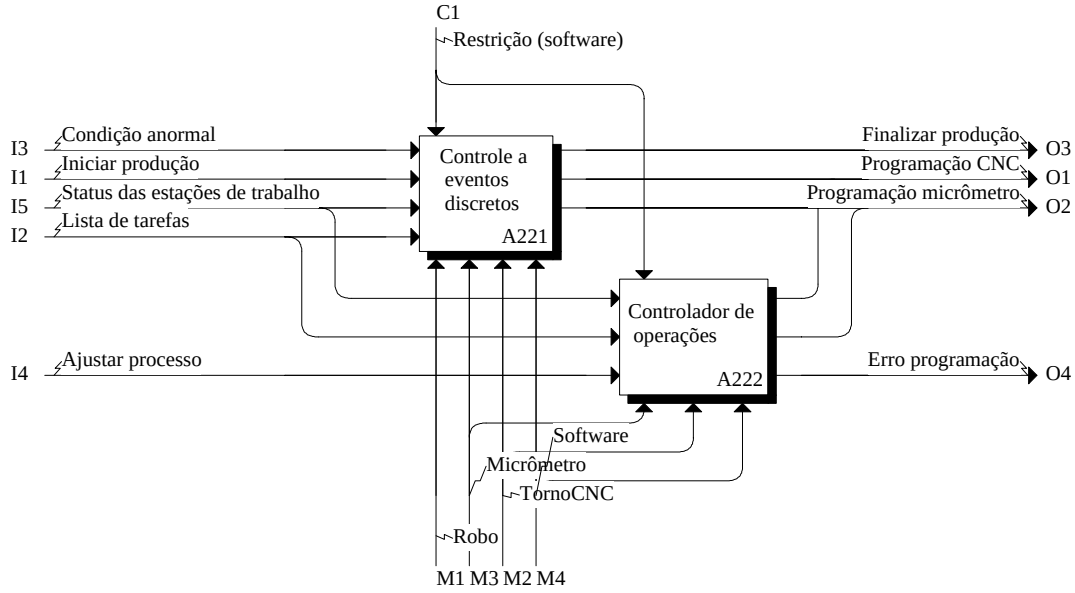

Figura 3.14 - Diagrama filho da função A22.

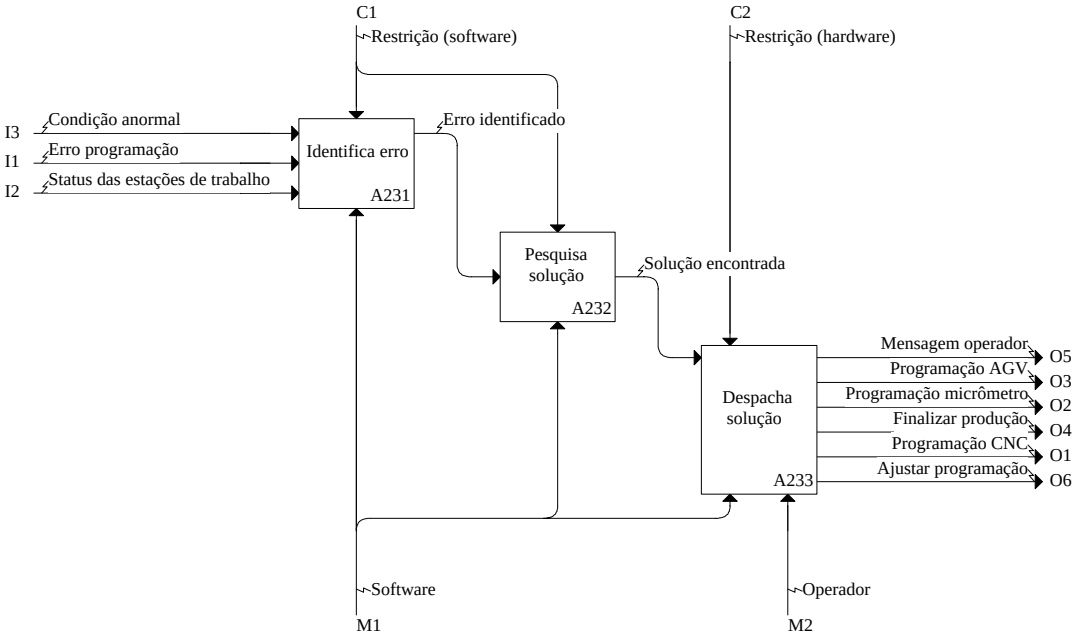

Figura 3.15 - Diagrama filho da função A23.

A função EmergênciaFMC (A23) é responsável pelo tratamento das condições anormais de funcionamento da célula (figura 3.15). As condições de erro que podem ser tratadas são previamente conhecidas e registradas na base de dados. Desse modo, esta função Identifica o erro (A231), Pesquisa a solução (A232) e Despacha a solução (A233) para os controladores das estações de trabalho notificando ao escalonador as eventuais alterações. Na impossibilidade de tratar a condição anormal identificada, uma mensagem é enviada ao operador informando a necessidade de intervenção humana.

3.3.1.3 – Monitor

Monitor executa o monitoramento do progresso das ordens em cada estação de trabalho. Isso inclui o monitoramento das atividades (monitor de eventos, monitor virtual e monitor em tempo real) e a análise da eficiência do sistema de produção (controle de qualidade e emissão de relatórios) (Teixeira *et at*, 2006).

A função *Monitor* é subdividida em outras três funções: Monitorar eventos (A31), Controle da qualidade (A32) e Apresentação dos resultados (A33) (figura 3.16). Monitorar eventos (A311) é a função responsável pela varredura periódica nos controladores das estações de trabalho com o intuito de verificar seus estados. O resultado é enviado à função Atualizar Estados (A312) que atualiza os estados dos controladores lógicos das estações de trabalho da MgU (figura 3.17).

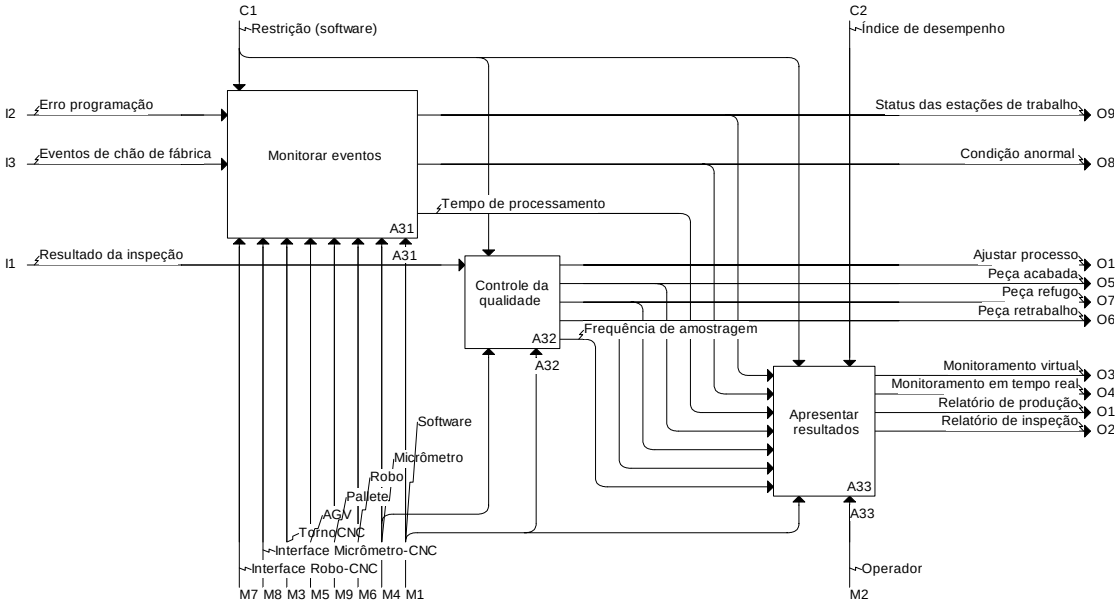

Figura 3.16 - Diagrama filho da função A3.

A função Controle da qualidade (A32) é subdividida em: Qualificar o processo (A321), Operação (A322) e Calcular freqüência de amostragem (A323) (figura 3.18). A função qualificar o processo tem por objetivo verificar a capacidade do processo em atender as especificações do projetista. Desse modo, as inspeções são monitoradas até que cinco amostras consecutivas estejam na zona central. Verificada a presença de duas peças consecutivas na zona de alerta ou uma na zona vermelha, uma mensagem (ajustar processo) é enviada a função *Dispatcher* (A2).

Calcular a freqüência de amostragem (A323) é a função utilizada para estimar a freqüência de amostragem em que as peças deverão ser amostradas. A freqüência de amostragem é calculada a dividindo-se por seis o período decorrido entre dois ajustes consecutivos. Como não existe a possibilidade de efetuar a programação *off-line* do manipulador, definida a programação, todas as peças podem ou não ser inspecionadas. Contudo, após o cálculo da freqüência de amostragem, parte das leituras obtidas deverá ser ignorada pela MgU.

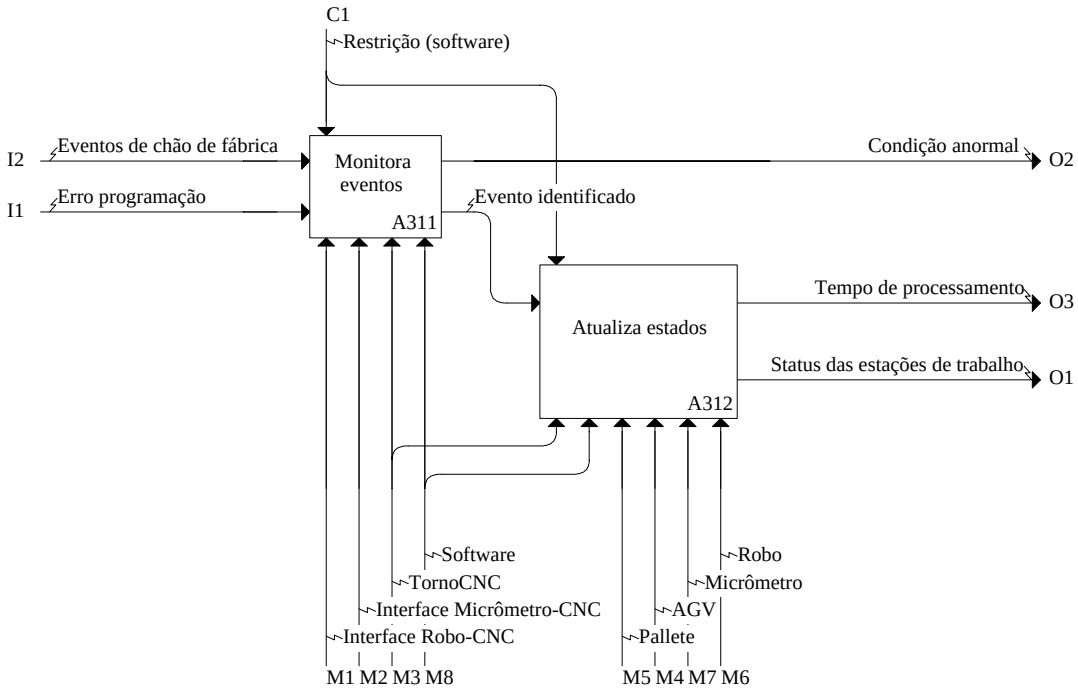

Figura 3.17 - Diagrama filho da função A31.

A função Apresentar resultados (A33) (figura 3.16) é subdivida em: Supervisão virtual (A331), Supervisão em tempo real (A332) e Relatório de atividades (A333) (figura 3.19). A supervisão

virtual é a função que encapsula o monitoramento virtual das estações de trabalho operando. A Supervisão em tempo real (A332) apresenta imagens de quatro câmeras estrategicamente instaladas na célula, bem como o áudio em tempo real das estações de trabalho operando.

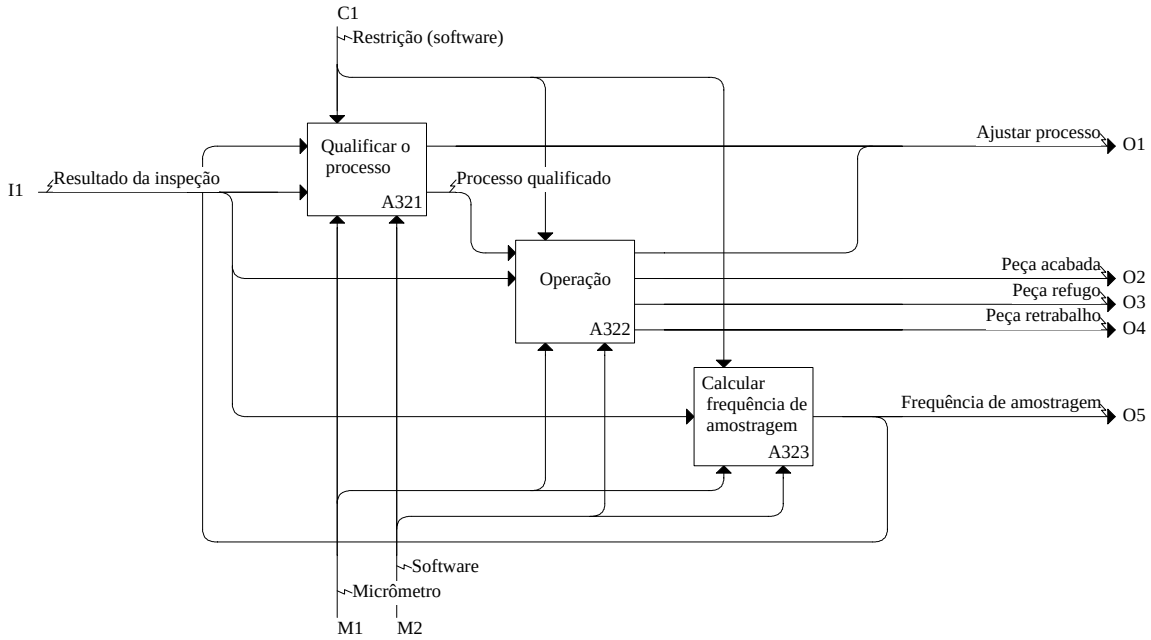

Figura 3.18 - Diagrama filho da função A32.

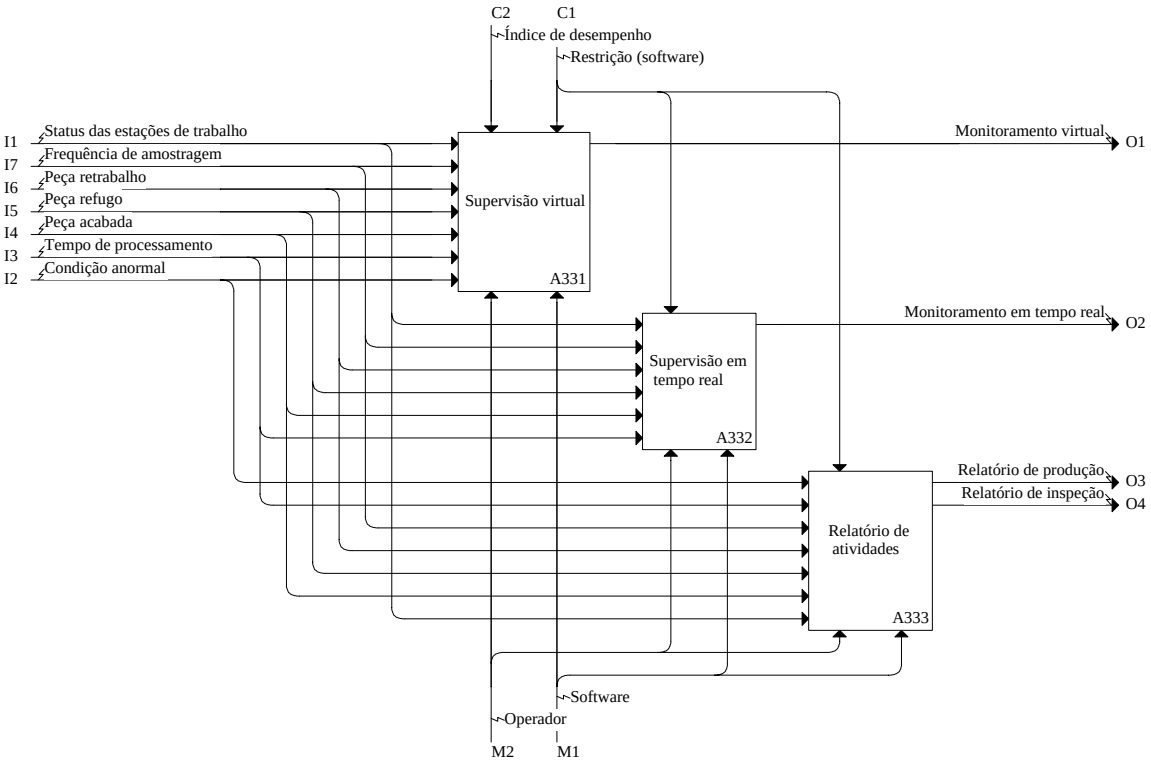

Figura 3.19 - Diagrama filho da função A33.

Relatório de atividades (A333) (figura 3.19) é a função que gerencia a emissão de relatórios de produção e de inspeção. Os relatórios reúnem as informações referentes à produção e à inspeção das peças (tempo de processamento, peças acabadas, peças refugo, etc.) como forma de avaliar a produtividade no chão-de-fábrica.

3.3.2 – Modelagem de dados (IDEF1x)

IDEF1x é o método utilizado para desenvolver o modelo de informação da Unidade de Gerenciamento. A partir do modelo da informação (representado pelo modelo entidade x relacionamento e pela normalização dos dados) é possível iniciar a construção do modelo físico ou banco de dados relacional (Oliveira, 2002). O modelo de informação da MgU é composto por um conjunto de tabelas conforme ilustra a figura 3.20.

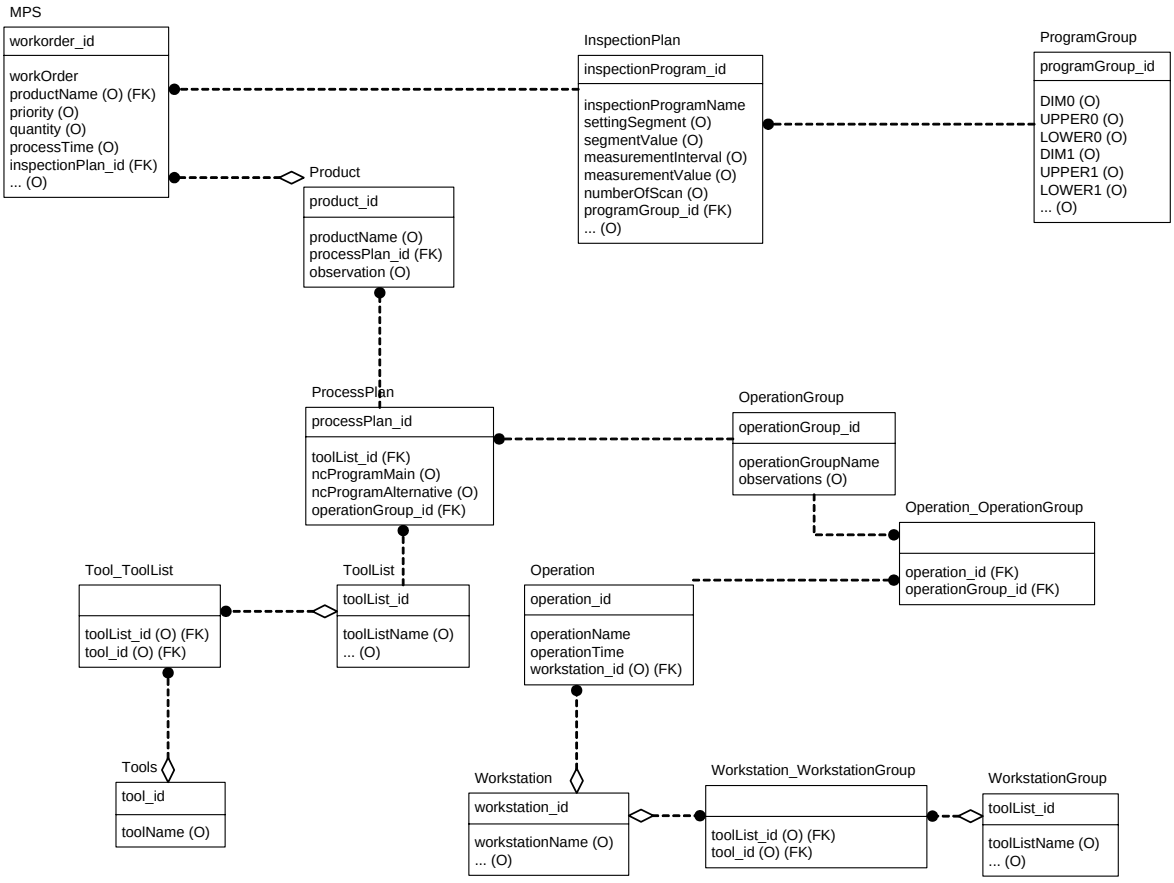

Figura 3.20 - Modelo da informação da Unidade de Gerenciamento.

Na tabela MPS estão registradas todas as ordens de trabalho (*WorkOrder*) cadastradas na base de dados WebFMC. Cada ordem possui um atributo *STATUS* que define a situação da mesma no sistema (*To produce* – para produzir, *In production* - em produção e *Produced* - produzida). Os

atributos *productName* e *inspectionProgram_id* são declarações de chaves estrangeira (*Foreign Key*). Isto é necessário para garantir que todas as ordens de trabalho cadastradas na base de dados estejam associadas a um produto e a um programa de inspeção.

Cada registro da tabela *Product* possui um atributo chave estrangeira (*Foreign Key*) *processPlan_id*. Consequentemente, cada registro de produto está associado a um plano de processo. Na tabela *ProcessPlan* são armazenados os planos de processos gerados pelo sistema CAD/CAPP/CAM (*Webmachining*). *OperationGroup_id*, *NCProgram_id* e *ToolList_id* são chaves estrangeiras que relacionam um registro da tabela *ProcessPlan* a um registro de *OperationGroup*, *NCProgram* e *ToolList*.

Na tabela *Operation* estão registradas as possíveis operações que as estações de trabalho podem executar. O atributo *operationTime* deve ser setado com o valor obtido a partir da execução do *try-out*. A tabela *OperationGroup* reúne o grupo de operações necessárias para a fabricação de um produto. A cardinalidade do relacionamento entre as tabelas *Operation* e *OperationGroup* é de muitos para muitos (m:n). Como este tipo de relacionamento não pode ser implementado na prática (Oliveira, 2002) uma tabela associativa distributiva (*Operation_OperationGroup*) relaciona das tabelas *Operation* e *OperationGroup*.

4 – WEBFMC: IMPLEMENTAÇÃO DA CÉLULA

Este capítulo descreve em detalhes a implementação da FMC. A implementação foi estrategicamente dividida em duas etapas: integração física e integração lógica. Na integração física é apresentada a montagem da célula, desde o projeto e construção do carro posicionador à conexão dos controladores das estações de trabalho.

Na fase de integração lógica é descrito em detalhes o projeto lógico de integração Robô-Torno-Micrômetro, desde a arquitetura CNC/PMC/Controlador do robô aos detalhes da programação da unidade de manipulação e transporte de materiais. Como último assunto é apresentado a estrutura hierárquica da célula descrevendo passo a passo o funcionamento integrado com os demais sistemas.

4.1 – INTEGRAÇÃO FÍSICA DA CÉLULA

Integrar fisicamente a célula consiste em disponibilizar as condições físicas necessárias para iniciar as atividades no chão de fábrica (Teixeira *et at*, 2006). Isto inclui desde o posicionamento das estações de trabalho à conexão dos respectivos controladores. Dentre as atividades necessárias para integrar fisicamente a célula estão:

- ✓ Projeto e construção de um armazenador de peças;
- ✓ Projeto e construção de um carro posicionador;
- ✓ Projeto e construção de um suporte para o micrômetro;
- ✓ Projeto e construção de dedos para a garra do robô;
- ✓ Conectar os controladores das estações de trabalho.

Das atividades anteriormente mencionadas, a conexão dos controladores das estações de trabalho merece um destaque especial. Os detalhes do projeto e construção do carro posicionador, dos dedos para a garra, do armazenador de peças e do suporte para o micrômetro estão descritos no apêndice B.

4.1.1 – Conexão dos controladores das estações de trabalho

O maior passo rumo à integração consiste em conectar as máquinas em uma rede de comunicação, já que ao estabelecer a comunicação entre as estações de trabalho e o sistema

externo é possível controlar as máquinas remotamente, bem como a troca de informações entre as estações de trabalho (Leitão *et at*, 1996). Este trabalho de integração tem sido um grande desafio para muitos pesquisadores, por causa da diversidade de máquinas e pela falta de uma padronização nos protocolos de comunicação (Teixeira *et at*, 2006).

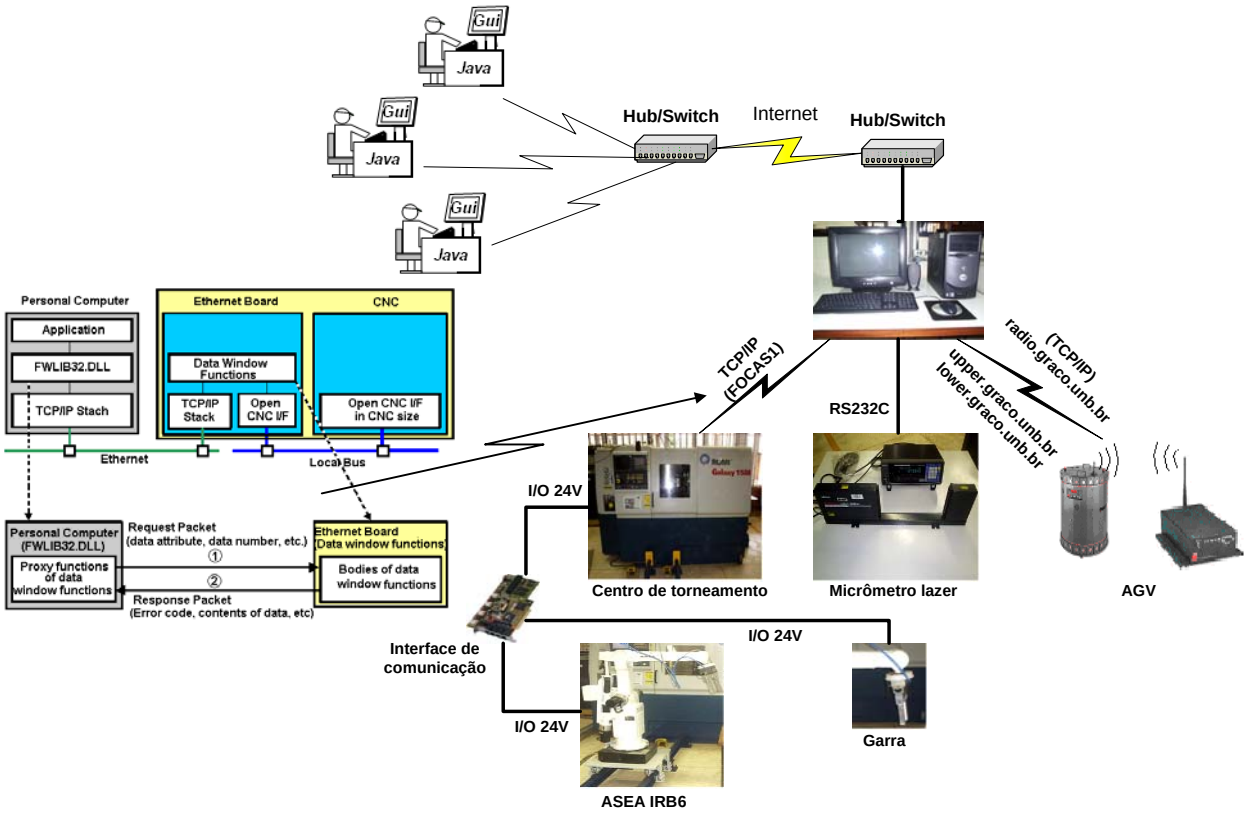

Figura 4.1 - Estrutura de comunicação da FMC.

A figura 4.1 apresenta a estrutura de comunicação da célula. A comunicação com o Centro de Torneamento Galaxy 15M (CNC Fanuc 18i-ta) é estabelecida por meio de uma interface *Ethernet*, utilizando o protocolo TCP/IP, associada às funções programadas do FOCAS1. O FOCAS1 é constituído por *drivers* e uma biblioteca de funções programáveis que promove a comunicação e o acesso programável para um sistema CNC baseado no PC (GEFanuc, 2006). Esta biblioteca dispõe cerca de 300 chamadas de funções que podem ser implementada por uma aplicação customizada.

Um sistema de Radio/*Ethernet* é utilizado para estabelecer a comunicação com o AGV. Este sistema possui uma interface *Proxin RangeLan2* responsável pela conexão com a rede e pela comunicação com um servidor *bridge*. Este servidor, por sua vez, conecta o robô a rede local utilizando o protocolo TCP/IP (Álvares *et at*, 2004). Contudo, é oferecido acesso aos principais

padrões e mecanismos de rede (ftp, *telnet*, TCP/IP, *sockets*) de tal forma que o robô possa ser operado como uma estação de trabalho.

A comunicação com a unidade de inspeção da célula é estabelecida mediante uma interface RS232C. Esta comunicação se restringe a um conjunto de 23 comandos programáveis definidas pela Mitutoyo. Estes comandos permitem desde a programação remota dos parâmetros geométricos da peça (diâmetro e tolerâncias), às condições em que a inspeção será realizada.

A comunicação com a unidade de manipulação e transporte está limitada a 13 I/O's digitais (7 entradas e 6 saídas). Esta restrição levou ao projeto, em parceria com o fabricante do Centro de Torneamento (Romi), de uma interface dedicada necessária para estabelecer uma comunicação indireta com o manipulador baseada na arquitetura CNC/PMC/Controlador do robô.

4.1.2 – Integração Física ROBO-TORNO-MICRÔMETRO.

A necessidade de conectar os controladores das estações de trabalho conduziu ao projeto e construção de duas interfaces dedicadas: a interface robótica e a do sistema de medição. A figura 4.2 apresenta as interfaces robóticas e de medição desenvolvidas pelo fabricante do Centro de Torneamento.

Figura 4.2 - Interface robótica e do sistema de medição.

A interface robótica foi projetada para suportar a integração do PMC (PLC) do Centro de Torneamento ao controlador do manipulador para que mensagens possam ser trocadas entre estas estações de trabalho. A interface do sistema de medição permite a conexão do sistema de

medição ao PMC do Centro de Torneamento, conexão necessária para informar ao manipulador o resultado da inspeção (peça acabada, peça re-trabalho e peça refugo). No apêndice C é apresentado o diagrama elétrico das conexões com a interface robótica, com a interface do sistema de medição e com a linha de emergência.

Conforme especificado nos diagramas elétricos da interface robótica e do sistema de medição[1], as linhas CN1-A1, CN1-A2, CN1-A3, CN1-A4, CN1-A5, CN1-A6, CN1-A7, CN1-A8, CN1-A22, CN1-A23, CN1-A24 devem ser conectadas através de relés para isolar eletricamente os controladores das estações de trabalho. Desse modo, 11 relés industriais (modelo T2RC3-24v, corrente nominal: 56mA) foram instalados para que estas linhas pudessem ser conectadas. A figura 4.3 mostra os relés industriais utilizados na conexão das interfaces robótica e do sistema de medição.

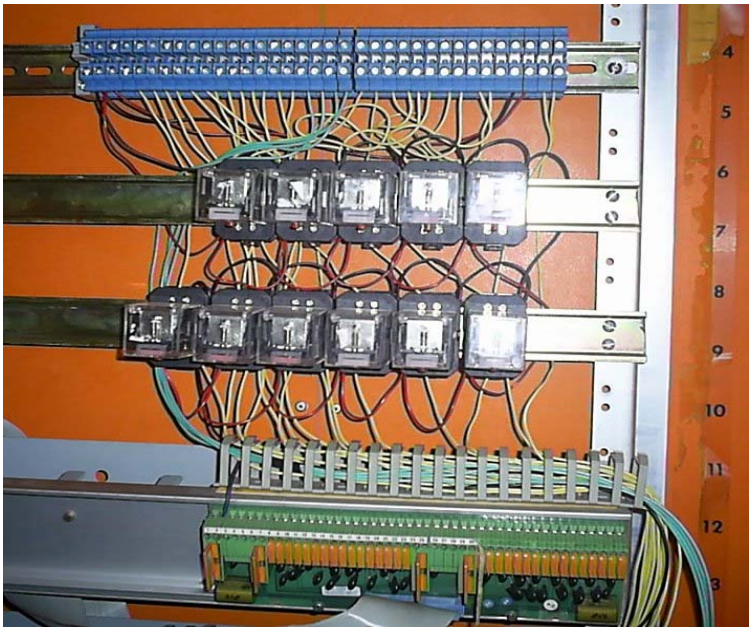

Figura 4.3 - Relés industriais utilizados na integração Robô-Torno-Micrômetro.

4.1.3 – Interface de comunicação

Para conectar os controladores das estações de trabalho, monitorar as trocas de mensagens e obter os sinais de posicionamento da garra foi desenvolvida uma interface de comunicação. Esta interface conecta o controlador do robô à interface dedicada Robô-PMC, como também coleta os sinais do sensor da garra e do controlador do robô para os acionamentos da garra.

[1] Os diagramas estão disponíveis no apêndice C.

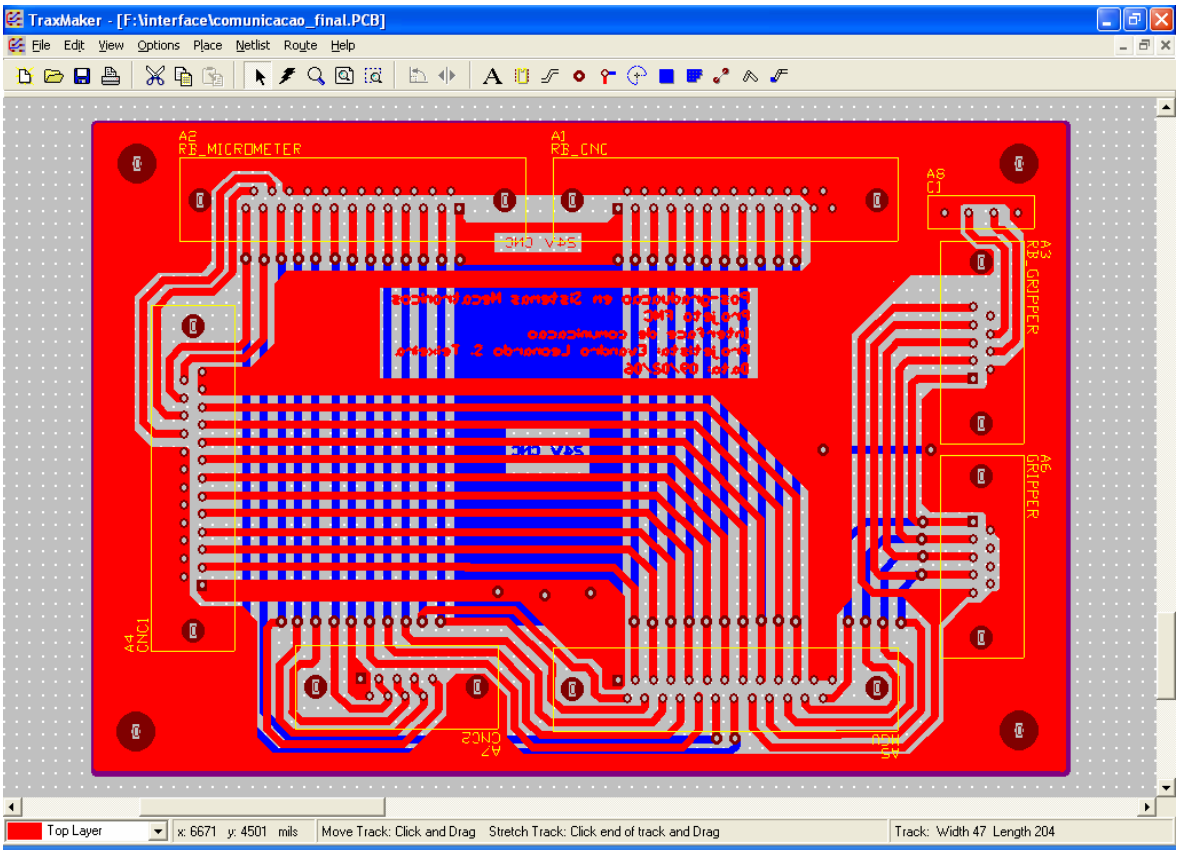

Figura 4.4 - Projeto da interface de comunicação – PCB.

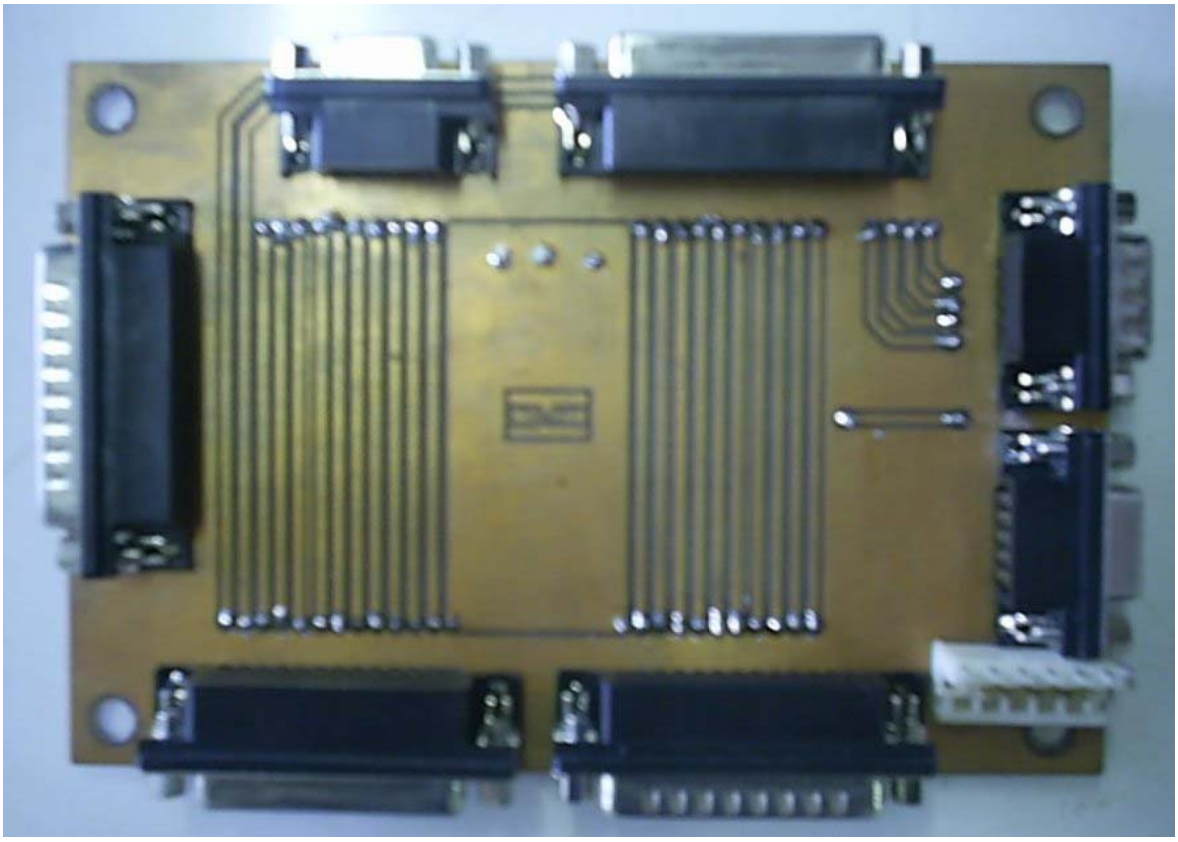

Figura 4.5 - Interface de comunicação.

As figuras 4.4 e 4.5 mostram respectivamente o projeto e a construção da interface de comunicação. Conforme apresenta a figura 4.4, os sinais das estações de trabalho, bem como das interfaces dedicadas foram divididos em blocos. Isto adiciona mais flexibilidade nas conexões e permite que as estações de trabalho sejam eventualmente utilizadas individualmente em outros projetos. Para projetar a placa de circuito impresso, utilizou-se o aplicativo TraxMaker da ferramenta CircuitMaker. Esta ferramenta fornece suporte ao projeto de circuitos eletrônicos, gerando *netlist* para o TraxMaker ou outras ferramentas PCB (CircuitMaker, 2000).

4.2 – INTEGRAÇÃO LÓGICA DA CÉLULA

A integração lógica consiste em fornecer as condições lógicas necessárias para iniciar as atividades no chão de fábrica. Neste contexto é apresentada a integração lógica Robô – Torno – Micrômetro, a arquitetura CNC/PMC/Controlador do robô e os detalhes da programação do Centro de Torneamento e da programação na unidade de manipulação e transporte de materiais.

4.2.1 Integração lógica ROBO-TORNO-MICRÔMETRO

A segunda fase do projeto de integração desenvolvido em parceria com o fabricante do Centro de Torneamento resultou na definição da arquitetura CNC/PMC/Controlador do robô. Esta arquitetura além de permitir a sincronização dos eventos do chão-de-fábrica, permite a comunicação indireta com o manipulador robótico utilizando o PMC do Centro de Torneamento. A figura 4.6 ilustra a arquitetura CNC/PMC/Controlador do robô.

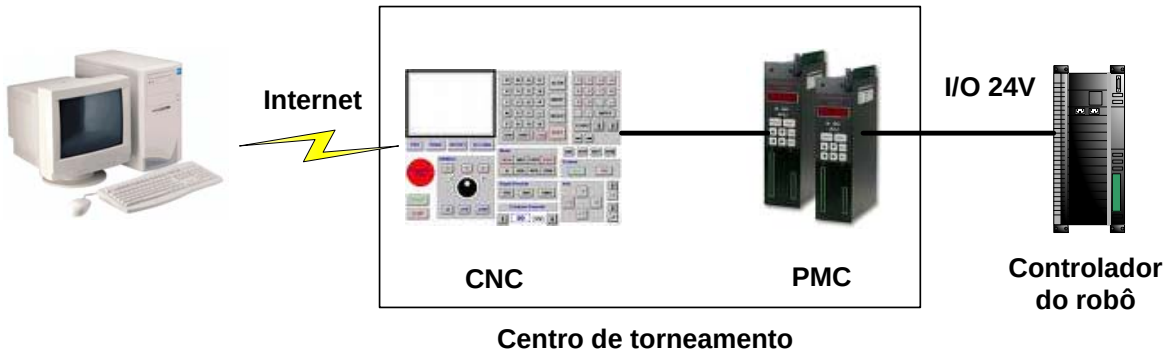

Figura 4.6 - Arquitetura CNC/PMC/Controlador do robô.

