PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 42/95

Design Baseado em Troca de Mensagens entre Subsistemas Autônomos de um Sistema para Controle e Monitoramento de um Processo de Litografia por Feixe de Elétrons
- COMONLIFE -

Maria Luiza d'Almeida Sanchez
Bruno Maffeo

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 42/95
Editor: Carlos J. P. Lucena Dezembro, 1995

Design Baseado em Troca de Mensagens entre Subsistemas Autônomos de um Sistema para Controle e Monitoramento de um Processo de Litografia por Feixe de Elétrons - COMONLIFE - *

Maria Luiza d'Almeida Sanchez **
Bruno Maffeo

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

** Universidade Federal Fluminense, Departamento de Engenharia de Telecomunicações.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br

Design Baseado em Troca de Mensagens entre Subsistemas Autônomos de um Sistema para Controle e Monitoramento de um Processo de Litografia por Feixe de Elétrons
- COMONLIFE -

Maria Luiza d'Almeida Sanchez
UFF- Departamento de Engenharia de Telecomunicações
DI/PUC - Rio
e-mail: mluiza@caa.uff.br
Bruno Maffeo
DI/PUC - Rio
e-mail: maffeo@inf.puc-rio.br

PUC - Rio. Inf. MCC - 42/95

Abstract:
This work presents the design of COMONLIFE, which is based on the exchange of messages between Autonomous Subsystems. It pursues the emphasis on the experimental approach to Software Engineering initiated with the conceptual modeling of this same system. The main objective is the assessing of a design method which deeply exploits the information hiding concept and the partitioning of a complex system into more manageable subsystems intended for reuse in other systems that have to perform similar functions.

Keywords :
Real-Time Systems, Process-Control Systems, Systems Design, Message Passing, Authonomous Subsystems, Object Oriented Design

Resumo:
Este trabalho apresenta o design do COMONLIFE, o qual baseia-se na Troca de Mensagens entre Subsistemas Autônomos. Prossegue-se aqui a ênfase aplicada à vertente experimental da Engenharia de Software iniciada com a modelagem conceitual desse mesmo sistema. O objetivo principal é a valiação de um método de design que explora fundamentalmente o domínio da complexidade de sistemas através do particionamento em subsistemas de menor porte, independentes uns dos outros e projetados para reúso, onde o critério mais importante de segmentação do sistema é baseado no encapsulamento de dados.

Palavras Chaves:
Sistemas de Tempo-Real, Sistemas Processo-Controle, Design de Sistemas, Troca de Mensagens, Subsistemas Autônomos, Design Orientado a Objetos

Sumário

1 INTRODUÇÃO **1**

2 SÍNTESE DO MODELO DA ESSÊNCIA **6**

2.1 RELAÇÃO ENTRE SERVIÇOS E EVENTOS EXTERNOS 6
2.2 SERVIÇOS X ASSUNTOS 8
2.3 DERIVANDO SUBSISTEMAS BÁSICOS 10
2.3.1 Alocação de Atividades Essenciais a Classes *12*
2.3.2 Determinação das Portas *20*
2.4 DERIVANDO ORGANIZAÇÃO HIERÁRQUICA 21

3 PROJETO BÁSICO **23**

3.1 VISÃO DE CONTEXTO 23
3.1.1 S0 - Comonlife *23*
3.1.2 Agregações de Entidades *24*
3.1.3 Agregações de Fluxos *25*
3.2 ESTRUTURA DO SISTEMA 26
3.2.1 S0 - COMONLIFE *26*
3.2.2 Gerencia Geração de Camada *32*
3.2.3 Controla Elementos de Traçado *37*
3.3 DIAGRAMA DOS SUBSISTEMAS BÁSICOS 42
3.4 ÁRVORE DE SUBSISTEMAS 43

4 CONCLUSÃO **44**

BIBLIOGRAFIA **46**

APÊNDICES

A. SUBSISTEMA BÁSICO - INTERFACE USUÁRIO MÁQUINA **48**

A.1 ESPECIFICAÇÃO 48
A.1.1 Descrição *48*
A.1.2 Diagrama de Contexto *48*
A.1.3 Descrição dos Serviços *48*
A.1.3.1 Serviços Externos 48
A.1.3.2 Serviços Internos 50
A.1.3.3 Serviços Requisitados 50
A.1.4 Descrição Detalhada dos Dados *51*
A.1.4.1 Definições de Tipos 51
A.1.4.2 Estruturas de Dados Encapsuladas 52
A.1.4.3 Estrutura das Mensagens 52
A.2 DESIGN DO SUBSISTEMA BÁSICO 53
A.2.1 Diagrama de Estrutura *53*
A.2.2 Diagrama de Comportamento *54*
A.2.3 Descrição dos Processos *54*
A.3 SISTEMAS ONDE É EMPREGADO 57

B. SUBSISTEMA BÁSICO - MONITORA MICROSCÓPIO **58**

B.1 ESPECIFICAÇÃO 58
B.1.1 Descrição *58*
B.1.2 Diagrama de Contexto *58*
B.1.3 Descrição dos Serviços *58*
B.1.3.1 Serviços Externos 58
B.1.3.2 Serviços Internos 59
B.1.3.3 Serviços Requisitados 59
B.1.4 Descrição Detalhada dos Dados *60*

B.1.4.1 Definições de Tipos 60
B.1.4.2 Estruturas de Dados Encapsuladas 60
B.1.4.3 Estrutura das Mensagens 60
B.2 DESIGN DO SUBSISTEMA BÁSICO 60
B.2.1 Diagrama de Estrutura *60*
B.2.2 Diagrama de Comportamento *61*
B.2.3 Descrição dos Processos *61*
B.3 SISTEMAS ONDE É EMPREGADO 63

C. SUBSISTEMA BÁSICO: GERA MÁSCARA 64

C.1 DESCRIÇÃO 64
C.1.1 Diagrama de Contexto *64*
C.1.2 Descrição dos Serviços *64*
C.1.2.1 Serviços Externos 64
C.1.2.2 Serviços Internos 67
C.1.2.3 Serviços Requisitados 67
C.1.3 Descrição Detalhada dos Dados *68*
C.1.3.1 Definições de Tipos 68
C.1.3.2 Estruturas de Dados Encapsuladas 69
C.1.3.3 Estrutura das Mensagens 70
C.2 DESIGN DO SUBSISTEMA BÁSICO 72
C.2.1 Diagrama de Estrutura *72*
C.2.2 Diagrama de Comportamento *73*
C.2.3 Descrição dos Processos *74*
C.2.4 Descrição dos Dados Introduzidos *78*
C.3 SISTEMAS ONDE É EMPREGADO 78

D. SUBSISTEMA BÁSICO - GERENCIA EXECUÇÃO DE COMANDO 79

D.1 ESPECIFICAÇÃO 79
D.1.1 Descrição *79*
D.1.2 Diagrama de Contexto *79*
D.1.3 Descrição dos Serviços *79*
D.1.3.1 Serviços Externos 79
D.1.3.2 Serviços Internos 82
D.1.3.3 Serviços Requisitados 82
D.1.4 Descrição Detalhada dos Dados *83*
D.1.4.1 Definições de Tipos 83
D.1.4.2 Estruturas de Dados Encapsuladas 84
D.1.4.3 Estrutura das Mensagens 85
D.2 DESIGN DO SUBSISTEMA BÁSICO 87
D.2.1 Diagrama de Estrutura *87*
D.2.2 Diagrama de Comportamento *87*
D.2.3 Descrição dos Processos *88*
D.3 SISTEMAS ONDE É EMPREGADO 88

E. SUBSISTEMA BÁSICO - ALINHA WAFER 89

E.1 ESPECIFICAÇÃO 89
E.1.1 Descrição *89*
E.1.2 Diagrama de Contexto *89*
E.1.3 Descrição dos Serviços *89*
E.1.3.1 Serviços Externos 89
E.1.3.2 Serviços Internos 90
E.1.3.3 Serviços Requisitados 91
E.1.4 Descrição Detalhada dos Dados *91*
E.1.4.1 Definições de Tipos 91
E.1.4.2 Estruturas de Dados Encapsuladas 91
E.1.4.3 Estrutura das Mensagens 91
E.2 DESIGN DO SUBSISTEMA BÁSICO 92
E.3 SISTEMAS ONDE É EMPREGADO 93