Nesta arquitetura, foram declaradas quatro variáveis de PMC as quais o controlador do robô tem acesso direto: Y104.3, Y104.2, Y104.1 e Y104.0. Combinando-se estas variáveis via código

BCD, um total de até quinze instruções[2] podem ser enviadas ao controlador do robô. Apesar do protocolo FOCAS1 não fornecer acesso direto ao PMC do Centro de Torneamento, através do CNC é possível realizar operações de leitura/escrita nas variáveis de PMC. Desse modo, para enviar um comando ao controlador do robô, a MgU escreve nas variáveis de PMC, o código BCD referente ao comando, via FOCAS1.

4.2.2 – Programação do Centro de Torneamento

Além de instalar interfaces dedicadas e alterar o *ladder* do PMC do Centro de Torneamento, a Romi desenvolveu um conjunto de funções miscelâneas para que os eventos entre as estações de trabalho, utilizando como controlador mestre o PMC do Centro de Torneamento, possam ser programados:

- ✓ M200 – Ciclo de carregamento;
- ✓ M201 – Ciclo de descarregamento;
- ✓ M202 – Zeramento da peça;
- ✓ M203 – Repasse de peça.

A função miscelânea M200 é utilizada para programar o ciclo de carregamento da célula. Desse modo, quando um bloco de um programa NC contém esta instrução, o Centro de Torneamento inicia o ciclo de carregamento verificando se as condições de início de ciclo são válidas e enviando um sinal ao manipulador notificando o início do ciclo.

A função M201 é utilizada para declarar o ciclo de descarregamento. Neste caso, uma série de eventos e, por conseguinte, troca de mensagens entre as estações de trabalho deve ocorrer para que a unidade de manipulação e transporte possa retirar a peça do Centro de Torneamento. A figura 4.7 apresenta os detalhes da programação do torno.

As funções M202 e M203 servem para declarar subprogramas utilizados no zeramento da peça e no repasse da peça respectivamente. Desse modo, todas as vezes que uma destas funções miscelâneas forem identificadas em um bloco de instrução do programa NC, o CNC lê as variáveis de macro #510 e #511 o nome da sub-rotina de zeramento e repasse de peça

[2] Apesar da combinação das variáveis fornecerem até 16 combinações, por questões de segurança, a combinação em que todas as variáveis assumem valor nulo não é utilizada.

respectivamente. A figura 4.8 apresenta as variáveis utilizadas na integração com o manipulador robótico (a esquerda) e os subprogramas registrados nas variáveis de macro (a direita).

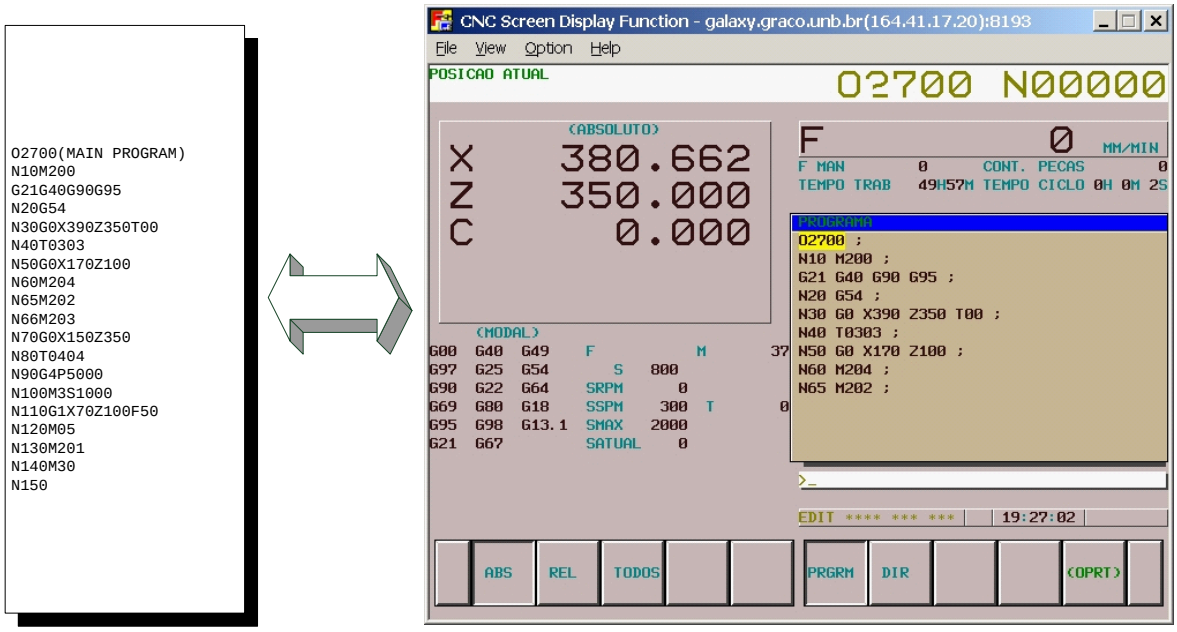

Figura 4.7 - Detalhes da programação do Centro de Torneamento – parte 1.

Ao final de cada sub-rotina a função M99 deve ser programada para que a execução do programa retorne para o programa principal. É importante lembrar que somente serão executadas estas sub-rotinas se uma mensagem de peça re-trabalho for enviada pela MgU. O fluxograma detalhado da programação do Centro de Torneamento é apresentado no apêndice C (figuras C.4 e C.5).

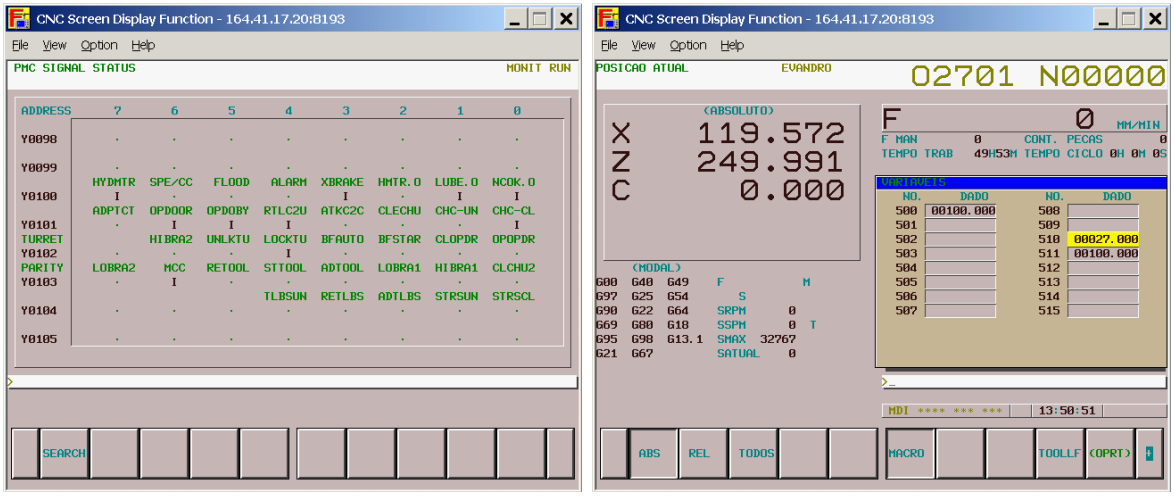

Figura 4.8 - Detalhes da programação do Centro de Torneamento – parte 2.

4.2.3 – Programação da unidade de manipulação e transporte

A programação do manipulador robótico é baseada no protocolo estabelecido pela arquitetura CNC/PMC/Controlador do robô. As restrições de comunicação impostas pelo manipulador (13 I/O's - 7 Entradas e 6 Saídas) levaram a adoção da lógica BCD como solução para expandir o número de possíveis combinações das entradas do robô. A figura 4.9 apresenta os detalhes da programação do manipulador robótico.

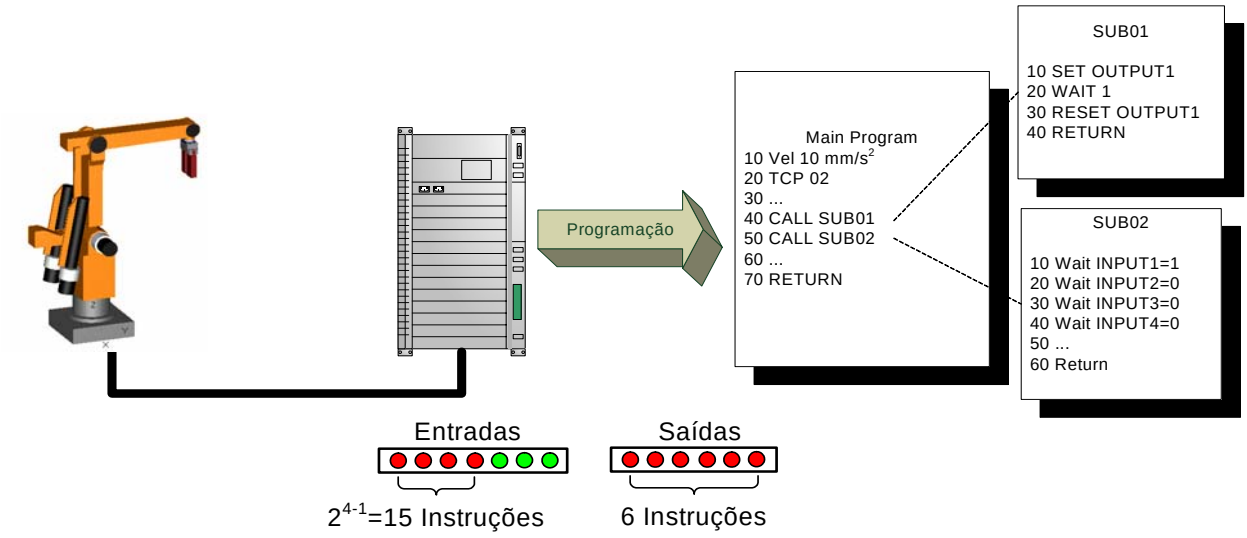

Figura 4.9 - Detalhes da programação do manipulador robótico.

Ao receber uma mensagem do PMC, uma sub-rotina deve verificar sequencialmente as *Inputs* utilizadas na integração (OS1, OS2, OS3 e OS4) e avaliar uma a uma de acordo com as estruturas *IF's* aninhadas. Esta avaliação consiste na leitura e comparação seqüencial com os valores definidos na tabela 4.1 para identificar a instrução enviada pelo PMC do Centro de Torneamento.

Tabela 4.1 - Lista de I/O's – Interface robótica.

Lista de I/O'S - Interface robótica											
Inputs	**OS4**	**OS3**	**OS2**	**OS1**	**Outputs**	**IS5**	**IS4**	**IS3**	**IS2**	**IS1**	**IS0**
Placa aberta	0	0	0	1	Ciclo de troca em andamento	0	0	0	0	0	1
Placa Fechada	0	0	1	0	Ciclo de troca concluído	0	0	0	0	1	0
Porta Aberta	0	0	1	1	Abre placa	0	0	0	1	0	0
Porta Fechada	0	1	0	0	Fecha Placa	0	0	1	0	0	0
Contra Ponto Avançado	0	1	0	1	Avança contra-ponta	0	1	0	0	0	0
Contra Ponto Recuado	0	1	1	0	Recua contra-ponta	1	0	0	0	0	0
Troca Proibida	0	1	1	1							
Carga solicitada	1	0	0	0							
Descarga solicitada	1	0	0	1							
Peça Boa	1	0	1	0							
Peça Retrabalho	1	0	1	1							
Peça Ruim	1	1	0	0							
Erro de Programação	1	1	0	1							

Identificado o comando recebido, a sub-rotina deve setar um registro e retornar o controle ao programa principal. Ao retomar a execução do programa, o programa principal verifica o valor do registro e decide a partir deste valor qual a ação de controle deverá ser executada. A tabela 4.1 descreve em detalhes as instruções enviadas e recebidas pelo controlador do robô.

4.3 – ESTRUTURA HIERÁRQUICA DA CÉLULA

A filosofia CIM estabelece que todas as operações da empresa relacionadas à produção são incorporadas em um sistema computacional integrado, em que a saída de uma atividade serve como entrada para a próxima atividade, através de uma cadeia de eventos que inicia com o pedido do cliente e termina com a entrega do produto (Groover, 2003). A FMC possui um ambiente integrado de informação (banco de dados comum) baseado no conceito da filosofia CIM e da metodologia *e-manufacturing* (Teixeira, Cano e Álvares, 2006).

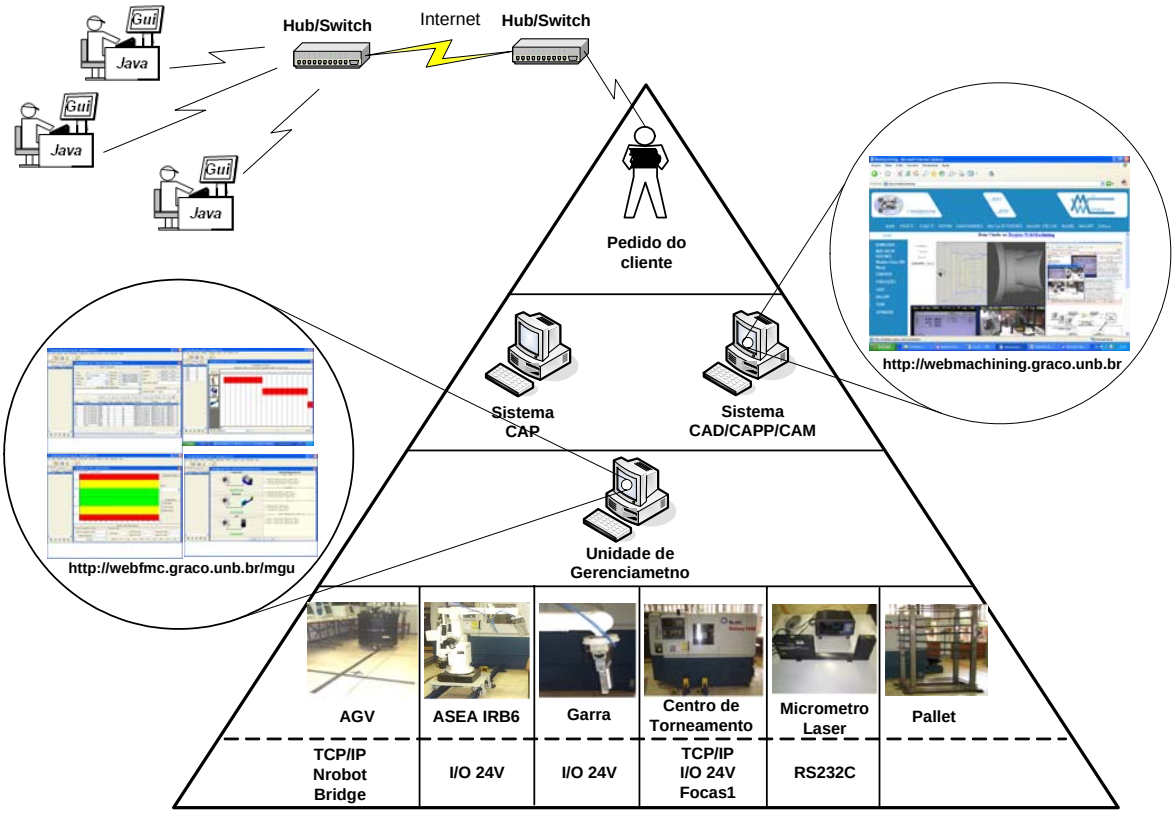

Figura 4.10 - Estrutura hierárquica da célula.

A figura 4.10 ilustra a estrutura hierárquica da célula. As atividades iniciam com o registro do pedido no cliente através do site: http://webfmc.graco.unb.br. Em cada pedido o cliente deve fornecer as informações do produto, a quantidade, a possível data de entrega, dentre outras que

se façam necessárias. Estas informações são posteriormente compartilhadas com os demais níveis hierárquicos inferiores através de um banco de dados comum. A figura 4.11 apresenta o *website* do projeto que serve como *front-end* para a utilização da FMC.

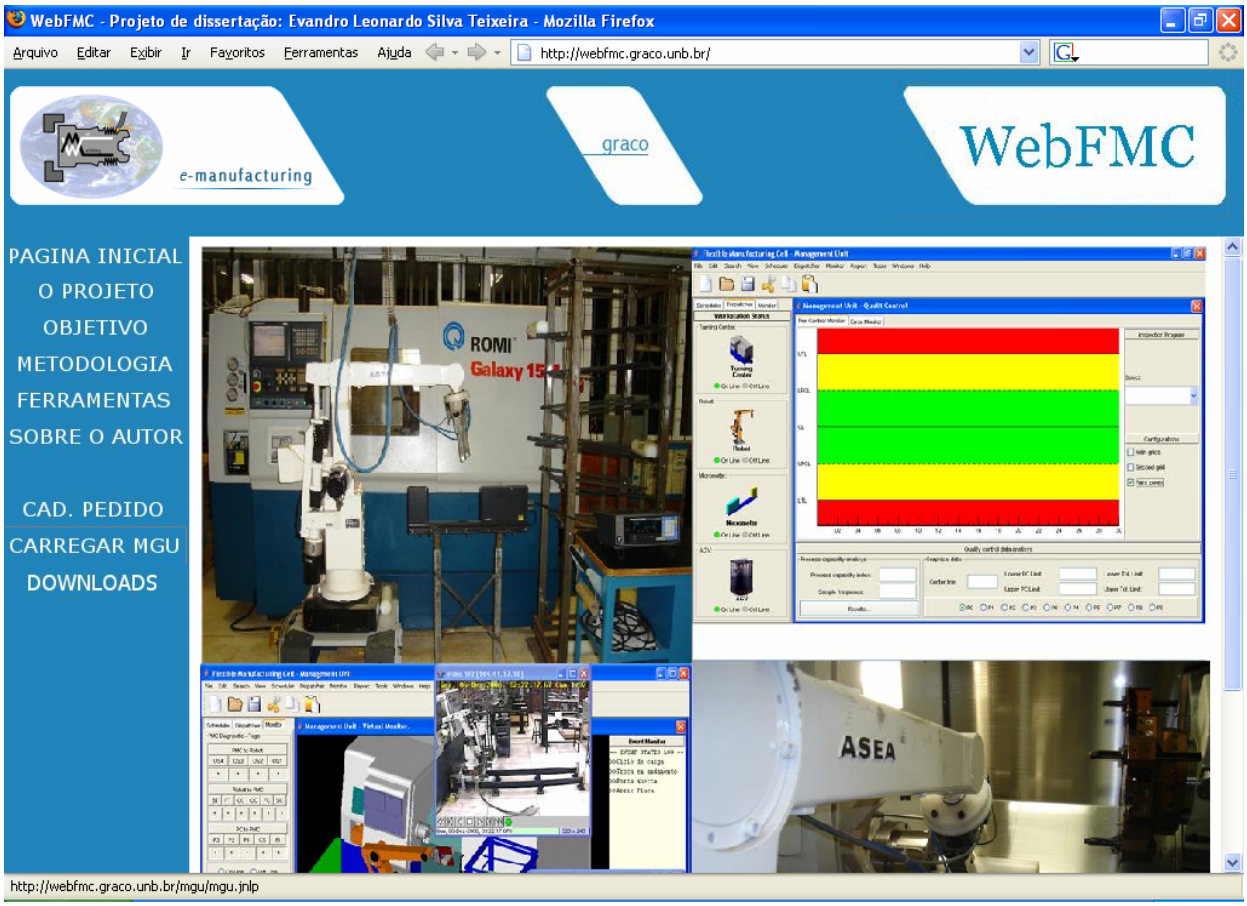

Figura 4.11 – Página *web* do projetoWebFMC.

A engenharia do produto, através de um sistema baseado na metodologia *Webmachining* (sistema CAD/CAPP/CAM – http://webmachining.alvarestech.com) inicia o projeto do produto e finaliza com o plano de processo (PP) (Álvares *et at*, 2005). O projeto do produto consiste na definição das formas geométricas, bem como na modelagem utilizando a ferramenta *WebCadByFeature*. Finalizado o projeto do produto, o processista utiliza a ferramenta *WebCAPP* para obter o plano de processo que será utilizado na fabricação do produto.

A engenharia de produção, através de um sistema CAP, converte o pedido do cliente em ordens de produção e as adiciona ao plano mestre da produção. Nesta etapa são avaliados os aspectos produtivos (capacidade de máquinas/recursos) e demais questões relacionadas ao planejamento da produção no nível tático. O plano mestre da produção, gerado pelo sistema CAP e o plano de

processo gerado pelo sistema CAD/CAPP/CAM são repassados à MgU para efetuar a programação das estações de trabalho.

A Unidade de Gerenciamento (executada através do *website* http://webfmc.graco.unb.br/mgu/mgu.jnlp) recebe o plano mestre da produção elaborado e efetua a programação da produção. Na programação da produção, as ordens de produção são selecionadas e efetivamente convertidas em ordens de trabalho e agrupadas em famílias de peças. Posteriormente, o escalonamento da produção converte as ordens de trabalho em lista de tarefas.

Estabelecida a lista de tarefas a serem enviadas para o chão-de-fábrica, a MgU efetua o carregamento de instruções nas estações de trabalho. Confirmado o carregamento e execução do *try-out* das estações de trabalho, a produção é iniciada. Funções de monitoramento virtual, monitoramento em tempo real e controle da qualidade fornecem ao operador a possibilidade de supervisionar remotamente as atividades no chão-de-fábrica.

5 – WEBFMC: CICLO DE VIDA DA UNIDADE DE GERENCIAMENTO

Este capítulo descreve o ciclo de vida da MgU. O ciclo de vida é baseado na abordagem metodológica do Processo Unificado da Rational para Sistemas de Engenharia (RUP SE) em que se divide o desenvolvimento da arquitetura do sistema em duas dimensões: ponto de vista (empresa, computacional, engenharia, informação e processo) e níveis do modelo (contexto, análises, projeto e implementação) (Rational, 2006). Utilizam-se a linguagem de modelagem unificada (UML) e a ferramenta Rational Rose 2003 para a modelagem visual e desenvolvimento dos diagramas respectivamente.

5.1 – CONTEXTO: DIAGRAMA DE CASOS DE USO

Os requisitos funcionais da Unidade de Gerenciamento foram documentados através de diagramas de casos de uso. Os casos de uso são utilizados para modelar as interações entre os componentes especificando a ação ou a seqüência de ações que o mesmo deve desempenhar na ocorrência de certos eventos (Anglani *et at*, 2002).

As funcionalidades que a MgU deve oferecer foram divididas em dois diagramas de casos de uso. No primeiro (figura 5.1) são agrupadas sequencialmente, em 11 casos de usos, as funcionalidades oferecidas ao operador. Os casos de uso Programar a Inspeção, Programar a produção e Escalonar a produção encapsulam as funções do subsistema *Scheduler*.

Verificar o *status* das estações de trabalho, Executar *setup* FMC, Iniciar a produção, Emergência e Finalizar a produção são funções do subsistema *Dispatcher*. As duas primeiras são funções preparatórias que devem ser executadas antes mesmo da produção ser iniciada. Após o início da produção, o operador pode interagir com este subsistema recebendo uma notificação da necessidade de intervenção humana (emergência) ou finalizando a produção.

Monitorar a produção e Relatórios são os casos de uso que representam as funcionalidades disponibilizadas pelo subsistema *Monitor*. Monitorar a produção inclui o monitoramento virtual das estações de trabalho, o monitoramento em tempo real e o controle da qualidade. Relatórios englobam o levantamento dos dados de produção e inspeção que servirão como embasamento técnico para avaliar os índices de desempenhos estabelecidos.

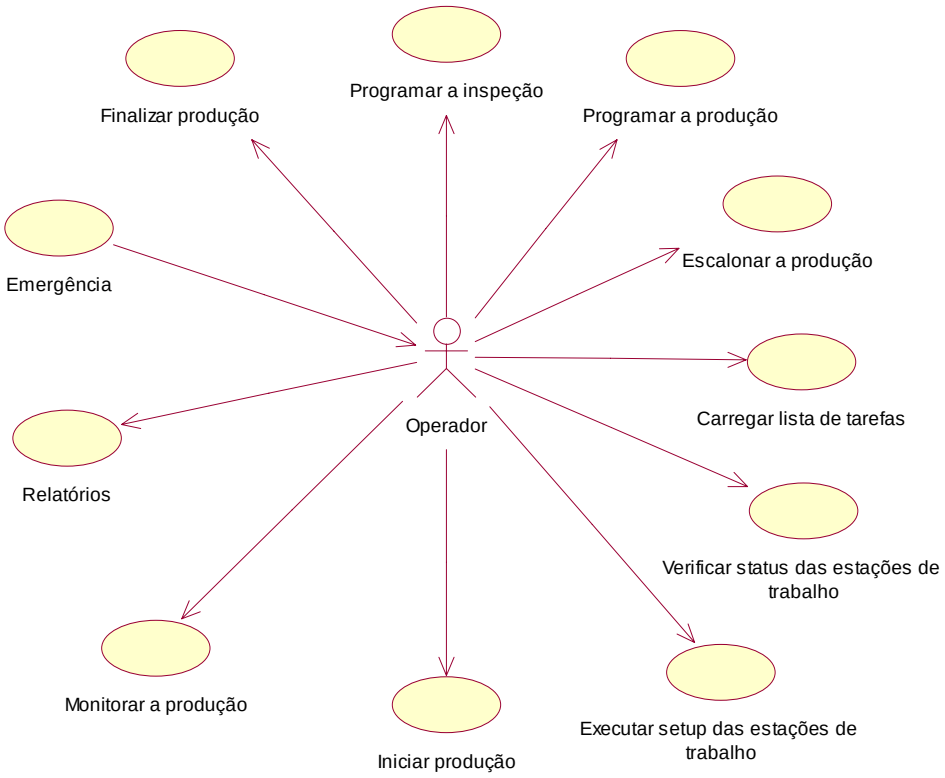

Figura 5.1 - Diagrama de casos de uso – Operador.

O segundo diagrama (figura 5.2) mostra a interação entre a MgU e o controlador das estações de trabalho (ator). Neste diagrama as funcionalidades foram agrupadas em 9 casos de uso. Para executar o *setup* das estações de trabalho, a Unidade de Gerenciamento deve ser capaz de programar o Centro de Torneamento (Programação CNC), programar o micrômetro (Programação micrômetro) e programar o AGV (Programação AGV). Iniciar a produção é o caso de uso que encapsula o envio do sinal de *CICLO START* ao Centro de Torneamento.

Inspeção é o caso de uso que encapsula a leitura efetuada pelo micrômetro que necessita ser processada pela MgU. Neste caso, o sentido do relacionamento é do ator para o caso de uso. O resultado da inspeção é enviado ao controlador das estações de trabalho para que o manipulador possa depositar a peça em seu devido lugar (armazém de peças acabadas, de peças re-trabalho ou de peças refugo). Eventos do chão-de-fábrica e emergência são casos de uso utilizados para atualizar o *status* das estações de trabalho e notificar o estado de emergência ao subsistema *Dispatcher* respectivamente.

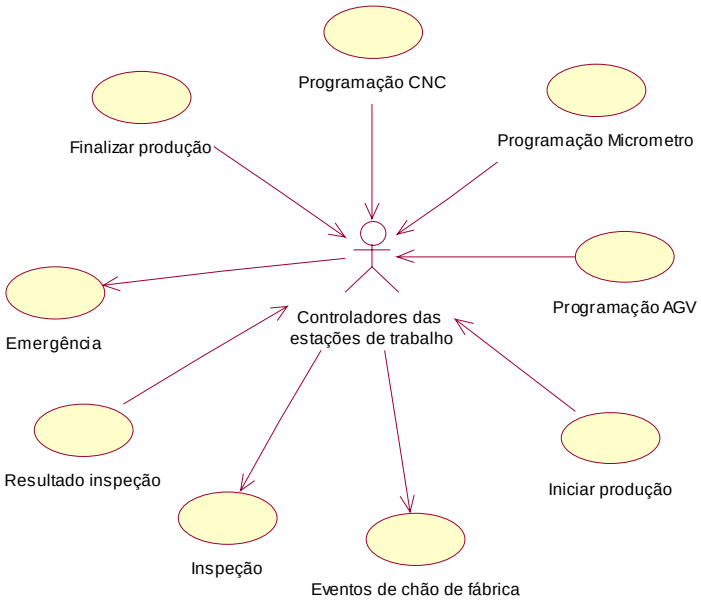

Figura 5.2 - Diagrama de casos de uso – controladores das estações de trabalho.

5.2 – ANÁLISE: DIAGRAMA DE ESTADOS

Diagramas de estados podem ser utilizados para documentar o nível de análise da arquitetura. Neste nível, não há uma preocupação em especificar os recursos a serem utilizados; Ao invés disso, a análise deve refletir aproximações do projeto para dividir o que o sistema necessita fazer e como o esforço computacional deve ser distribuído (Rational, 2006).

A figura 5.3 mostra o diagrama de atividades[1] do subsistema *Scheduler*. Este subsistema é composto por dois módulos: programação da inspeção e programação da produção. O primeiro módulo é responsável pelo cadastramento dos programas de inspeção. Após o cadastramento dos programas de inspeção, o módulo de programação da produção gerencia o cadastramento e a seleção das ordens de trabalho que serão enviadas para o chão-de-fábrica.

Definidas as ordens de trabalho que serão fabricadas, a atividade seguinte consiste em agrupar as ordens de trabalho em família de peças (agrupamento restrito à lista de ferramentas da família definidas pelo plano de processo). Ao finalizar este agrupamento, as famílias de peças e suas respectivas ordens de trabalhos são escalonadas. O resultado é apresentado em um gráfico de Gantt no qual operações de cada tarefa são alocadas em um gráfico que relaciona o tempo de

[1] O diagrama de atividades é considerado como um caso particular do diagrama de estados (Furlan, 1998).

processamento e os recursos de produção (Asprova, 2006). Se desejar, o operador poderá salvar a lista de tarefas para posteriormente efetuar o *setup* das estações de trabalho.

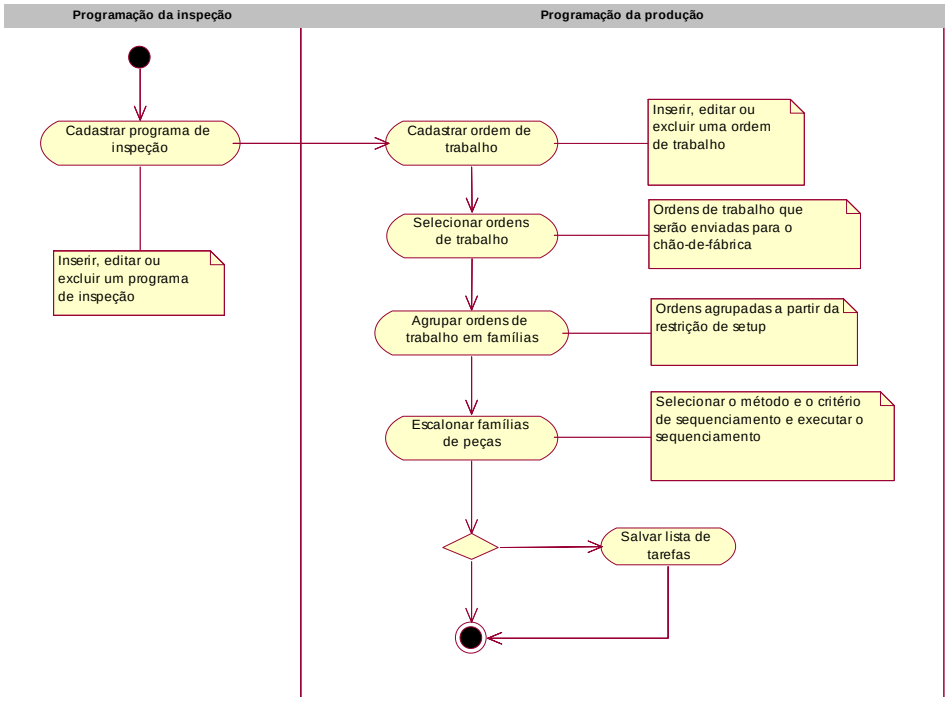

Figura 5.3 - Diagrama de atividades - *Scheduler*.

A figura 5.4 mostra o diagrama de estados do subsistema *Dispatcher*. Neste diagrama, os estados de atividade SetupFMC, OperaçãoFMC e EmergênciaFMC representam os módulos do subsistema *Dispatcher*. Cada estado possui um diagrama de atividades que representam as atividades executada por cada módulo do subsistema *Dispatcher*.

No estado SetupFMC a marcação inicial aponta para a primeira atividade relacionada ao carregamento da lista de tarefas. A lista de tarefas é um registro das informações necessárias para efetuar o *setup* das estações de trabalho. Posteriormente, verifica-se o *status* de cada estação de trabalho definida no *WorkstationGroup*, executa-se a programação das estações de trabalho e por último o *try-out*.

Ao iniciar a produção, uma transição de um estado complexo simultâneo demonstra a inicialização das atividades concorrentes do estado OperaçãoFMC: verificar os eventos de chão-de-fábrica e o controlador de operações. A primeira atividade encapsula um controlador a eventos discretos cuja principal finalidade é verificar a ocorrência de um evento não desejado. O controlador de operações re-programa as estações de trabalho, notifica a necessidade de efetuar

um novo *setup* (caso uma nova família de peças será produzida), como também notifica ao estado de atividade EmergênciaFMC eventuais perturbações.

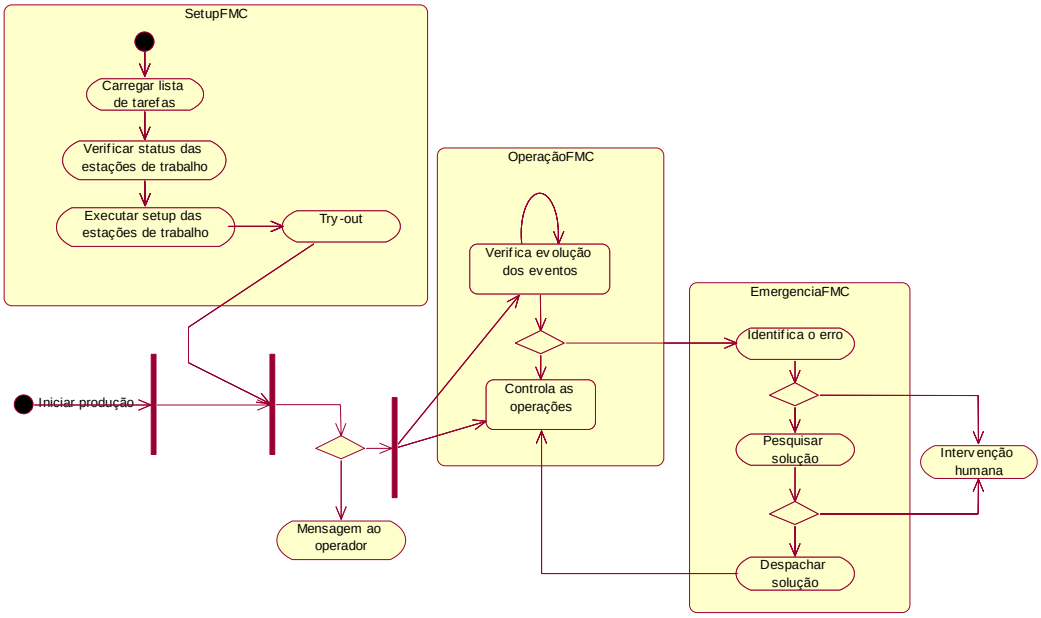

Figura 5.4 - Diagrama de estados – *Dispatcher*.

O estado de emergência realiza uma série de atividades na tentativa de resolver o problema observando a menor função de custo ao sistema. A primeira atividade busca identificar o erro. Caso seja identificado, a atividade Pesquisar solução é executada. Caso a solução seja encontrada uma mensagem é enviada ao controlador de operações para atuar na re-programação das estações de trabalho. Na impossibilidade de tratar o erro, o fluxo de controle passa a atividade Intervenção humana.

A figura 5.5 apresenta o diagrama de estados subsistema *Monitor*. Cada estado possui um diagrama de atividades que descreve seu funcionamento interno. Ao receber o sinal de início de atividades, o subsistema *Monitor* inicia as atividades presentes nos estados Monitorar eventos e Controle da qualidade. Caso o monitor de eventos identifique uma alteração nos estados das estações de trabalho, a atividade Atualizar estados é executada. Após a atualização dos estados, o fluxo de controle retorna ao monitor de eventos.

No estado Controle da qualidade, caso o processo seja qualificado, uma transição é disparada para colocar as atividades Calcular tempo de amostragem e Operação em estado de execução.

Esta atividade tem a responsabilidade de monitorar o índice de capabilidade do processo para evitar a produção de peças não-conformes, enquanto aquela calcula a freqüência em que as peças produzidas devem ser amostradas.

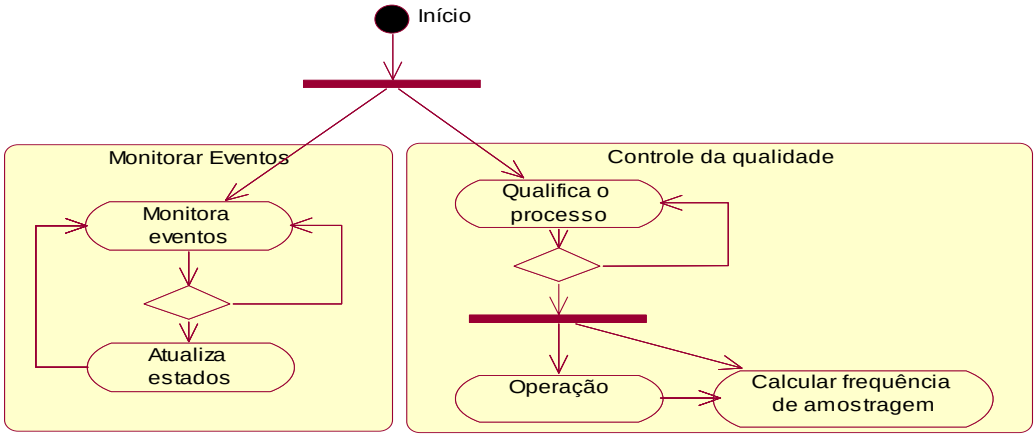

Figura 5.5 - Diagrama de estados - Monitor.

5.3 – PROJETO: DIAGRAMA DE CLASSES E SEQUÊNCIA

O diagrama de classes descreve os objetos que compõem um sistema e as relações estáticas estabelecida entre eles (Fowler *et at*, 2000). Um diagrama de classes é utilizado para representa um ponto de vista estático do projeto de um sistema, além de permitir a construção de sistemas executáveis por meio da engenharia reversa e de produção (Booch *et at*, 2000).

A figura 5.6 apresenta o diagrama de classes simplificado da Unidade de Gerenciamento. Neste diagrama, são especificados os subsistemas que compõem a MgU, bem como os dispositivos externos. A classe *ManagementUnit* se relaciona com as classes *Scheduler*, *Dispatcher* e *Monitor* por meio de uma agregação[2]. Essas classes representam à parte do relacionamento enquanto aquela representa o todo. Contudo, toda instância da classe *ManagementUnit* possui uma instância das classes *Scheduler*, *Dispatcher* e *Monitor*.

Uma associação pode especificar uma relação comum de troca de serviços entre classes. A associação estabelecida entre a classe *ManagementUnit* e *DataBase* (figura 5.6) é necessária para a troca de informações geradas pelas classes *Scheduler* (programação da produção e da inspeção) e *Monitor* (*status* das estações de trabalho, tempo de processamento, dentre outros).

[2] Um diamante ligado à classe todo representa este tipo de relacionamento.

Por outro lado, a associação entre as classes *ManagementUnit* e *WorkstationController* tem por finalidade especificar a comunicação entre os controladores das estações de trabalho e a MgU.

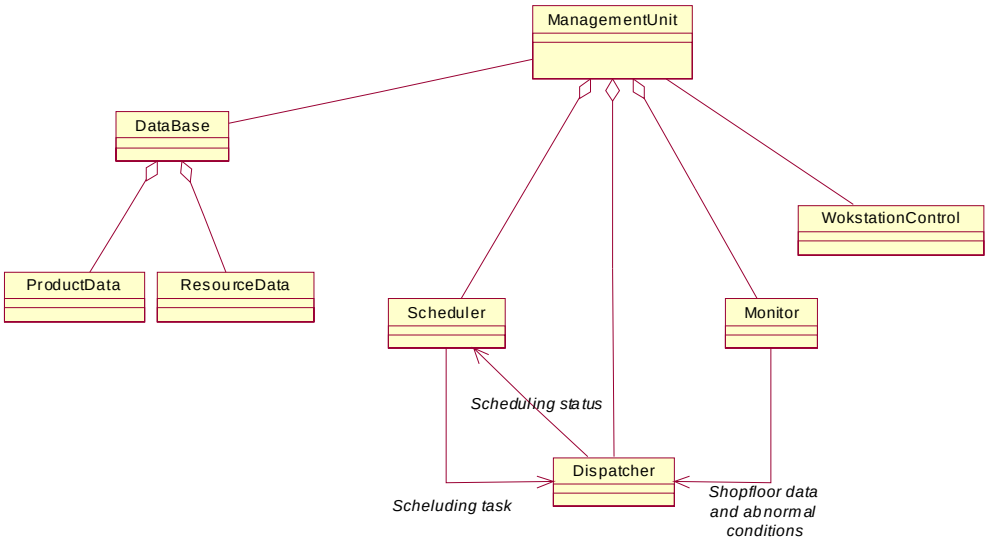

Figura 5.6 - Diagrama de classes simplificado.

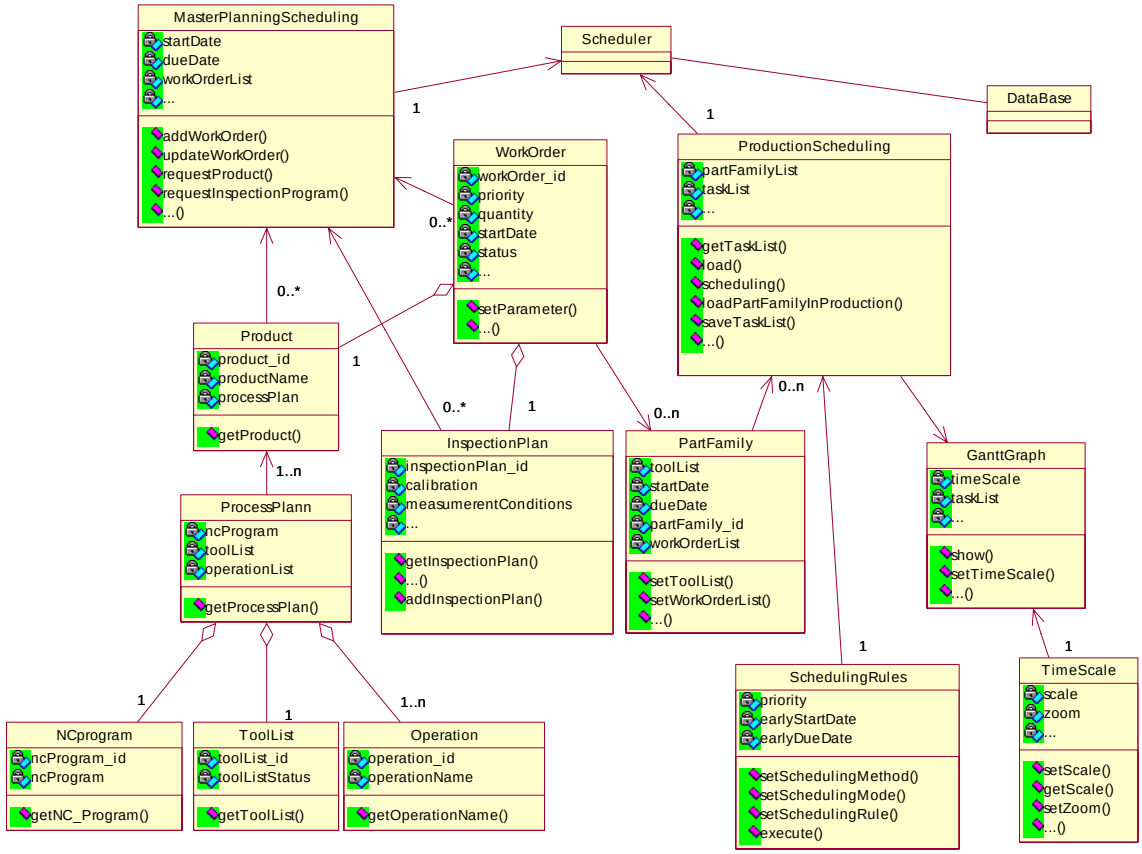

Figura 5.7 - Diagrama de classes Scheduler.

No diagrama *Scheduler* (figura 5.7), a classe *ProcessPlan* se relaciona com as classes *NCProgram*, *ToolList* e *Operation*. Estas duas primeiras encapsulam as informações geradas pelo plano de processo, enquanto a terceira as operações alocadas por cada ordem de trabalho para a fabricação do produto. A cardinalidade do relacionamento entre as classes *PartFamily* e *ProductionScheduling* indica uma relação de 0 para muitos (0:n), isto significa que um objeto *ProductionScheduling* pode conter zero, uma ou mais instâncias da classe *PartFamily*.

As figuras 5.8 e 5.9 mostram respectivamente o diagrama de classes *Dispatcher* e *Monitor*. No diagrama *Dispatcher* as classes *RobotController*, *TurningCenterController*, *MicrometerController* e *AGVControler* representam os controladores lógicos das estações de trabalho. Cada controlador possui uma instância *Connection* e uma instância *Workstation* que encapsulam a conexão e o estado lógico respectivamente da estação de trabalho a qual faz referência.

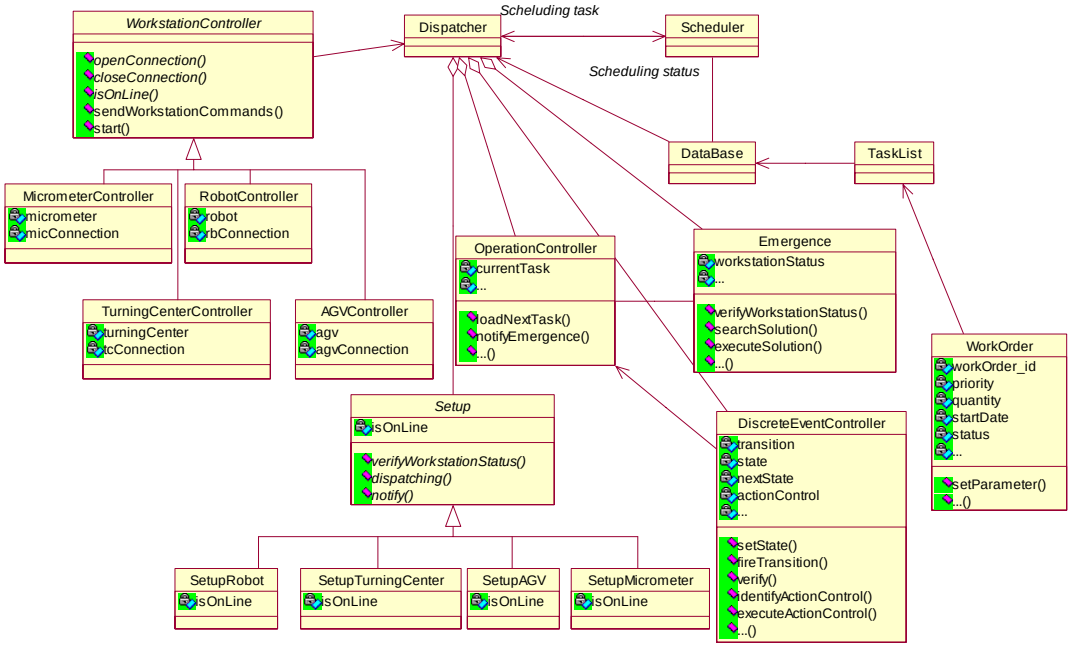

Figura 5.8 - Diagrama de classes Dispatcher.

Os controladores das estações de trabalho possuem um conjunto de serviços comuns (*verifyWorkstationStatus()*, *sendWorkstationCommands()*, etc). Estes serviços foram então encapsulados em uma classe abstrata *WorkstationController*. Por este motivo, o relacionamento

entre as classes *RobotController*, *TurningCenterController*, *MicrometerController*, *AGVControler* e a classe *WorkstationController* é um relacionamento de generalização[3].

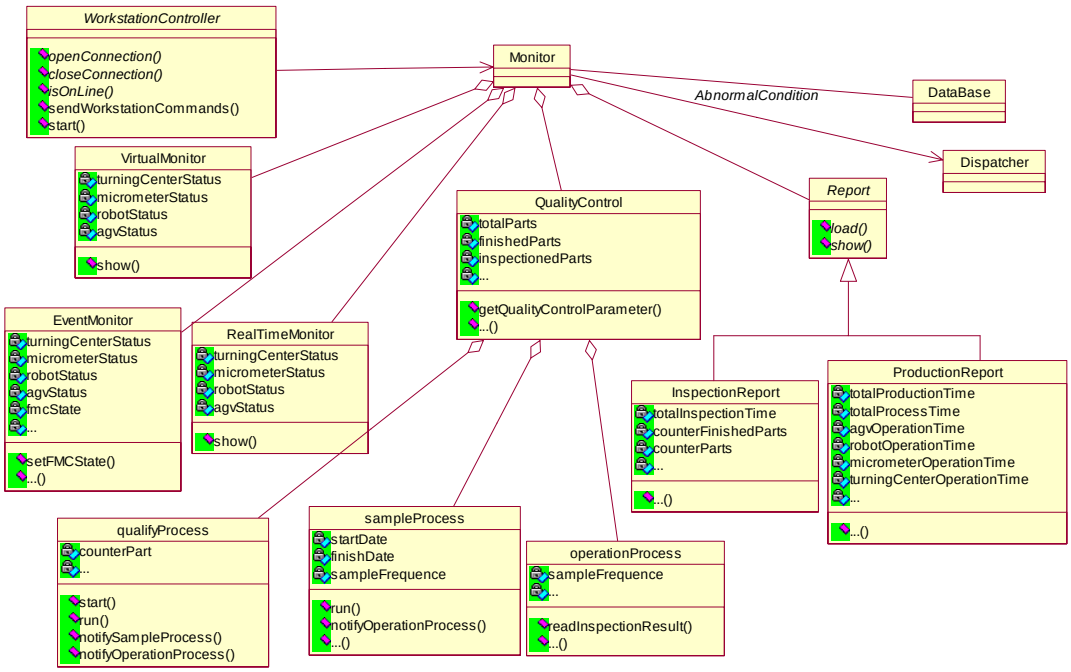

Figura 5.9 - Diagrama de classes Monitor.

No diagrama de classes *Monitor* (figura 5.9) uma relação de agregação indica as principais classes que compõem o subsistema *Monitor*. *EventMonitor* é uma classe cuja instância é responsável pelo monitoramento periódico dos eventos no chão-de-fábrica. *VirtualMonitor* e *RealTimeMonitor* são classes que encapsulam a visualização virtual e em tempo real respectivamente das estações de trabalho operando. *QualityControl* encapsula instâncias das classes *SampleProcess* e *OperationProcess*. Os serviços comuns prestados pelas classes *ProductionReport* e *InspectionReport* foram agrupados em uma classe abstrata *Report*.

Na linguagem de modelagem unificada, um diagrama de seqüência é utilizado para modelar os aspectos dinâmicos do sistema enfatizando a ordenação temporal das mensagens (Booch *et at*, 2000). Um diagrama de seqüência é composto por um conjunto de objetos, atores e sistemas que interagem entre si para realizar uma atividade em particular.

[3] Na UML a generalização é o mecanismo utilizado para especificar uma relação de herança entre classes.

O diagrama de seqüência da MgU é apresentado na figura 5.10. O operador[4] adiciona a ordem de trabalho (*WorkOrder*) no sistema através do método *addWorOrder()*. Adicionada as ordens de trabalho no sistema e estabelecida a programação da produção, a lista de tarefas é carregada pelo subsistema *Dispatcher* através do método *loadTaskList()*.

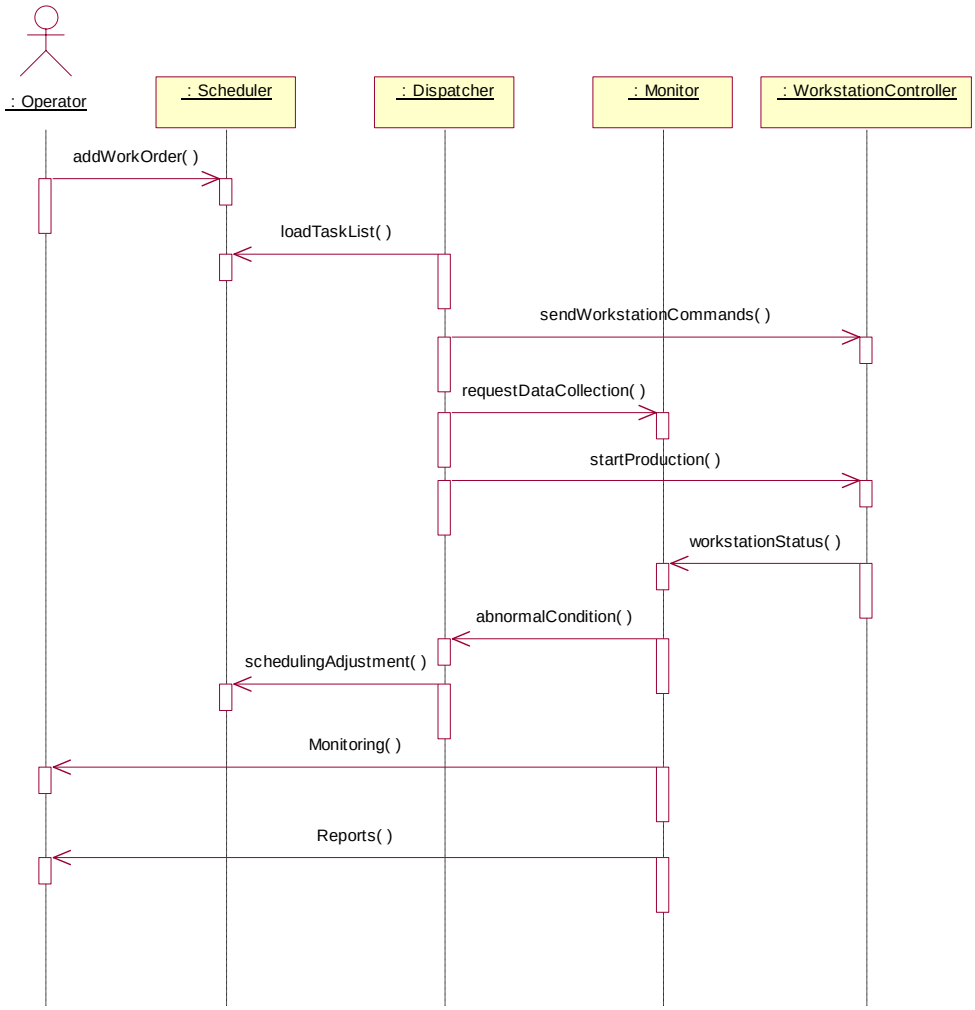

Figura 5.10 - Diagrama de seqüência da Unidade de Gerenciamento.

O subsistema *Dispatcher* carrega a lista de tarefas a ser processada e as converte em instruções para as estações de trabalho, através do método *loadTaskList()*. Estas instruções são posteriormente envidadas para as estações de trabalho através do método *sendWorkstationCommands()*. Confirmado o carregamento das instruções, *Dispatcher* envia

[4] Em uma futura integração o operador deverá ser substituído por um sistema CAP (*Computer Aided Production*).

uma mensagem para o subsistema *Monitor* (*requestDataCollection()*) notificando a necessidade de coleta de dados.

Uma mensagem *startProduction()* é enviada para o objeto *WorkstationController* para que a produção seja iniciada. A cada evento de chão-de-fábrica, o objeto *WorkstationController* envia uma mensagem *workstationStatus()* ao subsistema *Monitor*. A presença de condições anormais ou de eventuais problemas (parada de emergência, tarugo fora das especificações, dentre outros) verificada pelo subsistema *Monitor* é enviada a *Dispatcher* através do método *abnormalCondition()*, este por sua vez notifica o subsistema *Scheduler* (*schedulingAdjustment()*) eventuais alterações no escalonamento estabelecido durante a programação da produção.

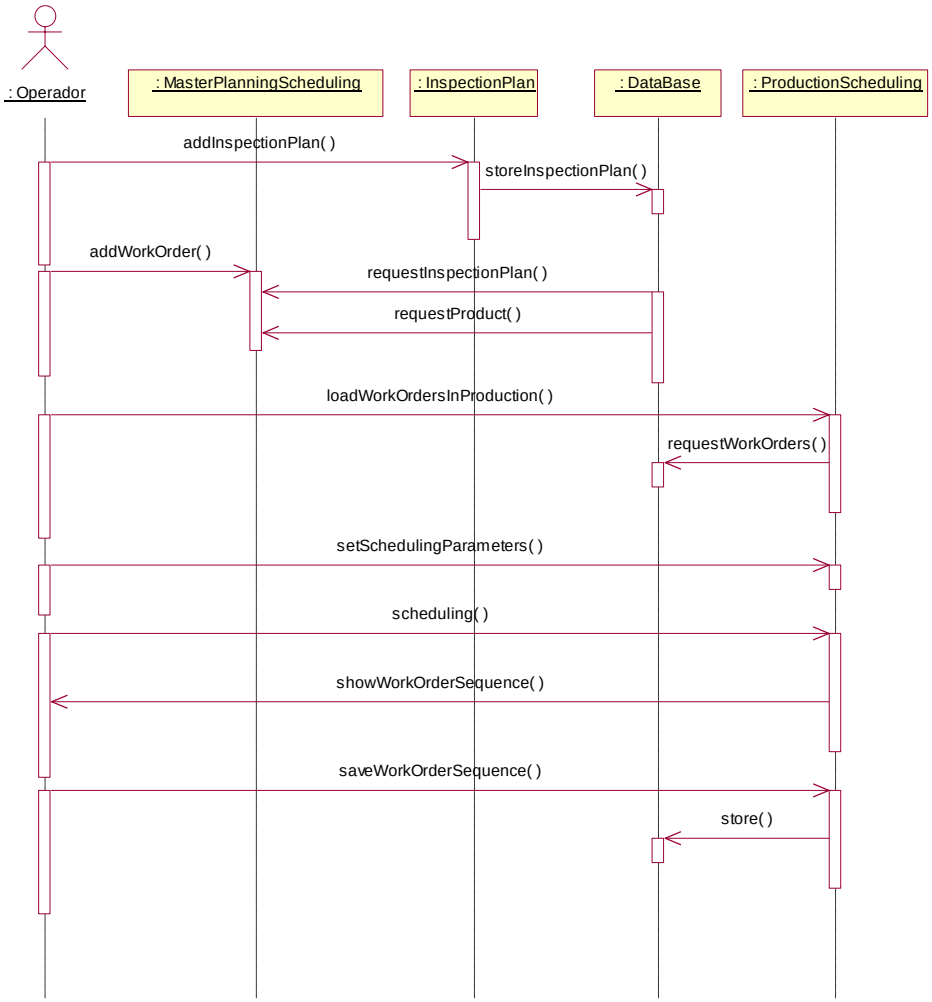

Figura 5.11 - Diagrama de seqüência do subsistema *Scheduler*.

O diagrama de seqüência da figura 5.11 apresenta em maiores detalhes as interações entre os objetos do subsistema *Scheduler*. A necessidade de adicionar um programa de inspeção, *addInspectionProgram(),* antes de adicionar a ordem de trabalho se deve a restrição de que toda ordem está vinculada a um programa de inspeção. Todavia, nada impede que o operador adicione somente uma ordem de trabalho vinculando-a a um programa de inspeção previamente cadastrado.

Definidas as ordens de trabalho que serão processadas, o próximo passo consiste em estabelecer a seqüência em que as ordens serão produzidas. O operador, ao enviar uma mensagem *loadWorkOrderInProduction(),* solicita ao objeto *ProductionScheduling* o carregamento das ordens de trabalho em produção. Após estabelecer os parâmetros de escalonamento (*setSchedulingParameters()*) o operador então envia uma mensagem *scheduling()* ao objeto *ProductionScheduling* para que o escalonamento seja efetivamente executado.

O diagrama da figura 5.12 apresenta o comportamento dinâmico do subsistema *Dispatcher*. As atividades iniciam com o operador solicitando ao objeto *DispatcherMainController* o carregamento da lista de tarefas (*loadTaskList()*). Este objeto, por sua vez, solicita à base de dados a lista de tarefas com as respectivas informações utilizadas no carregamento da lista de tarefas nas estações de trabalho (programa NC, lista de ferramentas, dentre outras).

Após o carregamento da lista de tarefas uma mensagem *verifyWorkstationStatus()* é enviada ao objeto *Setup*. Este objeto encapsula a lógica utilizada na verificação do *status* das estações de trabalho. Caso as estações de trabalho estejam operando em regime normal de funcionamento ao operador é permitido solicitar o carregamento de instruções nas estações ao objeto *Setup* através do método *sendWorkstationCommand().*

Finalizado o carregamento das instruções nas estações de trabalho o objeto *Setup* envia uma mensagem ao subsistema *Monitor* para ativar o módulo de coleta de dados. Feito isso ao operador é permitido enviar uma mensagem solicitando o início da produção. A cada evento do chão-de-fábrica o objeto *WorkstationController* envia uma mensagem *workstationStatus()* para atualizar o controlador em tempo real.

Na presença de perturbações ou condições anormais de funcionamento o subsistema *Monitor* notifica ao objeto *Emergence* a necessidade de intervenção. Este, por sua vez, verifica as

possíveis soluções e encaminha ao objeto *WorkstationController* o ajuste da programação através do método *sendSoluction()*. Caso seja necessário modificar o escalonamento da produção uma mensagem *shortAdjustment()* é enviada ao subsistema *Scheduler*. Na impossibilidade de tratar a situação de emergência, o objeto *Emergence* notifica a necessidade da intervenção humana ao operador através do método *emergence()*.

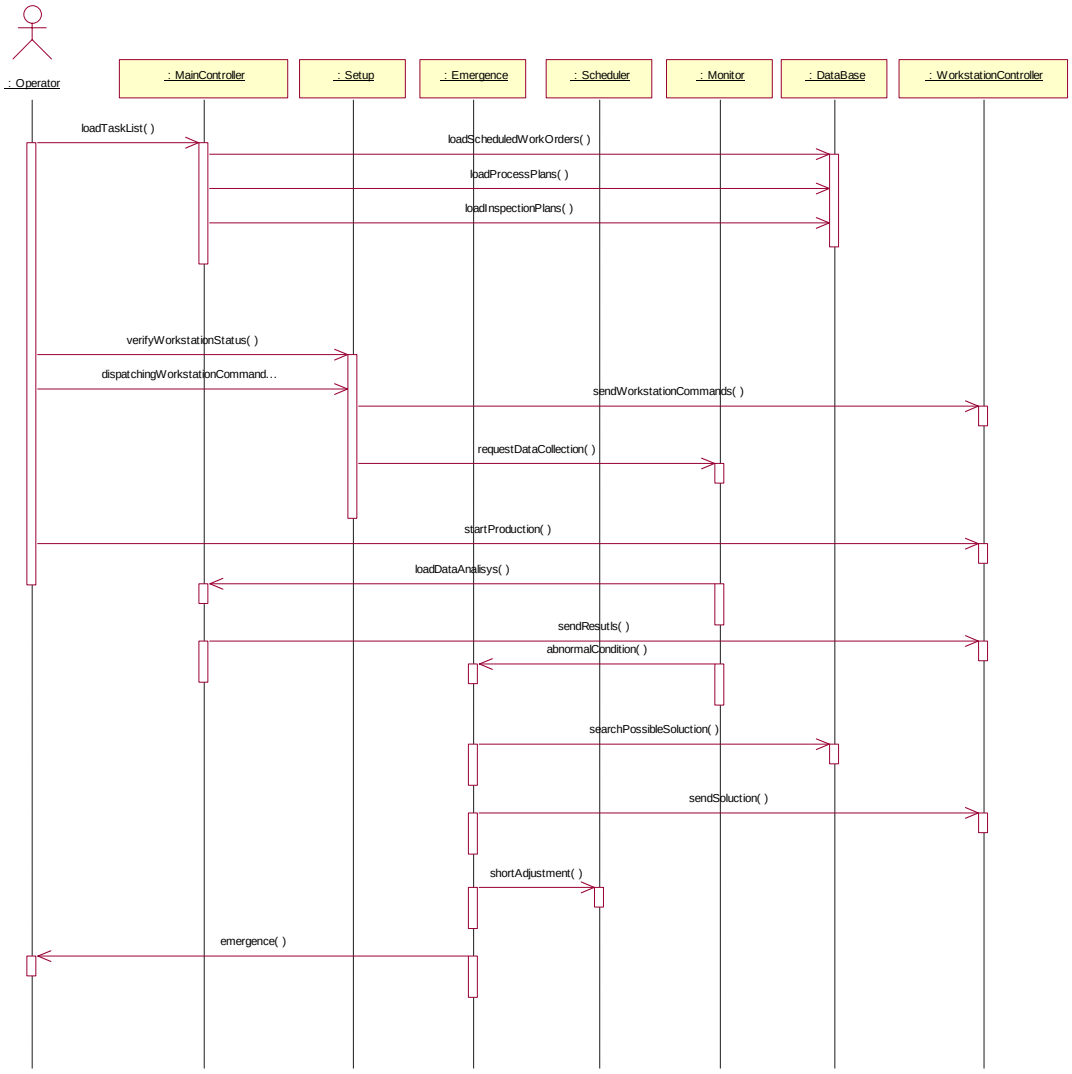

Figura 5.12 - Diagrama de seqüência do subsistema *Dispatcher*.

A figura 5.13 mostra o diagrama de seqüência *Monitor*. O subsistema *Monitor* possui três objetos que encapsulam suas principais funcionalidades: *EventMonitor*, *QualityControl* e *Report*. *EventMonitor* é o objeto responsável pelo acompanhamento da evolução dos eventos no

chão-de-fábrica, enquanto os objetos *QualityControl* e *Report* são utilizados no controle da qualidade e na apresentação de resultados respectivamente.

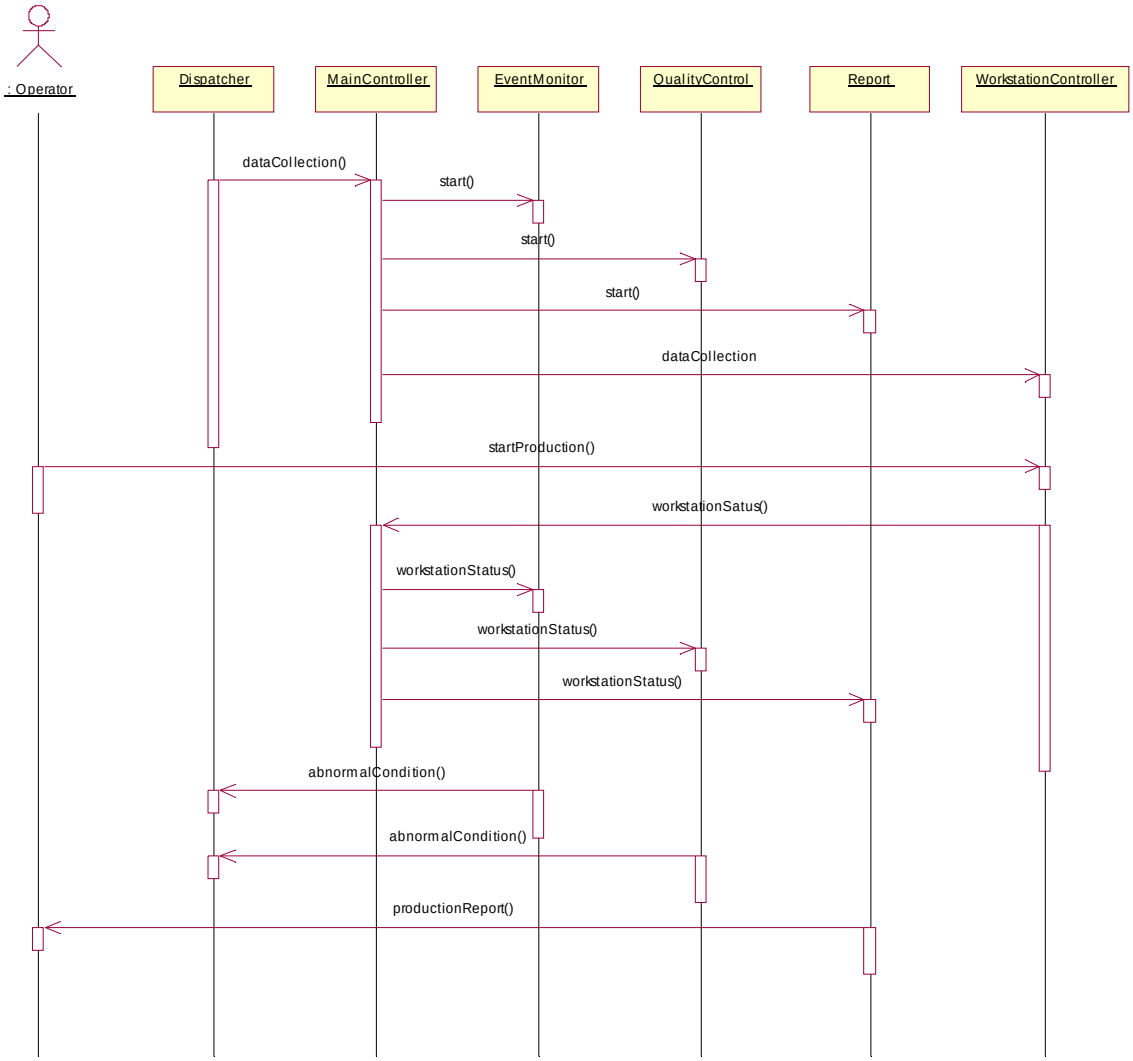

Figura 5.13 - Diagrama de seqüência do subsistema *Monitor*.

As atividades iniciam com a solicitação *dataCollection()* do subsistema *Dispatcher*. Esta solicitação é repassada pelo objeto *MainController* aos objetos *EventMonitor*, *QualityControl* e *Report* na forma de mensagens *start()*. Após a inicialização dos módulos, o subsistema *Monitor* está apto (*MonitorListener*) a receber mensagens *workstationStatus()*.

Ao iniciar a produção, o objeto *WorkstationController* varre periodicamente os controladores das estações de trabalho para atualizar os atributos *STATUS*. Estes atributos são repassados ao objeto *MainController* na forma de mensagens *workstationStatus()*. Os argumentos deste

método (*TurningCenterStatus*, *MicrometerStatus* e *AGVStatus*) encapsulam as informações utilizadas na atualização e análise da produção realizada pelos módulos do subsistema *Monitor*.

A presença de perturbações pode ser detectada tanto pelo objeto *EventMonitor* quanto pelo objeto *QualityControl*. *EventMonitor* é responsável pela identificação de uma condição anormal (evento não esperado) enquanto *QualityControl* pelo monitoramento da produção de peças não-conformes. Ao ser identificado qualquer uma destas condições uma mensagem *abnormalCondition()* é enviada ao subsistema *Dispatcher* com o intuito de avaliar a perturbação e atuar na resolução do problema.

5.4 – IMPLEMENTAÇÃO: DIAGRAMA DE PACOTES/COMPONENTES

Em se tratando da tecnologia de programação a ser utilizada (neste caso Java), os módulos do sistema possuem as responsabilidades distribuídas em pacotes. Um pacote é um mecanismo de propósito geral utilizado para organizar elementos de um mesmo grupo (Booch *et at*, 2000). Classes, interfaces e componentes que possuem funcionalidades semelhantes são agrupadas em pacotes[5].

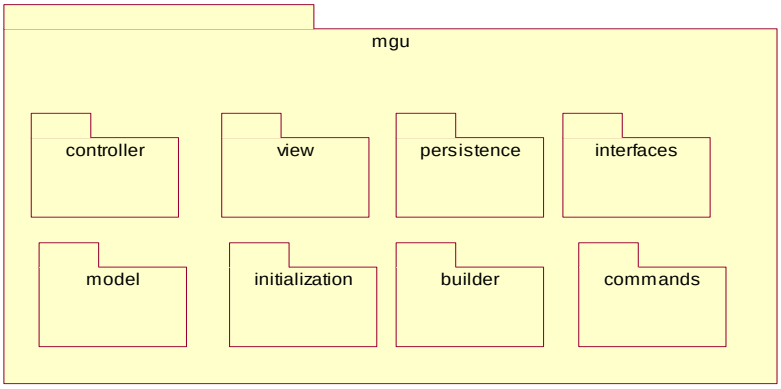

Figura 5.14 - Diagrama de Pacotes da Unidade de Gerenciamento.

A figura 5.14 mostra o diagrama de pacotes da Unidade de Gerenciamento. Cada pacote possuiu um conjunto de elementos responsável por parte da funcionalidade do sistema. *Initialization* é o pacote que contém uma única classe denominada *Initialization*. Esta classe possui um único método (o método *main ()*) invocado todas as vezes que a MgU é inicializada. Este método, ao

[5] As classes, interfaces e componentes de um pacote são posteriormente compactados em um arquivo JAR (*Java ARchive*). Geralmente, estes arquivos são utilizados para distribuir a aplicação através de um servidor *Web*.

ser invocado, tem como único propósito instanciar a controladora de navegação (*NavigatorController*) do sistema.

O pacote *Controller* armazena as classes controladoras. Este tipo de classe descreve a lógica de funcionamento do sistema. As controladoras podem ser de dois tipos: *FrameController* e *Controller*. As controladoras de *frame* (*FrameController*) capturam as iterações do usuário com as GUI's enquanto as controladoras (*Controller*) gerenciam a navegabilidade e troca de serviços entre camadas.

Dentre as classes contidas no pacote *Builder* estão: *BuilderScheduler, BuilderDispatcher* e *BuilderMonitor*. Essas classes são responsáveis pela criação dos subsistemas *Scheduler, Dispatcher* e *Monitor* respectivamente. Isto se resume em instanciar as controladoras de *frame* e as controladoras de cada subsistema e posteriormente interconectá-las através de relacionamentos. A funcionalidade de um objeto *Builder* se limita somente à criação de objetos ou subsistemas complexos (Gamma *et at*, 2000), o que justifica a destruição do objeto após o cumprimento da tarefa a qual se destinam.

Interfaces é o pacote que agrupa as interfaces utilizadas pela Unidade de Gerenciamento. Uma interface é um mecanismo utilizado para reduzir o grau de acoplamento entre os objetos. Quando a troca de serviços entre camadas de software é regida por uma interface as alterações feitas em uma classe não se propagam por todo o sistema, o que facilita a expansibilidade e a manutenção do aplicativo.

No pacote *View* estão todas as *GUI's* desenvolvidas para a MgU. Objetos customizados como o *DateChooseButton*, também são armazenados neste mesmo pacote. Estes objetos ajudam a manter o princípio do ocultamento da informação (ao encapsularem funcionalidades comuns) além de evitarem a repetição de código.

O pacote *Command* agrupa todas as classes *command*. Uma instância de uma classe *command* encapsula uma solicitação como um objeto. Consequentemente, o objeto que invoca a operação não precisa saber como ela deve ser executada. Na MgU, um objeto *SQLInsertWorkOrder* ao ser instanciado, recebe um objeto *WorkOrderData* que encapsula os atributos de uma ordem de trabalho (data de entrega, prioridade, etc). Ao invocar o método *execute()* de

SQLInsertWorkOrder, uma *String* contendo uma instrução SQL é criada para que a nova ordem possa ser armazenada na base de dados.

No pacote *Persistence* estão todas as classes que estabelecem uma comunicação com os dispositivos externos. Instância de classes como *DBConnection*, *MicrometerConnection*, *TurningCenterConnection*, *RobotConnection* e *AGVConnection* possuem a responsabilidade de estabelecer conexões com o banco de dados, com o micrômetro, com o manipulador robótico, com o Centro de Torneamento e com o AGV respectivamente.

A figura 5.15 mostra o diagrama de componentes da Unidade de Gerenciamento baseado na arquitetura cliente-servidor. As *GUI's* e funcionalidades de alto nível estão disponíveis no módulo cliente, enquanto que as funcionalidades de baixo nível (conexão direta com as estações de trabalho, etc) são disponibilizadas no módulo servidor. A comunicação entre estes dois módulos é estabelecida através de *sockets*, utilizando-se o protocolo TCP/IP.

O cliente (aqui representado pelo usuário remoto) se conecta ao servidor WebFMC através da URL: http://webfmc.graco.unb.br/mgu/mgu.jnlp. Este link aponta para um arquivo JNLP[6], isto é, um documento XML que especifica como os arquivos JAR's, que compõem o módulo cliente da MgU, deverão ser baixados para o computador do usuário remoto. Finalizado o *download* de todos os arquivos especificados no documento XML, a MgU é então executada.

Além de incorporar todas as vantagens oferecidas pelos *applets* (permitir a execução de programa via *Web* sem a necessidade de instalação, política de restrições, etc), a tecnologia JNLP permite o *download* incremental do aplicativo. Isto significa que todas as vezes que o aplicativo for executado no computador cliente, somente os arquivos JAR's modificados no servidor *Web* serão baixados. Outra vantagem desta tecnologia está no fato de permitir que o aplicativo seja executado sem a dependência do *browser*, diferentemente dos *applets* que estão ligados a página do navegador.

[6] Utiliza-se a tecnologia JNLP para distribuir, na *Web*, o módulo cliente da MgU. Esta tecnologia é um protocolo com um ambiente de aplicação que permite que um aplicativo Java seja executado através de um servidor *Web* (JNLP, 2006).

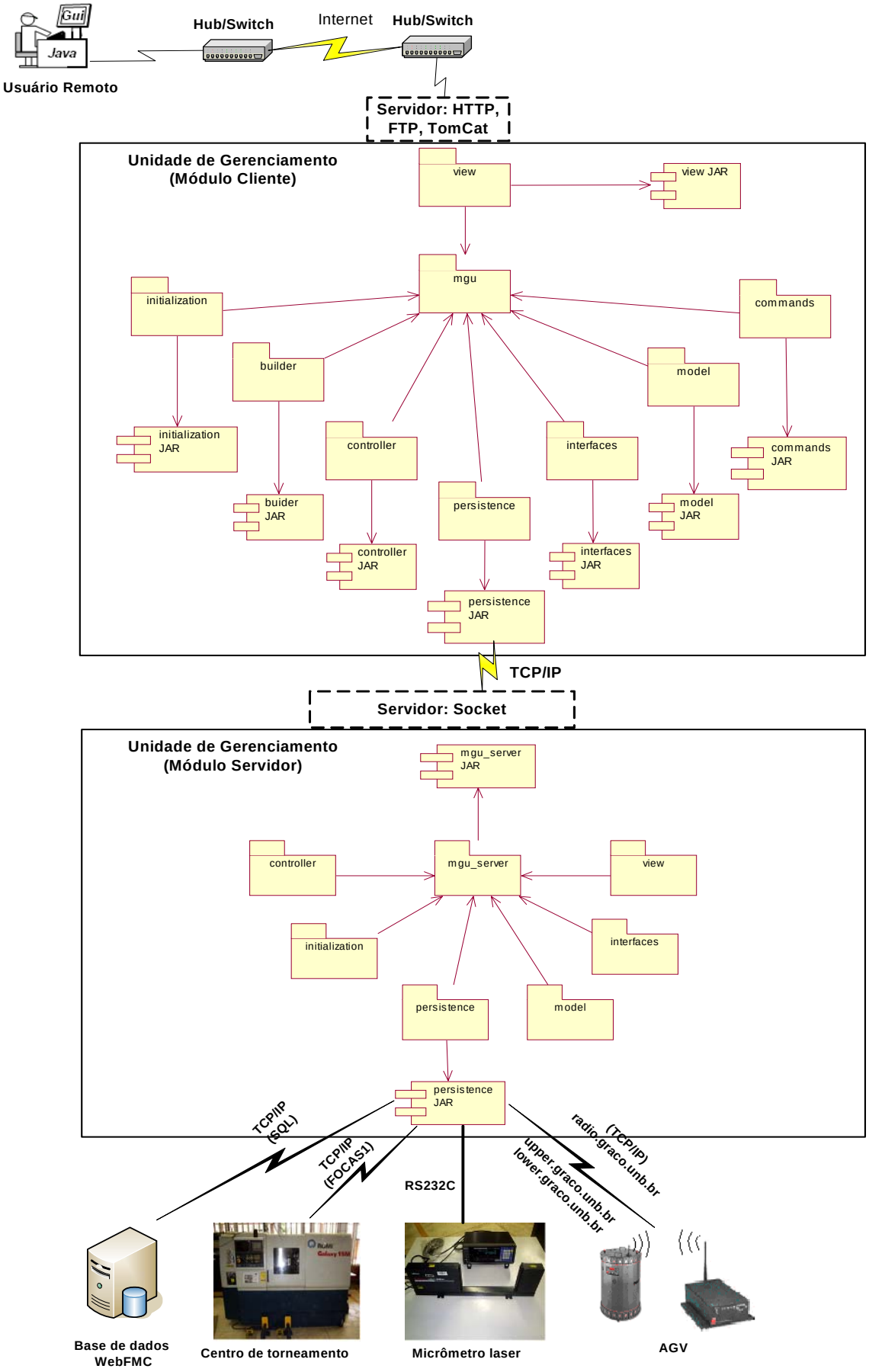

Figura 5.15 - Diagrama de componentes da MgU – parte 1.

A figura 5.16 mostra o segundo diagrama de componentes da MgU. Neste diagrama, as instâncias das classes *SchedulerController*, *DispatcherController* e *MonitorController* (módulo cliente) encapsulam funcionalidades de alto nível (como *requestMasterPlanningScheduling*() e *sendWorkstationCommands*(), *verifyWorkstationStates*() respectivamente) e estabelecem a comunicação com os controladores lógicos das estações de trabalho, presentes na camada de persistência, através de suas respectivas interfaces.

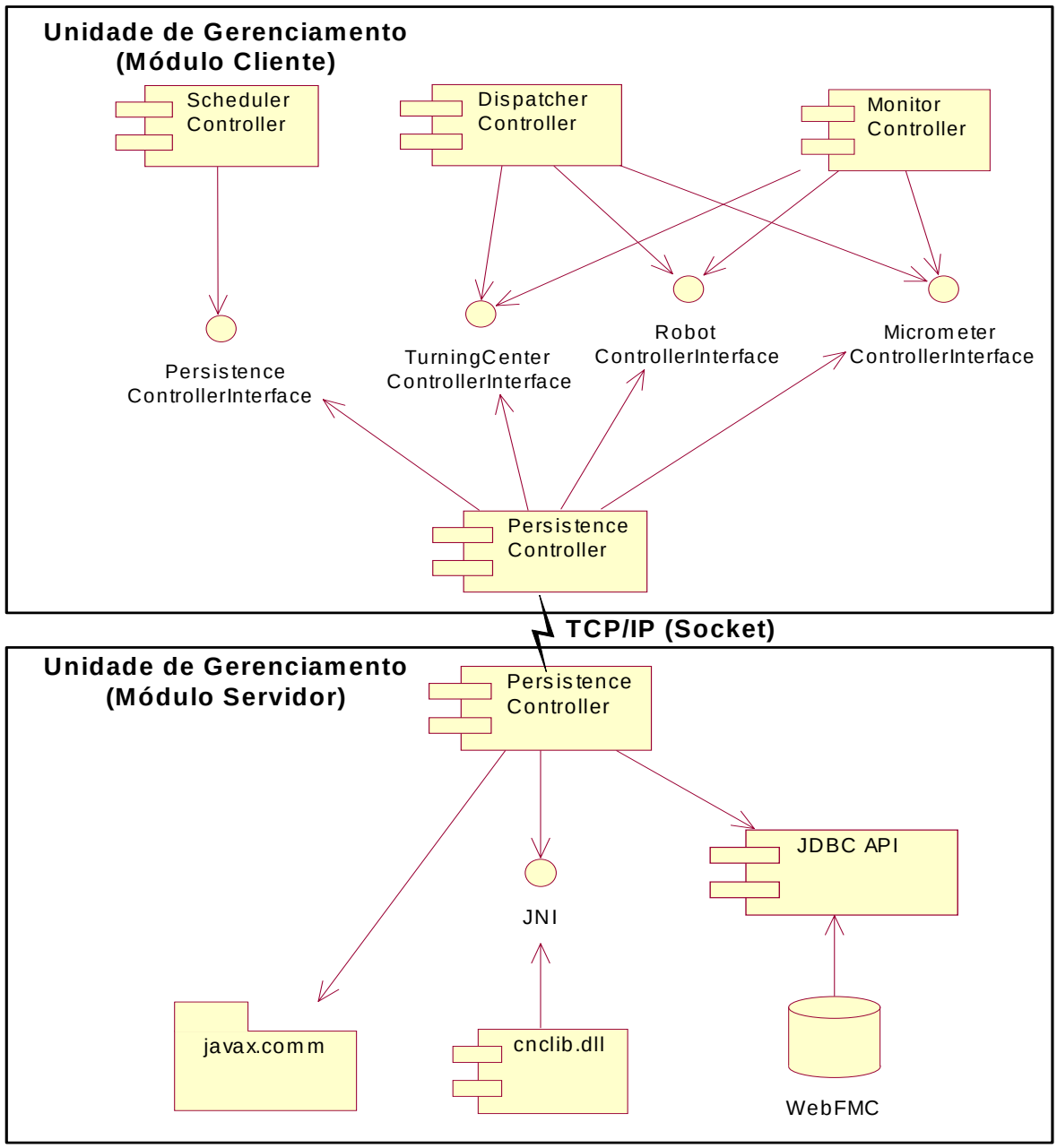

Figura 5.16 - Diagrama de componentes da MgU - parte 2.

Os controladores lógicos do módulo servidor implementam as funcionalidades de alto nível, oferecidas ao módulo cliente, e estabelecem a comunicação direta com os controladores físicos das estações de trabalho. Contudo, o módulo servidor da MgU estabelece a comunicação com o Centro de Torneamento via JNI[7]. Esta interface permite que a Unidade de Gerenciamento (escrita em Java) se comunique com a biblioteca desenvolvida em C++ (cnclib.dll), criada para encapsular algumas das funções programáveis oferecidas pela *API FOCAS1* e utilizadas na comunicação com o Centro de Torneamento.