F. SUBSISTEMA BÁSICO - CONTROLA MESA_θ 94

F.1 ESPECIFICAÇÃO 94

F.1.1 Descrição *94*
F.1.2 Diagrama de Contexto *94*
F.1.3 Descrição dos Serviços *94*
F.1.3.1 Serviços Externos 94
F.1.3.2 Serviços Internos 96
F.1.3.3 Serviços Requisitados 96
F.1.4 Descrição Detalhada dos Dados *96*
F.1.4.1 Definições de Tipos 96
F.1.4.2 Estruturas de Dados Encapsuladas 96
F.1.4.3 Estrutura das Mensagens 96
F.2 DESIGN DO SUBSISTEMA BÁSICO 97
F.3 SISTEMAS ONDE É EMPREGADO 97

G. SUBSISTEMA BÁSICO - CONTROLA MESA_XY 98

G.1 ESPECIFICAÇÃO 98
G.1.1 Descrição *98*
G.1.2 Diagrama de Contexto do Subsistema Básico *98*
G.1.3 Descrição dos Serviços *98*
G.1.3.1 Serviços Externos 98
G.1.3.2 Serviços Internos 100
G.1.3.3 Serviços Requisitados 100
G.1.4 Descrição Detalhada dos Dados *100*
G.1.4.1 Definições de Tipos 100
G.1.4.2 Estruturas de Dados Encapsuladas 100
G.1.4.3 Estrutura das Mensagens 100
G.2 DESIGN DO SUBSISTEMA BÁSICO 101
G.2.1 Árvore de módulos *101*
G.2.2 Objetivos dos Módulos *101*
G.3 SISTEMAS ONDE É EMPREGADO 101

H. SUBSISTEMA BÁSICO - CONTROLA MESA 102

H.1 ESPECIFICAÇÃO 102
H.1.1 Descrição *102*
H.1.2 Diagrama de Contexto *102*
H.1.3 Descrição dos Serviços *102*
H.1.3.1 Serviços Externos 102
H.1.3.2 Serviços Internos 104
H.1.3.3 Serviços Requisitados 104
H.1.4 Descrição Detalhada dos Dados *104*
H.1.4.1 Definições de Tipos 104
H.1.4.2 Estruturas de Dados Encapsuladas 104
H.1.4.3 Estrutura das Mensagens 104
H.2 DESIGN DO SUBSISTEMA BÁSICO 105
H.2.1 Diagrama de Comportamento *106*
H.2.2 Descrição dos Processos *106*
H.3 SISTEMAS ONDE É EMPREGADO 108

I. SUBSISTEMA BÁSICO - CONTROLA OBTURADOR FEIXE 109

I.1 ESPECIFICAÇÃO 109
I.1.1 Descrição *109*
I.1.2 Diagrama de Contexto *109*
I.1.3 Descrição dos Serviços *109*
I.1.3.1 Serviços Externos 109
I.1.3.2 Serviços Internos 110
I.1.3.3 Serviços Requisitados 110
I.1.4 Descrição Detalhada dos Dados *110*
I.1.4.1 Definições de Tipos 110
I.1.4.2 Estruturas de Dados Encapsuladas 110
I.1.4.3 Estrutura das Mensagens 111
I.2 DESIGN DO SUBSISTEMA BÁSICO 111
I.2.1 Diagrama de Estrutura *111*
I.2.2 Diagrama de Comportamento *112*
I.2.3 Descrição dos Processos *112*

I.2.4 Descrição dos Dados Introduzidos *114*
I.3 SISTEMAS ONDE É EMPREGADO 114

1 Introdução

O presente trabalho foi desenvolvido ao longo de uma linha de pesquisa que busca abordar questões como eficácia, qualidade e confiabilidade de sistemas, relacionando-as especialmente ao desenvolvimento de sistemas de tempo-real para controle de processos, de grande porte, em ambiente distribuído. O objetivo primordial de um sistema processo-controle ("process-control system") é manter, ao longo do tempo, algumas propriedades do processo associadas a valores específicos das variáveis que as representam. Além de depender da dinâmica do processo controlado, a porção de controle desse tipo de sistema deve também levar em conta as condições a serem satisfeitas pelo processo [Levenson90]. Assim sendo, os requisitos do subsistema de controle devem refletir, através de informações de controle e de engenharia de sistemas, a maioria das relações existentes entre os componentes do sistema processo-controle.

Em sistemas processo-controle, pequenas distorções comportamentais no subsistema de controle podem ter graves conseqüências relacionadas à segurança de equipamentos e pessoas [Levenson91]. Devido à complexidade dessa classe de sistemas, cada projeto constitui investimento de risco, o que implica uma probabilidade significativa de não haver sucesso no desenvolvimento e, conseqüentemente, de inexistir retorno sobre o investimento realizado.

Nessas condições, é fundamental que a equipe de projeto possa empregar ferramentas conceituais e técnicas de desenvolvimento que:

- durante a construção do subsistema de controle, permitam a minimização do custo e do risco associado ao projeto e, após a implantação desse subsistema, permitam a minimização do custo de sua manutenção evolutiva e/ou corretiva,
- apoiem o estudo da concorrência inerente ao software de controle e
- aumentem a portatilidade do software de controle para diversas arquiteturas de hardware,

tudo isso, sem prejuízo da qualidade do produto final, que deve ser assegurada durante sua construção. O atendimento eficaz a esse tipo de objetivos torna viável o desenvolvimento de projetos importantes que não são atualmente empreendidos pois o cliente/usuário não se dispõe a enfrentar todos os ônus que podem dele decorrer.

Na pesquisa que originou este trabalho, quatro métricas foram consideradas primordiais na avaliação da qualidade de sistemas de software: Modularidade, Manutenibilidade, Portatilidade e Reúso. Essas métricas balizaram a escolha de uma estrutura de modelagem específica [Ward85, Maffeo92] e, em particular, a definição de um método de design [Sanchez95.1] que enfatiza o reúso dos subsistemas que compõem a porção de controle do sistema processo-controle. A ênfase em reúso justifica-se em função de sua grande relevância econômica e porque, ao buscar a otimização dessa métrica, estar-se-á atendendo também às demais.

Conforme [Veldwiljk94], a interpretação a ser dada atualmente ao termo "crise do software" relaciona-se menos à falta de produtividade associada ao processo de construção do software - implicando as tão comentadas violações de cronogramas e custos - ou à falta de confiabilidade associada ao produto gerado - implicando a presença de grande quantidade de erros no software dado como "pronto" - e mais às dificuldades associadas à fase de manutenção do software construído - implicando custos excessivos alocados a essa fase do ciclo de vida e uma extensa fila de espera ("backlog") para novos projetos. Por outro lado, também segundo [Veldwiljk94], as dificuldade ainda encontradas na fase de manutenção devem-se primordialmente à falta de flexibilidade associada ao design do sistema de software. Assim sendo, aumentar a produtividade durante a fase de construção do sistema não resolverá necessariamente o problema da manutenção do sistema construído. Da mesma forma, apenas disciplina de desenvolvimento e documentação de

alto nível e poder de comunicação também não resolverão sozinhas esse tipo de problema. No entanto, maior flexibilidade, alcançada através de um design representado em alto nível de abstração - que esconde detalhes sem, no entanto, omiti-los -, permite aumentar significativamente a produtividade durante a fase de manutenção e reduzir a quantidade e a complexidade das atividade inerentes a essa fase. É o caso, por exemplo, da flexibilidade alcançada quando utiliza-se encapsulamento de informação como princípio de design. O emprego de ferramentas e técnicas de modelagem consistentes com o objetivo de garantir flexibilidade ao design do sistema deve fundamentar um processo de construção e manutenção baseado em software altamente reusável. As considerações apresentadas em [Veldwiljk94] relacionadas ao formato atual da "crise do software" constituem um forte respaldo para os objetivos de qualidade fixados como base para a derivação do método de design adotado neste trabalho. Esse método, inserido em um processo global de modelagem de sistemas logicamente baseado em uma versão conceitual do sistema a ser construído e no enunciado rigoroso problema que ele deve resolver, vem demonstrando, através de sucessivas experimentações controladas, sua capacidade de gerar estruturas que garantem grande flexibilidade ao sistema resultante.

A estrutura de modelagem empregada baseia-se nos chamados **Métodos Estruturados**, seja orientados a atividades seja a objetos, que buscam primordialmente identificar as funções de um sistema e as informações por ele tratadas. Essa estrutura de modelagem segmenta o processo de desenvolvimento em três etapas destinadas a gerar modelos com diferentes graus de abstração. Esses modelos [Sanchez94, Sanchez93, Maffeo92] representam em detalhe toda a concepção e implementação do sistema e conservam uma relação de precedência lógica que, entretanto, não define necessariamente a ordem cronológica mais adequada para sua construção. São eles:

♦ **Modelo da Essência** (ME), representando o resultado da elicitação, análise e especificação dos requisitos essenciais (isto é, dos requisitos invariantes em relação a alternativas de implementação) do sistema em questão;

♦ **Modelo da Implementação** (MI), sucessor lógico do ME, incorporando os requisitos essenciais ao design do sistema, do qual o presente trabalho é um exemplo.

♦ **Modelo da Automação** (MA), sucessor lógico do MI, incorporando o design da alternativa de implementação selecionada aos recursos tecnológicos de nível de abstração mais baixo (linguagem de programação, processadores, ...).

Este trabalho corresponde à etapa de construção do Modelo da Implementação e refere-se à porção de controle de um sistema processo-controle cujo Modelo da Essência [Sanchez95.2] foi totalmente elaborado previamente.

O Modelo da Implementação constitui uma planta de construção que agrega características não essenciais do sistema ao conjunto de requisitos essenciais. Esse modelo resulta de uma reorganização e expansão do Modelo da Essência, adicionando-se a este atividades e informações que visam superar a hipótese não realista referente ao emprego de tecnologia ideal para implementar o sistema [McMenamin84]. Dessa forma são completados os requisitos necessários ao atendimento às necessidades do cliente/usuário.

No método de design definido em [Sanchez95.1], o **Modelo da Implementação** é composto de três porções distintas:

♦ a definição dos **Requisitos de Concepção de Sistema**, agregando ao ME requisitos não-essenciais que especificam características de implementação relevantes, principalmente aquelas referentes a desempenho e tolerância a falhas. Essa definição impõe restrições à tecnologia não-

ideal a ser empregada na implementação e é precondicão para a escolha das arquiteturas de hardware e software básico;

♦ a definição da **Interface Usuário-Máquina**, que determina as características interativas do sistema e sua forma de operação. Implica selecionar ou construir os equipamentos de interface com os operadores e o software a ser incorporado ao sistema para estabelecer sua interatividade;

♦ o **design do sistema,** que o segmenta em subsistemas com um padrão de cooperação para atender aos requisitos essenciais, não essenciais e de comunicação com operadores.

Abrangendo o contexto descrito no parágrafo precedente, o presente trabalho incorpora o **design** do software do sistema de **C**ontrole e **Mon**itoramento de **L**itografia por **F**eixe de **E**létrons (COMONLIFE), futuramente em uso no Departamento de Ciência dos Materiais e Metalurgia (DCMM) da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Ele constitui um dos resultados de uma experimentação controlada relativa a um processo de desenvolvimento específico, visando testar as ferramentas conceituais e técnicas de design descritas em [Sanchez95.1]. A falta de definição relativa a alguns aspectos do hardware a ser empregado e à especificação da Interface Usuário-Máquina implicou a omissão de alguns detalhes que serão definidos em fase posterior do processo de desenvolvimento.

A estratégia de design empregada para atingir os objetivos de qualidade especificados anteriormente recomenda a segmentação do processo de design do sistema em três etapas:

♦ construção do **Projeto Básico**, que evidencia os **subsistemas básicos,** suas **portas** e **conexões**, bem como a eventual necessidade de concorrência e sincronismo entre os serviços alocados a cada subsistema básico;

♦ **especificação detalhada** de cada **subsistema básico**, que permite seu desenvolvimento autônomo e facilita sua recuperação para reúso;

♦ **design** de cada subsistema básico como unidade independente de desenvolvimento, envolvendo implementação e reúso;

♦ **configuração do sistema**, que evidencia a **configuração de software** (definição de portas e conexões) e a **configuração de hardware** (mapeamento dos subsistemas básicos e seus processos no hardware).

O **Projeto Básico**, resultado de uma ação de síntese aplicada sobre o Modelo da Essência previamente elaborado, contém a definição dos **subsistemas básicos**. Essa definição incorpora a especificação, a partir de agregações de Eventos Externos identificados no Modelo da Essência, dos **serviços** prestados pelo sistema associados a cada um desses subsistemas. **Subsistemas básicos,** definidos a partir de critérios de síntese aplicados ao Modelo do Comportamento e baseados no encapsulamento de informação (Tipos de Entidade) e operações (Atividades Essenciais Operacionais e de Controle), englobam um conjunto resumido e bem definido de serviços passível de reúso em outros sistemas.

Os **subsistemas básicos** são **unidades de implementação** e **unidades de reúso**. Unidades de implementação, cada uma prestando-se ao design, manuseio, produção e manutenção de grandes sistemas que encapsulam informação e operações de modo a garantir a inexistência de áreas de dados compartilhadas. Unidades de reúso de código fonte, cada uma sendo desenvolvida e mantida por uma equipe pequena e sendo indivisível quando reusada.

Visando proporcionar elevado nível de **reúso**, subsistemas devem ser concebidos de modo que cada um tenha conhecimento apenas de si próprio, isto é, sem incorporar referências a outros subsistemas. Para obter essa independência, cada subsistema deve possuir **portas,** de entrada e saída de dados e controle, que definam o protocolo de comunicação com o ambiente externo ao subsistema, regulando o **envio/recebimento de mensagens**. Essas mensagens são transportadas por uma **interface** composta de **canais de comunicação** que conectam **portas** de subsistemas, inclusive aqueles que pertençam ao ambiente externo. A comunicação e o sincronismo entre esses subsistemas ocorrem através dessa interface.

O **Sistema** é representado como uma organização hierárquica de subsistemas aos quais são alocados os serviços prestados. Por definição, **subsistema básico** é um subsistema-folha dessa hierarquia. A segmentação do sistema em subsistemas básicos obedece à restrição de que cada serviço só pode ser alocado a apenas um subsistema. Um subsistema que necessite de um serviço alocado a outro subsistema deverá solicitar sua execução ao subsistema ao qual o serviço esteja alocado. A requisição de um serviço alocado a outro subsistema ocorrerá a partir de uma interação interna ao subsistema cliente envolvendo uma de suas portas de saída. Pelo design do sistema, essa porta estará conectada a uma porta de entrada do subsistema servidor. Como os subsistemas básicos encapsulam toda a informação tratada pelo sistema, nenhum dos níveis da organização hierárquica apresentará área de dados compartilhada. Um subsistema pode possuir várias portas de entrada (simples ou com resposta associada) e, portanto, prestar mais de um serviço concorrentemente. O atendimento a esses requisitos garante ao sistema resultante alta **modularidade** (alocação de cada serviço a apenas um subsistema e não compartilhamento de áreas de dados) e alta **manutenibilidade** (mudança na implementação de um dado serviço concentra as alterações em um único subsistema).

O Projeto Básico especifica o conjunto de serviços pertencente a cada subsistema básico de uma forma ainda abstrata e sucinta. Entretanto, para que a implementação seja feita por equipes diferentes, cada subsistema básico necessita de uma especificação detalhada independente. Isto é, o serviço especificado no Projeto Básico ainda não estará descrito com o nível de detalhe necessário à implementação. A partir dele e do Modelo da Essência do sistema, será feita uma especificação detalhada do subsistema básico. Nessa especificação, para facilitar o reúso do subsistema básico, limites e restrições devem, sempre que possível, identificar o tipo de parametrização a ser empregado na implementação [Poulin94]. A separação entre a documentação do design e a documentação que especifica detalhadamente cada subsistema básico facilita a divisão do trabalho entre equipes responsáveis por um ou mais subsistemas básicos, bem como a identificação dos subsistemas básicos, já desenvolvidos, que podem ser reusados.