A comunicação com a Unidade de processamento do micrômetro é estabelecia mediante a *API* Java *Communication*. Esta *API* (também conhecida como javax.comm) permite que o aplicativo tenha acesso a interfaces seriais RS232 (porta serial) e o acesso limitado a porta paralelas (IEEE-1284) (Java Communication, 2006).

O módulo servidor da MgU acessa a base de dados WebFMC através da *API JDBC*. Esta *API* é um padrão industrial estabelecido para a conectividade entre a linguagem Java e uma variedade de base de dados, utilizando-se da linguagem SQL para a comunicação (JDBC technology, 2006). Consequentemente, se houver a necessidade de migrar para outra base de dados, não é necessário modificar o servidor da MgU, mas apenas o *driver* que estabelece a conexão.

[7] A JNI (*Java Native Interface*) é uma interface nativa de programação que permite com que o código executado dentro da JMV interaja com outras aplicações e bibliotecas escritas em outras linguagens de programação como C, C++, etc (JNI, 2006).

6 – WEBFMC: IMPLENTAÇÃO COMPUTACIONAL E ALGORITMOS CONCEBIDOS

Este capítulo descreve em detalhes a implementação computacional e os algoritmos concebidos da Unidade de Gerenciamento. São apresentados os principais algoritmos de cada módulo da MgU, as interfaces gráficas (*GUI's*) implementadas e uma explicação detalhada como cada módulo deve funcionar.

6.1 – PROGRAMAÇÃO DA INSPEÇÃO

O plano de inspeção reúne as informações utilizadas na programação, no controle e monitoramento da inspeção. Este plano é composto por um conjunto de programas de inspeção previamente cadastrados na base de dados. Cada programa possui as informações geométricas da peça (diâmetro e as tolerâncias das *features* a serem inspecionadas), bem como as condições de medição (unidade, referência, escala, dentre outras).

Figura 6.1 - Programa de inspeção da Unidade de Gerenciamento.

A figura 6.1 ilustra um típico programa de inspeção. No micrômetro, as medições são automaticamente realizadas de acordo com as informações geométricas registradas (programadas) incluindo o segmento (*feature* a ser medida) e o critério de julgamento GO/NG (Mitutoyo, 2003). Desse modo, cabe ao operador definir a *feature* da peça a ser medida, cadastrar as respectivas informações geométricas (diâmetro e tolerâncias) e as condições de medição em que a inspeção deve ser efetuada.

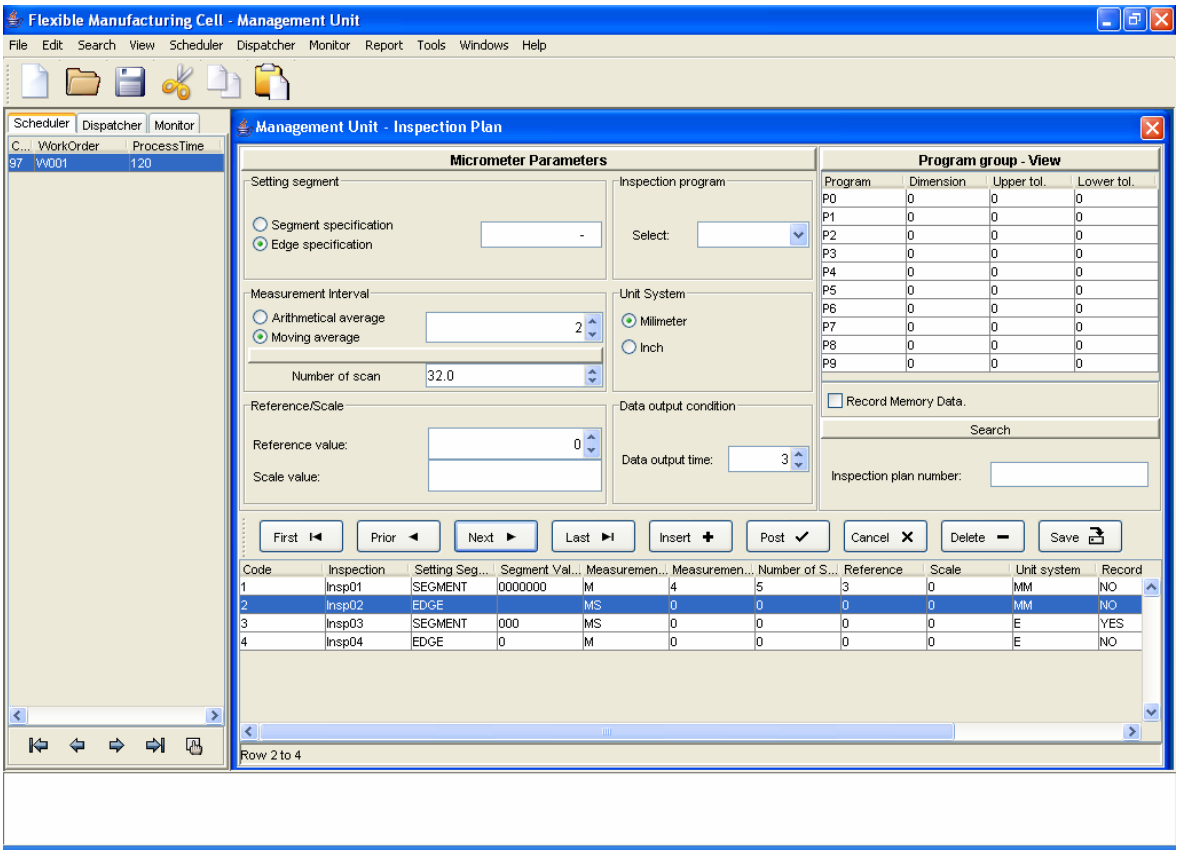

Figura 6.2 - *GUI InspectionPlan*.

A *GUI InspectionPlan* da Unidade de Gerenciamento (figura 6.2) foi implementada para oferecer a possibilidade de adicionar, editar e excluir programas no plano de inspeção. Cada programa tem a capacidade de armazenar as informações geométricas de até dez *features*. Isto significa que o mesmo programa de inspeção pode ser utilizado para inspecionar diversas peças sem a necessidade de re-programar micrômetro, desde que as condições de medição sejam as mesmas.

6.2 – PROGRAMAÇÃO DA PRODUÇÃO

6.2.1 – Plano mestre da produção

O Plano mestre da produção reúne as ordens de trabalho (*WorkOrder*) cadastradas na base de dados. Uma ordem de trabalho possui um conjunto de atributos (como prioridade, data de entrega, tempo de processamento, etc.) que serão utilizados pelos demais módulos da MgU. A *GUI MasterPlanningScheduling* (figura 6.3) foi desenvolvida, assim como a *GUI*

InspectionPlan, para oferecer as funcionalidades de adicionar, editar ou até mesmo excluir ordens de trabalho na base de dados.

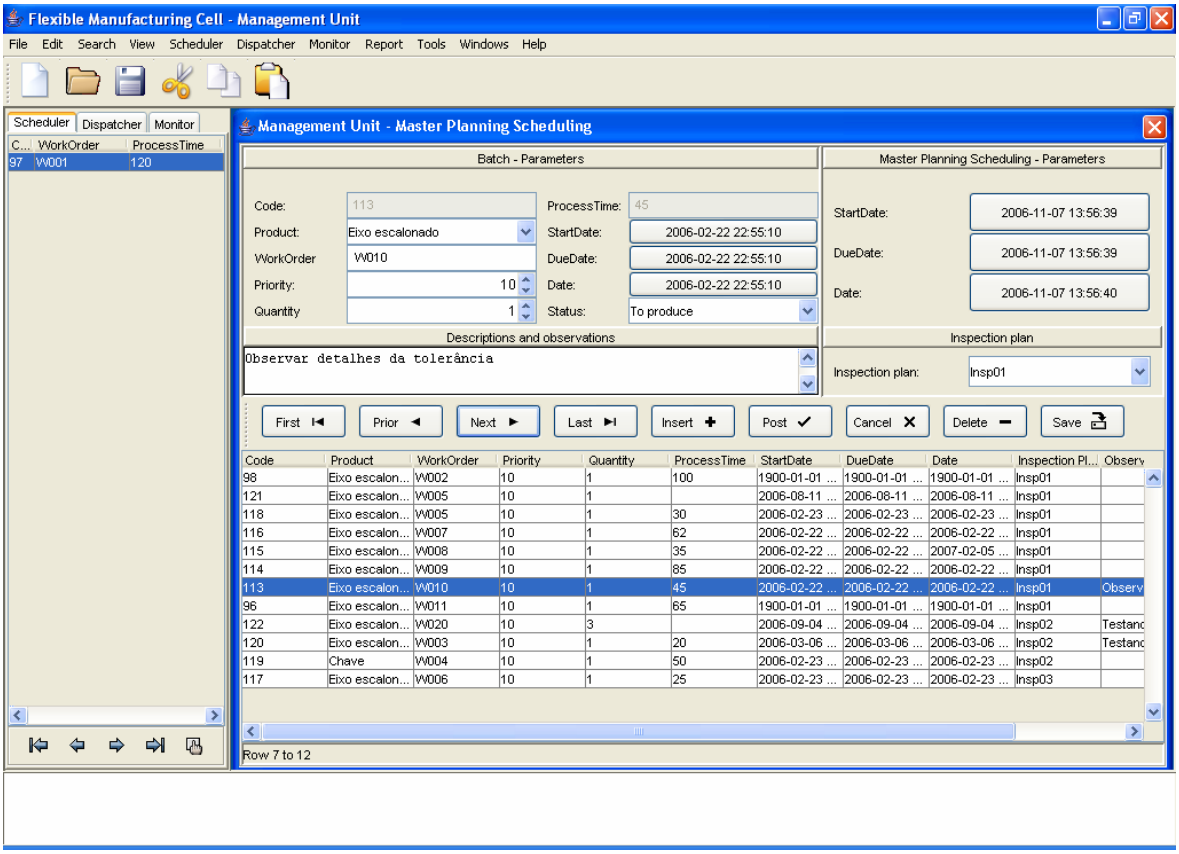

Figura 6.3 - *GUI MasterPlanningScheduling*.

Cada ordem de trabalho possui um atributo *STATUS*. Este atributo define a situação da ordem para o sistema (*To produce* – para produzir, *In production* - em produção e *Produced* - produzida). Contudo, para efetuar a programação da produção o operador deve selecionar as ordens de trabalho que serão produzidas setando o atributo *STATUS* como *In Production*.

6.2.2 – Formação das famílias de peças

A lógica de formação das famílias de peças é baseada no algoritmo de ordem por ranqueamento proposto por King *et at* (King *et at*, 1980). Neste caso, a matriz de incidência máquina-peça (também conhecida apenas como matriz máquina-peça) foi adaptada para atender ao problema particular de formação da família na FMC, isto é, as peças que possuem as mesmas características de fixação e de ferramentas são agrupadas em uma mesma família. A figura 6.4 apresenta o algoritmo *PartFamily*.

```
Algoritmo PartFamily
1.0 boolean flagColuna=false, flagLinha=false
2.0 Enquanto (!flagColuna && !flagLinha)
    2.1 Para cada coluna da matriz ferramenta-peça
        2.1.1 Calcular  o peso total binário da coluna
    2.2 Se os pesos das colunas estiverem em ordem ascendente
        2.2.1 flagColuna==true
    2.3 Senão
        2.3.1 Ordene as colunas em ordem ascendente
    2.4 Para cada linha da matriz ferramenta-peça
        2.4.1 Calcular  o peso total binário da linha
    2.5 Se os pesos das linhas estiverem em ordem ascendente
        2.5.1 flagColuna==true
    2.6 Senão
        2.6.1 Ordene as linhas em ordem ascendente
3.0 Fim algoritmo
```

Figura 6.4 - Algoritmo *PartFamily*.

PartFamily é um algoritmo cuja finalidade é agrupar peças em famílias e suas respectivas ferramentas em lista de ferramentas. Contudo, este algoritmo tem como objetivo agrupar uma família de peças de tal forma que possam ser processadas com o mesmo grupo de ferramentas, dispensando a necessidade de um novo *setup* de ferramentas todas as vezes que uma nova peça que pertença à mesma família for processada.

6.2.3 – Escalonamento da produção

Definida as famílias de peças que serão processadas, o passo seguinte consiste em estabelecer a seqüência em que as famílias (e as ordens de trabalho) serão processadas. O método de escalonamento selecionado é baseado em regras de prioridade (Starbek, 1993). A figura 6.5 apresenta a *GUI GanttGraph* implementada, com a seqüência das famílias e suas respectivas ordens de trabalho.

Caso o operador selecione o modo manual de escalonamento, uma *JDialog* relaciona as famílias de peças para que a seqüência seja definida manualmente. Após estabelecer a seqüência das famílias de peças, *JDialogs* são instanciadas para que a seqüência das ordens de trabalho de cada família seja estabelecida. As figuras 6.6 e 6.7 apresentam os algoritmos de sequenciamentos utilizados.

Caso o método automático seja o selecionado, o algoritmo verifica o método de programação (*Forward* ou *Backward*) e o critério de sequenciamento (*Priority*, *Earliest Due Date*, *First In First Out* ou *Shortest Processing Time*). O sequenciamento automático é executado com o

auxílio da arquitetura unificada *framework collection* fornecida pela plataforma Java 2. Esta arquitetura representa e manipula objetos *collection*[1] permitindo que os mesmos sejam manipulados independentemente da sua representação (Sun Microsystem, 2006).

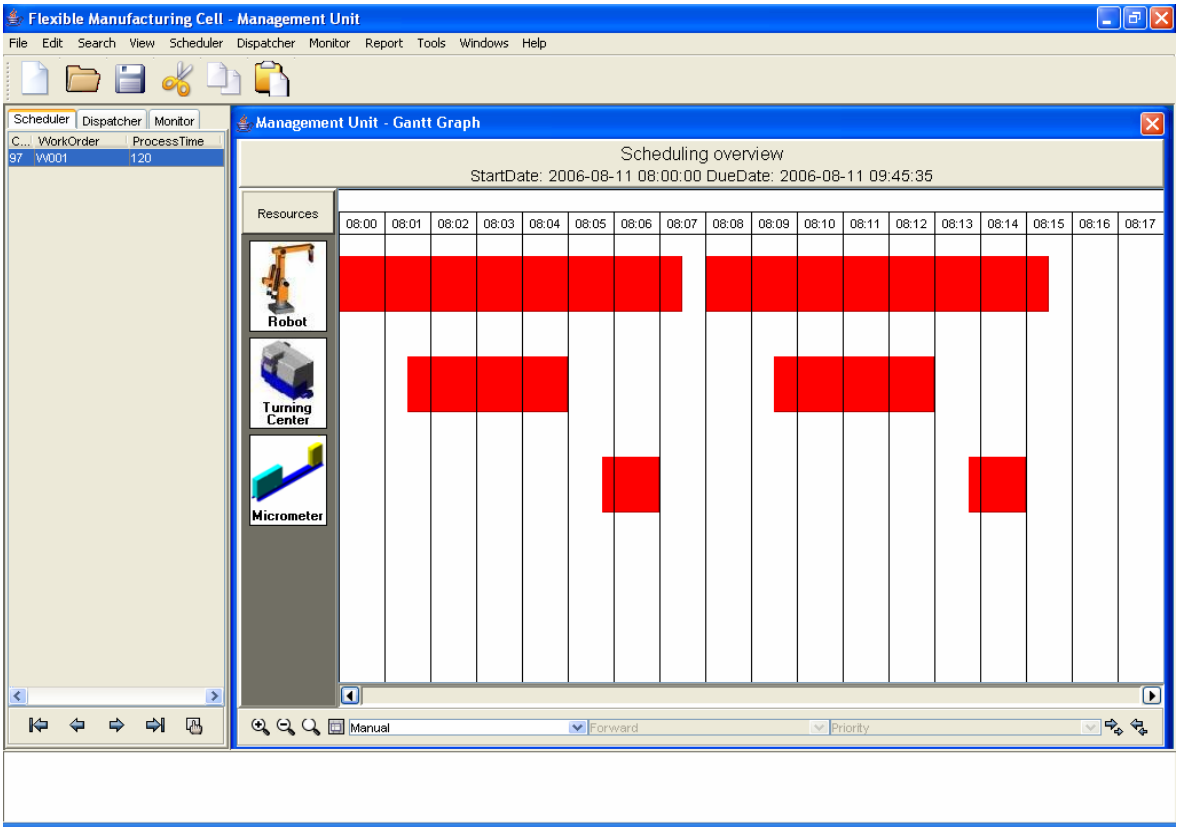

Figura 6.5 - *GUI GanttGraph.*

Algoritmo Scheduling
1.0 Verifica qual o modo de escalonamento selecionado
2.0 Se modo manual foi o selecionado
2.1 Solicita ao operador definir a sequência das famílias de peças
2.2 Para cada família de peças
2.2.1 Solicita ao operador definir a sequência das ordens de trabalho
3.0 Senão
3.1 Verifica método de sequenciamento selecionado
3.2 Verifica critério de sequenciamento selecionado
3.3 Executa o sequenciamento automático
4.0 Fim da estrutura Senão
5.0 Fim da estrura Se
6.0 Apresenta resultado em um gráfico de gantt
7.0 Fim algoritmo

Figura 6.6 - Algoritmo *Scheduling*.

[1] Um *collection* é um objeto utilizado para representar um grupo de objetos.

Algoritmo AutomaticScheduling
1.0 Para cada elemento do vetor partFamily
2.0 Seta o método e o critério de sequenciamento
3.0 Ordena vetor partFamily
4.0 Para cada elemento do vetor part family
4.1 vetor workOrderList = (Vector) partFamily[elemento]
4.2 Para cada elemento de workOrderList
4.2.1 Seta o método e o critério de sequenciamento
4.3 Ordena vetor workOrderList
5.0 Apresenta resultado em um gráfico de gantt
6.0 Fim algoritmo

Figura 6.7 - Algoritmo *Automatic Scheduling*.

Um objeto *collection*, através do método estático *sort()*, ordena os elementos do vetor *partFamilyList* e do vetor *workOrderList* em ordem ascendente de acordo com a ordenação natural dos elementos. Todo objeto *partFamily* e *workOrder* deve implementar a interface *Comparable* e consequentemente o método *toCompare(Object obj1, Object obj2)* para serem mutuamente comparados. Casos os objetos sejam considerados iguais eles não serão ordenados.

6.3 – INTEGRAÇÃO COM O SISTEMA CAD/CAPP/CAM

A integração entre a MgU e o sistema CAD/CAPP/CAM é baseada na elaboração do escalonamento após a elaboração do plano de processo. Apesar do problema levantado por Kumar e Rajotia[2] (Kumar *and* Rajotia, 2003) para este tipo de abordagem, ao considerar que a seqüência de processamento de cada tarefa dentro da célula é fixa, em outras palavras, o "roteamento" interno é rígido (Rabelo, 1997), esta abordagem pode ser aplicada.

A figura 6.8 mostra a arquitetura de integração cliente-servidor da base de dados. A base de dados WebFMC foi projetada em um banco de dados MySQL e possui um conjunto de quatro tabelas (*Product*, *ProcessPlan*, *NCProgram* e *ToolList*) utilizadas na integração. A tabela *Product* possui uma chave estrangeira da tabela *ProcessPlan*, e esta uma chave estrangeira das tabelas *NCProgram* e *ToolList*. Isto significa que todo registro da tabela *Product* está vinculado a um registro de *ProcessPlan*.

[2] Kumar e Rajotia (Kumar and Rajotia, 2003) afirmam que o principal problema relacionado a integração CAPP/*Scheduling* quando o escalonamento é realizado após a elaboração do plano de processo é a possibilidade de se utilizar um plano de processo não-ótimo do ponto de vista do escalonamento.

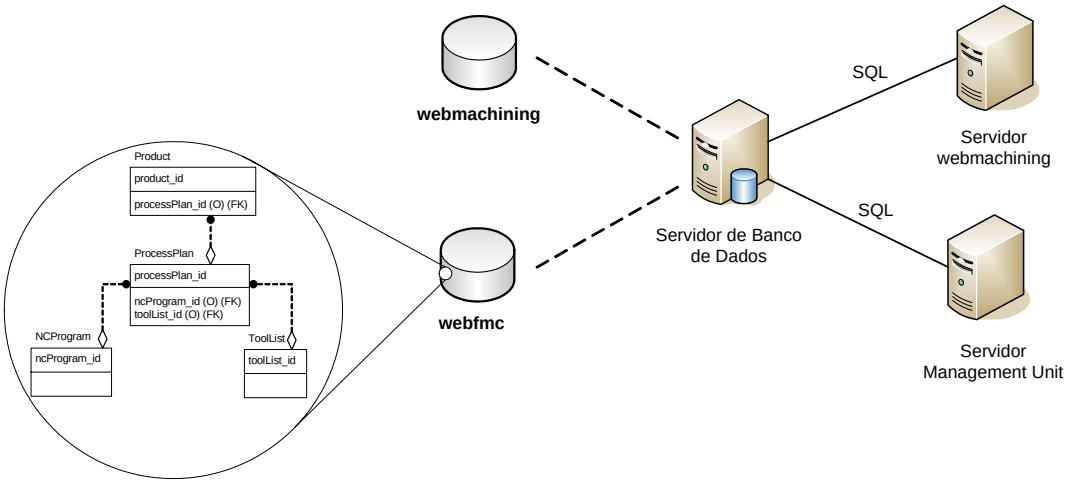

Figura 6.8 - Arquitetura de integração cliente/servidor – base de dados.

Ao finalizar o projeto e o plano de processo de um novo produto, as informações geradas pelo módulo CAPP do sistema CAD/CAPP/CAM devem ser replicadas na base de dados WebFMC. Consequentemente, ao gerar uma nova lista de tarefas, a Unidade de Gerenciamento estará carregando as informações oriundas do sistema CAD/CAPP/CAM de tal forma a garantir a integração e o fluxo de informações entre os diferentes níveis hierárquicos.

6.4 – SETUP DAS ESTAÇÕES DE TRABALHO

6.4.1 – Carregamento da lista de tarefas

Estabelecida a seqüência em que as ordens de trabalho serão enviadas para o chão–de–fábrica, o passo seguinte consiste em carregar a lista de tarefas. O carregamento da lista de tarefas requer a leitura de um arquivo com o registro da seqüência das ordens de trabalho, instanciar objetos *task* e armazenando-os em um objeto *taskList*. A figura 6.9 apresenta o algoritmo *LoadTaskList* utilizado no carregamento da lista de tarefas.

Algoritmo LoadTaskList
1.0 Solicita o carregamento da lista de tarefas
2.0 Lê o arquivo contendo a lista de tarefas
3.0 Para cada tarefa
3.1 Instancia um objeto task e seta seus atributos conforme a informação obtida do arquivo
3.2 Adiciona o objeto instanciado no passo 2.1 ao vetor listTask
4.0 Fim algoritmo

Figura 6.9 - Algoritmo *LoadListTask*.

As atividades iniciam com o operador solicitando o carregamento da lista de tarefas. Após esta solicitação, o subsistema *Dispatcher* instancia uma *JDialog* para que o operador possa informar ao sistema o nome do arquivo contendo uma descrição das ordens de trabalho e a seqüência em que as mesmas deverão ser carregadas.

Se for encontrado, o arquivo é lido e a seqüência das ordens de trabalho, bem como as respectivas informações do plano de processo (programa NC e lista de ferramentas) e do plano de inspeção (programa de inspeção) são carregadas. Estas informações são encapsuladas em um objeto *task* posteriormente adicionado a um objeto *taskList*. Estes passos são repetidos até que todos os objetos *task* tenham sido instanciados. Finalizado o carregamento da lista de tarefas, uma mensagem é enviada ao operador indicando o sucesso da operação.

6.4.2 – Verificando o *status* das estações de trabalho

As tarefas só poderão ser despachadas mediante uma prévia verificação do *status* das estações de trabalho. Esta verificação é necessária para garantir que as estações de trabalho estão aptas a receber a programação a elas destinada. A *GUI VerifyWorkstationStatus* (figura 6.10) foi implementada para facilitar a interação do operador na verificação dos *status* das estações de trabalho.

A MgU estabelece a comunicação com as estações de trabalho para verificar a condição normal de funcionamento. A medida que esta verificação acontece, um registro (*log*) é impresso (a direita na *GUI*) para que o operador possa acompanhar o andamento da verificação. Se por algum motivo houver alguma falha, o operador é impedido de efetuar a o carregamento das instruções nas estações de trabalho. A figura 6.11 apresenta a o algoritmo implementado para verificar o *status* das estações de trabalho.

VerifyWorkstationStatusCommand, um objeto *Command*, encapsula o algoritmo *VerifyWorkstationStatus* e retorna como resultado um objeto *WorkstationStatusData* que encapsula os *status* das estações de trabalho. Ao se invocar o método *execute()* de *VerifyWorkstationStatusCommand*, verifica-se quais serão as estações de trabalho alocadas, e para cada estação, um objeto *threadVerifyStatus* que *extends* uma *Thread* é instanciado. O objeto *threadVerifyStatus* sobrescreve o método *run()* e em seu corpo invoca o método *verifyWorkstationStatus()* do controlador da estação de trabalho a qual faz referência.

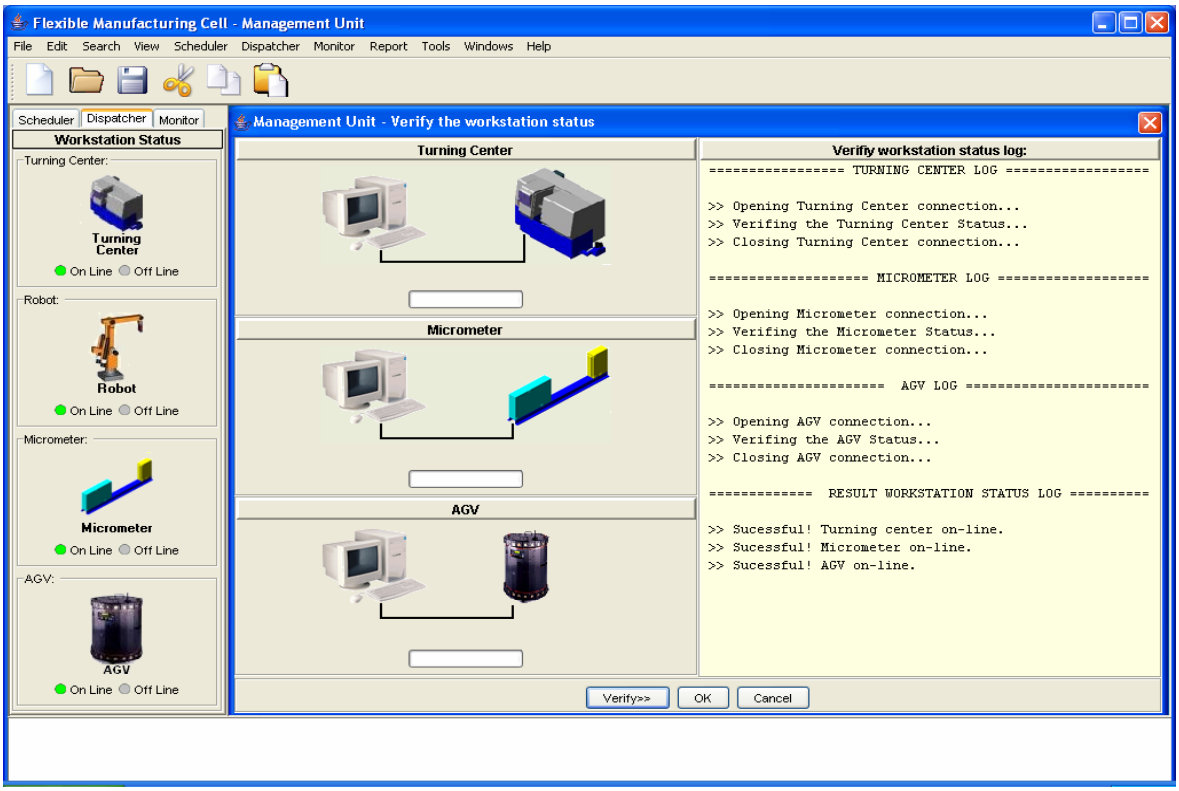

Figura 6.10 - *GUI VerifyWorkstationStatus*.

Algoritmo VerifyWorkstationStatus
1.0 Verifica quais serão as estações de trabalho utilizadas
2.0 Para cada estação de trabalho
2.1 Instancia um objeto threadVerifyStatus
2.2 Invoca o método start da threadVerifyStatus
3.0 Instancia um objeto TimeOut
4.0 Invoca o método start sobre o objeto instanciado no passo 3.0
5.0 Enquanto TimeOut não notificar
5.1 Verifica status das estações de trabalho
6.0 Fim enquanto
7.0 Retorna o status das estações de trabalho
8.0 Fim algoritmo

Figura 6.11 - Algoritmo *VerifyWorkstationStatus*.

Depois que todos os objetos *threadVerifyStatus* foram instanciados, um objeto *TimeOut* que *extends* um objeto *Timer* é instanciado. Este objeto dispara um evento *ActionListener()* todas as vezes que em que houver um *overflow*. Como *VerifyWorkstationStatusCommand* implementa a interface *ActionListener()*, consequentemente deve implementar o método *actionPerformed(ActionEvent actionEvent)*. No corpo deste método todas as referências que apontam para os objetos *threadVerifyStatus* recebem *null* sendo posteriormente invocado o método *stop()* sobre o objeto *TimeOut*.

6.4.3 – Despacho de instruções para as estações de trabalho

O despacho de instruções consiste no carregamento e verificação das instruções nas estações de trabalho, no intuito de programar as operações necessárias para processar a matéria prima. Nesta atividade, devem-se considerar as particularidades envolvidas, como o protocolo de comunicação, o formato das mensagens, dentre outras.

A figura 6.12 apresenta o algoritmo *TurningCenterDispatching*. Este algoritmo descreve passo a passo a lógica de carregamento de um programa NC no Centro de Torneamento. Após abrir a conexão, a MgU verifica se o programa a ser enviado já está cadastrado no diretório de programas do CNC através da função *searchNCProgram()*. Caso o programa já esteja cadastrado, uma mensagem é enviada ao operador verificando se o programa deve ser sobrescrito. Se o operador optar por sobrescrever o programa a função *uploadNCProgram()* envia o novo programa para o Centro de Torneamento.

Algoritmo TurningCenterDispatching

1.0 Abre conexão com o centro de torneamento
2.0 Verifica se o programa NC já está cadastrado
3.0 Se o programa NC estiver cadastrado
    3.1 Verifica se o operador deseja sobrescreve-lo
    3.2 Se o operador desejar sobrescrevê-lo
        3.2.1 Sobrescreve o programa
        3.2.2 Senão se o operador deseja utilizar o programa cadastrado
            3.2.2.1 Utiliza o programa cadastrado
4.0 Senão envia o programa NC
5.0 Verifica o programa NC
6.0 Se o programa NC estiver OK
    6.1 Efetua a ativação do programa
7.0 Senão envia uma mensagem ao operador notificando falha na operação
8.0 Fecha conexão com o centro de torneamento
9.0 Fim algoritmo

Figura 6.12 - Algoritmo *TurningCenterDispatching*.

Posteriormente, através da função *verifyNCProgram()*, a Unidade de Gerenciamento verifica se o programa NC é passível de ser executado. Caso seja possível executar programa, ele é carregado na memória do CNC do Centro de Torneamento através das funções *writePMC()* e *readPMC()*[3].

[3] As funções *readPMC()* e *writePMC()* são utilizadas para ler e escrever respectivamente nas variáveis de PMC do Centro de Torneamento. Isto significa que é possível emular o teclado do Centro de Torneamento (através do CNC via API FOCAS1) e assim efetuar o carregamento do programa NC na memória do CNC.

A figura 6.13 mostra o algoritmo *MicrometerDispatching*. A programação do micrômetro consiste no envio de instruções via RS232C no formato estabelecido pelo fabricante. O algoritmo de programação desenvolvido envia inicialmente um comando *LOCK* para garantir que o teclado da unidade de processamento não tenha uma entrada validada durante a programação *on-line*. Posteriormente, um comando *CLEAR* é enviado. Para cada seção a ser medida, os respectivos comandos são enviados para programar as informações geométricas (6.1) e os parâmetros de inspeção (6.2):

$$P0, SET, LL\ dd.dddd, LH\ ddd.dddd \tag{6.1}$$

$$SET, SG\ ssssss, M\ m, MN\ nnnn, REF \pm ddd.dddd, SCLc,\ PRT\ ttt \tag{6.2}$$

Em que:

Pp = Número do programa;
LL = Limite inferior;
LH = Limite superior;
SG = Segmentação;
Mn = Intervalo de medição;
REF = Referência;
SCL = Escala;
PRT = Tempo de periodicidade de impressão da resposta.

Finalizada a programação das informações geométricas e dos parâmetros de inspeção, um comando *UNLOCK* é enviado para garantir que o teclado possa então ser manualmente operado. Como último passo, a conexão com a unidade de processamento do micrômetro é fechada.

Algoritmo MicrometerDispatching

1.0 Abre conexão com o micrômetro
2.0 Trava o teclado da unidade de processamento
3.0 Para cada feature a ser inspecionada
  3.1 Programa as informações geométricas
  3.2 Programa os parâmetros de inspeção
4.0 Destrava teclado da unidade de processamento
5.0 Fecha conexão com o micrômetro
6.0 Fim algoritmo

Figura 6.13 - Algoritmo *MicrometerDispatching*.

O despacho de instruções para o AGV é realizado através do envido de uma lista de ferramentas concatenada na forma de uma *String*. Esta lista será pronunciada por um comando *N_speak()* no *NOMAD* solicitando ao operador a deposição das ferramentas necessárias para realizar o *setup* do Centro de Torneamento. Um servidor C foi desenvolvido para encapsular as funcionalidades de programação do AGV e permitir a comunicação com a MgU via *sockets* (ver figura 6.14). A figura 6.15 descreve os principais passos do algoritmo *AGVDispatching*.

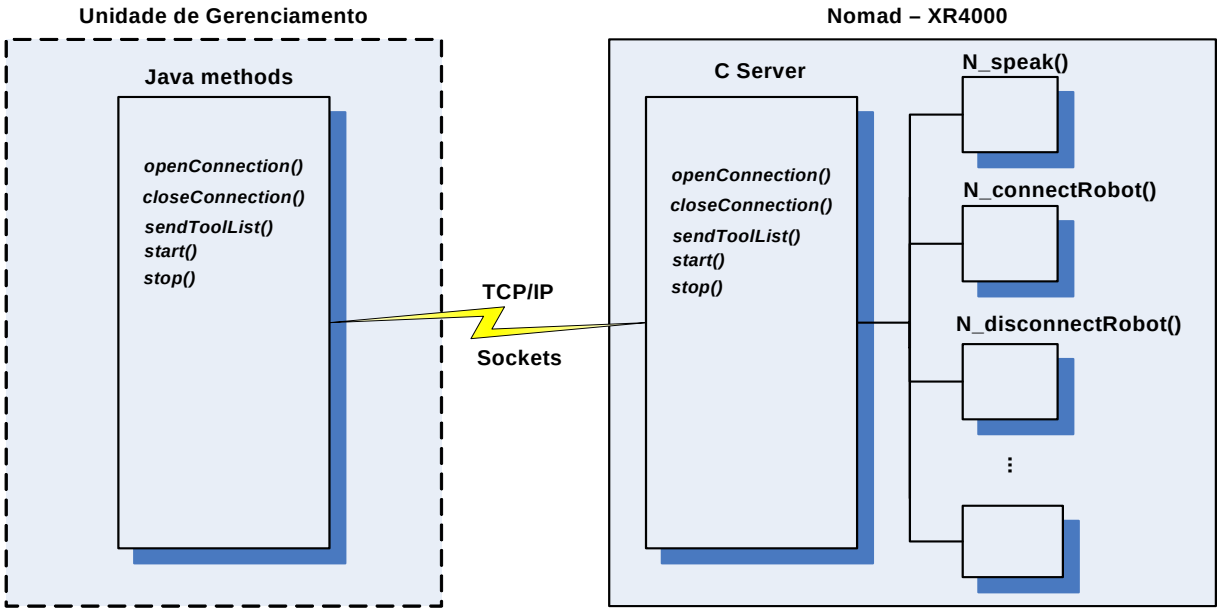

Figura 6.14 - Programação do *AGV*.

Algoritmo AGVDispatching
1.0 Abre conexão com o AGV
2.0 Envia string que encapsula lista de ferramentas
3.0 Fecha conexão com o AGV
4.0 Fim algoritmo

Figura 6.15 - Algoritmo *AGVDispatching*.

6.5 – O CONTROLADOR A EVENTOS DISCRETOS

O controlador a eventos discretos, incorporado ao módulo OperaçãoFMC do subsistema *Dispatcher*, tem como responsabilidade verificar a evolução dos eventos no chão-de-fábrica. Este controlador realiza o controle supervisório das estações de trabalho e identifica eventuais alterações na evolução dos eventos que poderiam levar as estações de trabalho a uma situação de *deadlock*. O algoritmo *DiscreteEventController* (figura 6.16) mostra os principais passos executados pelo controlador a eventos discretos.

A cada transição disparada, isto é, a cada evento de chão-de-fábrica identificado pelo módulo *EventMonitor* subsistema *Monitor*, o controlador recebe um objeto que encapsula os estados das estações de trabalho. A partir destas informações o controlador identifica a transição, verifica o estado posterior, verifica o estado posterior e compara com o obtido.

Algoritmo DiscreteEventController

2.0 Aguarda notificação
3.0 Recebe notificação de que o estado das estações de trabalho variou
3.1 Identifica transição
3.2 Verifica estado anterior
3.3 Verifica estado posterior
3.3 Compara estado posterior com o obtido
3.4 Se estado posterior confere com o obtido
3.4.1 Retorna para o passo 1.0
3.5 Senão
3.5.1 Executa ação de controle
3.6 Fim Senão
3.5 Fim Se
4.0 Fim algoritmo

Figura 6.16 - Algoritmo *DiscreteEventController*.

O resultado desta comparação influencia na ação de controle a ser tomada. Caso o estado posterior seja equivalente ao estado obtido, o controlador retorna ao estado de espera, até que receba uma nova notificação de que os estados das estações de trabalho foram alterados. Caso contrário, o controlador executa a ação de controle (notifica o operador da necessidade de intervenção humana ou solicita a atuação do controlador de operações).

Este controlador foi desenvolvido a partir da Rede de Petri detalhada da célula[4]. A implementação foi realizada baseada na abordagem não procedimental em que cada transição é considerada uma regra de transformação de estado (Cardoso *et at*, 1997). Consequentemente, as transições são associadas à eventos externos e os lugares aos estados alcançáveis das estações de trabalho.

6.6 – O CONTROLADOR DE OPERAÇÕES

O controlador de operações realiza o gerenciamento das operações utilizadas no funcionamento da célula. Este módulo tem como principal responsabilidade identificar a necessidade de re-programar as estações de trabalho (re-escalonamento), identificar a necessidade de efetuar um novo *setup* na célula (quando houver um erro de programação) ou até mesmo acionar o módulo de emergência da MgU.

[4] A Rede de Petri detalhada da célula está descrita no apêndice A.

O algoritmo *OperationController* (figura 6.17) mostra a estrutura do controlador de operações. A cada notificação recebida, o controlador de operações verifica a origem da notificação. Se a notificação vier do controlador a eventos discretos, o controlador de operações verifica o tipo de notificação recebida. Se a notificação estiver relacionada com a produção de uma nova peça (comparação com o sinal obtido do sensor da garra), este módulo verifica se as dimensões do tarugo correspondem ao da próxima tarefa a ser executada. Caso consista, o programa NC e o programa de inspeção são despachados para as respectivas estações de trabalho. Caso contrário, um sinal de erro de programação é enviado para o módulo EmergênciaFMC.

Algoritmo OperationController

1.0 Recebe notificação
2.0 Verifica a origem da notificação
2.1 Caso notificação veio do controlador a eventos discretos:
2.1.1 Verifica dimensões do tarugo
2.1.2 Verifica dimensões do tarugo da próxima tarefa
2.1.3 Se as dimensões corresponderem
2.1.3.1 Despacha programa NC
2.1.3.2 Despacha programa de inspeção
2.1.4 Senão
2.1.4.1 Envia sinal de erro de programação ao módulo EmergenciaFMC
3.0 Fim algoritmo

Figura 6.17 - Algoritmo *OperationController*.

6.7 – MONITORAMENTO DA PRODUÇÃO

6.7.1 – Monitor de eventos

O monitor a eventos discretos tem a responsabilidade de fazer uma varredura periódica nos controladores físicos das estações de trabalho e atualizar os estados dos controladores lógicos da Unidade de Gerenciamento. A figura 6.18 descreve o algoritmo implementado para realizar estas atividades.

Algoritmo EventMonitor

1.0 Inicializa a thread MonitorThreadController
2.0 Enquanto a produção não é finalizada verifica os controladores das estações de trabalho
2.1 Lê as tags do PMC do centro de torneamento
2.2 Verifica porta serial do micrômetro
3.0 Se ao menos um estado variou atualiza estados das estações de trabalho
3.1 Atualiza os estados do centro de torneamento
3.2 Atualiza os estados do robô
3.3 Atualiza os estados do micrômetro
4.0 Fim enquanto
5.0 Fim algoritmo

Figura 6.18 - Algoritmo *EventMonitor*.

Ao finalizar o despacho de instruções para as estações de trabalho, o método *start()* do objeto *MonitorThreadController* é invocado. Este objeto extends uma *Thread* e sobrescreve o método *run()*. No método *run()* uma estrutura *WHILE* com o argumento *TRUE* garante a execução do método indefinidamente. Ao se invocar o método *this.sleep(300)* dentro da estrutura *WHILE* anteriormente citada, a *thread MonitorThreadController* adormece em intervalos de 300 milisegundos.

Decorrido o tempo em que *MonitorThreadController* adormece, a *Thread* entra em estado de execução e faz uma varredura nos controladores físicos das estações de trabalho, isto é, no PMC do Centro de Torneamento (atualizando os estados do manipulador robótico e do Centro de Torneamento) e na unidade de processamento do micrômetro (atualizando os estados do micrômetro).

Em seguida, verifica-se se os estados de pelo menos uma das estações de trabalho variam, comparando-se os estados anteriores com os obtidos. Caso isto tenha ocorrido, os estados das estações de trabalho são atualizados com os valores obtidos da leitura feita nos controladores físicos.

6.7.2 – O monitor virtual

O *frame VirtualMonitor* (figura 6.20) desenvolvido proporciona ao operador o monitoramento virtual das estações de trabalho. Na guia monitor (a esquerda) são apresentadas as *tags* do PMC do Centro de Torneamento utilizadas na integração das estações de trabalho. No centro são apresentadas imagens virtuais das estações de trabalho operando. A direita um *JPanel* com o registro (*log*) dos eventos ocorridos no chão-de-fábrica. A figura 6.19 descreve o algoritmo implementado para o monitor virtual das estações de trabalho.

Algoritmo VirtualMonitor
1.0 Mostra o frame VirtualMonitor
2.0 Aguarda notificação
3.0 Recebe notificação de que o estado das estações de trabalho variou
3.1 Atualiza PMC diagnostic tags
3.2 Atualiza imagem virtual das estações de trabalho operando
3.3 Atualiza o registro (log) eventMonitor
4.0 Retorna para o passo 2.0
5.0 Fim algoritmo

Figura 6.19 - Algoritmo *VirtualMonitor*.