O design do sistema resulta em um conjunto de subsistemas básicos (SB) independentes, cuja integração será efetuada através de uma **linguagem de configuração** cuja função é definir as conexões, verificar a consistência entre as portas conectadas e mapear os processos de um dado SB na Arquitetura de Hardware. O uso de uma linguagem de configuração determina a hierarquização vertical do software em duas camadas [Sanchez 95.1]. A camada de nível mais baixo (**Interface de Compatibilização**) interfaceia diretamente com o hardware e o software básico e oferece uma interface padrão de serviços prestados para a camada de nível mais alto (a **Aplicação** propriamente dita). O software desta última (que implementa os requisitos da aplicação) pode, então, ser projetado em nível de abstração compatível com o empregado no design. O uso dessa linguagem de configuração, permitindo gerar uma unidade de implementação responsável por configurar o sistema independentemente da implementação de cada SB, garante ao software resultante alta **portatilidade** em relação à arquitetura de hardware e software básico: uma mudança na plataforma de hardware exige alterações apenas na configuração dos subsistemas básicos; uma mudança na plataforma de software básico exige alterações apenas na Interface de Compatibilização. Garante

também alta **manutenibilidade** para o software da aplicação pois a substituição de um SB por outro, que implemente seus serviços e possua interface compatível, pode ser feita apenas com a alteração da configuração, sem alteração de nenhum outro SB componente do sistema.

Vale ressaltar que um SB pode ser considerado um Tipo Abstrato de Dados pois encapsula uma estrutura de dados juntamente com todas as operações de acesso a essa estrutura. Assim sendo, o método aqui empregado incorpora características básicas de métodos de design orientado a objetos. Coerente com isso, o encapsulamento de informação associado a um SB não se restringe apenas à informação essencial do sistema ou a informação dela derivada. Por exemplo, um SB pode englobar um conjunto de ações representadas no Modelo da Essência por uma ou mais Atividades de Controle. Um SB desse tipo coordena a execução de tarefas baseando-se em informação que modela estados internos do sistema - estados esses que refletem na realidade estados do processo controlado - e não em informação que conste da modelagem da essência do sistema. Uma diferença relativa a métodos tradicionais de orientação a objetos [Rumbaugh91, Booch94] reside no fato de que um SB pode gerenciar o comportamento de vários objetos de uma mesma classe, cuja característica de processamento seja seqüencial. O número desses objetos pode ser determinado em tempo de geração do sistema ou, dinamicamente, em tempo de execução.

A seção 2 deste trabalho descreve a derivação, efetuada a partir do Modelo da Essência, dos subsistemas básicos que constituem o COMONLIFE. A seção 3 apresenta todo o Projeto Básico desse sistema de controle/monitoramento. A seção 3.1 que fornece a visão de contexto desse subsistema de controle/monitoramento sob o enfoque de serviços. A seção 3.2 descreve a organização hierárquica dos subsistemas do COMONLIFE até a obtenção dos subsistemas básicos, folhas dessa hierarquia. Cada nível da organização hierárquica corresponde a uma descrição estrutural, em termos de subsistemas, empregando a linguagem de Diagramas de Estrutura, e a uma descrição comportamental, empregando a linguagem de Diagramas de Comportamento, revelando a concorrência e/ou seqüencialização existente entre os subsistemas. São também apresentadas as justificativas das decisões de design que levaram a cada agregação, ressaltando as características que identificam cada subsistema de acordo com os critérios propostos em [Sanchez95.1]. As seções 3.3 e 3.4 apresentam, respectivamente, o Diagrama de Subsistemas Básicos e a Árvore de Subsistemas. A seção 4 apresenta as conclusões deste trabalho. Nos apêndices, encontram-se as especificações detalhadas de cada um dos Subsistemas Básicos que compõem o COMONLIFE.

2 Síntese do Modelo da Essência

2.1 Relação entre Serviços e Eventos Externos

Quando existe alguma associação, esta seção apresenta a relação entre os serviços oferecidos pelo sistema e os Eventos Externos a cuja ocorrência o sistema deve reagir, identificados durante a modelagem da essência. Essa relação é fundamental para o processo de construção do design. Dessa forma, é possível garantir a completeza do design em relação à especificação dos requisitos essenciais para o sistema. Em última análise, isso permite garantir que o sistema a ser implementado a partir desse modelo possuirá toda a funcionalidade que atende aos objetivos de eficácia correspondentes às necessidades do cliente/usuário.

Serviço "Gerar Camada Genérica"

Não existe evento externo associado diretamente a este serviço pois sua execução depende da requisição feita por outros componentes do sistema.

Serviço "Iniciar Wafer"

1. Operador necessita iniciar um wafer com parâmetros de operação específicos.
   (sinalizado por "pedido_iniciação").

Serviço "Alinhar Wafer"

2. Operador necessita iniciar alinhamento de wafer específico.
   (sinalizado por "pedido_início_alinhamento");
3. Operador necessita confirmar posição de marca de alinhamento.
   (sinalizado por "confirmação_marca");
4. Operador necessita ajustar posição de marca de alinhamento na direção x.
   (sinalizado por "ajuste_marca_x");
5. Operador necessita ajustar posição de marca de alinhamento na direção y.
   sinalizado por "ajuste_marca_y").

Serviço "Gerar Camada"

6. Operador necessita interromper o processo de geração de camada.
   (sinalizado por "interrupção_máscara");
7. Operador necessita relatório sobre wafer gravado.
   (sinalizado por "pedido_relatório_wafer");
8. Com periodicidade predeterminada, OPERADOR necessita que informações sobre o andamento do processo sejam armazenadas.
   (sinalizado por "é_tempo_de registrar variáveis_processo");

29. Decorrido tempo necessário à estabilização do microscópio eletrônico, variáveis associadas ao Feixe de Elétrons não atingem valor dentro da faixa desejada.

(sinalizado por "é_tempo_de diagnosticar estado_do_microscópio");

Serviço "Posicionar Mesa xy"

9. Variável x, associada à posição da mesa, atinge valor dentro da faixa desejada.

(sinalizado por valor específico de "x_mesa");

10. Variável y, associada à posição da mesa, atinge valor dentro da faixa desejada.

(sinalizado por valor específico de "y_mesa");

12. Variável x, associada à posição da mesa, sai da faixa desejada ultrapassando o valor máximo.

(sinalizado por valor específico de "x_mesa");

13. Variável y, associada à posição da mesa, sai da faixa desejada ultrapassando o valor máximo.

(sinalizado por valor específico de "y_mesa");

15. Variável x, associada à posição da mesa, sai da faixa desejada ultrapassando o valor mínimo.

(sinalizado por valor específico de "x_mesa");

16. Variável y, associada à posição da mesa, sai da faixa desejada ultrapassando o valor mínimo.

(sinalizado por valor específico de "y_mesa");

18. Decorrido período predeterminado após início da tentativa de posicionamento, variável x, associada à posição da mesa, não atinge a faixa desejada.

(sinalizado por "é_tempo_de interromper processo");

19. Decorrido período predeterminado após início da tentativa de posicionamento, variável y, associada à posição da mesa, não atinge a faixa desejada.

(sinalizado por "é_tempo_de interromper processo").

Serviço "Posicionar Mesa θ"

11. Variável θ, associada à posição da mesa, atinge valor dentro da faixa desejada.

(sinalizado por valor específico de "θ_mesa);

14. Variável θ, associada à posição da mesa, sai da faixa desejada ultrapassando o valor máximo.

(sinalizado por valor específico de "θ_mesa");

17. Variável θ, associada à posição da mesa, sai da faixa desejada ultrapassando o valor mínimo.

(sinalizado por valor específico de "θ_mesa");

20. Decorrido período predeterminado após início da tentativa de posicionamento, variável q, associada à posição da mesa, não atinge a faixa desejada.

(sinalizado por "é_tempo_de interromper processo").

Serviço "Monitorar Microscópio Eletrônico"

21. Variável tensão, associada à tensão na Fonte de Alta Tensão, sai da faixa desejada ultrapassando o valor máximo.

    (sinalizado por valor específico de "tensão_fonte");

22. Variável tensão, associada à tensão na Fonte de Alta Tensão, sai da faixa desejada ultrapassando o valor mínimo.

    (sinalizado por valor específico de "tensão_fonte");

23. Variável corrente_emissão_filamento, associada à quantidade de elétrons gerada pelo Filamento, sai da faixa desejada ultrapassando o valor máximo.

    (sinalizado por valor específico de "corrente_emissão_filamento");

24. Variável corrente_emissão_filamento, associada à quantidade de elétrons gerada pelo Filamento, sai da faixa desejada ultrapassando o valor mínimo.