Decorrido o tempo em que *MonitorThreadController* adormece é verificado se as *tags* das estações de trabalho variaram. Caso tenham variado, um objeto *MonitorEvent* que *extends ObjectEvent*[5] é instanciado. *MonitorEvent* é passado como argumento ao método *fireMonitorEventListener()*. Este método envia uma mensagem *workstationStateChanged(MonitorEvent monitorEvent)* a todos os objetos que implementam a interface *MonitorEventListener*.

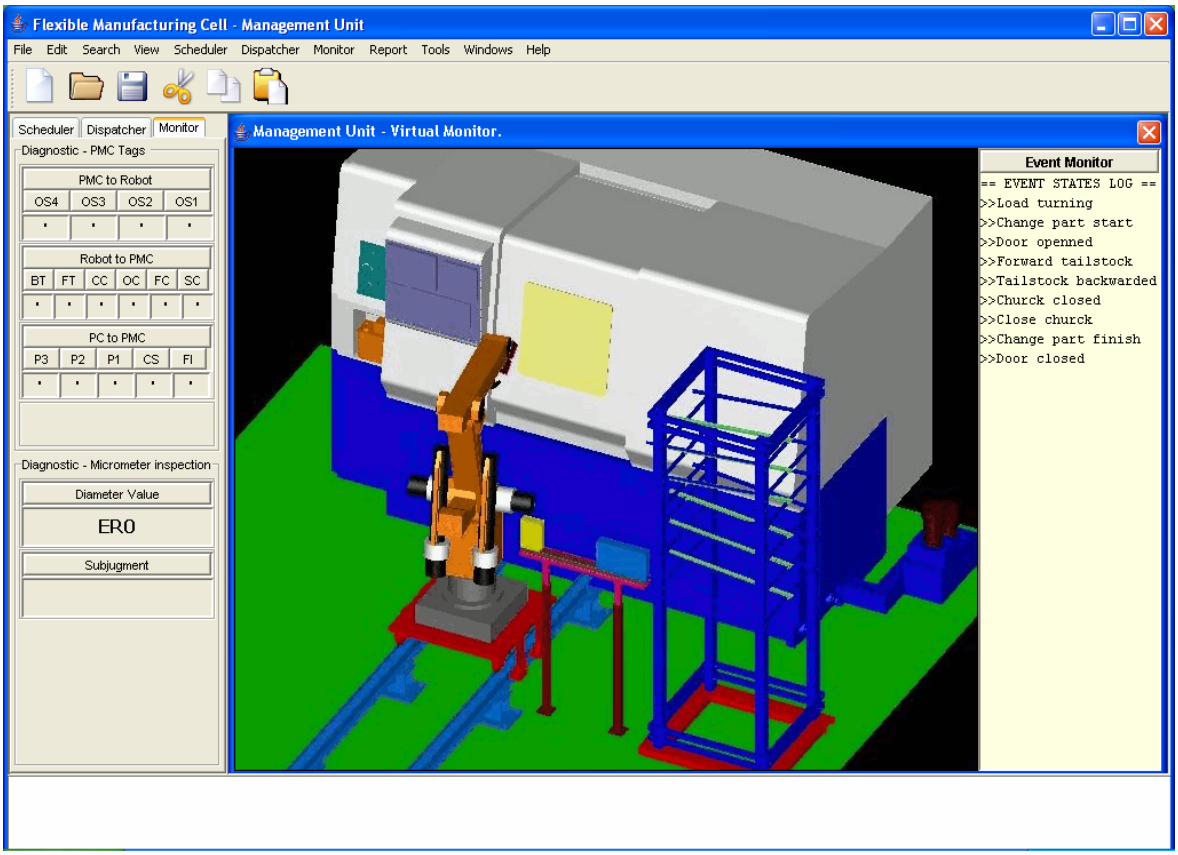

Figura 6.20 - *GUI Virtual Monitor*.

MonitorStatusPanel e *VirtualMonitorFrame* implementam a *MonitorEventListener*, em outras palavras, um contrato é estabelecido entre o objeto que disparou o evento *(MonitorThreadController*) e os que se declaram ouvintes de evento (*MonitorStatusPanel* e *VirtualMonitorFrame*). Contudo, todas as vezes que os estados das estações de trabalho variarem, *MonitorStatusPanel* e *VirtualMonitorFrame* recebem um objeto *MonitorEvent* para que os componentes visuais de monitoramento sejam atualizados.

[5] *EventObjects* são objetos imutáveis, portanto não podem ser modificados pelos ouvintes de evento. Isto é fundamental para que um ouvinte não modifique a forma como os demais irão processar o evento (Stanchfield, 2000).

6.7.3 – O monitor em tempo real

O monitoramento em tempo real consiste na apresentação de imagens de quatro câmeras estrategicamente instaladas na célula e na transmissão de áudio em tempo real via Internet. Este sistema de monitoramento foi herdado do módulo *WebTurning* do sistema *Webmachining* (Álvares, 2005). A arquitetura *WebTurning* é baseada na arquitetura cliente-servidor composta por três servidores e um cliente:

- ✓ Servidores de vídeo/audio *streaming* (*WebCam*);
- ✓ Servidor FOCAS1 (*Fanuc Open CNC API Specification*) localizado no Centro de Torneamento, sendo representado pelo CNC Fanuc 18i-ta;
- ✓ Servidor de teleoperação WebDNC que atua como uma camada intermediária entre o CNC e o cliente usando mecanismos de acesso via *Web*, como CGI e *inetd*;
- ✓ Interface gráfica (*GUI*) em Java e HTML.

Figura 6.21 - *GUI Real Time Monitor*.

Dos servidores disponibilizados por esta arquitetura, apenas o servidor *WebCam* é utilizado. Este servidor é responsável pela captura e transmissão de vídeo/áudio *streming* via Internet. As imagens são capturas a uma taxa de 30 *frames*/segundo para cada processador de imagem da

placa e apresentadas ao cliente através de uma interface *HTML/applet*. O servidor de vídeo está disponível na URL: http://video.graco.unb.br, enquanto que o servidor de áudio *stream* está disponível na URL: http://WebMachining.AlvaresTech.com:8000/audioturning.ogg. Para reproduzir o som capturado pelo servidor *WebCam* é necessário instalar um *plugin* para reproduzir arquivos no formado *Ogg Vorbis* (http://www.winamp.com).

6.7.4 – Controle da qualidade

O processo de inspeção inicia com o posicionamento da peça usinada na unidade de leitura do micrômetro. Estando a peça posicionada e decorrido o tempo de resposta programado (*DataOutputTimer*), o micrômetro escreve o resultado da leitura na interface RS232C. O comando de resposta apresenta o seguinte formato (Mitutoyo, 2003):

$$Pp,(GO/NG\ judment\ result)\pm ddd.ddd\ desviation \tag{6.3}$$

Em que:

Pp = Número do programa;
GO/NG = Julgamento GO/NG. Quando ativado resulta em –NG, GO ou +NG.
±ddd.ddd = Valor de desvio[6] (Valor medido – referência).

O número do programa é o registro das informações geométricas da seção que está sendo inspecionada (diâmetro e tolerância). Com o critério de julgamento GO/NG ativado, a unidade de processamento avalia o resultado da medição e decide se o mesmo está dentro dos limites de tolerância especificados pelo projetista. Os resultados podem ser –NG (medição menor que o limite inferior), G0 (dentro dos limites de tolerância) e +NG (medição maior que o limite superior).

A *GUI quality control* (figura 6.22) foi desenvolvida para que o operador possa acompanhar as inspeções realizadas. O método estatístico selecionado para controlar o processo é baseado no pré-controle. A aplicação este método é baseada em três etapas (qualificação do processo, operação e cálculo da freqüência de amostragem). As figuras 6.23, 6.24 e 6.25 apresentam os algoritmos deste método estatístico de controle da qualidade.

[6] Só aparece no comando de resposta quando a referência é setada.

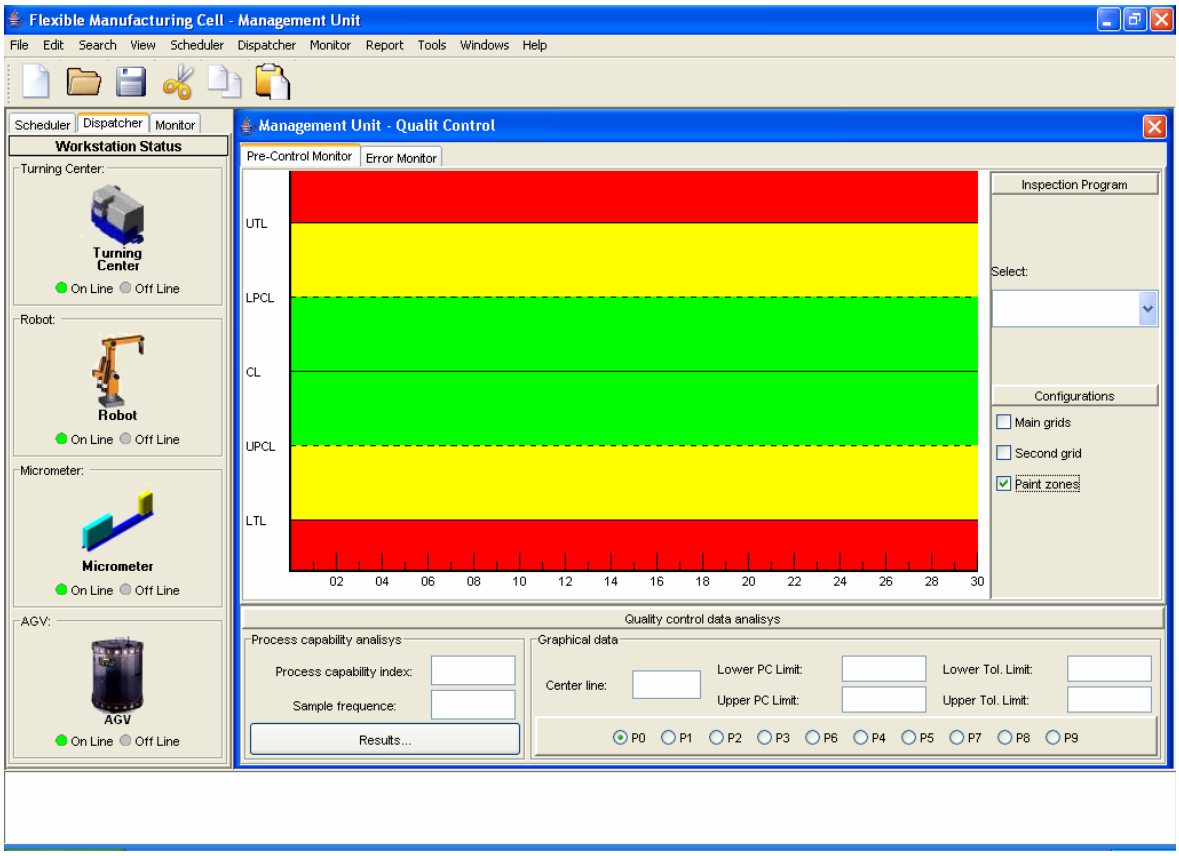

Figura 6.22 -*GUI Quality Control*.

Algoritmo QualifyQualityControl

1.0 Aguarda Notificação
2.0 Recebe notificacao
2.1 Se a leitura >=LowerPreControlLimit e leitura <=UpperPreControlLimit //Zona verde
2.1.1 Se numeroPecaZonaVerde<5
2.1.1.1 Incrementa numeroPecaZonaVerde
2.1.2 Senão
2.1.2.1 processoQualificado

2.2 Senão se LowerToleranceLimit<=leitura<=LowerPreControlLimit //Zona amarela
2.2.1 Se numeroPecaZonaAmarela>0
2.2.1.1 ajustarProcesso
2.2.2 Senão
2.2.2.1 Incrementa numeroPecaZonaAmarela

2.3 Senão se UpperPreControlLimit<=leitura<=UpperToleranceLimit //Zona amarela
2.3.1 Se numeroPecaZonaAmarela>0
2.3.1.1 ajustarProcesso
2.3.2 Senão
2.3.2.1 Incrementa numeroPecaZonaAmarela

2.4 Senão se leitura<=LowerToleranceLimit ou leitura>=UpperToleranceLimit //Zona vermelha
2.4.1 ajustarProcesso

3.0 Fim algoritmo

Figura 6.23 - Algoritmo *QualifyQualityControl*.

Algoritmo OperationQualityControl

1.0 Recebe a leitura do micrômetro
1.1 Se a leitura >=LowerPreControlLimit e leitura <=UpperPreControlLimit //Zona verde
1.1.1 pecaBoa

1.2 Senão se LowerToleranceLimit<=leitura<=LowerPreControlLimit //Zona amarela
1.2.1 Se LowerToleranceLimit<=leituraAnterior<=LowerPreControlLimit //Zona amarela mesmo lado
1.2.1.1 pecaBoa
1.2.2 Senão //Lado oposto
1.2.2.1 ajustarProcesso

1.3 Senão se UpperPreControlLimit<=leitura<=UpperToleranceLimit //Zona amarela
1.3.1 Se UpperPreControlLimit<=leituraAnterior<=UpperToleranceLimit //Zona amarela mesmo lado
1.3.1.1 pecaBoa
1.3.2 Senão //Lado oposto
1.3.2.1 ajustarProcesso

1.4 Senão se leitura<=LowerToleranceLimit ou leitura>=UpperToleranceLimit //Zona vermelha
1.4.1 ajustarProcesso

2.0 Fim algoritmo

Figura 6.24 - Algoritmo *OperationQualityControl*.

Algoritmo SampleFrequencyQualityControl

1.0 Registra a data e a hora da primeira amostra
2.0 Recebe notificacao de ajuste do processo
3.0 Registra a data e a hora da notificação
4.0 Calcula a frequência de amostragem
5.0 Fim algoritmo

Figura 6.25 - Algoritmo *SampleFrequencyQualityControl*.

6.8 – EMERGÊNCIA

O tratamento de condições adversas é feito por um módulo de emergência integrado ao subsistema *Dispatcher*. Especificamente, duas são as condições que a Unidade de Gerenciamento é capaz de tratar: o re-escalonamento e a parada de emergência da FMC. A figura 6.26 mostra o algoritmo *Emergence* projetado para atuar sob condições adversas.

Algoritmo Emergence

1.0 Recebe notificação de emergência
2.0 Identifica as dimensões do tarugo
3.0 Lê a próxima task do vetor listTask
3.1 Se as dimensões correspondem a do taguro task
3.1.1 Notifica o controlador de operações
3.2 Senão
3.2.1 Se for a ultima task do vetor listTask
3.2.1.1 Notifica intervenção humana
3.2.2 Senão
3.2.2.1 Retorna para o passo 3.0
4.0 Fim algoritmo

Figura 6.26 - Algoritmo *Emergence*.

O re-escalonamento das tarefas é realizado quando as dimensões do tarugo não conferem com as especificadas no plano de processo. Neste caso, módulo de emergência pesquisa na lista de tarefas escalonadas (respeitando a ordem de prioridade) qual tarefa as dimensões do tarugo correspondem com a identificada pelo sensor da garra.

Caso esta tarefa seja encontrada, as estações de trabalho são re-programadas e uma mensagem é enviada ao subsistema *Scheduler* solicitando o ajuste da programação. Por outro lado, se não for encontrada uma tarefa cujas especificações competem com as obtidas pelo sensor da garra, uma mensagem é enviada ao operador notificando a necessidade de intervenção humana.

A segunda condição está relacionada com a parada de emergência da FMC. Ao se pressionar o botão de emergência do Centro de Torneamento ou do manipulador robótico, uma mensagem é enviada imediatamente a Unidade de Gerenciamento notificando a parada de emergência. Nesta situação, as atividades são interrompidas e uma mensagem é enviada ao operador notificando a necessidade de intervenção humana.

7 – ESTUDO DE CASO

Neste capítulo é apresentado um estudo de caso para demonstrar o ciclo de fabricação de duas famílias de peças utilizando os recursos disponibilizados pela FMC. São apresentadas todas as fases de fabricação, desde o projeto das peças (realizada pelo sistema CAD/CAPP/CAM), a programação da produção ao monitoramento das estações de trabalho.

7.1 – INTRODUÇÃO

Neste estudo de caso é considerada a fabricação de cinco peças agrupadas em três ordens de trabalho. Cada ordem de trabalho está associada a um produto, e cada produto a um plano de processo. As tabelas 7.1, 7.2 e 7.3 apresentam respectivamente o plano de produção simplificado, as informações geométricas e os parâmetros de inspeção do plano de inspeção.

Tabela 7.1 - Plano de produção (simplificado).

Nº	*Ordem de trabalho*	*Produto*	*Prioridade*	*Quantidade*	*Data de entrega*	*Observação*
01	W001	Produto 1	0	2	4/12/06 14:00	Material: Tecnil
02	W002	Produto 2	5	2	6/12/06 10:00	Material: Tecnil
03	W003	Produto 3	7	1	7/12/06 16:00	Material: Tecnil

Tabela 7.2 - Plano de inspeção – Informações geométricas.

Nº	*Programa de inspeção*	*Seção*	*Diâmetro*	*Tolerência Superior*	*Tolerância inferior*
01	Insp01	I	46 mm	+0.01	-0.01
02	Insp02	II	60 mm	+0.01	-0.01
03	Insp03	I	46 mm	+0.01	-0.01

Tabela 7.3 - Plano de inspeção – Parâmetros de inspeção.

Nº	*Programa de inspeção*	*Segmentação*	*Valor da segmentação*	*Intervalo de medição*	*Valor do intervalo*	Scan	*Rerefência*	*Escala*	*Sistema de medida*	*Gravar dado na memória*	*Data Output Time*
01	Insp01	SEGMENT	4	M	4	4	0	1	MM	YES	3
02	Insp02	SEGMENT	4	M	4	4	0	1	MM	YES	3
03	Insp03	SEGMENT	4	M	4	4	0	1	MM	YES	3

7.2 – DESCRIÇÃO DO ESTUDO DE CASO

7.2.1 – Modelagem dos produtos e elaboração do plano de processo

As figuras 7.1, 7.2 e 7.3 apresentam respectivamente o modelo geométrico dos produtos elaborado pelo módulo *WebCadByFeatures* do sistema *Webmachining*. As tabelas 7.4, 7.5 e 7.6 mostram respectivamente o código G obtido a partir do plano de processo gerado pelo módulo *WebCAPP* do sistema *Webmachining*. Algumas funções miscelâneas (como M36 – abre porta, M37 – fecha porta, etc) foram suprimidas do código G pelo fato destas funções já estarem encapsuladas no ciclo de carregamento (M200) e descarregamento (M201). As funções relacionadas ao segundo *setup* (ciclo de faceamento) também foram retiradas já que os tarugos foram previamente faceados.

As funções miscelâneas destacadas (M200 e M201) foram acrescentadas manualmente para declarar respectivamente o ciclo de carregamento e descarregamento do Centro de Torneamento. A tabela 7.7 apresenta uma descrição das ferramentas que serão utilizadas na fabricação das peças.

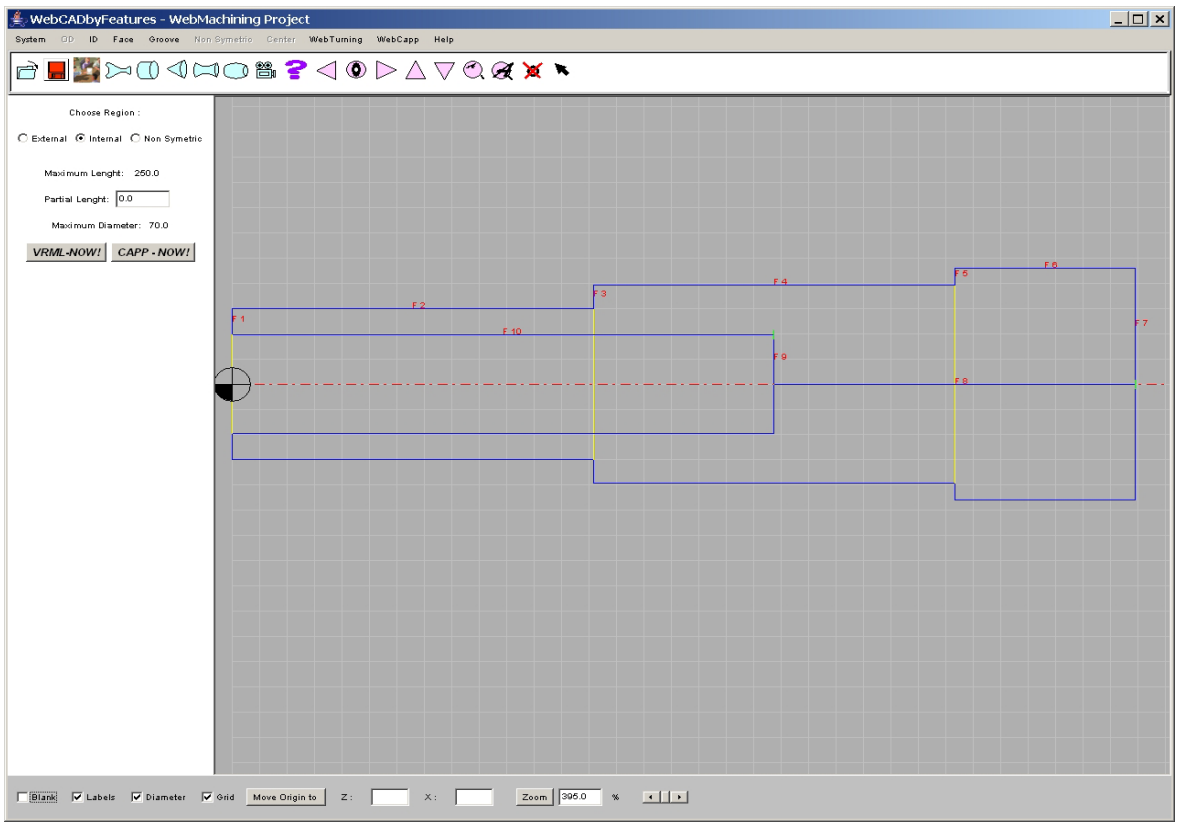

Figura 7.1 – Modelo geométrico do produto 1.

Tabela 7.4 - Código G gerado pelo WebCAPP (produto 1).

O1002(PRODUTO1)	N340G0X46
N10M200 (CICLO DE CARGA)	N350G42
N20G21G40G90G95	N360G1W-100
N30(INICIO SETUP1)	N370X60
N40G53	N380W-100
N50G0X390Z350T00	N390X70
N60T0404(T FURACAO)	N400W-50
N70G54	N410X72
N80G90	N420G70P370Q450
N90G97S800M15	N430G53G40
N100(O - CICLO FIXO G74 FURACAO)	N440G0X390Z350T00
N110X0Z262	N450T0909(T DESBASTE GERAL INTERNO)
N120X0Z257	N460G54
N130G74R2	N470G96S300
N140G74Z104Q15000F0.15	N480G92S2000
N150(WORKINGSTEP - FEATURE DE USINAGEM F2)	N490M4
N160G53	N500(O - CICLO FIXO G71-TIPO1 DESBASTE E G70 ACABAMENTO)
N170G0X390Z350T00	N510G0X18Z262
N180T0707(T DESBASTE GERAL EXTERNO)	N520G0X28Z257
N190G54	N530G71U1R2
N200G96S300	N540G71P610Q650U-0.2W0F0.2
N210G92S2000	N550G0X30
N220M4	N560G41
N230(O - CICLO FIXO G75 FACEAMENTO)	N570G1W-5
N240G0X82Z264	N580G1W-150
N250G0X77Z253.996	N590X28
N260G75X23Z250P800000Q2000R1F0.2	N600G70P610Q650
N270(WORKINGSTEP - FEATURE DE TORNEAMENTO F3)	N610(FIM SETUP1)
N280G0X72	N620G53
N290(O - CICLO FIXO G71-TIPO1 DESBASTE E G70 ACABAMENTO)	N630G0X390.0Z350.0T00
N300G0X82Z262	N640M00
N310G0X72Z257	N650M201(CICLO DE DESCARGA)
N320G71U1R2	N660M30 (FIM DA PECA)
N330G71P370Q450U0.2W0F0.2	

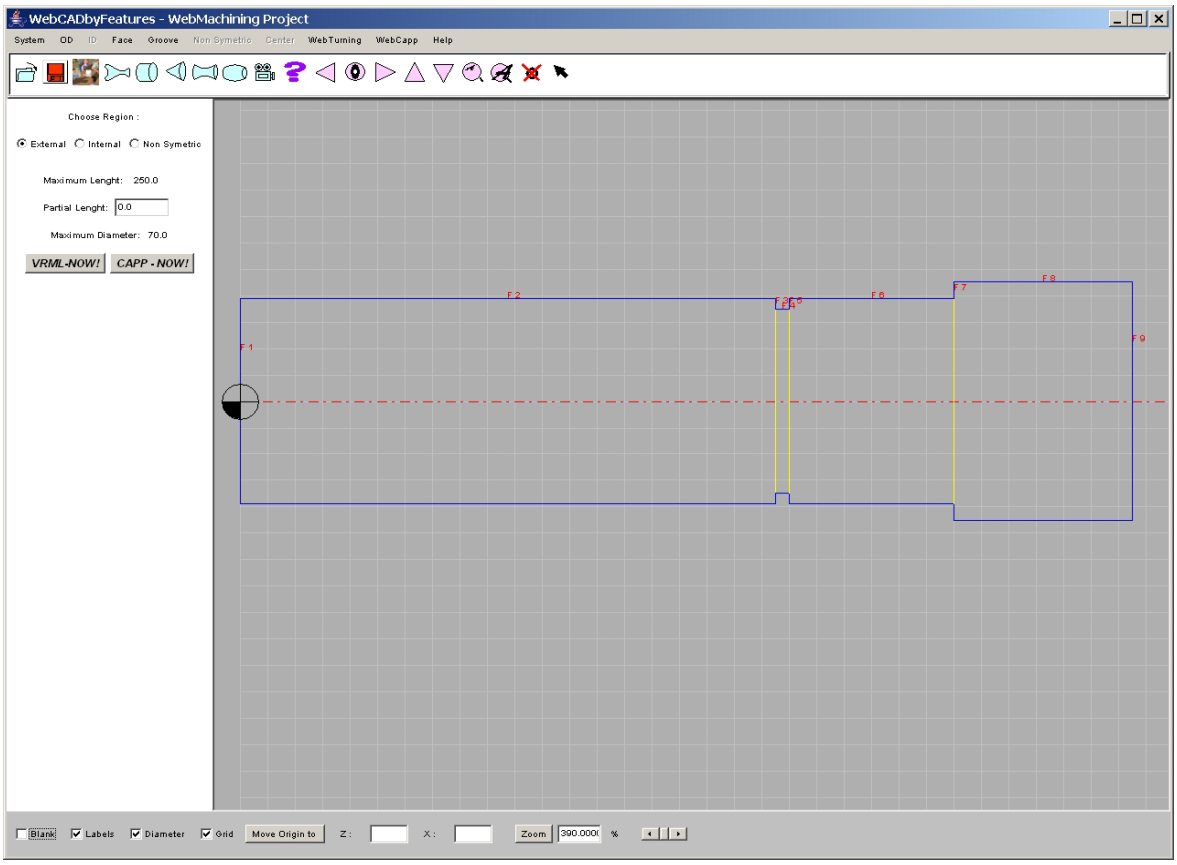

Figura 7.2 - Modelo geométrico do produto 2.

Tabela 7.5 - Código G gerado pelo WebCAPP (produto 2).

O1003(PRODUTO2)	N280X72
N20M200 (CICLO DE CARGA)	N290G70P250Q330
N30G21G40G90G95	N300(WORKINGSTEP - FEATURE DE TORNEAMENTO F3)
N40G53	N310G53G40
N50G0X430Z440T00	N320G0X430Z440T00
N60T0707(T DESBASTE GERAL EXTERNO)	N330T0303(T SANGRAMENTO EXTERNO)
N70G54	N340G54
N80G96S300	N350G96S300
N90G92S2000	N360G92S2000
N100M4	N370M4
N110M8	N380(O - CICLO FIXO G72 GROOVE COMPLEXO E G70 ACABAMENTO)
N120G0Z264	N390G0Z112
N130(WORKINGSTEP - FEATURE DE TORNEAMENTO F2)	N400G0X70Z102
N140G0X85	N410X65
N150(O - CICLO FIXO G71-TIPO1 DESBASTE E G70 ACABAMENTO)	N420G72W0.6R0
N160G0X82Z262	N430G72P490Q540U0W0F0.2
N170G0X72Z257	N440G0Z98
N180G71U1R2	N450G41
N190G71P250Q330U0.2W0F0.2	N460G1X60
N200G0X60	N470G1X54
N210G42	N480W4
N220G1W-5	N490X60
N230G1W-150	N500G70P490Q540
N240X60W-4	N510G53M5
N250W-46	N520G0X430Z440T00
N260X70	N530M201(CICLO DE DESCARGA)
N270W-5	N540M30(FIM DA PECA)

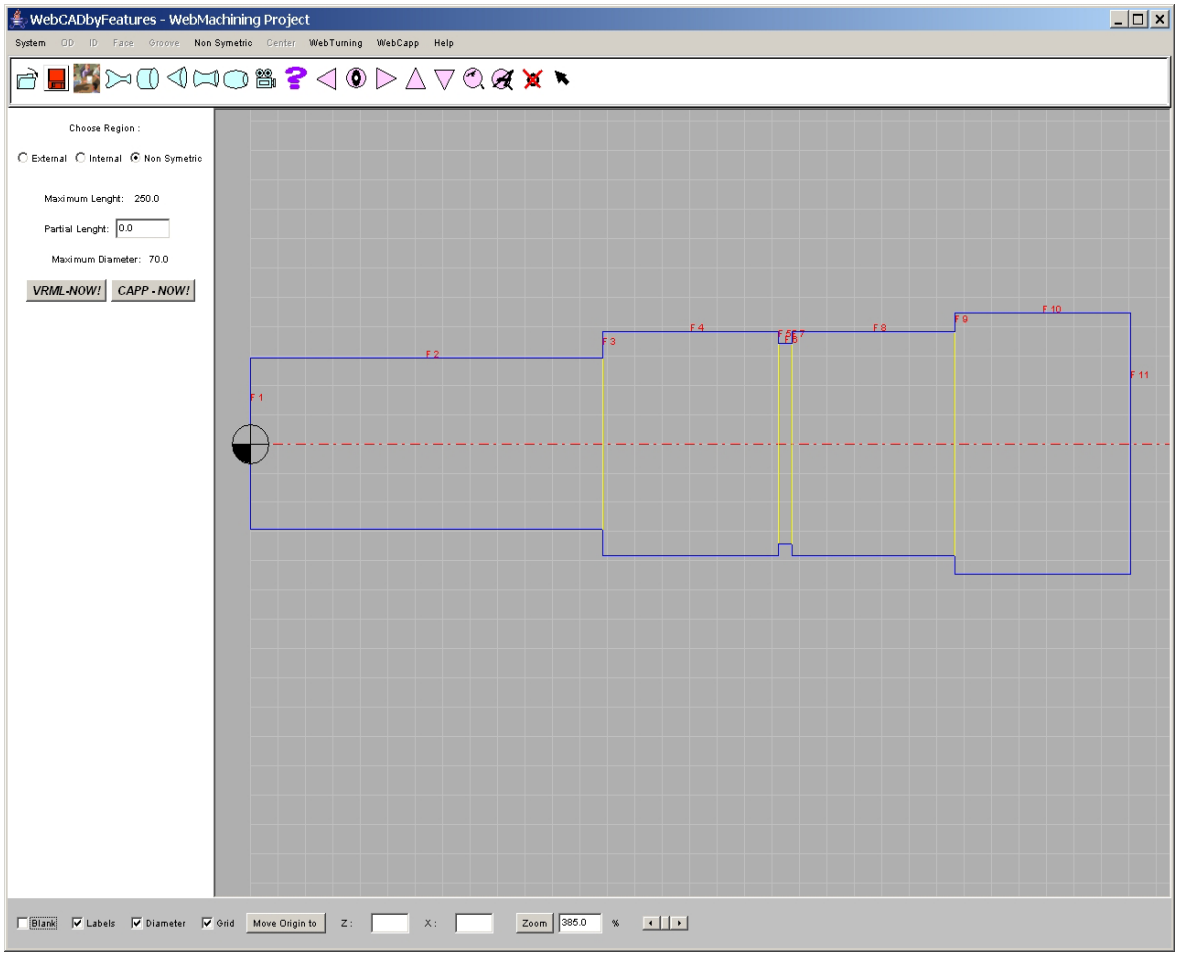

Figura 7.3 - Modelo geométrico do produto 3.

Tabela 7.6 - Código G gerado pelo WebCAPP (produto 3).

O10004 (PRODUTO3)	N410G54
N10M200(CICLO DE CARGA)	N420G96S300
N20G21G40G90G95	N390G92S2000
N30(INICIO SETUP1)	N400M4
N40(WORKINGSTEP - FEATURE DE USINAGEM F1)	N410(O - CICLO FIXO G72 GROOVE COMPLEXO E G70 ACABAMENTO)
N50G53	N420G0X70Z112
N60G0X390Z350T00	N430G0X65Z102
N70T0707(T DESBASTE GERAL EXTERNO)	N440G72W0.6R0
N80G54	N450G72P510Q560U0W0F0.2
N90G96S300	N460G0Z98
N100G92S2000	N470G41
N110M4	N480G1X60
N120(O - CICLO FIXO G75 FACEAMENTO)	N490G1X54
N130G0X82Z264	N500W4
N140G0X77Z253.996	N510X60
N150G75X0Z250P800000Q2000R1F0.2	N520G70P510Q560
N160(WORKINGSTEP - FEATURE DE TORNEAMENTO F2)	N530G53G40
N170G0X72	N540G0X390Z350T00
N180(O - CICLO FIXO G71-TIPO1 DESBASTE E G70 ACABAMENTO)	N550T1010(T FRESAMENTO EXTERNO)
N190G0X82Z262	N560G54
N200G0X72Z257	N570G96S300
N210G71U1R2	N580G92S2000
N220G71P250Q350U0.2W0F0.2	N590M4
N230G0X46	N600(O - CICLO MANUAL RASGO DE CHAVETA)
N240G42	N610G97S800M15
N250G1W-5	N620M19
N260G1W-100	N630G0X70Z162
N270X60	N640G28C0
N280W-50	N650G0X60Z157C0
N290W-4	N660G01Z127F0.16
N300W-46	N670G0Z157
N310X70	N680M17
N320W-50	N690(FIM SETUP1)
N330X72	N700G53
N340G70P250Q350	N710G0X390Z350T00
N350(WORKINGSTEP - FEATURE DE TORNEAMENTO F3)	N720M00
N360G53G40	N730M201 (CICLO DE DESCARGA)
N370G0X390Z350T00	N740M30(FIM DA PECA)
N380T0303(T SANGRAMENTO EXTERNO)	

Tabela 7.7 - Ferramentas *Sandvik* utilizada na fabricação das peças (fonte: Álvares, 2005).

nº	Porta-ferramenta	Inserto	Operação
01	LF123g20-2020B	N123G200300003-GM4025	sangrar (circular - 4 mm)
02	R416.2-0200C 3-31	LCMX030308-53 1020	furar (20 mm)
03	SVJBL-2020K-16	VBMT1604 08-MM2025	tornear externo
04	A16R-SDUPL 07-R	DPMT070204-PM4015	tornear interno
05	fresa - acionada	aço rápido	fresar (12 mm)

7.2.2 – Programação da inspeção

Estabelecido o plano de processo de cada produto, o passo seguinte consiste em cadastrar os programas de inspeção definidos na tabela 7.2 e 7.3. Os parâmetros de inspeção são exatamente os mesmos para todos os programas. Somente uma seção transversal de cada peça será inspecionada. A *GUI InspectionPlan* (figura 7.4) mostra os programas de inspeção cadastrados para realizar as inspeções das peças.

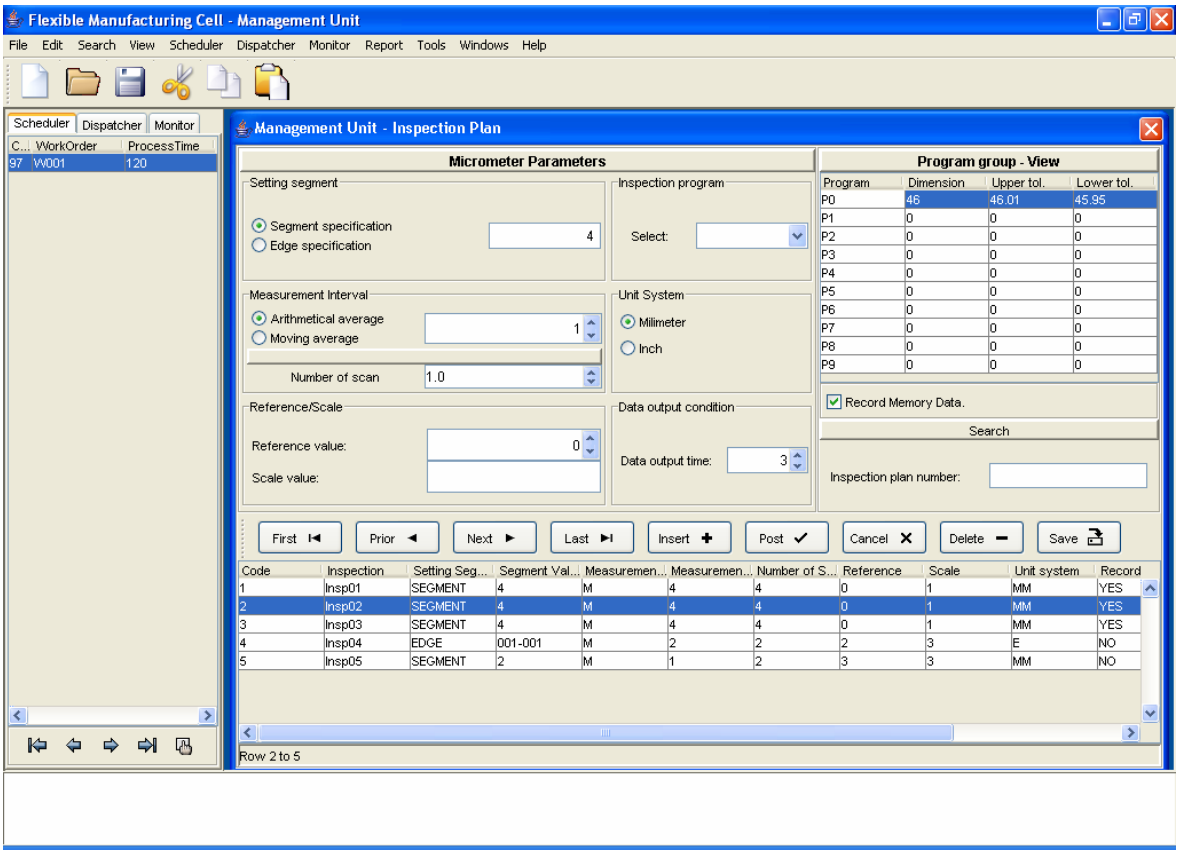

Figura 7.4 - Cadastramento dos programas de inspeção.

7.2.3 – Programação da produção

A primeira fase da programação da produção consiste no cadastramento das ordens de trabalho. Cada ordem está associada a um produto e a um programa de inspeção. Informações adicionais (como data de início, data de cadastro, etc) são acrescentadas para serem utilizadas pelos demais módulos da MgU. A figura 7.5 mostra a *GUI MasterPlanningScheduling* utilizada para efetuar o cadastramento das ordens de trabalho.

Finalizado o cadastramento das ordens de trabalho, o atributo *STATUS* de cada ordem é setado como *IN PRODUCTION*. Ao setar este atributo como *IN PRODUCTION*, a Unidade de Gerenciamento entende que a ordem foi selecionada para ser enviada para o chão-de-fábrica.

Definida as ordens de trabalho que serão processadas, o passo seguinte consiste em agrupá-las em famílias de peças. Cada família possui um conjunto de ordens de trabalho e sua respectiva lista de ferramentas. A lista de ferramentas, assim como o código G, foi obtida a partir do plano de processo gerado pelo módulo *WebCAPP*. A tabela 7.8 apresenta o resultado obtido a partir de dois ciclos de interação do algoritmo *PartFamily* (tabela 6.1). A Família 1 é composta pela

ordem de trabalho W001 e pela *ToolList1* (ferramentas 02, 03 e 04 – tabela 7.7), enquanto a Família 2 é composta pelas ordens W002 e W003 e pela *ToolList2* (ferramentas 01, 03 e 05[1] - tabela 7.7).

Tabela 7.8 – Agrupamento das ordens de trabalho em famílias.

Família 1									
Nº	*Ordem de trabalho*	*Produto*	*Prioridade*	*Quantidade*	*Data de início*	*Data de entrega*	*Programa de inspeção*	*Lista de ferramentas*	*Observação*
01	W001	Produto 1	0	2	4/12/06 8:00	4/12/06 14:00	Insp01	ToolList1	Material: Tecnil
Família 2									
02	W002	Produto 2	5	2	4/12/06 8:07	6/12/06 10:00	Insp02	ToolList2	Material: Tecnil
03	W003	Produto 3	7	1	4/12/06 8:00	7/12/06 16:00	Insp03	ToolList2	Material: Tecnil

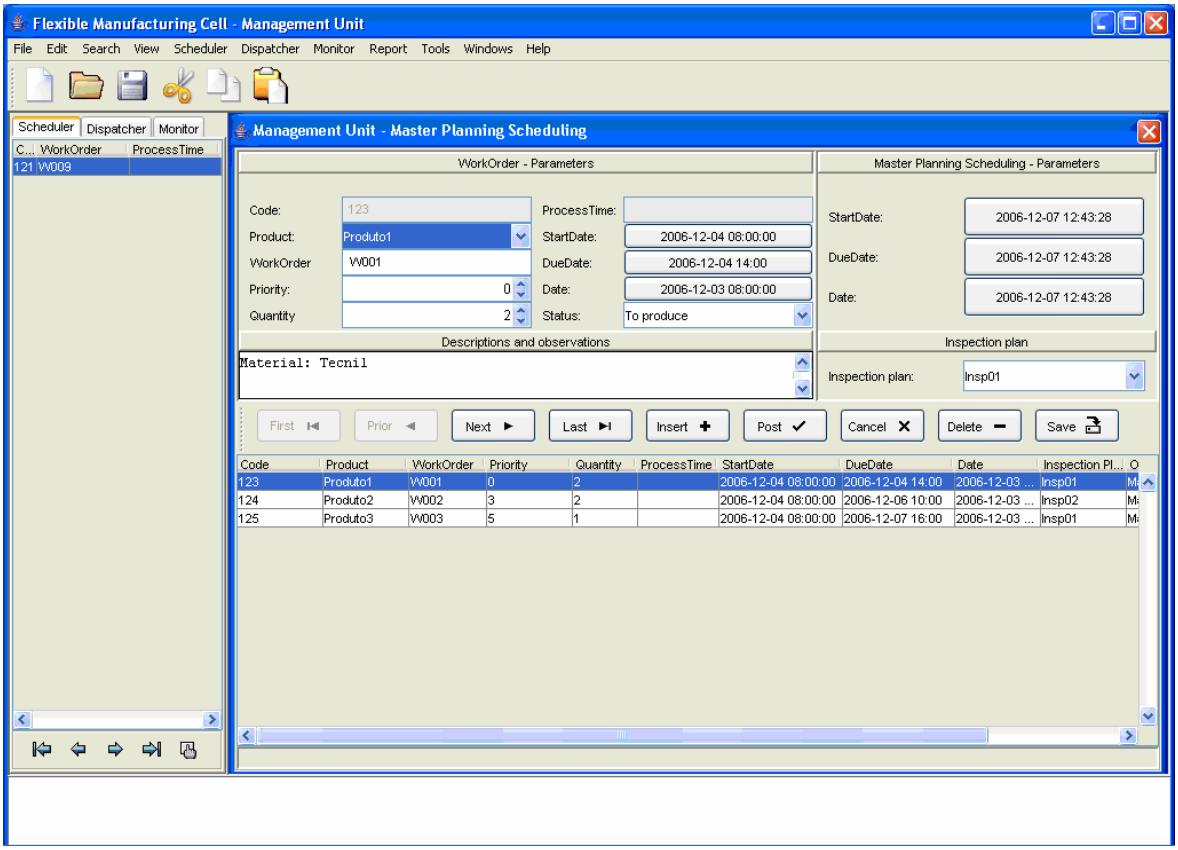

Figura 7.5 – Cadastramento das ordens de trabalho.

A última fase da programação da produção consiste em estabelecer a seqüência em que as famílias e suas respectivas ordens de trabalho serão processadas. Para este estudo de caso, foi adotada como critério de sequenciamento a regra *EarliestDueDate*. Desse modo, as ordens de trabalho que possuem a data de entrega mais próxima terão prioridade sobre as demais. A figura

[1] O Centro de Torneamento possui uma torre contendo 12 porta-ferramentas. Desse modo, caso todas as ferramentas estejam disponíveis, é possível agrupar as peças e as ferramentas que serão utilizadas em uma única família de peças e lista de ferramentas respectivamente.

7.6 ilustra a sequência das famílias de peças e suas respectivas ordens de trabalho apresentada em um gráfico de Gantt.

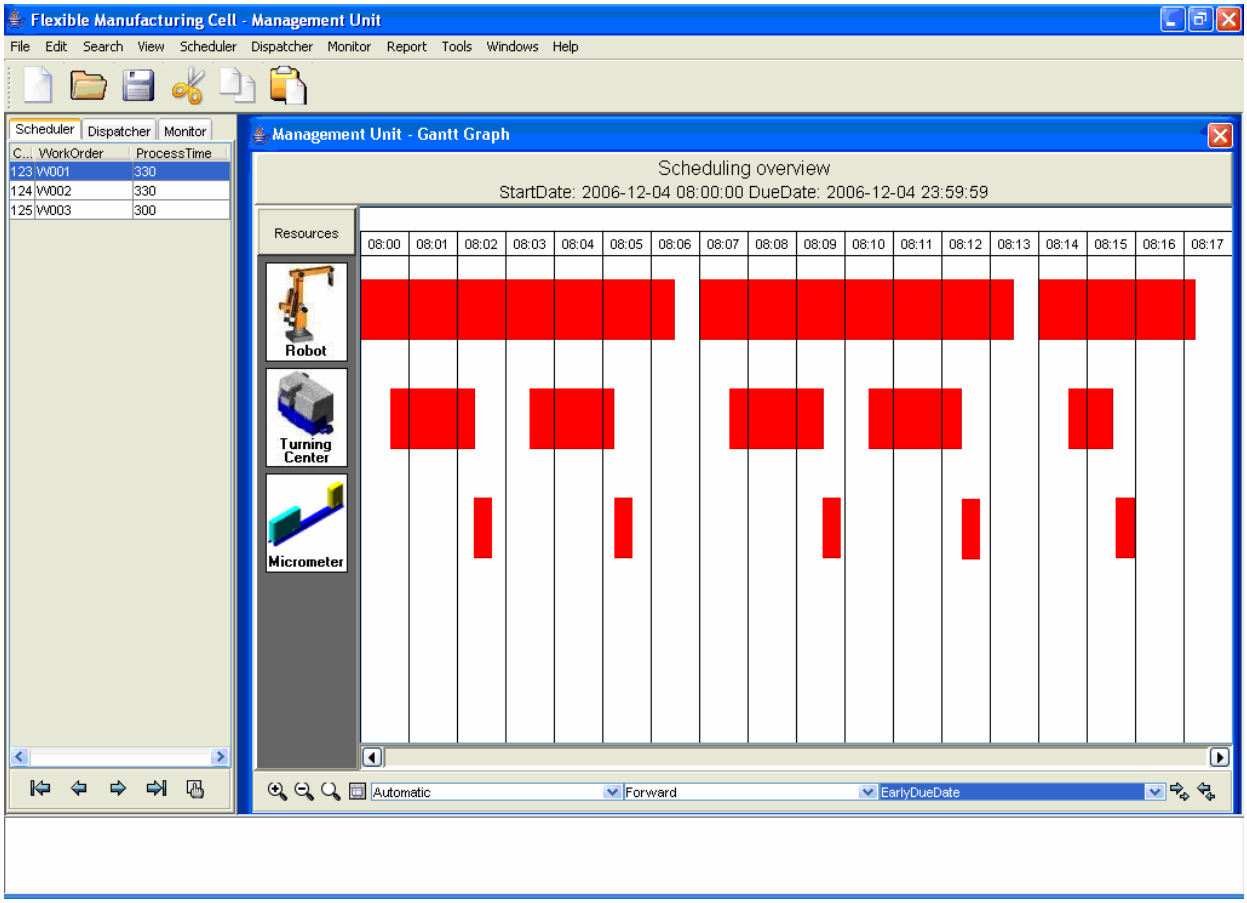

Figura 7.6 - Escalonamento das famílias de peças e suas respectivas ordens de trabalho.

Cada ordem de trabalho contém um conjunto de operações necessárias para a fabricação do produto. Para obter o tempo real de cada operação foi necessário executar um *try-out* na FMC para cada produto a ser produzido. A partir dos valores obtidos com o *try-out* foram setadas as variáveis *operationTime* de cada operação para que os valores de tempo apresentados no gráfico de Gantt reflitam os tempos reais necessários para executar as operações.

7.2.4 – Setup das estações de trabalho

A primeira fase do *setup* das estações de trabalho consiste no carregamento da lista de tarefas. Após efetuar o carregamento da lista de tarefas, o operador deve verificar o *status* das estações de trabalho. Através do menu *DISPATCHER – WORKSTATIONGROUP – GROUP2* o operador seleciona as estações de trabalho que serão utilizadas na fabricação das peças.

O *WorkstationGroup* selecionado neste estudo de caso é o *Group2*. Consequentemente a MgU deve verificar o status do Centro de Torneamento, do manipulador robótico e do micrômetro. Como não existe a possibilidade de comunicação direta com o manipulador robótico, considera-se o *status* desta estação de trabalho como *ON-LINE* somente se o Centro de Torneamento também estiver. A figura 7.7 mostra a *GUI VerifyWorkstationStatus* utilizada para verificar o *status* das estações de trabalho.

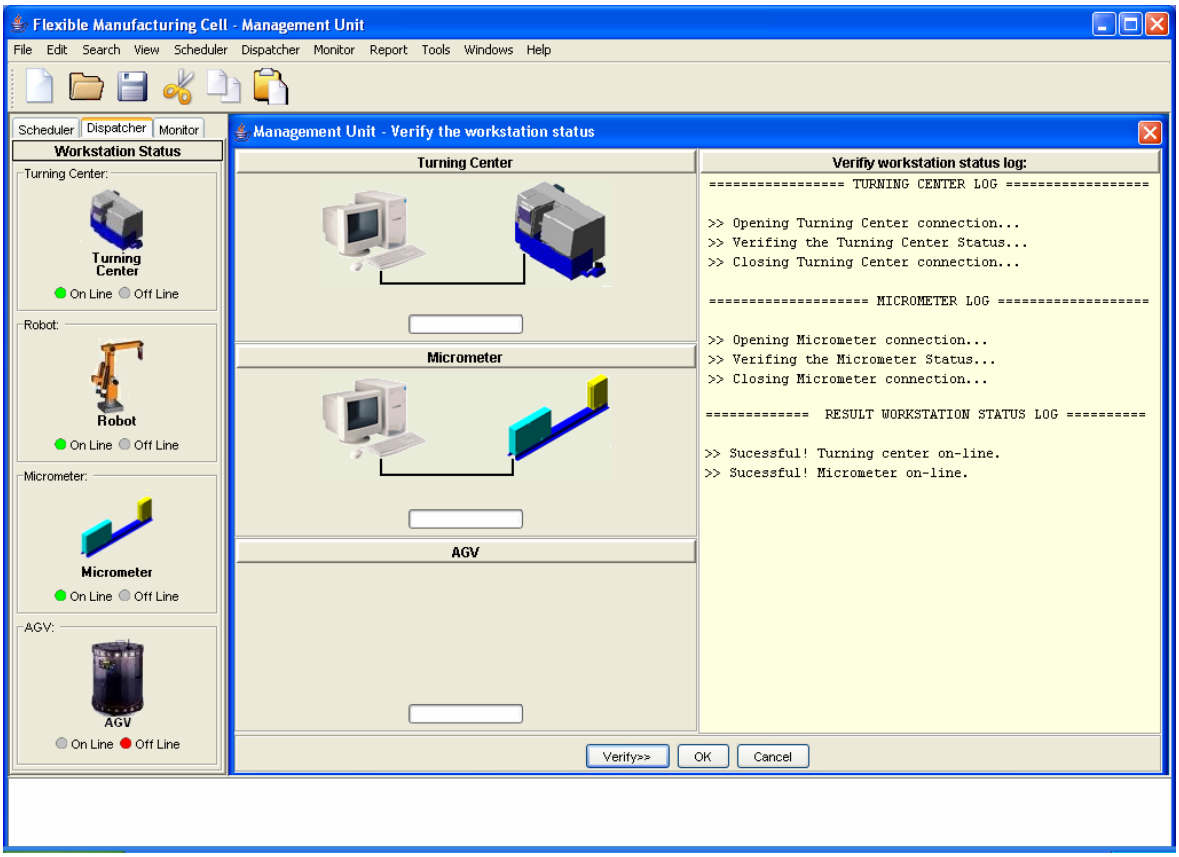

Figura 7.7 - Verificando o status das estações de trabalho.

Após verificar o *status* das estações de trabalho, a Unidade de Gerenciamento deve efetuar o carregamento de instruções nas estações de trabalho. O carregamento de instruções está parcialmente implementado, já que para se tornar completamente operacional, depende dos resultados obtidos com o algoritmo *LoadTaskLis*[2]*t* (tabela 6.3).

[2] Este algoritmo está parcialmente implementado.

7.2.5 – Programação da unidade de manipulação e transporte

A tabela 7.9 apresenta em detalhes a programação do manipulador robótico. Embora a integração física e lógica ofereça suporte para a troca de sinais entre a unidade de manipulação e transporte e o Centro de Torneamento, durante a realização deste estudo de caso, o Centro de Torneamento não estava enviando sinais para o manipulador robótico[3].

Tabela 7.9 – Programação do manipulador robótico.

PROGRAMA PRINCIPAL Nº 98

INSTR	DESCRIÇÃO
10	V = 1000mm/s to 2000mm/s
20	TCP9
30	V 100%
40	Gripper wait 0.5s
50	V 100%
60	Gripper wait 1s
70	V 100%
80	V 100%
90	SET OUTPUT1
100	Call 89
110	RESET OUTPUT1
120	Wait 5s
130	V 5%
140	Call 89
150	V 1%
160	SET OUTPUT5
170	CALL89
180	RESET OUTPUT5
190	Wait 3s
200	SET OUTPUT4
210	CALL89
220	RESET OUTPUT4
230	V 100%
240	SET OUTPUT2
250	CALL89
260	RESET OUTPUT2
270	CALL 88
280	SET OUTPUT1
290	Call 89
300	RESET OUTPUT1
310	V 5%
320	Gripper wait 0.5s
330	SET OUTPUT6
340	CALL89
350	RESET OUTPUT6
360	SET OUTPUT3
370	CALL89
380	RESET OUTPUT3
390	Wait 3s
400	V 1%
410	V 5%
420	Wait 3s
430	SET OUTPUT2
440	CALL89
450	RESET OUTPUT2
460	V100%
470	Wait 3s
480	V100%
490	Gripper wait 1s
500	V100%
510	Gripper wait 0.5s
520	Return

SUBPROGRAMA Nº89

INSTR	DESCRIÇÃO
10	Wait 1s
20	Return

SUBPROGRAMA Nº88

INSTR	DESCRIÇÃO
10	Wait 99s
20	Wait 99s
30	Wait 99s
40	Wait 40s
50	Return

[3] Este problema está sendo verificado junto ao setor de engenharia da Romi.

Consequentemente, na elaboração da programação do manipulador robótico, assumiu-se que todos os comandos enviados pelo manipulador ao Centro de Torneamento fossem devidamente executados, inserindo-se apenas temporalizações para garantir a finalização dos comandos[4]. O programa principal 89 (tabela 7.9) armazena as informações de posicionamento do manipulador e os comandos que devem ser executados pela garra do robô (abrir/fechar) e pelo Centro de Torneamento (abrir porta, abrir placa, etc), enquanto o subprograma 88 representa o tempo de usinagem da peça, anteriormente calculado a partir da execução de um ciclo de usinagem da peça.

7.2.6 – Fabricação das peças

Embora tenha sido prevista inicialmente a fabricação das três ordens de trabalho, apenas a ordem W002 (duas peças) foi efetivamente executada. Isto não prejudica a validação deste trabalho, já que o principal objetivo é apresentar o ciclo de fabricação e não a usinagem das peças. A figura 7.8 apresenta a simulação da usinagem do produto 2 enquanto que as figuras 7.9 a 7.14 o ciclo de fabricação da execução da ordem de trabalho W002.

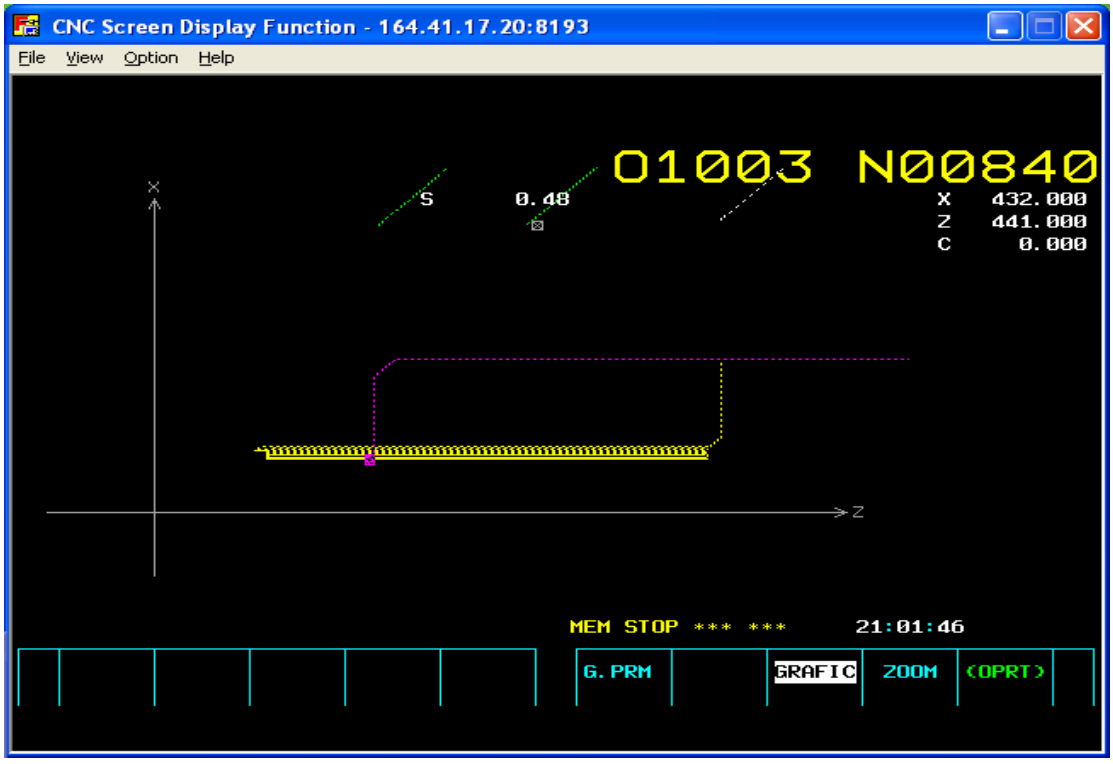

Figura 7.8 – Simulação da Usinagem do produto 2.

[4] Na prática, esta abordagem deve ser evitada, já que não garante a ausência de *deadlocks* durante a fabricação.

Figura 7.9 - Robô retirando o tarugo do *pallet*.

Figura 7.10 - Robô posicionando o *taguro* na placa do Centro de Torneamento.

Figura 7.11 - Robô aguardando a usinagem da peça.

Figura 7.12 - Robô posicionando a peça na unidade de medição do micrômetro.

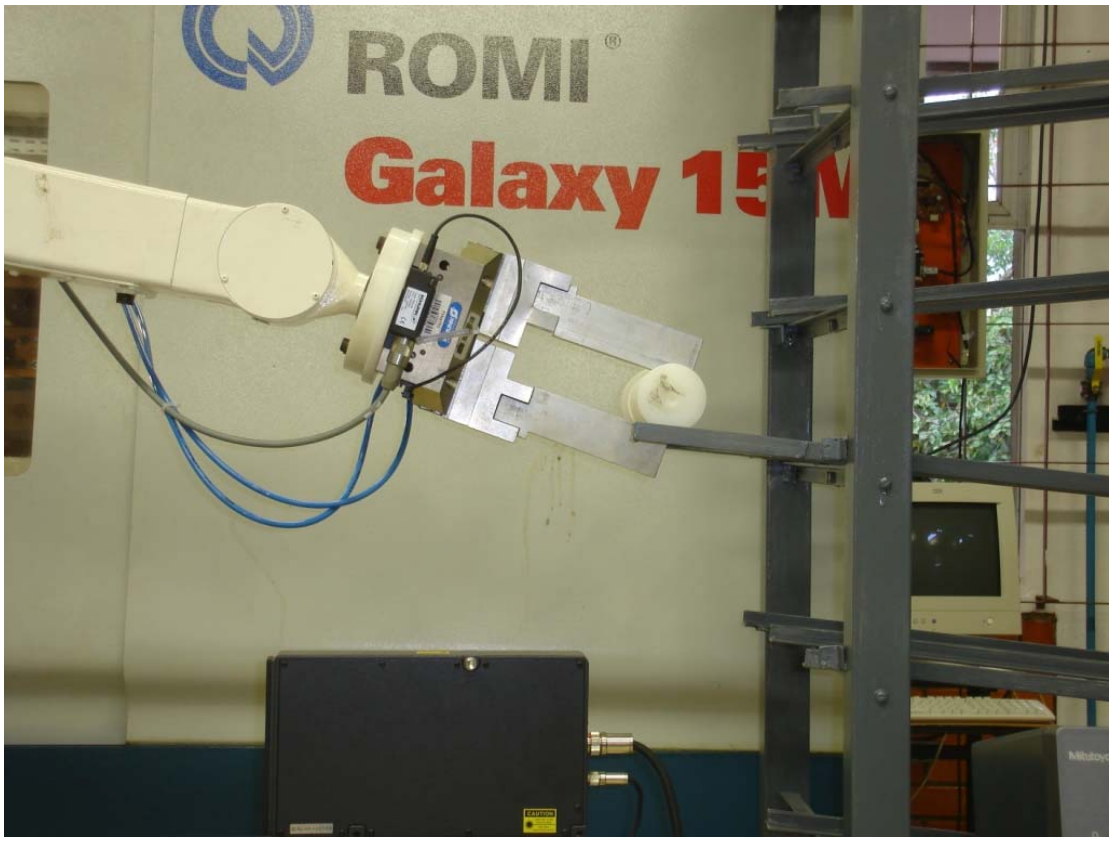

Figura 7.13 - Robô depositando a peça no *pallet*.

Figura 7.14 – Peças finais produzidas (ordem de trabalho W002).

7.2.7 – Monitoramento das estações de trabalho

A figura 7.15 mostra o monitoramento virtual das estações de trabalho operando. A esquerda estão as *tags* do PMC do Centro de Torneamento utilizadas na integração, bem como o resultado da medição e o critério de julgamento obtido durante a inspeção das peças. No centro estão as imagens virtuais das estações de trabalho operando (atualizadas a medidas que os eventos no chão-de-fábrica ocorrem), e a direita está o registro (*log*) dos eventos ocorridos durante a fabricação das peças.

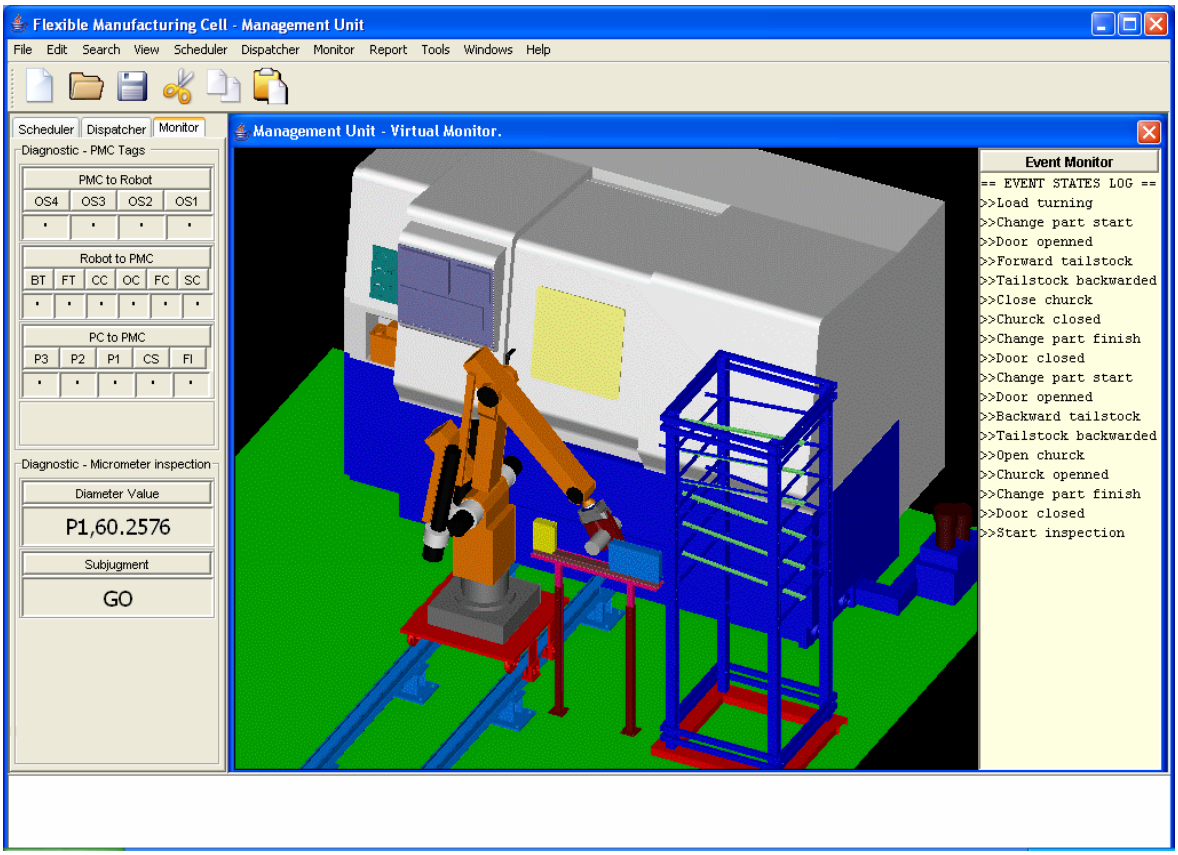

Figura 7.15 - Monitoramento virtual das estações de trabalho.

8 – CONCLUSÕES, CONTRIBUIÇÕES E TRABALHOS FUTUROS

Neste capítulo são apresentadas as considerações finais sobre a metodologia proposta, as conclusões sobre os principais resultados alcançados e as contribuições associadas ao trabalho de mestrado. Como último assunto, são propostas sugestões para trabalhos futuros que visam primordialmente complementar o desenvolvimento computacional da MgU.

8.1 – CONCLUSÕES

Este projeto resultou em uma metodologia (WebFMC) para o projeto e construção de Células Flexíveis de Manufatura, em que se utilizam os recursos oferecidos pela Internet e tecnologias associadas, para promover a fabricação remota de peças. Esta metodologia descreve um método de integração das estações de trabalho de uma FMC e uma arquitetura para a implementação do sistema de controle da célula (Unidade de Gerenciamento).

A metodologia WebFMC é baseada em aspectos de modelagem e implementação. Ela foi concebida sob os princípios da filosofia *e-manufacturing* ao enfatizar o uso da Internet e tecnologias associadas para promover a integração das operações de manufatura.

Esta metodologia pode ser empregada tanto na academia quanto na indústria. Na academia oferece uma sistemática para a construção de laboratórios remotos para auxiliar os cursos de engenharia no ensino a distância. Na indústria, concede as empresas (aqui representada pelo usuário remoto) à possibilidade de fabricarem seus produtos ainda que não possuam os recursos de fabricação necessários.

8.1.1 – Projeto e implementação da Célula Flexível de Manufatura

As técnicas de modelagem utilizadas contribuíram para o levantamento dos requisitos funcionais da célula. A modelagem por simulação, utilizando como ferramenta o software *workspace*, permitiu estabelecer o *layout* da célula, bem como a área do manipulador robótico e do *AGV*. A modelagem a eventos discretos, através das Redes de Petri, resultou no levantamento do mapa de estados alcançáveis da célula e das suas respectivas estações de trabalho.

O mapa de estados alcançáveis consiste em uma descrição detalhada das mensagens que devem ser trocadas entre as estações de trabalho, com a evolução dos eventos, a fim de evitar os indesejáveis *deadlocks*. Este estudo serviu como base para o projeto da integração física e lógica dos controladores das estações de trabalho (alteração do *Ladder* do PMC do Centro de Torneamento, criação de funções miscelâneas, instalação de interfaces dedicadas, etc).

A arquitetura de integração CNC/PMC/Controlador do robô, concebida em parceria com o fabricante do Centro de Torneamento, permite estabelecer uma comunicação indireta com o controlador do robô via PMC. Uma relação mestre-escravo entre o PMC do Centro de Torneamento (mestre) e o controlador do robô (escravo) permite que, através das funções programáveis do *FOCAS1*, a MgU escreva nas variáveis de PMC a serem lidas pelo controlador do robô.

A tabela 8.1 apresenta um resumo dos resultados alcançados no projeto e implementação da FMC. Nesta tabela, cada atividade tem sua proposta inicial compara com seu respectivo resultado alcançado.

Tabela 8.1 - Projeto e Implementação da Célula Flexível de Manufatura.

Nº	Proposta Inicial	Projeto	Implementação
1.0	**Modelagem da FMC**		
	Modelagem por simulação via workspace	-	Concluída
	Modelagem a eventos discretos	-	Concluída
2.0	**Integração física da FMC**		
	Projeto e construção de um carro posicionador	Concluído	Concluída
	Projeto e construção da unidade de armazenamento - pallet	Concluído	Parcialmente concluída
	Projeto e construção dos dedos para a garra do robô	Concluído	Concluída
	Projeto e construção do suporte para o micrômetro	Concluído	Concluída
	Projeto e construção da Interface de comunicação	Concluído	Concluída
	Integração física ROBO-TORNO-MICRÔMETRO	Concluído	Parcialmente concluída
3.0	**Integração lógica da FMC**		
	Integração lógica ROBO-TORNO-MICRÔMETRO	Concluído	Concluída
	Programação da Unidade de manipulação e transporte	Concluído	Concluída

8.1.2 – Projeto e implementação da Unidade de Gerenciamento

A Unidade de Gerenciamento, como sistema computacional, é responsável pelo planejamento e controle da produção no chão-de-fábrica. Embora nem todas as funcionalidades previstas na arquitetura foram implementadas, o sistema está operando com o mínimo de funcionalidades necessárias para a fabricação do produto.

O modelo funcional projetado em IDEF0 documenta, em um nível mais abstrato, a relação entre cada módulo do sistema, bem como as respectivas transformações que cada função realiza. Este modelo serve como referência para a continuidade da implementação da MgU ou até mesmo como base para novos sistemas de controle. Em nenhuma das referências consultadas, a arquitetura foi modelada com o nível de detalhamento apresentado nessa dissertação. Já o modelo da informação gerado a partir do método IDEF1x serve como base para a implementação da base de dados relacional WebFMC.

O projeto e a documentação da arquitetura baseada no Processo Unificado resultam em uma descrição detalhada, centrada na arquitetura, que permitiu o desenvolvimento interativo e incremental baseado em fluxos de trabalho. Os diagramas UML gerados servem como modelos para que futuras alterações na implementação computacional da MgU possam ser realizados mais facilmente.

A primeira versão computacional da MgU pode ser acessada através da URL: http://webfmc.graco.unb.br/mgu/mgu.jnlp. Embora nem todas as funcionalidades estejam implementadas, os algoritmos concebidos e os diagramas constituem uma importante documentação para que futuros desenvolvedores continuem os trabalhos de implementação. Toda a documentação gerada durante este projeto pode ser obtida a partir da URL: http://webfmc.graco.unb.br/downloads/documentations.

A implementação baseada na tecnologia Java permite que o operador execute a MgU via Internet, sem a necessidade de instalação do aplicativo. O cliente deve possuir apenas uma máquina virtual instalada (JRE) que suporte Java 2.

A implementação, baseada na arquitetura cliente-servidor, encapsula os serviços que fazem chamadas a funções do sistema operacional (funções programáveis do FOCAS1 e comunicação via interface RS232C) no módulo servidor da MgU. Consequentemente, a portabilidade herdada da tecnologia Java é mantida, o que permite que a MgU seja executada em diferentes sistemas operacionais, devendo apenas o módulo servidor ser instalado na plataforma Windows. A tabela 8.2 mostra os detalhes do projeto e da implementação da Unidade de Gerenciamento.

Tabela 8.2 – Projeto e implementação da Unidade de Gerenciamento.

Nº	Proposta Inicial	Algoritmo	Implementação
1.0	**Modelagem da MgU**	**-**	Concluída
2.0	**Implementação computacional da MgU**		
	Programação da Inspeção	-	Concluído
	Programação da produção	-	Concluído
	Formação das famílias de peças	Concluído	Não implementado
	Escalonamento da produção	Concluído	Parcialmente concluída
	Setup das estações de trabalho	Concluído	Parcialmente concluída
	Monitoramento das estações de trabalho	Concluído	Concluído
	Controle da qualidade	Concluído	Parcialmente concluída
	Emissão de relatórios	-	-

8.2 – CONTRIBUIÇÕES

A principal contribuição deste trabalho constitui a proposição de uma nova metodologia (WebFMC) baseada em aspectos de modelagem e implementação. A FMC implementada servirá como unidade de aprendizado e pesquisa as disciplinas relacionadas a Sistemas Integrados de Manufatura (SIM), como também poderá ser utilizada por outras instituições de ensino que não possuem semelhantes recursos de fabricação. Em se tratando de contribuições específicas relacionadas ao projeto e implementação da célula:

- ✓ Projeto e construção de componentes mecânicos: Um armazenador de peças (*pallet*), um carro posicionador, um suporte para o micrômetro, dedos para a garra do robô;

- ✓ Projeto e construção de interfaces dedicadas: Interface de comunicação, interfaces dedicadas Robô – Torno e Sistema de Medição – Torno;

- ✓ Integração física e lógica dos controladores das estações de trabalho: A abordagem adotada oferece uma sistemática para que os controladores das estações de trabalho possam ser integrados, ainda que possuam diferentes protocolos de comunicação;

- ✓ Arquitetura de integração CNC/PMC/Controlador do robô: Controle indireto de uma estação de trabalho utilizando como mestre o controlador de outra estação de trabalho. Isto inclui desde a instalação das interfaces dedicadas à definição da estrutura de programação;

- ✓ Implementação de um banco de dados relacional: Projeto e construção de um banco de dados relacional utilizado para armazenar a informação da camada de persistência do sistema e para suportar a integração com o sistema CAD/CAPP/CAM;

- ✓ Supervisão, monitoramento e teleoperação de um micrômetro laser (modelo LM6100): Envio e recepção de comandos via interface RS232C que permitem a programação das informações geométricas às condições de medição. Essas funções foram implementadas e encapsuladas em um algoritmo de programação baseado no exemplo disponibilizado pelo manual do fabricante.

- ✓ Supervisão, monitoramento e teleoperação de Centro de Torneamento Romi (modelo Galaxy 15M – CNC Fanuc 18i-ta): Envio de instruções ao CNC para alterar os parâmetros de CNC ou de PMC do Centro de Torneamento. Utilizam-se as funções programadas da *API FOCAS1* e da Interface *JNI*;

- ✓ Supervisão, monitoramento e teleoperação de um manipulador robótico ASEA IRB6: Apesar da limitação em termos de comunicação, através da arquitetura CNC/PMC/Controlador do robô, é possível enviar até 15 instruções ao manipulador robótico, escrevendo-se nas variáveis de PMC do Centro de Torneamento via *FOCAS1*;

- ✓ Supervisão, monitoramento de um *AGV* (Nomad XR4000): Apesar de não estar completamente implementado, o servidor desenvolvido permite que uma lista de ferramentas seja enviada ao *Nomad* via TCP/IP. Para finalizar a integração do mesmo com a célula é necessário ainda desenvolver o sistema de navegação do *AGV*;

As contribuições relacionadas ao projeto e implementação da Unidade de Gerenciamento estão encapsuladas em um conjunto de funcionalidades disponibilizadas. É importante lembrar que nem todas as funcionalidades foram completamente implementadas:

- ✓ Programação da inspeção: Incluem desde algoritmos a interfaces gráficas desenvolvidas no intuito de programar os atributos geométricos das peças e as condições em que a medição será efetuada;

- ✓ Programação da produção: Algoritmos e interfaces gráficas que permitem o cadastro de ordens de trabalho, a formação de famílias de peças e a definição da seqüência em que as famílias e ordens de trabalho serão enviadas para o chão-de-fábrica;

- ✓ *Setup* das estações de trabalho: Algoritmos e interfaces gráficas para carregar a lista de tarefas, para verificar o *status* das estações de trabalho e para efetuar o carregamento de instruções nas estações de trabalho;

- ✓ Supervisão e monitoramento das estações de trabalho: Monitoramento virtual das estações de trabalho, monitoramento em tempo real herdado do sistema *Webmachining* (Álvares *et at*, 2005), controle da qualidade parcialmente implementado;

- ✓ Tratamento de condições adversas: O algoritmo concebido para o tratamento de condições adversas permite o re-escanolamento das ordens de trabalho ou até mesmo a solicitação de intervenção humana.

8.3 – TRABALHOS FUTUROS

Embora a MgU estejam operando com o mínimo das funcionalidades necessárias, existe ainda muito trabalho a ser realizado para que a Unidade de Gerenciamento opere com todas as funcionalidades descritas na arquitetura gerada por esta metodologia. A seguir estão relacionadas as principais propostas de trabalhos futuros que visam complementar a implementação da MgU:

- ✓ Otimização do controle estatístico do processo: Um importante trabalho seria a continuidade no desenvolvimento deste módulo, incorporando o ajuste automático do processo via corretor de ferramentas fornecido pelo CNC FANUC 18i-ta. É previsto também a utilização de planos de processo não lineares, utilizando-se das funções miscelâneas M202 e M203 para realizar o processamento do produto;

- ✓ Otimização do módulo de emergência da célula: Esta primeira versão do módulo de emergência necessita ser otimizada. Um importante trabalho seria a incorporação do tratamento de condições adversas, a partir de planos de processos alternativos gerados pelo sistema CAD/CAPP/CAM.

- ✓ Criação de um help para documentar a utilização das funcionalidades: Este documento constitui a criação de um *help* para que docentes e discente utilizem as funcionalidades disponibilizadas pela MgU. Um pequeno tutorial descreveria passo a passo como executar um ciclo de fabricação na célula utilizando as funcionalidades oferecidas.

REFERÊNCIAS BIBLIOGRÁFICAS

Alexander, C. (1979) *"The Timeless Way of Building."* ed. Oxford University, New York.

Álvares, A. J. and Andriolli, G. F. and Dutra P. R. C. and Sousa, M. M. and Ferreira, J. C. E. (2004) "A navigation and path planning system for the nomad XR4000 mobile robot with remote web monitoring." In: *ABCM Symposium Series in Mechatronics*, vol. 1, pp. 18-24.

Álvares, A. J. (2005). *Uma metolodogia para a integração CAD/CAPP/CAM voltada para a manufatura remota de peças rotacionais baseada na internet*, Tese de Doutorado, Departamento de Engenharia Mecânica, UFSC, 329p.

Álvares, A. J. and Ferreira, J. C. E. (2005) "WebMachining: implementation of a collaborative CAD/CAPP/CAM system for e-manufcturing through the Internet." In: *The 38th Cirp, International Seminar on Manufacturing Systems*, florianópolis, SC.

Anglani, A. and Grieco, A. and Pacella, M. and Tolio, T. (2002). "Object-oriented modeling and simulation of flexible manufacturing systems: a rule-based procedure." In: *Simulation Modelling Practice and Theory 10*, pp. 209-234.

ARP2-4 (2006) "Petri Net Analyser ARP version 2.4."

In:http://www.lcmi.ufsc.br/~farines/das6604/das6604.html (25/10/2006).

Asprova, (2006). "Advanced Planning & Scheduling (APS) - Asprova."

In: http://www.asprova.com/en/asprova/gantt_chart.html#FreeSample (22/11/2006).

Bedworth, D., Henderson, M.R. and Wolfe, P.M. (1991) *"Computer-Integrated Design and Manufacturing"*, New York: McGraw-Hill.

Blazewicz, J. and Domschke, W. and Pesch, E. (1996) "The job shop scheduling problem: Conventional and new solution techniques." In: *European Journal of Operational Research 93*, pp. 1-33.

Booch, B. e Rumbaugh, J. e Jacobson, I. (2000) *"UML guia do usuário."*, ed. Campus, Rio de Janeiro, 472p.

Cardoso, J. e Valette, R. (1997) *"Redes de Petri."*, ed. UFSC, Florianópolis, 212p.

Cheng, C. H and Goh,C, and Lee, A. (2001). "Designing group technology manufacturing systems using heuristics branching rules." In: *Computer & Industrial Engineering 40.* pp. 117-131

Chen, S. and Lu, F. Y. (2002) "Web-based simulation of power systems." In: *IEEE Computer Applications in Power 1*, pp. 35-40.

Cinnéide, M. Ó. (2000). *Automated Application of Design Patterns: A Refactoring Approach*, PhD Thesis, Department of Computer Science, Trinnity College, Dublin, 151p.

CircuitMaker (2000). “CircuitMaker – the virtual electronic lab” In: http://www.circuitmaker.com/pdfs/cm_usermanual.pdf (20/11/2006).

Cisco, (2006). “Cisco Ethernet to the Factory.”

In:http://www.cisco.com/web/strategy/docs/manufacturing_ettf_overview0808.pdf 16/11/2006).

Conway, R.W. and Maxwell, W. L. and Miller, L.W. (1967) *“Theory of Scheduling”*, ed. Addison-Wesley, Reading, MA.

Duncan, A. J. (2001) *“Quality control and industrial statistics”*, 5º edition, ed. Boston.

Feng, S.C., and Song, E. Y. (2000) “Information Modeling of Conceptual Process Planning Integrated with Conceptual Design.” In: *The International Mechanical Engineering Congress and Exposition*, November, pp.5 – 10.

Fowler, M. and Scott, K. (2000) *“Uml essencial : Um breve guia para a linguagem-padrão de modelagem de objetos.”*, 2º edição, Bookman, Porto Alegre, 169p.

Furlan, J.D. (1998) *“Modelagem de objetos através da UML – The Unifield Modeling Language.”*, ed. Makron Book, São Paulo, Brasil.

GEFanuc. (2006) “Application Development - FOCAS1 (Drivers & Programming Libraries)” In: http://www.geindustrial.com/cwc/products?pnlid=2&id=cnc_mec_39 (04/09/2006).

Gamma, E., Helm, R., Johnson, R., et at (2000) *“Padrões de Projeto: Soluções reutilizáveis de software orientados a objeto.”* Ed. Bookman, Porto Alegre, Rio Grande do Sul.

Girault, C. and Valk, R. (2002) *“A Petri nets for systems engineering – a guide to modeling, verification, and applications.”*, ed. Springer.

Groover, M. P. (2003) *“Automation, Production Systems, and Computer-Integrated manufacturing”*, second edition, ed. Prentice Hall.

Gu, T. and Bahri, P. A. (2002). “A survey of Petri net applications in batch processes.” In: *Computers in Industry 47*, pp. 99-111.

Hannemann, J. and Kiczales, G. (2002). “Design pattern implementation in Java and aspectJ” In: *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 161-173.

IDEF0, (1993) “Integration Definition for Functional Modeling (IDEF0)” In: *Federal Information Processing Standards Publication 18*.

Jeon, G. and Leep, H. R. (2006). “Forming part families by using genetic algorithm and designing machine cells under demand changes.” In: *Computers & Operations Research 33*, pp. 263-283.

Java Communication (2006) “Java Communications.”

In: http://java.sun.com/products/javacomm/ (21/08/2006).

JDBC technology, (2006). “Java SE - Java Database Connectivity (JDBC).” In: http://java.sun.com/javase/technologies/database/index.jsp (30/11/2006).

JNI (2006) “Java Native Interface Overview.”

In: http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/intro.html#wp725 (15/01/2006).

JNLP (2006) “JNLP Specification & API Documentation.”

In: http://java.sun.com/products/javawebstart/download-spec.html (15/01/2006).

Juran, J. M., Gryna, F. M. (1992). *Controle da qualidade - HandBook*, **vol**. **6**, Makron Books, São Paulo, Brasil.

Kim, C. and Weston, R. H. and Hodgson, A. and Lee, K. (2002) “The complementary use of IDEF and UML modelling approaches.” In: *Computers in Industry 50*, pp.35 – 56.

King, J. R. (1980) “Machine-component grouping in production flow analysis: An approach using a rank order clustering algorithm.” In: *International Journal of Production Research 18*, pp. 213-232.

Krar, S. and Arthur, G. (2003) “Exploring Advanced Manufacturing Technologies”, In: *New York, Industrial Press*.

Kumar, M. and Rajotia, S. (2003) “Integration of scheduling with computer aided process plan.” In: *Journal of Materials Processing Techology 138*, pp. 297-300.

Lee, J. (2003) “E-manufacturing: fundamental, tools and transformation. In: *Robotics and Computer Integrated Manufacturing 19*, p. 501-507.