    (sinalizado por valor específico de "corrente_emissão_filamento");

25. Variável corrente_objetiva, associada à quantidade de elétrons que passa pela Lente Objetiva, sai da faixa desejada ultrapassando o valor máximo.

    (sinalizado por valor específico de "corrente_objetiva");

26. Variável corrente_objetiva, associada à quantidade de elétrons que passa pela Lente Objetiva, sai da faixa desejada ultrapassando o valor mínimo.

    (sinalizado por valor específico de "corrente_objetiva");

27. Variável corrente_condensadora, associada à quantidade de elétrons que passa pela Lente Condensadora, sai da faixa desejada ultrapassando o valor máximo.

    (sinalizado por valor específico de "corrente_condensadora");

28. Variável corrente_condensadora, associada à quantidade de elétrons que passa pela Lente Condensadora, sai da faixa desejada ultrapassando o valor mínimo.

    (sinalizado por valor específico de "corrente_condensadora").

Serviço "Controlar Obturador do Feixe"

30. Decorrido período predeterminado após início da tentativa de abrir obturador, valor de "corrente_absorção" não indica obturador aberto.

    (sinalizado por "é_tempo_de interromper processo");

31. Decorrido período predeterminado após início da tentativa de fechar obturador, valor de "corrente_absorção" não indica obturador fechado.

    (sinalizado por "é_tempo_de interromper processo").

2.2 Serviços x Assuntos

A tabela apresentada a seguir associa a cada **serviço**, prestado pelo sistema, o **assunto** pertinente referido na Descrição de Operações e do Modelo da Essência (Modelo do Contexto). Para cada serviço ou assunto, foram registrados os eventos externos correspondentemente alocados. Ficam também explícitas nessa tabela as distinções entre serviço e assunto, ressaltando-se para cada

serviço a possibilidade de requisição de outros serviços que proporcionam o atendimento a eventos externos presentes na descrição do assunto correspondente e não associados ao serviço em questão.

Serviço	Assunto
Serviço: Gerar Camada Genérica • requisita serviços de posicionamento de mesa e controle de obturador para traçado de linhas.	Assunto: Geração de Camada Genérica (camada de marcas ou camada do chip). • Traçar linha. (eventos externos 9 e 10)
Serviço: Iniciar Wafer (evento externo 1) • requisita serviço de movimentação linear da mesa para a posição de repouso.	Assunto: Iniciação de Wafer. • Iniciar wafer. (evento externo 1) • Levar a mesa para sua posição de repouso. (eventos externos 9 e 10)
Serviço: Alinhar Wafer (eventos externos 2, 3, 4, 5) • requisita serviço de movimentação linear da mesa xy para a posição de repouso ou para posição de marca. • requisita serviço de movimentação angular da mesa θ.	Assunto: Alinhamento de Wafer. • Iniciar alinhamento de wafer. (evento externo 2) • Estabelecer alinhamento de wafer. (eventos externos 3, 4, 5) • Levar a mesa para sua posição de repouso. (evento externo 9 e 10) • Levar a mesa para posição de ajuste na direção x. (evento externo 9) • Levar a mesa para posição de ajuste na direção y. (evento externo 10) • Levar a mesa para posição de uma marca de alinhamento. (eventos externos 9 e 10) • Levar a mesa para posição de alinhamento na direção θ. (evento externo 11)
Serviço: Gerar Camada (eventos externos 6, 7, 8, 29) • requisita serviço de geração de camada genérica. • requisita serviço de monitoramento de microscópio.	Assunto: Geração de Camada do Chip. • Iniciar geração de camada do chip em wafer alinhado. (evento externo 29) • Interromper geração de camada do chip. (evento externo 6) • Avaliar qualidade da gravação de wafer. (eventos externos 7, 8)
Serviço: Posicionar Mesa xy (eventos externos 9, 10,12,13,15, 16, 18, 19)	• Assunto: Movimentação da Mesa nas Direções x, y e θ. • Posicionar a Mesa na direção x. (eventos externos 9,12, 15, 18) • Posicionar Mesa na direção y. (eventos externos 10, 13, 16, 19)

Serviço: Posicionar Mesa θ (evento externo 11, 14, 17, 20)	Assunto: Movimentação da Mesa nas Direções x, y e θ. • Posicionar Mesa na direção θ. (evento externo 11, 14, 17, 20)
Serviço: Controlar Obturador do Feixe (eventos externos 30, 31)	Assunto: Monitoramento do Microscópio Eletrônico. • Interromper ou restabelecer fluxo de elétrons. (eventos externos 30, 31)
Serviço: Monitorar Microscópio Eletrônico (eventos externos 21, 22, 23, 24, 25, 26, 27, 28)	Assunto: Monitoramento do Microscópio Eletrônico. • Fornecer energia aos elétrons do feixe. (evento externo 21, 22) • Determinar a densidade de corrente do feixe de elétrons. (evento externo 23, 24) • Focalizar o feixe de elétrons. (evento externo 25, 26) • Uniformizar e dar forma ao feixe de elétrons. (evento externo 27, 28)

2.3 Derivando Subsistemas Básicos

Subsistemas básicos são derivados do Modelo da Essência do COMONLIFE (Modelo do Comportamento) aplicando-se os critérios descritos em [Sanchez95.1]. A Figura 1(a) apresenta a tabela de definição das classes (uma para cada Tipo de Entidade presente no Esquema da Memória Essencial) e explicita as atividades essenciais relacionadas a cada Tipo de Entidade identificando o tipo de acesso de cada atividade aos depósitos internos correspondentes. A Figura 1(b) relaciona os descritores das atividades essenciais visando facilitar o entendimento da semântica associada a cada uma delas. Para maiores detalhes do comportamento dessas atividades, referir-se ao Modelo do Comportamento do COMONLIFE, em particular: Organização Hierárquica do Esquema de Atividades e Listas de Pré- e Pós- Condições.

Critério Classe	acesso escrita inserção/remoção	acesso escrita alteração	acesso leitura
WAFER	F1.4.1	F1.1.1 F1.2.3 F1.2.1.1 F1.2.1.5 F1.2.2.3	F1.3 F1.1.2 F1.1.4 F1.2.1.2 F1.2.1.3 F1.2.1.4 F1.2.2.5 F1.4.2
PROJETO_CHIP	F1.4.1	F1.2.1.1 F1.2.1.5 F1.4.2	F1.3 F1.1.1 F1.1.2 F1.1.4 F1.2.3 F1.2.1.2 F1.2.1.3 F1.2.1.4 F1.2.2.3 F1.2.2.5 F1.4.2
Detalhe de LITOGRAFIA	F1.4.1	F1.2.2.5	F1.1.1 F1.1.4 F1.2.2.3
MÁSCARA_CHIP	F1.4.1		F1.2.3
CAMADA	F1.4.1		F1.3 F1.1.2 F1.2.3 F1.2.1.2 F1.2.1.3 F1.2.1.4
Detalhe de GRAVAÇÃO	F1.2.1.2		F1.3
MÁSCARA CORRENTE	F1.2.3 F1.4.1	F1.1.1	F1.1.4
FIGURA	F1.1.1		F1.1.2 F1.1.4
MOVIMENTO θ	F1.2.2.4	F2.3.2	F2.3.1
MOVIMENTO X-Y	F1.1.2 F1.1.3 F1.2.2.1 F1.2.2.2 F1.2.2.3	F2.2.2	F2.2.1
DESLOCAMENTO	F1.2.2.1 F1.2.2.2		F1.2.2.4 F1.2.2.5
POSIÇÃO θ	F2.3.1		F2.3.2

POSIÇÃO X-Y	F2.2.1		F2.2.2
ESTADO	F2.1.1 F2.1.2		F2.1.3

Figura 1(a)- Tabela de Associação de Atividades

F1.3 imprime relatório wafer
F1.1.1 interpreta figura
F1.1.2 calcula movimentação para posição traçando
F1.1.3 calcula movimentação para posição sem traçar
F1.1.4 emite status
F1.2.3 valida pedido alinhamento
F1.2.1.1 atualiza estado wafer
F1.2.1.2 registra varáveis processo
F1.2.1.3 acompanha estado do microscópio
F1.2.1.4 verifica estado do microscópio
F1.2.1.5 restaura estado wafer
F1.2.2.1 alinha mesa na direção x
F1.2.2.2 alinha mesa na direção y
F1.2.2.3 determina a posição da marca
F1.2.2.4 calcula ajuste_θ
F1.2.2.5 calcula origem x-y
F1.4.1 valida pedido iniciação wafer
F1.4.2 calcula tensão e corrente
F2.1.1 abre obturador feixe
F2.1.2 fecha obturador feixe
F2.1.3 verifica obturador feixe
F2.2.1 envia comando para mesa x-y
F2.2.2 avalia posição da mesa x-y
F2.3.1 envia comando para mesa θ
F2.3.2 avalia posição da mesa θ