Lee, J. and Korbaa, O. (2004) “Modeling and scheduling of ration-driven FMS using unfolding time Petri nets.” In: *Computers & Industrial Engineering 46*, pp. 639-653.

Leitão, P. and Machado, J. and Lopes, J. (1996) “A Manufacturing Cell Integration Solution.” In: *Proceedings of 2nd IEEE/ECLA/IFIP International Conference on Architectures and Design Methods for Balance Automation Systems, L. Camarinha-Matos and H. Afsarmanesh (eds), Chapman & Hall*, Lisboa, Portugal, pp. 209-216.

Leitão, P. and Restivo, F. (1999). “Layered Approach to Distributed Manufacturing.” In: *Proceedings of 1999 Advanced Summer Institute - LIFE Cycle Approaches to Production Systems: Management, Control, Supervision, Leuven, Belgium*.

Leitão, P. (2004). *An Agile and Adaptive Holonic Architecture for Manufacturing Control*, PhD Thesis, Faculty of Engineering, University of Porto, Portugal, 273p.

Ledolter, J. and Swersey, A. (1997) “An evaluation of pre-control.” In: *Journal of Quality Technology 29*, pp. 163.

MEEM Laboratory, (2006). “Manufacturing Engineering and Engineering Management: City University of Hong Kong.” In: http:// www.cityu.edu.hk/me/gallery/meem01-052.jpg (25/11/2006).

Minas, M. and Frey, G. (2002). “Visual PLC-programming using signal interpreted Petri nets.” In: *Proceedings of the American Control Conference, Anchorage, AK*, pp. 5029-5024.

Mitutoyo, (2003). “Laser Scan Micrometer (Display Unit).” In: *User Manual.*

Nomadic Tecnologies, (1999). “Nomad XR4000 Hardware Manual Release 1.0.” In: *User Manual.* Califórnia, EUA.

Oliveira, C. H. P. (2002) *“SQL Curso prático.”*, ed. Novatec, São Paulo, 272p.

Ou-Yang, C. and Guan, T. Y. and Lin, J. S. (2000) “Developing a computer shop floor control model for a CIM system - using object modeling technique.” In: *Computers in Industry 41*, pp. 213-238.

Pacella, M. and Semeraro, Q. and Anglani, A. (2004) “Manufacturing quality control by means of a Fuzzy ART network trained on natural process data.” In: *Engineering Applications of Artificial Intelligence 17*, pp. 83-96.

Pearn, W. L. and Shu, M. (2003) “Manufacturing capability control for multiple power-distribution switch processes based on modified Cpk MPPAC.” In: *Microelectronics Reliability 43*, pp. 963 - 975.

Pels, H.J. and Wortmann, J.C. and Zwegers, A.J.R. (1997) “Flexibility in manufacturing: An architecture point of view” In: *Computer in industry 33*, 271-283.

Quintas, A. and Leitão, P. (1997) “A Cell Controller Architecture Solution: Description and Analysis of Performance and Costs.” In: *Proceedings of Integrated and Sustainable Industrial Productions*, Lisboa, Portugal.

Rabelo, R. J. (1997). *Um enquadramento para o desenvolvimento de sistemas de escalonamento ágil da produção – uma abordagem multiagente*, Tese de Doutorado, Faculdade de Ciências e Tecnologia, Universidade de Nova Lisboa, Portugal, 287p.

Rational, (2006). “Rational Unified Process for Systems Engineering.” In: http://www-306.ibm.com/software/rational/sw-library/#White%20papers (14/10/2006).

Rockwell, (2006). “Integrated Architecture.”

In: http://www.rockwellautomation.com/solutions/integratedarchitecture/ (16/11/2006).

Rodammer, F. A. and White Jr, P. (1988) “A Recent Survey of Production Scheduling.” In: *IEE Transactional on Systems, Man, and Cybernetics, vol. 18*, nº 06.

Ryan, J. and Heavey, C. (2006) “Process modeling for simulation.” In: *Computer Industry 57*, pp. 437-450.

Sauer, J. (1997). *Knowledge-Based Systems Techniques and Applications in Scheduling*, San Diego, Academic Press.

Spinellis, D. (2001). "Notable design patterns for domain-specific languages" In: *The Journal of System and Software 56*, pp. 91-99.

Sotto, E. and Pereira, M. (2001) "Implementing a Petri net specification in a FPGA using VHDL." In: *The International Workshop on Discrete-Event System Design DESDes'01*.

Stanchfield, S. (2000). "How do I create my own events to pass between objects." In: http://www.jguru.com/faq/view.jsp?EID=98547, (14/10/2006).

Starbek, M. and Kušar, J. and Brezovar, A. (2003). "Optimal Scheduling of Jobs in FMS." In: *Proceedings of CIRP - Journal of Manufacturing Systems vol. 32*, nº 5.

Steiner, S. H. (1998) "Pre-control and Some Simple Alternatives." In: *Quality Engineering 10*, pp. 65-74.

Sun BluePrints. (2002). "Designing Enterprise Applications with the J2EETM Platform." In*: Addison-Wesley*. Second edition.

Sun Microsystem (2006). "Collections Framework Overview."

In: http://java.sun.com/j2se/1.4.2/docs/guide/collections/overview.html (25/11/2006).

Teixeira, E. L. S. and Cano, C. H. V. and Álvares, A. J. (2005) "Modeling and Implementation of a Flexible Manufacturing Cell (FMC)." In: *18th International Congress of Mechanical Engineering*, COBEM2005, Ouro Preto, Minas Gerais.

Teixeira, E. L. S. and Álvares, A. J. (2006) "Modeling and Implementation of a Management Unit integrated to a CAD/CAPP/CAM System for a Flexible Manufacturing Cell." In: *Proceedings of Conference on Flexible Automation and Intelligent Manufacturing*, FAIM2006, Limerick, Ireland.

Teixeira, E. L. S. and Álvares, A. J. (2006) "An architecture for a Management Unit integrated to a CAD/CAPP/CAM system for a Flexible Manufacturing Cell." In: *IV Congresso Nacional de Engenharia Mecânica – CONEM2006*, Recife, Pernambuco.

Teixeira, E. L. S. e Cano, C. H. V. e Álvares, A. J. (2006) "Modelagem e implementação de uma Célula Flexível de Manufatura. Em: *Máquinas e Metais, v. XLIII*, p. 134-153.

Urdhwareshe, H. P. (2002) "The power of pre-control." In: *Quality & Productivity Journal* , 5p.

Wysk, R. A. and Smith, J. S. (1995) "A formal function characterization of shop floor control." In: *Computer Industry Engineering*, vol. 28, nº 3, pp. 631-643.

Workspace, (2006). "The Next Generation of Advanced Robotic Simulation Software." In: http://www.workspace5.com/ (20/11/2006).

APÊNDICES

APÊNDICE A – REDE DE PETRI DETALHADA DA CÉLULA

Este apêndice apresenta a estrutura da RdP detalhada da célula, bem como um relatório de simulação para a produção de três peças. A estrutura da RdP está de acordo com a linguagem descritiva utilizada pela ferramenta ARP2-4[1] (Analisador de Redes de Petri ARP versão 2.4), desenvolvida pelo laboratório de controle e microinformática da Universidade Federal de Santa Catarina (LCMI/DEFL/UFSC). Esta ferramenta permite a modelagem e validação de três tipos de RdP: ordinária, temporal e temporal estendida.

Composta por oito módulos (edição, compilação, gerenciamento de arquivos, simulação, enumeração dos estados alcançáveis, etc), a ferramenta ARP2-4 permite desde a definição e validação da estrutura da rede, como o estudo das propriedades estruturais e do comportamento dinâmico da mesma. Maiores detalhes sobre esta ferramenta podem ser obtidos através da URL: http://www.lcmi.ufsc.br/~farines/das6604/das6604.html.

A.1 – ESTRUTURA DA REDE

```
NET FMC;
CONST
  NPB=3;{Numero de pecas bruta}

NODES
  {Pallet}
  P_Bt: Place(NPB);{Peca bruta}
  P_Acab,          {Peca acabada}
  P_Retr,          {Peca retrabalho}
  P_Refu: Place;     {Peca refugo}

  {Robo Manipulador}
  {Status}
  Rb_Op: Place;
```

[1] Esta ferramenta pode ser obtida gratuitamente através da URL: http://www.das.ufsc.br/engsoft/arp2-4.zip.

Rb_Livre: Place(1);

{Posicionamento}
Rb_Pos1, {Robo se posiciona em frente ao pallet}
Rb_Pos2, {Robo se posiciona p/ pegar peca bruta}
Rb_Pos3, {Robo se retirando do torno p/ iniciar usinagem}
Rb_Pos4: Place; {Robo se aproxima para retirar peca: Setup}
Rb_Hom: Place(1); {Robo na posicao HOME}

{Robo Manipulador - Transporte}

Tr_Peca1, {Robo transportando tarugo p/ o torno}
Tr_Peca2, {Robo colocando tarugo na placa do torno}
Tr_Peca4, {Robo colocando a peca na placa do torno}
Tr_Peca5, {Robo transportando tarugo p/ o torno}
Tr_Peca6, {Robo coloca peca no pallet de pecas acabadas}
Tr_Peca7, {Robo coloca peca no pallet de pecas retrabalho}
Tr_Peca8, {Robo coloca peca no pallet de pecas refugo}

{Garra - Estados}
Rb_Ga_A, {Robo com a garra aberta}
Rb_Ga_P1, {Robo com a garra na posicao P1}
Rb_Ga_P2, {Robo com a garra na posicao P2}
Rb_Ga_P3: Place; {Robo com a garra na posicao P3}
Rb_Ga_F: Place(1); {Robo com a garra fechada}

{TornoCNC}
Tn_Livre: Place(1);
Tn_P_F, {Torno com a porta aberta}
Tn_Pl_F: Place(1); {Torno com a porta fechada}
Tn_Pl_A, {Torno com a placa aberta}
Tn_P_A: Place; {Torno com a porta aberta}
Tn_Us : Place; {Torno usinando peca}
Tn_Op : Place; {Torno alocado p/ usinagem}

```
  {Micrometro}
  Medicao: Place;      {Micrometro medindo peca}
  N_Medicao: Place(1); {Micrometro nao medindo peca}

  {FMC}
  Clc_Carr: Place; {Ciclo de carregamento}
  Clc_Desc: Place; {Ciclo de descarregamento}


 {Transicoes p atender a eventos do robo}
 T01,T02,T03,T04,T05,
 T06,T07,T08,T09,T10,
 T11,T12,T13,T14,T15,
 T16,T17,T18,T19,T20,
 T21,T22,T23,T24,T25,
 T26,T27,T28,T29,T30,
 T31,T32,T33,T34,T35,
 T36,T37,T38,T39,T40,
 T41,T42,T43,T44,T45,
 T46,T47,T48,T49,T50,
 T51,T52,T53: Transition;

STRUCTURE

{Inicializa robo}
T01: (Rb_Livre,P_Bt,Tn_Livre,Rb_Ga_F,Rb_Hom),(Rb_Op,Tn_Livre,Rb_Ga_A,Rb_Pos1);

{Robo avanca para pegar o tarugo}
T02:(Rb_Pos1,Rb_Ga_A),(Rb_Pos2,Rb_Ga_A);

{Robo fecha a garra na posicao 1,2,3 ou F}
T03:(Rb_Pos2,Rb_Ga_A),(Rb_Ga_P1,Rb_Pos2,Clc_Carr);
T04:(Rb_Pos2,Rb_Ga_A),(Rb_Ga_P2,Rb_Pos2,Clc_Carr);
```

T05:(Rb_Pos2,Rb_Ga_A),(Rb_Ga_P3,Rb_Pos2,Clc_Carr);
T06:(Rb_Pos2,Rb_Ga_A),(Rb_Ga_F,Rb_Pos2 ,Clc_Carr);

{Ciclo de carregamento - M200}
{Inicializa torno}
T07:(Tn_Livre,Clc_Carr),(Tn_Op,Clc_Carr);

{Abre a porta e a placa do torno}
T08:(Tn_Op,Clc_Carr,Tn_P_F,Tn_Pl_F),(Tn_Op,Clc_Carr,Tn_P_A,Tn_Pl_A);

{Robo transporta tarugo p/ o torno}
T09:(Rb_Pos2,Rb_Ga_P1,Tn_P_A,Clc_Carr),(Tr_Peca1,Rb_Ga_P1,Tn_P_A,Clc_Carr);
T10:(Rb_Pos2,Rb_Ga_P2,Tn_P_A),(Tr_Peca1,Rb_Ga_P2,Tn_P_A);
T11:(Rb_Pos2,Rb_Ga_P3,Tn_P_A),(Tr_Peca1,Rb_Ga_P3,Tn_P_A);
T12:(Rb_Pos2,Rb_Ga_F ,Tn_P_A),(Tr_Peca1,Rb_Ga_F ,Tn_P_A);

{Robo posiciona tarugo na placa}
T13:(Tn_P_A, Tn_Pl_A,Tr_Peca1),(Tr_Peca2,Tn_Pl_A,Tn_P_A);

{Torno Fecha placa}
T14:(Tr_Peca2,Tn_Pl_A),(Tn_Pl_F,Tr_Peca2);

{Robo Abre garra}
T15:(Tn_Pl_F,Tr_Peca2,Rb_Ga_P1),(Rb_Ga_A,Tn_Pl_F,Tr_Peca2);
T16:(Tn_Pl_F,Tr_Peca2,Rb_Ga_P2),(Rb_Ga_A,Tn_Pl_F,Tr_Peca2);
T17:(Tn_Pl_F,Tr_Peca2,Rb_Ga_P3),(Rb_Ga_A,Tn_Pl_F,Tr_Peca2);
T18:(Tn_Pl_F,Tr_Peca2,Rb_Ga_F), (Rb_Ga_A,Tn_Pl_F,Tr_Peca2);

{Robo se retira do torno}
T19:(Rb_Ga_A,Tr_Peca2),(Rb_Pos3,Rb_Ga_A);

{Torno fecha porta}
T20:(Rb_Pos3,Tn_P_A,Clc_Carr),(Tn_P_F,Rb_Pos3);
{Fim do ciclo de carregamento}

{INICIA PROCESSO USINAGEM}
T21:(Rb_Pos3,Tn_P_F,Tn_Op),(Rb_Pos3,Tn_Us,Tn_P_F);

{TERMINA USINAGEM}
T22:(Tn_Us,Tn_P_F),(Tn_Op,Tn_P_A,Clc_Desc);

{Ciclo de descarregamento - M201}
{Torno abre porta}
T23:(Rb_Pos3,Tn_P_F,Clc_Desc),(Tn_P_A,Rb_Pos3,Tn_Op,Clc_Desc);

{Robo se posiciona para retirar a peca}
T24:(Tn_P_A,Rb_Pos3,Clc_Desc),(Rb_Pos4,Tn_P_A,Clc_Desc);

{Robo fecha a garra na posicao 1,2,3 ou F}
T25:(Rb_Pos4,Rb_Ga_A),(Rb_Ga_P1,Rb_Pos4);
T26:(Rb_Pos4,Rb_Ga_A),(Rb_Ga_P2,Rb_Pos4);
T27:(Rb_Pos4,Rb_Ga_A),(Rb_Ga_P3,Rb_Pos4);
T28:(Rb_Pos4,Rb_Ga_A),(Rb_Ga_F,Rb_Pos4);

{Torno abre placa}
T29:(Rb_Pos4,Rb_Ga_P1,Tn_Pl_F),(Tn_Pl_A,Rb_Pos4,Rb_Ga_P1);
T30:(Rb_Pos4,Rb_Ga_P2,Tn_Pl_F),(Tn_Pl_A,Rb_Pos4,Rb_Ga_P2);
T31:(Rb_Pos4,Rb_Ga_P3,Tn_Pl_F),(Tn_Pl_A,Rb_Pos4,Rb_Ga_P3);
T32:(Rb_Pos4,Rb_Ga_F, Tn_Pl_F),(Tn_Pl_A,Rb_Pos4,Rb_Ga_F);

{Robo retira peca}
T33:(Tn_Pl_A,Rb_Pos4,Clc_Desc),(Tr_Peca4,Tn_Pl_A);

{Fim do ciclo de descarregamento}

{INICIA INSPECAO}
{Inicia inspecao}
T34:(Tr_Peca4,N_Medicao),(Medicao,Tr_Peca5);

{Resultado da medicao}
T35:(Medicao,Tr_Peca5),(N_Medicao,Tr_Peca6);
T36:(Medicao,Tr_Peca5),(N_Medicao,Tr_Peca7);
T37:(Medicao,Tr_Peca5),(N_Medicao,Tr_Peca8);

{Peca acabada - transporte p/ o pallete}
T38:(Tr_Peca6,Rb_Ga_P1),(Tr_Peca6,Rb_Ga_A);
T39:(Tr_Peca6,Rb_Ga_P2),(Tr_Peca6,Rb_Ga_A);
T40:(Tr_Peca6,Rb_Ga_P3),(Tr_Peca6,Rb_Ga_A);
T41:(Tr_Peca6,Rb_Ga_F),(Tr_Peca6,Rb_Ga_A);

{Peca retrabalho}
T42:(Tr_Peca7,Rb_Ga_P1),(Tr_Peca7,Rb_Ga_A);
T43:(Tr_Peca7,Rb_Ga_P2),(Tr_Peca7,Rb_Ga_A);
T44:(Tr_Peca7,Rb_Ga_P3),(Tr_Peca7,Rb_Ga_A);
T45:(Tr_Peca7,Rb_Ga_F),(Tr_Peca7,Rb_Ga_A);

{Peca refugo - transporte p/ o pallete}
T46:(Tr_Peca8,Rb_Ga_P1),(Tr_Peca8,Rb_Ga_A);
T47:(Tr_Peca8,Rb_Ga_P2),(Tr_Peca8,Rb_Ga_A);
T48:(Tr_Peca8,Rb_Ga_P3),(Tr_Peca8,Rb_Ga_A);
T49:(Tr_Peca8,Rb_Ga_F),(Tr_Peca8,Rb_Ga_A);

{Robo deposita peca e volta a posicao inicial}
T50:(Rb_Ga_A,Tr_Peca6,Rb_Op),(Rb_Ga_A,Rb_Hom,Rb_Livre, P_Acab);
T51:(Rb_Ga_A,Tr_Peca7,Rb_Op),(Rb_Ga_A,Rb_Hom,Rb_Livre, P_Retr);
T52:(Rb_Ga_A,Tr_Peca8,Rb_Op),(Rb_Ga_A,Rb_Hom,Rb_Livre, P_Refu);

{Robo livre - reinicializa processo}
T53:(Tn_Op,P_Bt,Rb_Hom,Rb_Livre),(Tn_Op,Rb_Pos1,Rb_Op);

ENDNET.

A.2 – RELATÓRIO DE SIMULAÇÃO

Simulation tracing for net FMC.

----> Trace on.
Depth : 0 (Memorized as Inicial State).
Marking: {3* P_Bt, Rb_Livre, Rb_Hom, Rb_Ga_F, Tn_Livre, Tn_Pl_F, Tn_P_F,
N_Medicao}
Bounds : {T01}

-> T01 fire
Depth : 1.
Marking: {2* P_Bt, Rb_Op, Rb_Pos1, Rb_Ga_A, Tn_Livre, Tn_Pl_F, Tn_P_F,
N_Medicao}
Bounds : {T02}

-> T02 fire
Depth : 2.
Marking: {2* P_Bt, Rb_Op, Rb_Pos2, Rb_Ga_A, Tn_Livre, Tn_Pl_F, Tn_P_F,
N_Medicao}
Bounds : {T03, T04, T05, T06}

-> T06 fire
Depth : 3.
Marking: {2* P_Bt, Rb_Op, Rb_Pos2, Rb_Ga_F, Tn_Livre, Tn_Pl_F, Tn_P_F,
N_Medicao, Clc_Carr}
Bounds : {T07}

-> T07 fire
Depth : 4.
Marking: {2* P_Bt, Rb_Op, Rb_Pos2, Rb_Ga_F, Tn_Pl_F, Tn_P_F, Tn_Op,
N_Medicao, Clc_Carr}
Bounds : {T08}

-> T08 fire

Depth : 5.

Marking: {2* P_Bt, Rb_Op, Rb_Pos2, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T12}

-> T12 fire

Depth : 6.

Marking: {2* P_Bt, Rb_Op, Tr_Peca1, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T13}

-> T13 fire

Depth : 7.

Marking: {2* P_Bt, Rb_Op, Tr_Peca2, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T14}

-> T14 fire

Depth : 8.

Marking: {2* P_Bt, Rb_Op, Tr_Peca2, Rb_Ga_F, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T18}

-> T18 fire

Depth : 9.

Marking: {2* P_Bt, Rb_Op, Rb_Ga_A, Tr_Peca2, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T19}

-> T19 fire

Depth : 10.

Marking: {2* P_Bt, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T20}

-> T20 fire
Depth : 11.
Marking: {2* P_Bt, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_F, Tn_Op,
N_Medicao}
Bounds : {T21}

-> T21 fire
Depth : 12.
Marking: {2* P_Bt, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_F, Tn_Us,
N_Medicao}
Bounds : {T22}

-> T22 fire
Depth : 13.
Marking: {2* P_Bt, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T24}

-> T24 fire
Depth : 14.
Marking: {2* P_Bt, Rb_Op, Rb_Pos4, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T25, T26, T27, T28}

-> T28 fire
Depth : 15.
Marking: {2* P_Bt, Rb_Op, Rb_Pos4, Rb_Ga_F, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T32}

-> T32 fire
Depth : 16.

Marking: {2* P_Bt, Rb_Op, Rb_Pos4, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T33}

-> T33 fire
Depth : 17.
Marking: {2* P_Bt, Rb_Op, Tr_Peca4, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao}
Bounds : {T34}

-> T34 fire
Depth : 18.
Marking: {2* P_Bt, Rb_Op, Tr_Peca5, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
Medicao}
Bounds : {T35, T36, T37}

-> T35 fire
Depth : 19.
Marking: {2* P_Bt, Rb_Op, Tr_Peca6, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao}
Bounds : {T41}

-> T41 fire
Depth : 20.
Marking: {2* P_Bt, Rb_Op, Rb_Ga_A, Tr_Peca6, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao}
Bounds : {T50}

-> T50 fire
Depth : 21.
Marking: {2* P_Bt, P_Acab, Rb_Livre, Rb_Hom, Rb_Ga_A, Tn_Pl_A, Tn_P_A,
Tn_Op, N_Medicao}
Bounds : {T53}

-> T53 fire

Depth : 22.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos1, Rb_Ga_A, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T02}

-> T02 fire

Depth : 23.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos2, Rb_Ga_A, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T03, T04, T05, T06}

-> T06 fire

Depth : 24.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos2, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao, Clc_Carr}

Bounds : {T12}

-> T12 fire

Depth : 25.

Marking: {P_Bt, P_Acab, Rb_Op, Tr_Peca1, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao, Clc_Carr}

Bounds : {T13}

-> T13 fire

Depth : 26.

Marking: {P_Bt, P_Acab, Rb_Op, Tr_Peca2, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao, Clc_Carr}

Bounds : {T14}

-> T14 fire

Depth : 27.

Marking: {P_Bt, P_Acab, Rb_Op, Tr_Peca2, Rb_Ga_F, Tn_Pl_F, Tn_P_A, Tn_Op, N_Medicao, Clc_Carr}

Bounds : {T18}

-> T18 fire

Depth : 28.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Ga_A, Tr_Peca2, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T19}

-> T19 fire

Depth : 29.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}

Bounds : {T20}

-> T20 fire

Depth : 30.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_F, Tn_Op,
N_Medicao}

Bounds : {T21}

-> T21 fire

Depth : 31.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_F, Tn_Us,
N_Medicao}

Bounds : {T22}

-> T22 fire

Depth : 32.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}

Bounds : {T24}

-> T24 fire

Depth : 33.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos4, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op, N_Medicao, Clc_Desc}
Bounds : {T25, T26, T27, T28}

-> T28 fire
Depth : 34.
Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos4, Rb_Ga_F, Tn_Pl_F, Tn_P_A, Tn_Op, N_Medicao, Clc_Desc}
Bounds : {T32}

-> T32 fire
Depth : 35.
Marking: {P_Bt, P_Acab, Rb_Op, Rb_Pos4, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao, Clc_Desc}
Bounds : {T33}

-> T33 fire
Depth : 36.
Marking: {P_Bt, P_Acab, Rb_Op, Tr_Peca4, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}
Bounds : {T34}

-> T34 fire
Depth : 37.
Marking: {P_Bt, P_Acab, Rb_Op, Tr_Peca5, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, Medicao}
Bounds : {T35, T36, T37}

-> T35 fire
Depth : 38.
Marking: {P_Bt, P_Acab, Rb_Op, Tr_Peca6, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}
Bounds : {T41}

-> T41 fire

Depth : 39.

Marking: {P_Bt, P_Acab, Rb_Op, Rb_Ga_A, Tr_Peca6, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T50}

-> T50 fire

Depth : 40.

Marking: {P_Bt, 2* P_Acab, Rb_Livre, Rb_Hom, Rb_Ga_A, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T53}

-> T53 fire

Depth : 41.

Marking: {2* P_Acab, Rb_Op, Rb_Pos1, Rb_Ga_A, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T02}

-> T02 fire

Depth : 42.

Marking: {2* P_Acab, Rb_Op, Rb_Pos2, Rb_Ga_A, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T03, T04, T05, T06}

-> T06 fire

Depth : 43.

Marking: {2* P_Acab, Rb_Op, Rb_Pos2, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao, Clc_Carr}

Bounds : {T12}

-> T12 fire

Depth : 44.

Marking: {2* P_Acab, Rb_Op, Tr_Peca1, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao, Clc_Carr}

Bounds : {T13}

-> T13 fire
Depth : 45.
Marking: {2* P_Acab, Rb_Op, Tr_Peca2, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}
Bounds : {T14}

-> T14 fire
Depth : 46.
Marking: {2* P_Acab, Rb_Op, Tr_Peca2, Rb_Ga_F, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}
Bounds : {T18}

-> T18 fire
Depth : 47.
Marking: {2* P_Acab, Rb_Op, Rb_Ga_A, Tr_Peca2, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}
Bounds : {T19}

-> T19 fire
Depth : 48.
Marking: {2* P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Carr}
Bounds : {T20}

-> T20 fire
Depth : 49.
Marking: {2* P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_F, Tn_Op,
N_Medicao}
Bounds : {T21}

-> T21 fire
Depth : 50.

Marking: {2* P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_F, Tn_Us,
N_Medicao}
Bounds : {T22}

-> T22 fire
Depth : 51.
Marking: {2* P_Acab, Rb_Op, Rb_Pos3, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T24}

-> T24 fire
Depth : 52.
Marking: {2* P_Acab, Rb_Op, Rb_Pos4, Rb_Ga_A, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T25, T26, T27, T28}

-> T28 fire
Depth : 53.
Marking: {2* P_Acab, Rb_Op, Rb_Pos4, Rb_Ga_F, Tn_Pl_F, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T32}

-> T32 fire
Depth : 54.
Marking: {2* P_Acab, Rb_Op, Rb_Pos4, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao, Clc_Desc}
Bounds : {T33}

-> T33 fire
Depth : 55.
Marking: {2* P_Acab, Rb_Op, Tr_Peca4, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op,
N_Medicao}
Bounds : {T34}

-> T34 fire

Depth : 56.

Marking: {2* P_Acab, Rb_Op, Tr_Peca5, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, Medicao}

Bounds : {T35, T36, T37}

-> T37 fire

Depth : 57.

Marking: {2* P_Acab, Rb_Op, Tr_Peca8, Rb_Ga_F, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T49}

-> T49 fire

Depth : 58.

Marking: {2* P_Acab, Rb_Op, Rb_Ga_A, Tr_Peca8, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {T52}

-> T52 fire

Deadlock detected in this state.

Depth : 59.

Marking: {2* P_Acab, P_Refu, Rb_Livre, Rb_Hom, Rb_Ga_A, Tn_Pl_A, Tn_P_A, Tn_Op, N_Medicao}

Bounds : {}

APÊNDICE B – DETALHES DA INTEGRAÇÃO FÍSICA

Neste apêndice são apresentados os desenhos dos componentes mecânicos projetados e construídos durante a integração física da célula. Nos desenhos (figuras B1 a B4) estão especificados as principais dimensões que cada componente possui. Maiores detalhes sobre o projeto e construção destes componentes podem ser obtidos a partir da *URL*: http://webfmc.graco.unb.br/downloads/documentation/cad/

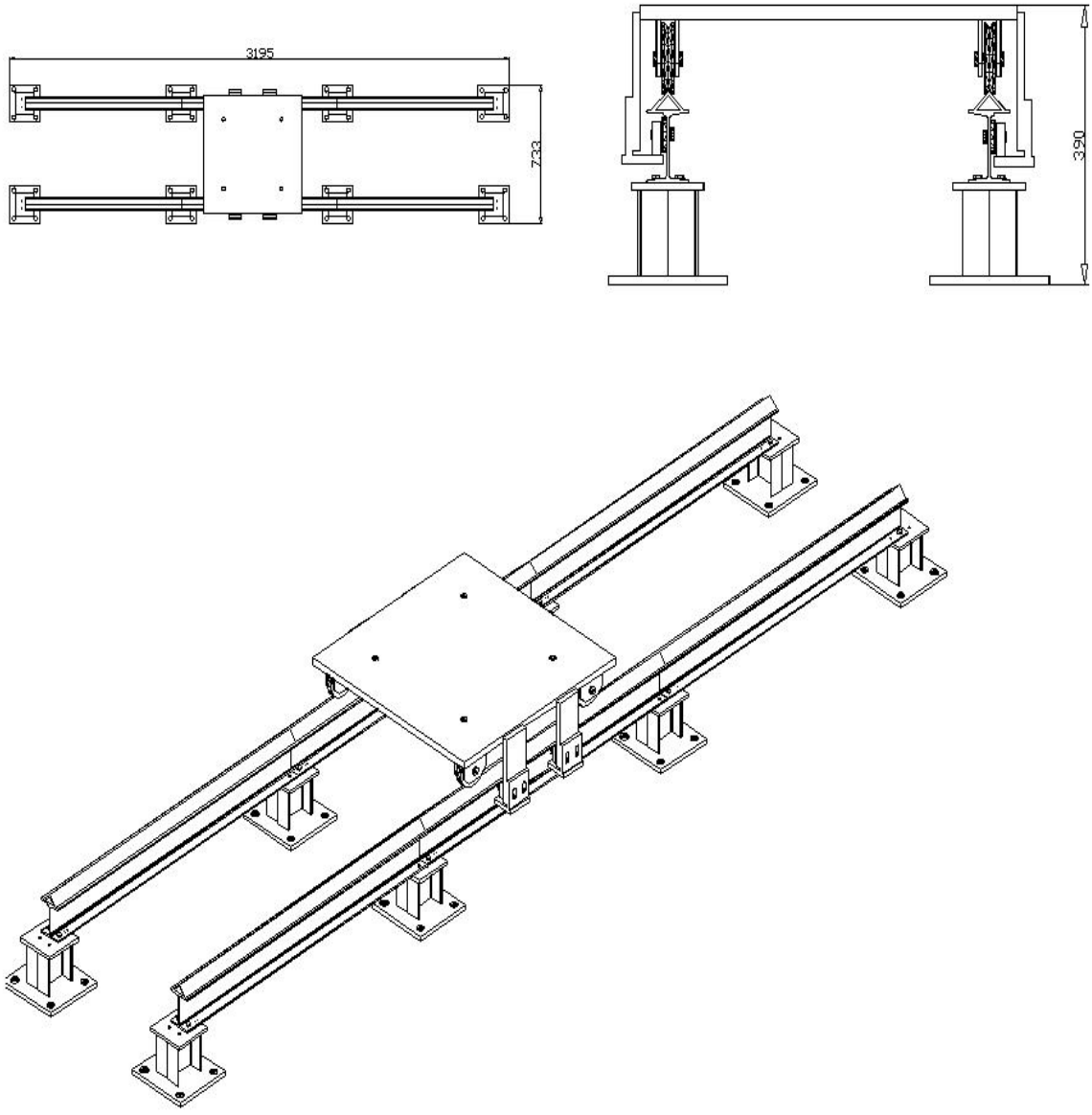

Figura B.1 - Desenho do carro posicionador.

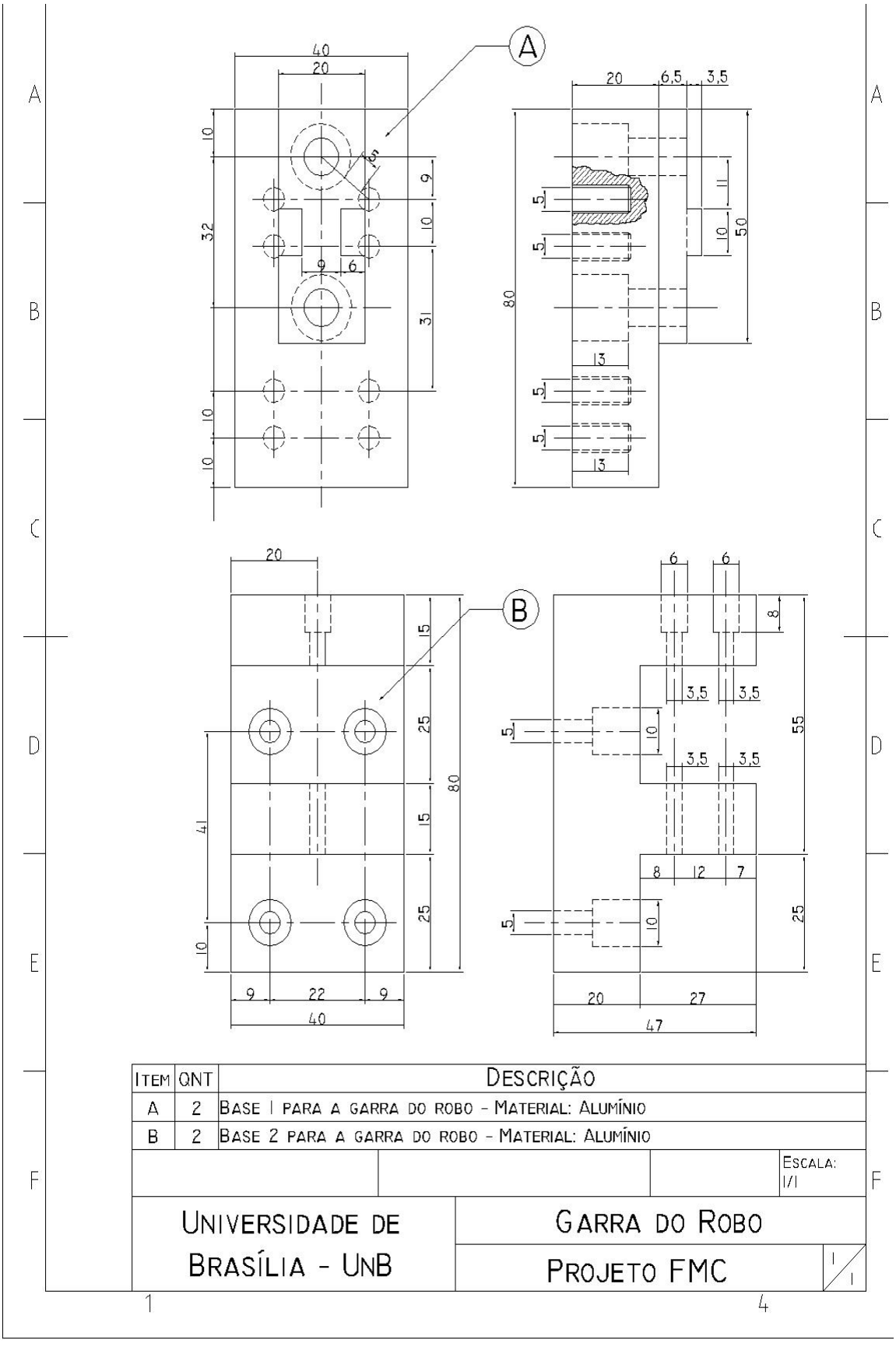

Figura B.2 - Desenho dos dedos da garra do robô.

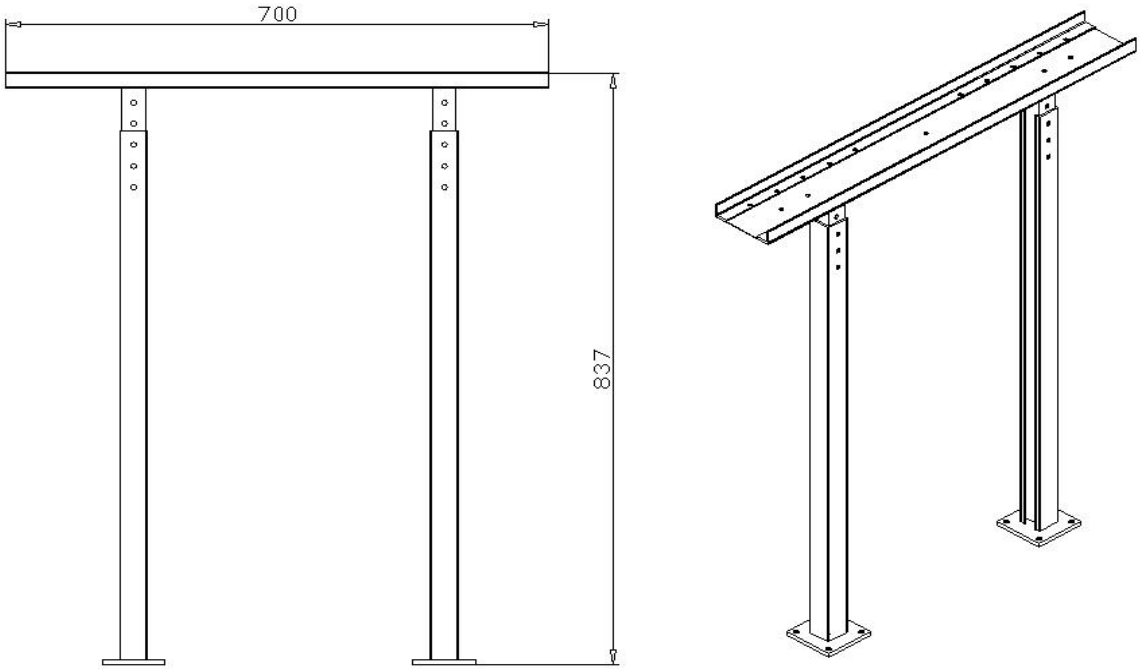

Figura B.3 - Desenho do suporte para o micrômetro.

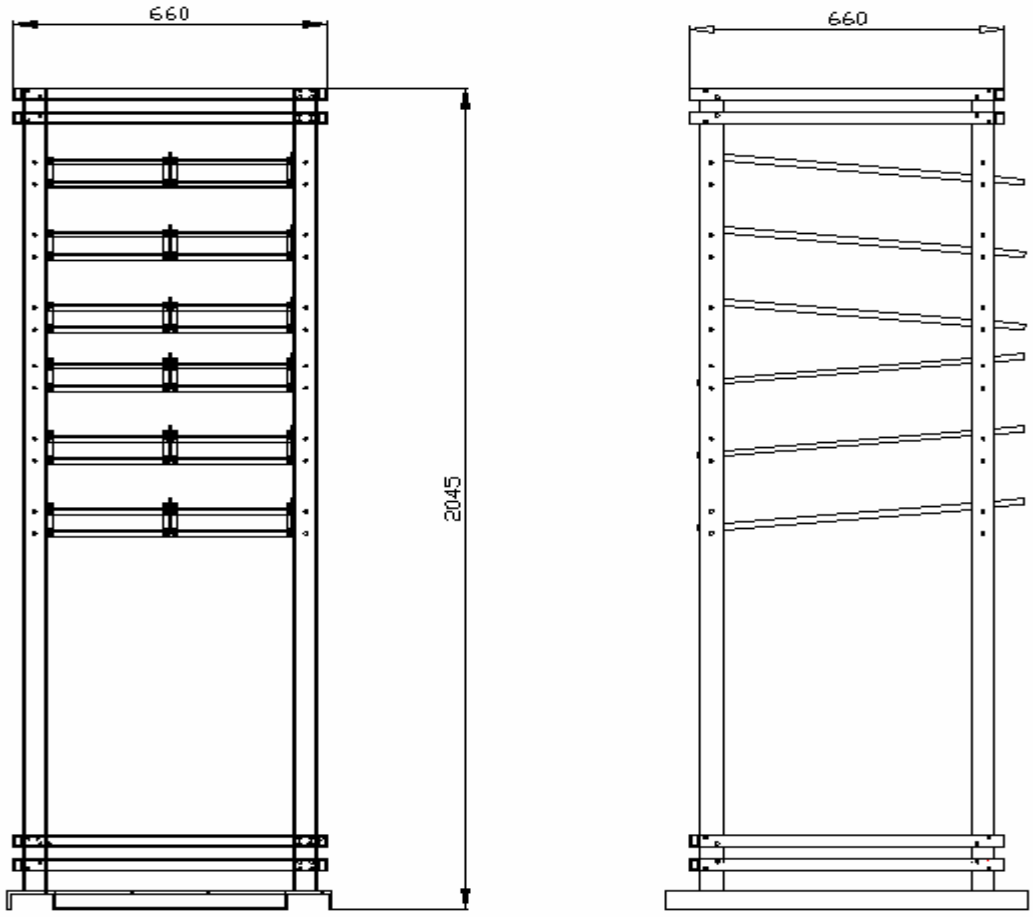

Figura B.4 - Desenho do pallet armazenador de peças.

APÊNDICE C – DETALHES DA INTEGRAÇÃO ROBO-TORNO-MICRÔMETRO

Os principais documentos adicionais da integração Robô-Torno-Micrômetro estão descritos neste apêndice. As primeiras figuras apresentam os diagramas elétricos da linha de emergência (figura C.1), da conexão Robô-Torno (figura C.2) e da conexão Micrômetro-Torno (figura C.3). Já as figuras C.4 e C.5 apresentam o fluxograma detalhado da programação de um ciclo de fabricação, enquanto a tabela C.1 demonstra a programação detalhada do manipulador robótico. Estes documentos também estão disponíveis no site do projeto e podem ser obtidos a partir da *URL*: http://webfmc.graco.unb.br/downloads/documentation/integration/.