Figura 1(b)- Descritores das Atividades Essenciais

Figura 1- Associação Estrutural entre Atividades Essenciais e Tipos de Entidade

2.3.1 Alocação de Atividades Essenciais a Classes

A alocação inicial das atividades essenciais a classes, representada na Figura 2, seguiu a regra de maior prioridade conforme o tipo de acesso da atividade à estrutura de informação instalada na classe, prioridade essa evidenciada na Figura 1. As atividades alocadas segundo uma dada prioridade não aparecem em classes onde seu tipo de acesso seria de menor prioridade. Só apresentam alocação múltipla as atividades que realizam acesso de mesma prioridade a tipos de entidades encapsuladas em classes diferentes.

Os identificadores das atividades essenciais alocadas a classes estão distinguidos na Figura 2 conforme sua prioridade da seguinte forma:

- com caracteres em negrito, no caso de acesso de escrita para inserção/remoção de entidades *(atividade primária)*;
- com caracteres em itálico, no caso de escrita para alteração de estado de entidades *(atividade secundária)*;
- com caracteres normais, no caso de leitura realizada sobre tipo de entidade *(atividade fundamental)*.

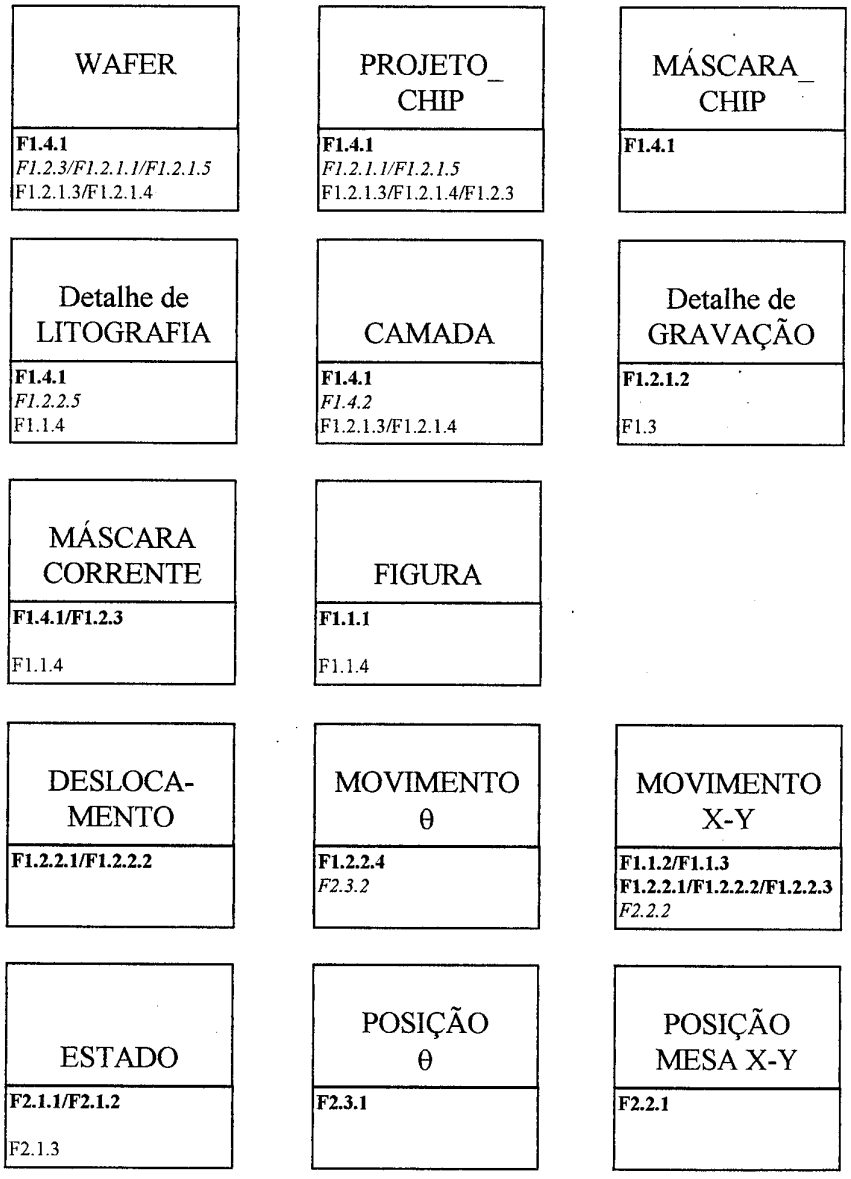


Figura 2- Alocação Inicial das Atividades Essenciais a Classes

A partir da alocação por prioridade do tipo de acesso e usando a semântica associada a cada atividade essencial, o Esquema de Classes inicial pode ser revisto considerando-se:

⇒ Características das atividades associadas às classes WAFER, PROJETO_CHIP, CAMADA, MÁSCARA _CHIP

- A atividade essencial F1.4.1 está associada com alta prioridade às classes que encapsulam WAFER, PROJETO_CHIP, CAMADA, Detalhe de LITOGRAFIA, MÁSCARA_CHIP e MÁSCARA CORRENTE.
- A atividade essencial F1.2.3 está associada com alta prioridade às classes que encapsulam WAFER.
- A semântica associada a essas duas atividades (F1.4.1- valida pedido iniciação wafer - e F1.2.3 - valida pedido alinhamento) indica um tipo de serviço - Gerência de Gravação de Wafer - que deveria agregar, devido à coesão funcional existente entre atividades associadas, todas as estruturas de informação acima mencionadas.
- No entanto, considerando-se:
  - ♦ que essas atividades geram MÁSCARA CORRENTE como resultado de sua execução (ver Listas de Pré- e Pós- Condições das atividades);
  - ♦ o fato de não existir um relacionamento entre MÁSCARA CORRENTE e as demais estruturas de informação indicadas;
  - ♦ o uso intensivo de MÁSCARA CORRENTE por F1.1.1, atividade principal associada a FIGURA (FIGURA é gerada pelo processamento de elementos de MÁSCARA CORRENTE) ;
  - ♦ MÁSCARA CORRENTE e FIGURA estão associadas em mesmo grau de prioridade a uma mesma atividade essencial - F1.1.4 -, que emite status durante a geração de camada sobre wafer;

  pode-se concluir que MÁSCARA CORRENTE não deve ser agregada às demais estruturas de informação e sim a FIGURA cujo objetivo principal está associado à semântica de F1.1.1 (interpreta figura), correspondente à geração de uma camada sobre o wafer a partir de um arquivo C.A.D. internalizado em MÁSCARA CORRENTE. A classe responsável pela Gerência de Gravação de Wafer deve pedir o serviço de geração de camada à classe que encapsula FIGURA e MÁSCARA CORRENTE - **Gerência de Geração de Camada** -, informando qual máscara (MÁSCARA CORRENTE) deverá ser processada e o número de réplicas a serem traçadas (valor constante para um dado wafer, armazenado na iniciação em Detalhe de LITOGRAFIA).
- Outras atividades, relacionadas por coesão funcional à classe Gerência de Gravação de Wafer, são:
  - F1.2.1.1 e F1.2.1.5, referentes à atualização do estado de gravação de um wafer;
  - F1.4.2, referente à atualização de parâmetros de operação após o término de iniciação com sucesso do wafer.
- F1.2.1.3 e F1.2.1.4, referentes à verificação e ao acompanhamento do estado do microscópio, fazem acesso de leitura a algumas das estruturas de informação que foram agregadas pela classe Gerência de Gravação de Wafer. No entanto, executam atividades não coesas com o objetivo principal associado a essa classe. Por essa razão, uma melhor alternativa de design é definir uma nova classe - **Monitoramento do Microscópio** - que encapsulará essas duas atividades e nenhuma estrutura de informação originária da essência do sistema será nela instalada. Essa alternativa favorece o requisito de forte coesão entre atividades de uma mesma classe e explicita a opção por definir uma classe para cada equipamento relevante que esteja sendo controlado/monitorado.