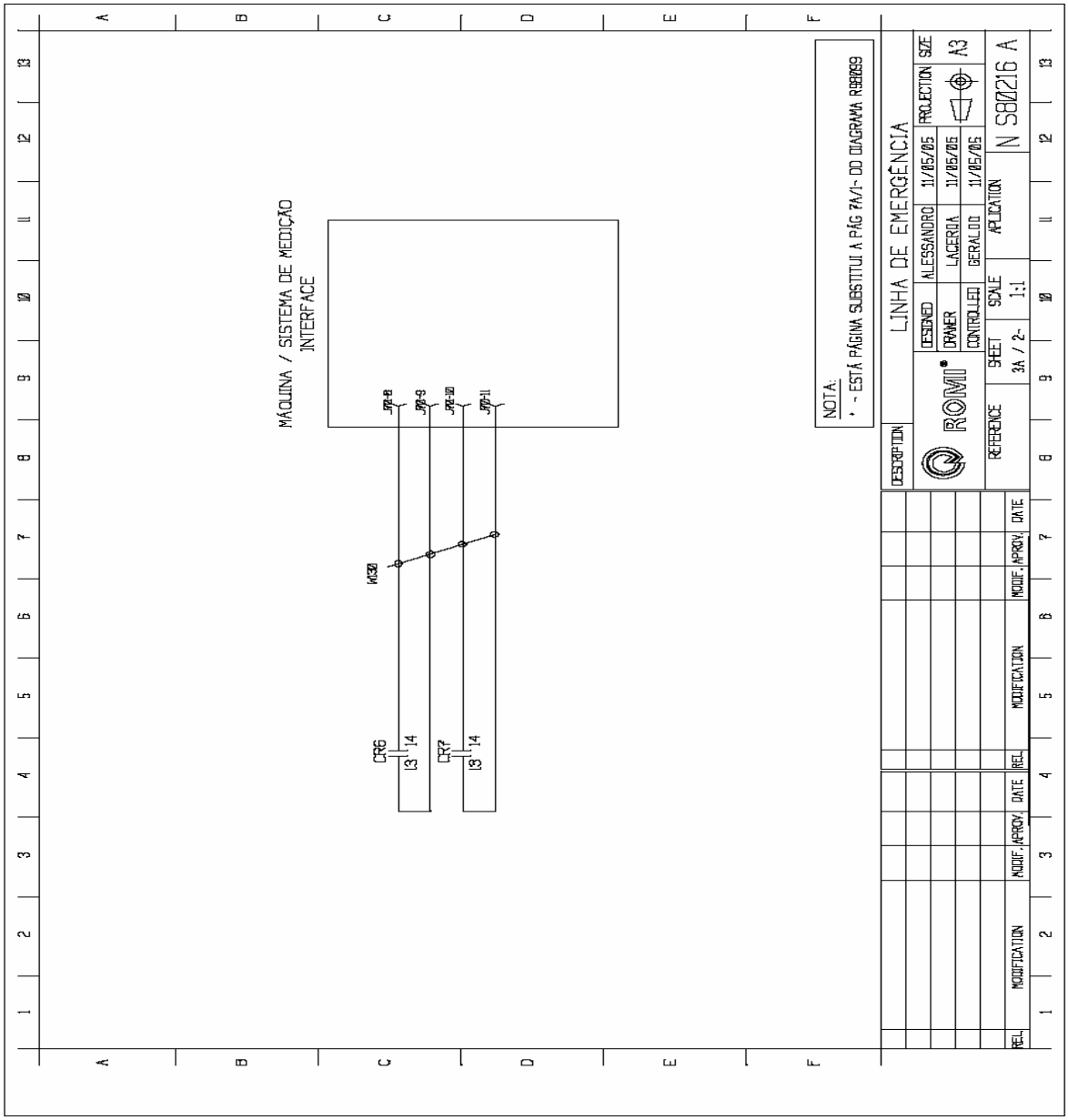

Figura C.1 - Diagrama elétrico da linha de emergência.

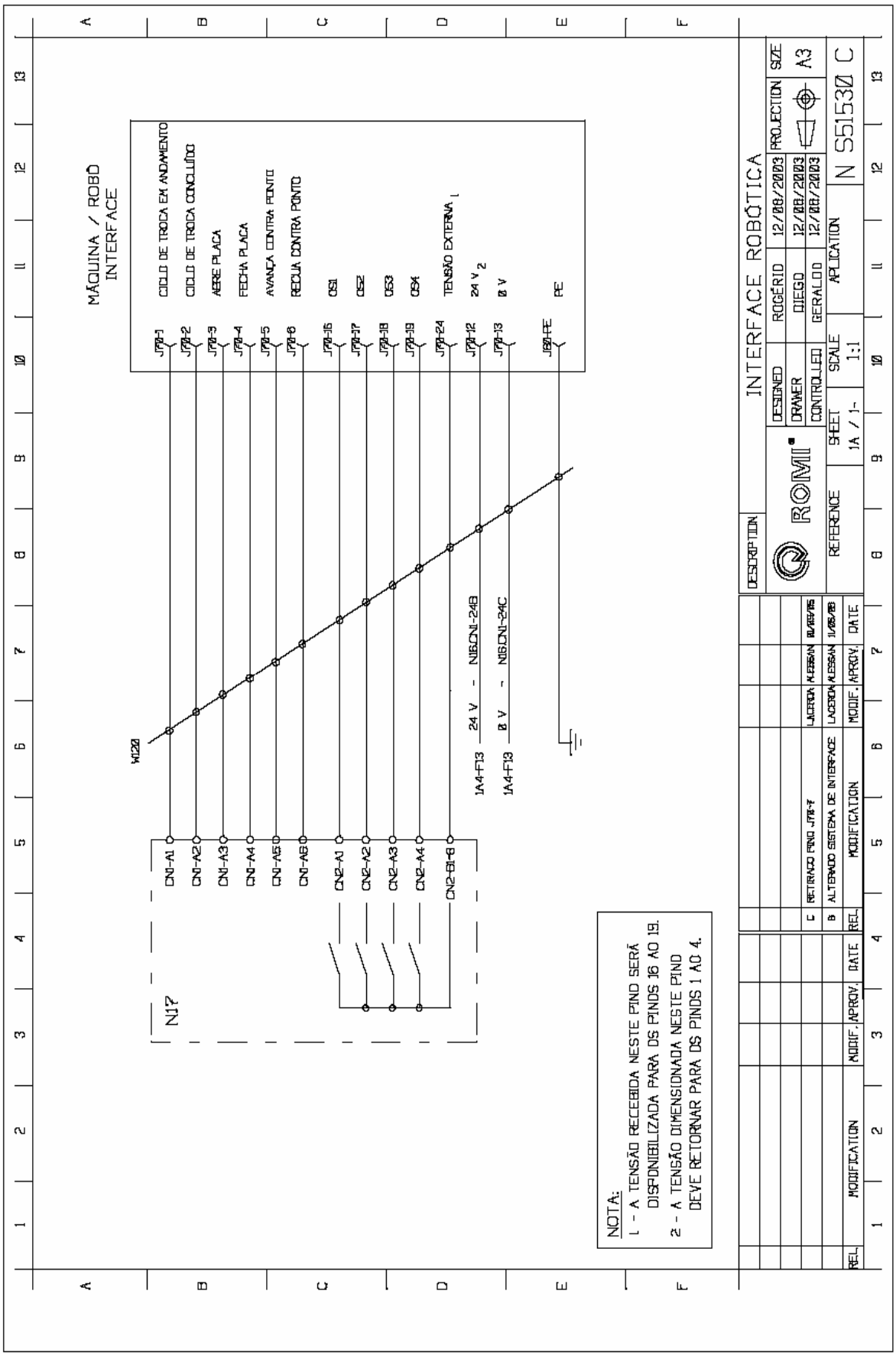

Figura C.2 - Diagrama elétrico da conexão com a interface robótica.

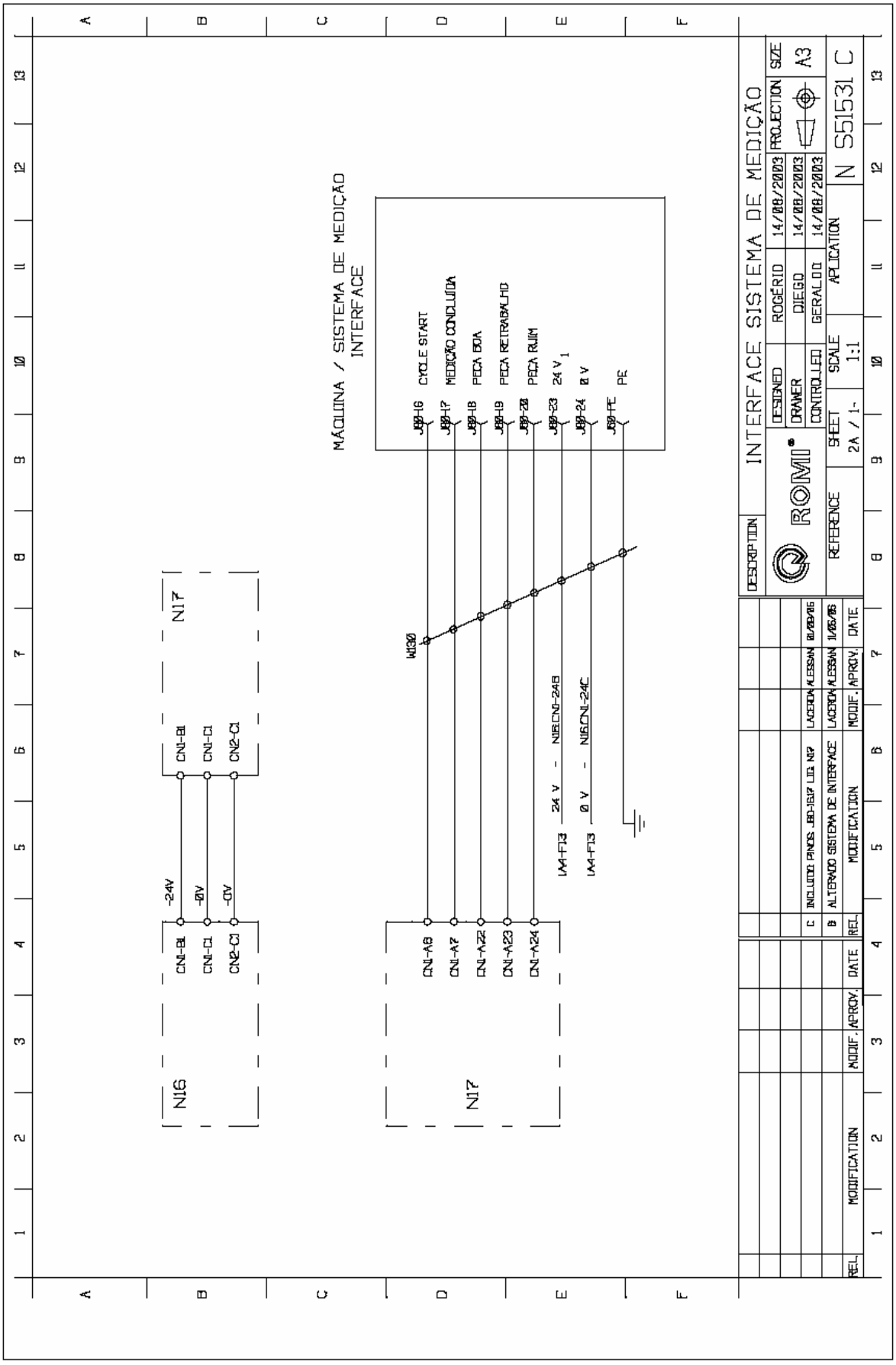

Figura C.3 - Diagrama elétrico da conexão com o sistema de medição.

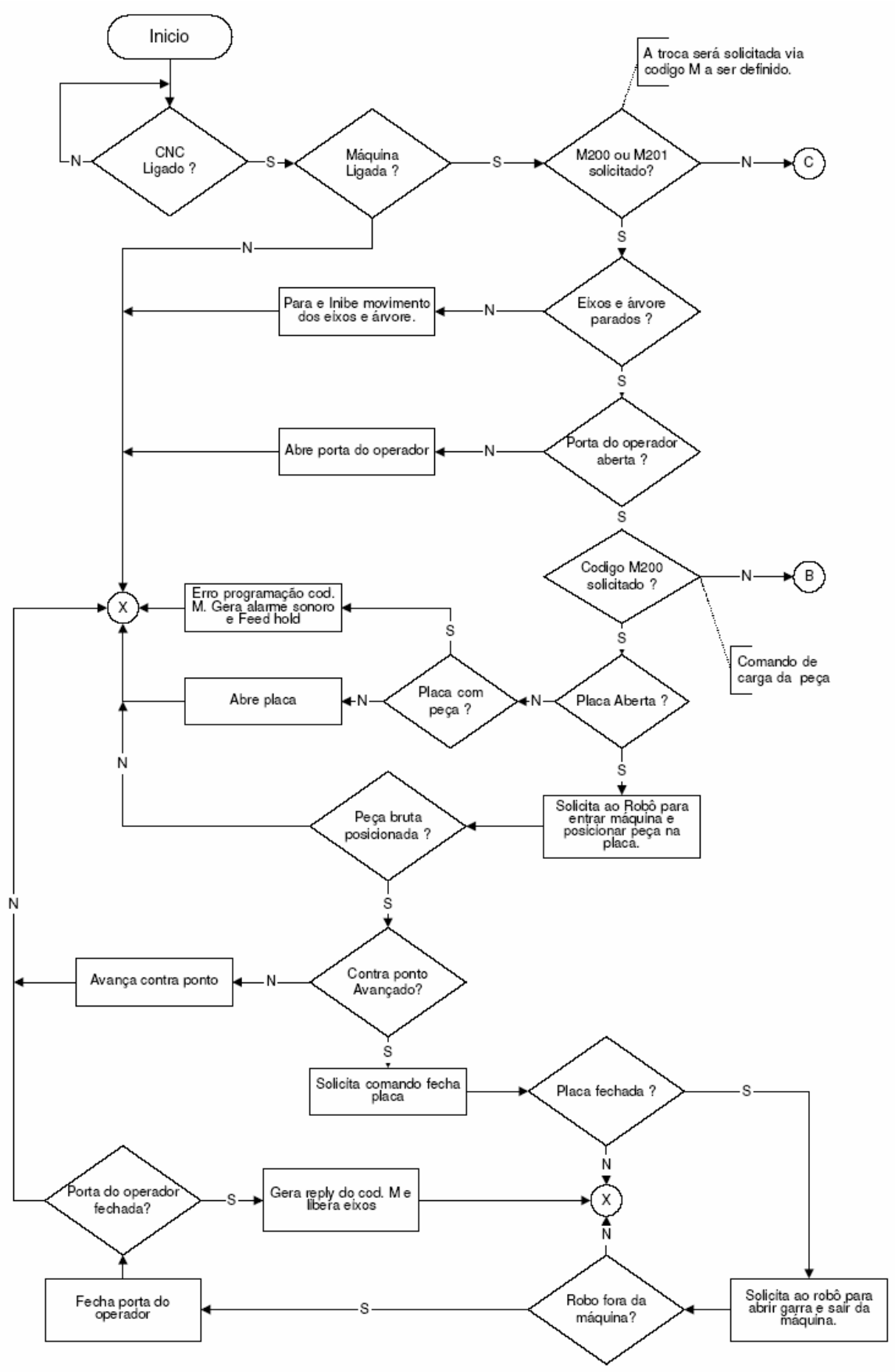

Figura C.4 - Fluxograma detalhado de programação de um ciclo de fabricação. – Parte 1.

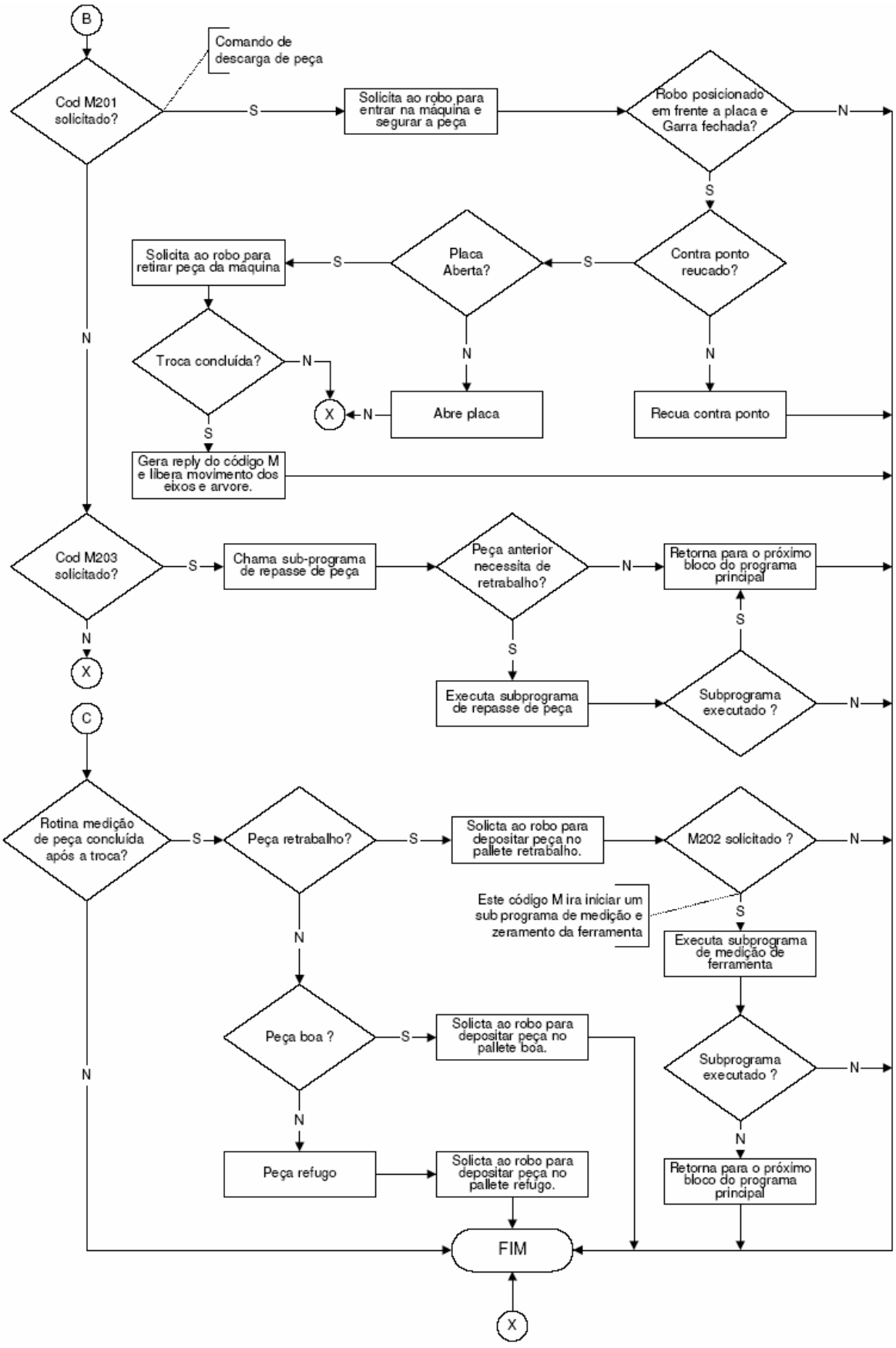

Figura C.5 - Fluxograma detalhado da programação de um ciclo de fabricação – Parte 2.

Tabela C.1 – Programação detalhada do manipulador robótico.

PROGRAMA PRINCIPAL

INSTR	DESCRIÇÃO
10	V = 1000mm/s to 2000mm/s
20	TCP9
30	V 100%
40	Gripper wait 0.5s
50	V 100%
60	Gripper wait 1s
70	V 100%
80	V 100%
90	CALL 87
100	SET OUTPUT1
110	Call 89
120	RESET OUTPUT1
130	CALL 86
140	Wait 3s
150	V 5%
160	Call 89
170	V 1%
180	SET OUTPUT5
190	CALL89
200	RESET OUTPUT5
210	CALL 85
220	Wait 3s
230	SET OUTPUT4
240	CALL89
250	RESET OUTPUT4
260	CALL 84
270	V 100%
280	SET OUTPUT2
290	CALL89
300	RESET OUTPUT2
310	CALL 83
320	SET OUTPUT1
330	Call 89
340	RESET OUTPUT1
350	CALL86
360	V 5%
370	Gripper wait 0.5s
380	SET OUTPUT6
390	CALL89
400	RESET OUTPUT6
410	CALL82
420	SET OUTPUT3
430	CALL89
440	RESET OUTPUT3
450	CALL81
460	Wait 3s
470	V 1%
480	V 5%
490	Wait 3s
500	SET OUTPUT2
510	CALL89
520	RESET OUTPUT2
530	CALL 83
540	V100%
550	Wait 3s
560	V100%
570	Gripper wait 1s
580	V100%
590	Gripper wait 0.5s
600	Return

SUBPROGRAMA Nº 87

INSTR	DESCRIÇÃO
10	WAIT INPUT4=1
20	WAIT INPUT3=0
30	WAIT INPUT2=0
40	WAIT INPUT1=0
50	Return

SUBPROGRAMA Nº86

INSTR	DESCRIÇÃO
10	WAIT INPUT4=0
20	WAIT INPUT3=0
30	WAIT INPUT2=1
40	WAIT INPUT1=1
50	Return

SUBPROGRAMA Nº85

INSTR	DESCRIÇÃO
10	WAIT INPUT4=0
20	WAIT INPUT3=1
30	WAIT INPUT2=0
40	WAIT INPUT1=1
50	Return

SUBPROGRAMA Nº84

INSTR	DESCRIÇÃO
10	WAIT INPUT4=0
20	WAIT INPUT3=0
30	WAIT INPUT2=1
40	WAIT INPUT1=0
50	Return

SUBPROGRAMA Nº83

INSTR	DESCRIÇÃO
10	WAIT INPUT4=1
20	WAIT INPUT3=0
30	WAIT INPUT2=0
40	WAIT INPUT1=1
50	Return

SUBPROGRAMA Nº82

INSTR	DESCRIÇÃO
10	WAIT INPUT4=0
20	WAIT INPUT3=1
30	WAIT INPUT2=1
40	WAIT INPUT1=0
50	Return

SUBPROGRAMA Nº81

INSTR	DESCRIÇÃO
10	WAIT INPUT4=0
20	WAIT INPUT3=0
30	WAIT INPUT2=0
40	WAIT INPUT1=1
50	Return

APÊNDICE D – ARQUITETURA MVC E PADRÕES DE PROJETO

D.1 – PADRÃO DE ARQUITETURA MVC

O conceito da arquitetura Modelo – Visão – Controle (MVC) foi inicialmente apresentado pela linguagem de programação *SmallTalk*, sendo muito utilizado em *Smalltalk*-80 para construir interfaces gráficas para o usuário (Chen *and* Lu, 2002). Esta arquitetura é particularmente bem adequada para aplicações *Web* interativas, em que os usuários interagem com um *website*, com múltiplas páginas interativas e múltipos *round-trip* requerendo e mostrando dados (Sun BluePrints, 2002).

Nesta arquitetura as funcionalidades do aplicativo são divididas em camadas. Cada camada possui uma responsabilidade específica associada a execução de uma atividade. O esforço inicial em construir o aplicativo em camadas é visivelmente compensado quando o mesmo necessita ser expandido ou modificado, isto porque ao dividir as funcionalidades entre camadas, o grau de acoplamento entre os objetos é reduzido.

A arquitetura MVC divide as funcionalidades em três camadas:

- ✓ Modelo;
- ✓ Visão;
- ✓ Controle.

A camada Modelo é composta por entidades que representam o modelo da informação do sistema. Os objetos que compõem esta camada constituem uma aproximação de software dos objetos do mundo real. A camada Visão traduz as informações do modelo para o meio externo, sendo estas informações traduzidas por meio de interfaces gráficas utilizadas tanto para apresentação dos dados quanto para capturar as interações com o usuário. A camada Controle implementa a lógica de funcionamento do sistema recebendo as solicitações do usuário e modificando adequadamente o modelo para refletir a iteração com o sistema. A figura D.1 mostra a arquitetura MVC e as iterações entre as camadas e o meio externo.

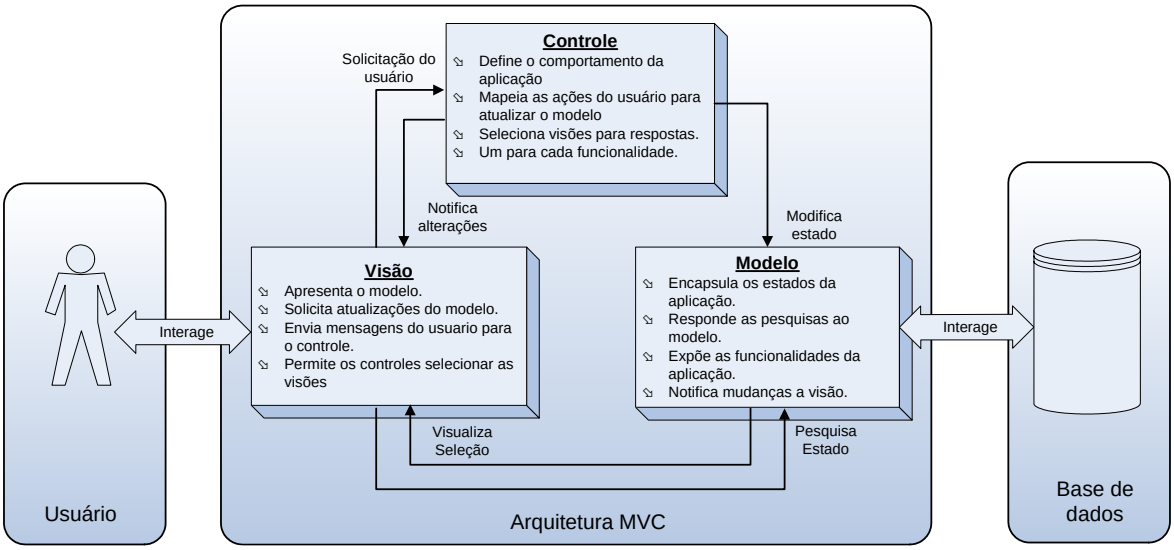

Figura D.1 - Arquitetura MVC.

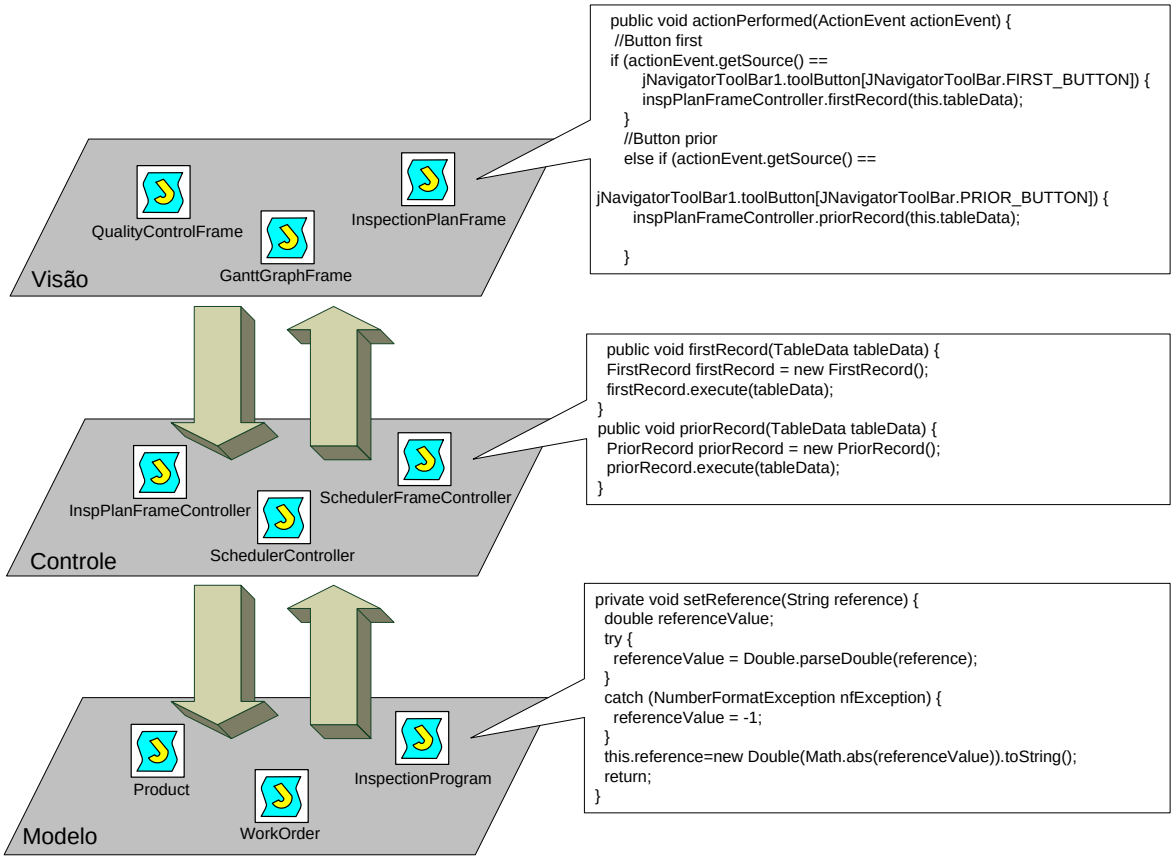

Figura D.2 - Unidade de Gerenciamento sob a arquitetura MVC.

Durante a implementação da Unidade de Gerenciamento três pacotes (*Model*, *View e Controller*) foram definidos para separar as classes de acordo com suas respectivas funcionalidades. No pacote *Model* são encontradas as classes que encapsulam o modelo da informação do sistema

enquanto nos pacotes *View* e *Controller* as *GUI's* e as controladoras da Unidade de Gerenciamento respectivamente.

InspectionPlanFrame é uma classe que descreve a *GUI InspectionPlanFrame* utilizada para apresentar os programas de inspeção cadastrados. Ao se invocar o método *actionPerformed(ActionEvent actionEvent)* desta classe e sendo o *firstButton* o botão pressionado, uma mensagem é enviada à controladora de *frame* (*InspPlanFrameController*) solicitando a execução do serviço *firstRecord(this.tableData).*

Ao se invocar o método *firstRecord()* da controladora *InspPlanFrameController*, um objeto *firstRecord* é instanciado e recebe como argumento um objeto *TableData*. *FirstRecord* é um objeto *command* cuja finalidade é modificar os atributos de *TableData* para que a primeira linha de uma tabela seja a linha selecionada.

D.2 – UTILIZANDO PADRÕES DE PROJETO NA IMPLEMENTAÇÃO

> Projetar software orientado a objeto é duro, mas projetar software reutilizável orientado a objetos é ainda mais duro (Gamma *et at*, 2000).

O arquiteto Chistopher Alexander propôs em 1977 o conceito de Padrões de Projeto. Segundo Alexander (Alexander *et at*, 1977) cada padrão descreve um problema particular e uma solução núcleo, de tal forma que esta solução possa ser utilizada várias vezes, sem nunca fazê-la da mesma maneira.

Posteriormente, este conceito foi estendido para diversas áreas do conhecimento (engenharia de software, sistemas de controle, sistemas da informação, dentre outras). Os padrões de projeto constituem uma forma de capturar, documentar, organizar e disseminar o conhecimento existente de uma dada área em um formato consistente e acessível (Spinellis, 2001). A idéia por trás deste conceito é utilizar a experiência acumulada na solução de problemas e utilizá-las repetidas vezes, evitando que o projetista redescubra-a novamente.

Na engenharia de software, Gamma (Gamma *et at*, 2000) propôs um catálogo contendo 23 padrões que podem ser utilizados na solução de problemas comuns enfrentados no dia – a – dia em desenvolvimento de software. Cada padrão compreende um número de partes, incluindo a

motivação, a aplicabilidade, a estrutura da solução e exemplos de implementação (Hannemann *et at*, 2002). Basicamente um padrão é descrito através de quatro elementos:

- ✓ O nome do padrão;
- ✓ O problema;
- ✓ A solução;
- ✓ As conseqüências.

O nome é uma palavra-chave utilizada para descrever sucintamente algo que relembre o padrão. Esta palavra-chave deve lembrar o projetista em qual contexto o padrão se aplica. O problema é a descrição do contexto, em outras palavras, o problema que se propõem em resolver. O terceiro elemento é a exposição da estrutura utilizada na solução do problema, especificando seus componentes, particularidades, aplicabilidade, dentre outros. As conseqüências são as análises e descrições das vantagens e desvantagens de se utilizar o padrão.

Gamma (Gamma *et at*, 2000) organizou um catálogo classificando os padrões por meio de dois critérios: finalidade e escopo. Os padrões podem ter finalidade de criação (criar objetos), estrutural (lidam com o comportamento de classes e objetos) e comportamental (as formas como os objetos interagem e distribuem suas responsabilidades). O escopo é a especificação ao qual o padrão se aplica. Os padrões com escopo de classe lidam com relacionamentos estabelecidos através do mecanismo de herança ao passo que os padrões com escopo de objetos lidam com relacionamentos dinâmicos.

Tabela D.1 - O espaço dos padrões de projeto (Gamma *et at*, 2000).

		Finalidade		
		Criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter (class)	Interpreter
				Template method
	Objeto	Abstract Factory	Adapter (object)	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singletron	Decorator	Mediator
			Façade	Menento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

A tabela D.1 apresenta os 23 padrões de projetos GoF classificados de acordo com a finalidade e com o escopo. Dos padrões apresentados *Builder, Singletron* e *Command* foram os utilizados no

desenvolvimento da Unidade de Gerenciamento. Desse modo, a explicação detalhada dos demais padrões foge do escopo dessa dissertação.

D.2.1 – Builder

O processo de inicialização de um sistema é composto basicamente pela criação dos objetos estruturais e pelo estabelecimento de suas respectivas relações. Em sistemas complexos, em que o número de objetos e relações é grande, separar a construção de um objeto da sua respectiva representação permite que o mesmo processo de construção possa criar diferentes representações.

Builder é o padrão de projeto criado com este propósito. Este padrão separa a construção de um sistema de sua respectiva representação encapsulando o processo de criação em um único objeto. Este objeto deve existir somente durante a criação de um sistema que se resume em instanciar os objetos estruturais e estabelecer suas respectivas relações.

A figura D.3 mostra o diagrama de classes que descreve como o padrão de projeto *Builder* foi aplicado no desenvolvimento da Unidade de Gerenciamento. O método *main()* da classe *Initialization* é invocado toda vez que a MgU é inicializada. No corpo deste método o objeto *NavigatorController* é instanciado. Este objeto estrutural é responsável pela navegabilidade entre os módulos da MgU.

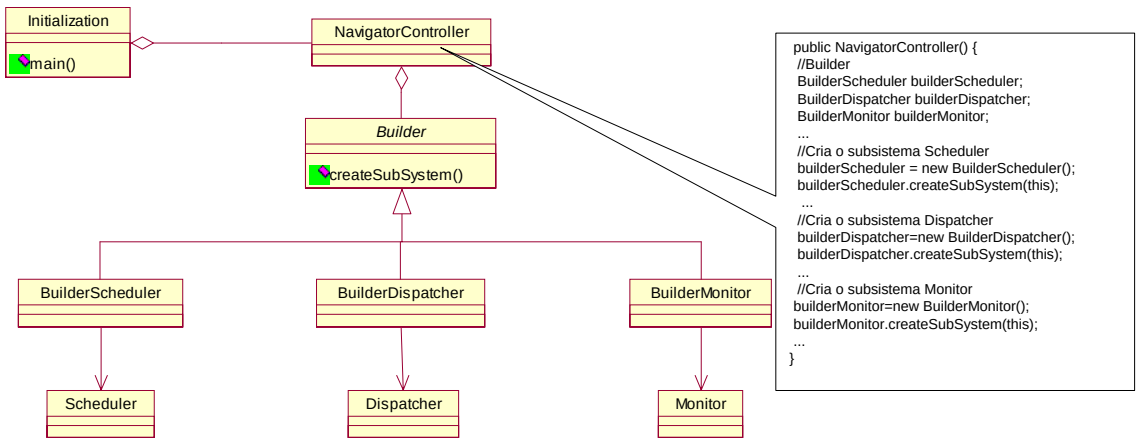

Figura D.3 - Utilizando o padrão de projeto *Builder*.

Ao se invocar o construtor da classe *NavigatorController*, os objetos *BuilderScheduler*, *BuilderDispatcher* e *BuilderMonitor* são instanciados (ver figura D.3). Estes objetos *extends* a classe abstrata *Builder*, consequentemente devem implementar o método *createSubSystem()*. Cada um destes objetos *Buider* implementa este método no intuito de descrever passo a passo a construção do subsistema ao qual faz referência.

Como a funcionalidade de um objeto *Builder* se limita a criação, após a execução desta atividade, estes objetos podem ser descartados. Por esse motivo *builderScheduler*, *builderDispatcher* e *builderMonitor* são declaradas como variáveis locais, em outras palavras, ao finalizar a execução do construtor do objeto *NavigatorController*, as referências que aponta para os objetos *BuilderScheduler*, *BuiderDispatcher* e *BuilderMonitor* são destruídas indicando que os mesmos podem ser coletados pelo *Java Garbage Colection*[2].

D.2.2 – Singletron

Na camada de persistência da MgU estão armazenadas as classes utilizadas na comunicação com os dispositivos externos. Classes como *TurningCenterController*, *AGVController* e *MicrometerController* do pacote *persistence.controller* implementam a lógica de comunicação com o Centro de Torneamento, o AGV e com o micrometro respectivamente considerando as particularidades de cada estação de trabalho (protocolo de comunicação, formato das mensagens, etc).

Como estas controladoras possuem métodos que acessam diretamente os dispositivos externos é importante garantir que exista somente uma instancia de cada uma dessas classes e que essas instâncias sejam facilmente acessíveis. Utilizar variáveis globais resolve o problema de acesso ao objeto instanciado, mas não inibe a criação de novos objetos (Gamma *et at*, 2000). Neste caso, utiliza-se o padrão *Singleton* para garantir que a classe tenha somente uma instância no sistema.

A figura D.4 apresenta o diagrama de classes que representa a utilização deste padrão de projeto no desenvolvimento da Unidade de Gerenciamento. O objeto *DataBaseController* é a controladora responsável pela comunicação com a base de dados. Este objeto possui uma

[2] Coletor de lixo Java.

variável global estática (*dataBaseController*), e um método público estático, *getInstance()*, e um construtor *private*.

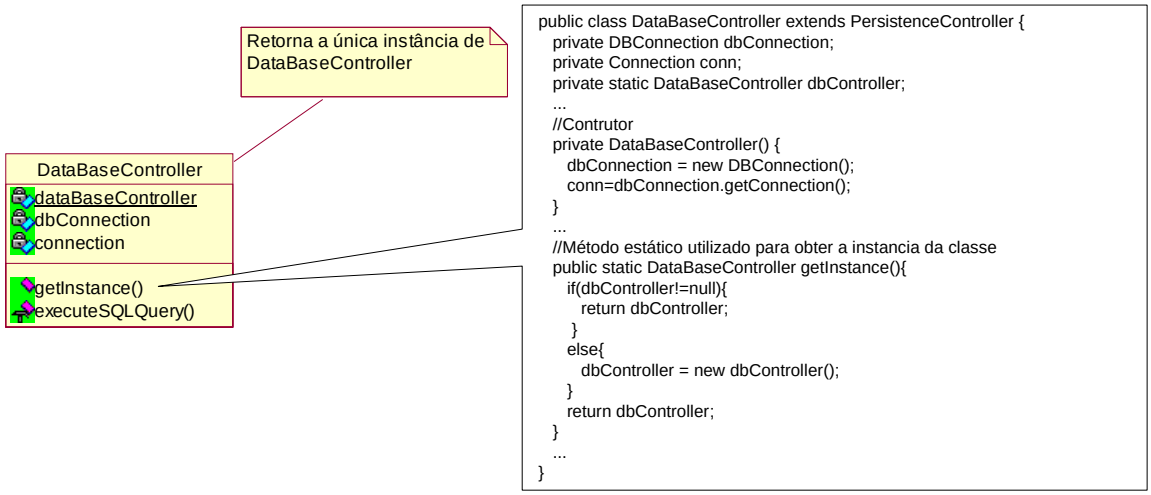

Figura D.4 - Utilizando o padrão de projeto *Singleton*.

Adicionar o modificador de acesso *private* no construtor da *DataBaseController* garante que somente *DataBaseController* instancia-se a si mesma. Se qualquer outra classe tentar instanciar *DataBaseController*, um erro de compilação é gerado. O método estático *getInstance()* retorna uma variável estática *dataBaseController*. No corpo deste método uma estrutura *IF* verifica se esta variável estática aponta para uma referência nula. Se apontar, a execução do método é finalizada com o retorno da variável estática *dataBaseController*, caso contrário, o objeto *DataBaseController* é instanciado e a variável que faz referência a este objeto é retornada. Esta lógica em que se estrutura o método *getInstance()* garante que exista somente uma instância do objeto *DataBaseController* em todo o sistema.

D.2.3 – Command

O objetivo deste padrão é desacoplar a relação entre o objeto que solicitou o serviço e o objeto que fornece o serviço (Cinnéide, 2000). Isto porque em alguns casos, nada se sabe sobre a operação solicitada ou até mesmo sobre o objeto que recebera a solicitação. Nestes casos, padrão *Command* é utilizado para encapsular a solicitação. A figura D.5 mostra a aplicação deste padrão de projeto no desenvolvimento da MgU.

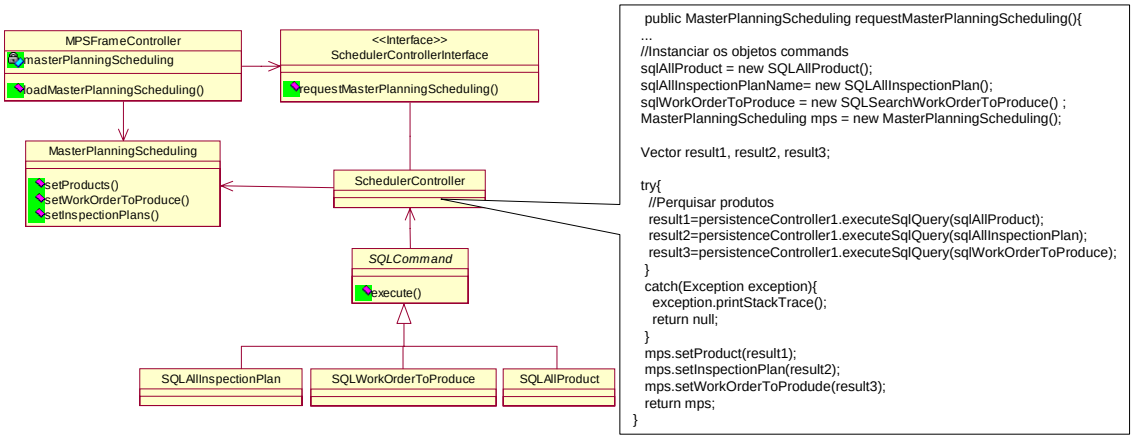

Figura D.5 - Utilizando o padrão de projeto *Command*.

Quando o usuário solicita o frame *MasterPlanningSchedulingFrame*, o objeto *MPSFrameController* invoca o método *requestMasterPlanningScheduling()* declarado na interface *SchedulerControllerInterface*. *SchedulerController* é a classe que implementa a interface *SchedulerControllerInterface* e portanto deve implementar o método *requestMasterPlanningScheduling()*.

No corpo deste método três objetos *Command* são instanciados (*SQLAllProduct, SQLAllInspectionProgram e SQLWorkOrderToProduce*). A cada um destes objetos é atribuída a responsabilidade de construir a instrução SQL a ser utilizada na pesquisa da base de dados. O objeto *DataBaseConnection* implementa o método *sqlQueryExecute(SQLCommand sqlCommand)* que recebe como argumento um objeto *SQLCommand*. Como os objetos *SQLAllProduct*, *SQLAllInspectionPlan* e *SQLWorkOrderToProduce extends SQLCommand*, eles podem ser passados como argumentos para o método *sqlExecuteQuery()*.