⇒ **Características das atividades associadas às classes MOVIMENTO X-Y, DESLOCAMENTO, Detalhe de Gravação**

- F1.1.2 e F1.1.3 fazem acesso de escrita a MOVIMENTO X-Y, mas sua finalidade é apenas registrar a informação referente à posição para onde a mesa xy deve ser movida durante o traçado de uma camada. Portanto, apesar de terem sido inicialmente associadas a MOVIMENTO X-Y, devem ser realocadas à classe **Gerência de Geração de Camada**.
- Apesar de F1.2.2.1, F1.2.2.2 e F1.2.2.3 fazerem acesso de escrita a MOVIMENTO X-Y, sua finalidade é apenas registrar a informação referente à posição para onde a mesa xy deve ser movida durante o alinhamento de wafer. Além disso, F1.2.2.1 e F1.2.2.2 foram também associadas a DESLOCAMENTO, em mesmo grau de prioridade de MOVIMENTO X-Y. Essa estrutura de informação está relacionada ao serviço de alinhamento de wafer, classe - **Alinhamento de Wafer** -. Portanto, essas Atividades serão alocadas à classe **Alinhamento de Wafer** pois seu acesso a MOVIMENTO X-Y é resultado de um pedido de serviço para movimentação de mesa durante o alinhamento do wafer**;**
- Devem também ser instaladas em **Alinhamento de Wafer** as seguintes atividades:
  - F1.2.2.5 faz acesso de escrita a Detalhe de LITOGRAFIA, mas sua finalidade é calcular o deslocamento nas direções x e y necessários para que o wafer retome a posição usada na sua iniciação;
  - F1.2.2.4 faz acesso de escrita a MOVIMENTO θ, mas sua finalidade é calcular o deslocamento angular necessário para que o wafer retome a posição usada na sua iniciação;

⇒ **Características das atividades associadas à classe Detalhe de GRAVAÇÃO**

- F1.2.1.2 cria Detalhe de Gravação e deve ser acompanhada pela atividade F1.3, única atividade que utiliza essa informação.
- Detalhe de Gravação pode ser incorporada à classe **Gerência de Geração de Camada** (que agrega FIGURA e MÁSCARA CORRENTE) já que essa estrutura de informação só existe a partir do serviço principal associado a essa classe.

⇒ **Características das atividades associadas à classe ESTADO**

- F2.1.1, F2.1.2 e F2.1.3 fazem acesso unicamente a ESTADO, cujo objetivo é o **Controle do Obturador do Feixe**.

⇒ **Características das atividades associadas às classes MOVIMENTO X-Y e POSIÇÃO X-Y**

- F2.2.1 e F2.2.2 fazem acesso a MOVIMENTO X-Y e POSIÇÃO X-Y, que devem ser agregadas em uma classe - **Controle da Mesa X-Y**. O serviço associado a essas estrutura de informação é a de movimentação da MESA X-Y.

⇒ **Características das atividades associadas às classes MOVIMENTO θ e POSIÇÃO θ**

- F2.3.1 e F2.3.2 fazem acesso a MOVIMENTO θ e POSIÇÃO θ, que devem ser agregadas em uma classe - **Controle da Mesa** θ. O serviço associado a essas estruturas de informação é a de movimentação da MESA θ.

⇒ **Características das Atividades de Controle**

- As atividades de controle são alocadas a classes que encapsulam o maior subconjunto das atividades operacionais por ela controladas. Tem-se portanto:
  - FC1.1.1 - controla impressão em wafer: alocada à classe que encapsula Gerência de Geração de Camada;
  - FC1.2.1.1 - controla geração de camada: alocada à classe que encapsula Gerência de Gravação de Wafer;
  - FC1.2.2.1 - controla alinhamento de wafer: alocada à classe que encapsula Alinhamento de Wafer;
  - FC1.4.1 - controla iniciação de wafer: alocada à classe que encapsula Gerência de Gravação de Wafer;
  - FC2.1.1 - controla abertura/fechamento de obturador do feixe: alocada à classe que encapsula Controle de Obturador do Feixe;
  - FC2.2.1 - controla movimentação da mesa x-y: alocada à classe que encapsula Controle da MESA X-Y;
  - FC2.3.1 - controla movimentação da mesa θ: alocada à classe que encapsula Controle da MESA θ.

Essas considerações levam a uma nova configuração de classes e alocação de atividades essenciais a essas classes, representada na Figura 3.

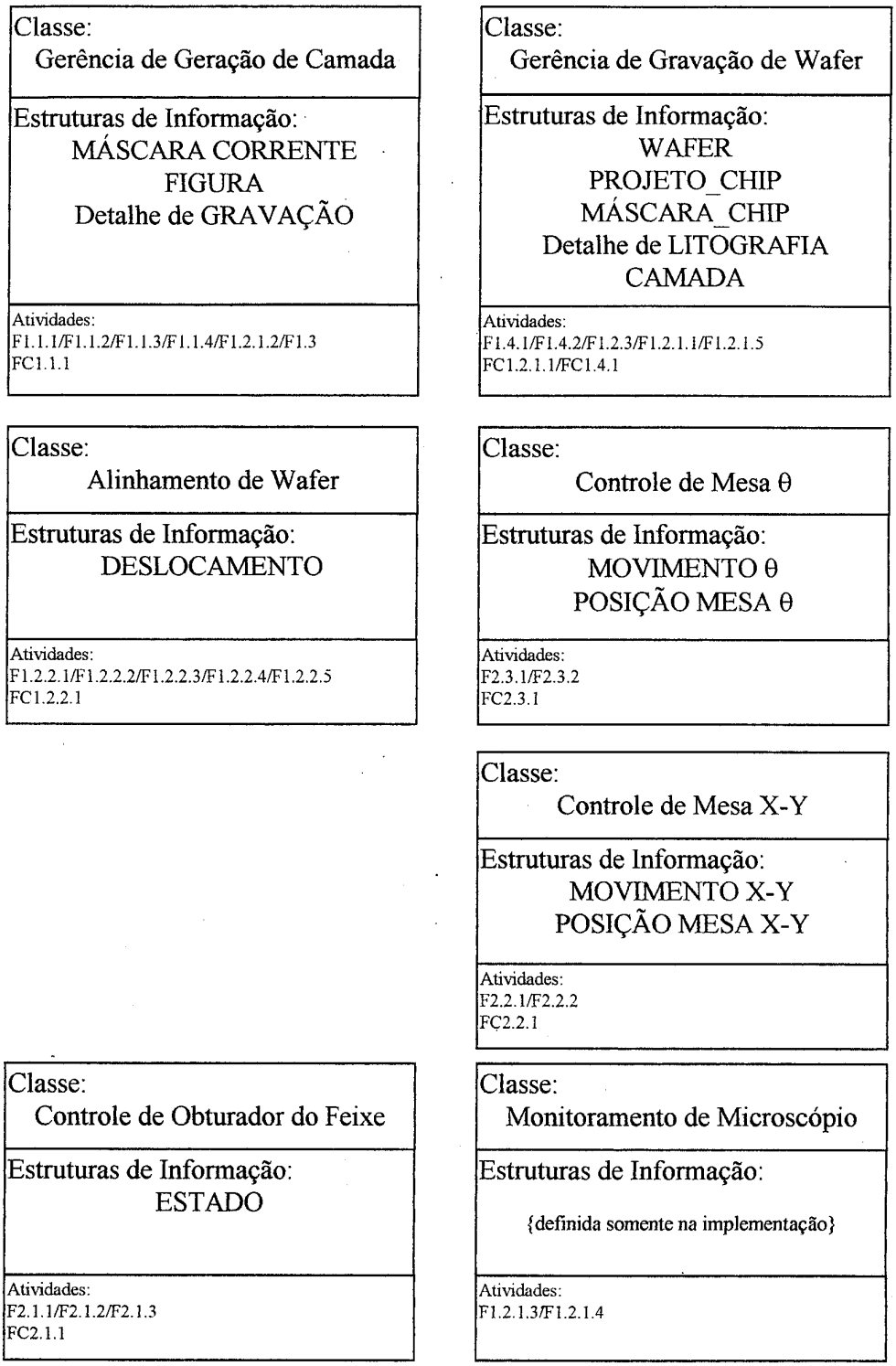


Figura 3- Alocação das Atividades Essenciais a Classes

A estrutura de classes representada na Figura 3 requer ainda a explicitação do tratamento de um conjunto sensor/atuador (vide Figura 4) ainda encapsulado na classe **MESA X-Y.** Essa explicitação,

obtida através da criação da classe **Controle de MESA em uma Direção** (x ou y), constitui melhor alternativa pois contempla um requisito importante do design que é minimizar a geração de código e evidenciar uma classe por equipamento que esteja sendo controlado.

O Esquema de Classes resultante contém mais uma classe, que será instanciada por dois subsistemas básicos, um para MESA X e outro para a MESA Y. A MESA X-Y é uma classe que permanece devido à característica do movimento integrado da mesa nas duas direções cujo serviço corresponde ao controle efetuado pela atividade de controle FC2.2.1.

Classe: Gerência de Geração de Camada	Classe: Gerência de Gravação de Wafer
Estruturas de Informação: MÁSCARA CORRENTE, FIGURA, Detalhe de GRAVAÇÃO	Estruturas de Informação: WAFER, PROJETO_CHIP, MÁSCARA_CHIP, Detalhe de LITOGRAFIA, CAMADA
Atividades: F1.1.1/F1.1.2/F1.1.3/F1.1.4/F1.2.1.2/F1.3 FC1.1.1	Atividades: F1.4.1/F1.4.2/F1.2.3/F1.2.1.1/F1.2.1.5 FC1.2.1.1/FC1.4.1
Classe: Alinhamento de Wafer	**Classe: Controle de MESA θ**
Estruturas de Informação: DESLOCAMENTO	Estruturas de Informação: MOVIMENTO θ, POSIÇÃO MESA θ
Atividades: F1.2.2.1/F1.2.2.2/F1.2.2.3/F1.2.2.4/F1.2.2.5 FC1.2.2.1	Atividades: F2.3.1/F2.3.2 FC2.3.1
Classe: Controle de MESA em uma Direção	**Classe: Controle de MESA X-Y**
Estruturas de Informação: MOVIMENTO LINEAR, POSIÇÃO MESA LINEAR	Estruturas de Informação: MOVIMENTO X-Y, POSIÇÃO MESA X-Y
Atividades: F2.2.1/F2.2.2	Atividades: FC2.2.1
Classe: Controle de Obturador do Feixe	**Classe: Monitoramento do Microscópio**
Estruturas de Informação: ESTADO	Estruturas de Informação: {definida somente na implementação}
Atividades: F2.1.1/F2.1.2/F2.1.3 FC2.1.1	Atividades: F1.2.1.3/F1.2.1.4

Figura 4- Esquema de Classes

O Esquema de Classes obtido conduz diretamente ao Esquema de Subsistemas Básicos, acrescentando-se duas encarnações para a classe **Controle de MESA em uma Direção**, uma para cada direção. Esses subsistemas básicos devem implementar o comportamento do sistema. A esse esquema devem ainda ser acrescentados os subsistemas básicos responsáveis pela Interface Usuário-Máquina.

Cada classe é transformada em subsistema e seus identificadores são alterados para evidenciar o serviço prestado pela classe. Em particular, a classe responsável pela gravação de wafer tem como identificador "Gerencia Execução de Comando" pois além do serviço de gravação de wafer, necessita seqüencializar a execução de comandos sobre wafer devido a existência de um único equipamento de gravação (vide Figura 5).

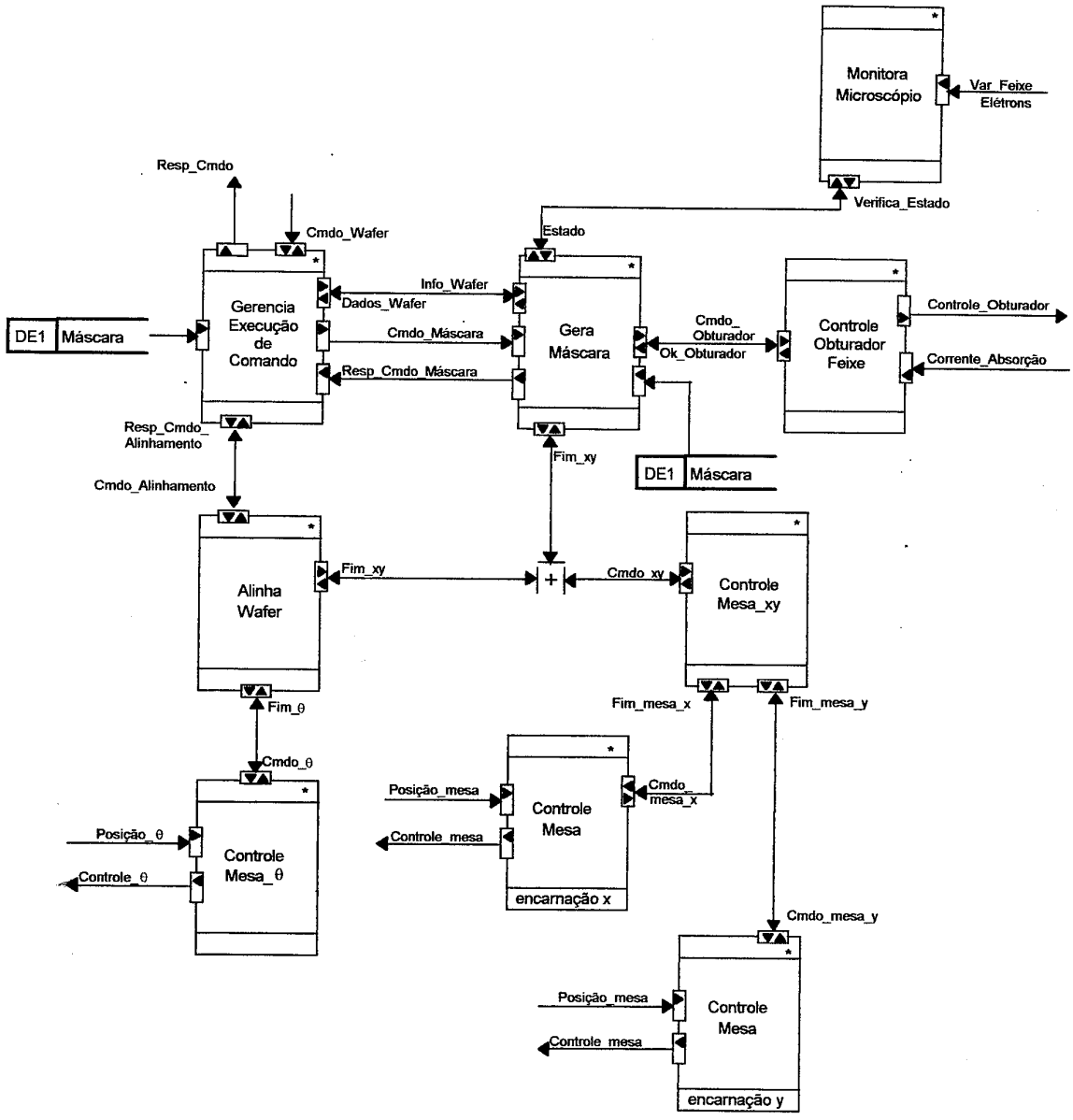


Figura 5- Esquema de Subsistemas Básicos

2.3.2 Determinação das Portas

A descrição que se segue reflete as decisões de design que conduziram aos subsistemas básicos identificados previamente. Essas decisões impõem uma configuração específica para as conexões entre esses subsistemas e o formato da descrição emprega um referenciamento cruzado de subsistemas (por exemplo, SB1 pede serviço a SB2). Evidentemente, esse referenciamento visa proporcionar uma descrição simples da configuração e não é contraditório com a característica de autonomia dos subsistemas. Cada um destes, na realidade, ao pedir um serviço, comunica-se apenas com uma de suas portas de saída, a qual foi projetada de modo a participar da conexão correta com o subsistema que atenderá à requisição.

⇒ **Para o subsistema "Gerencia Execução de Comando"**

- pede serviço a **Gera Máscara**, que deve efetivamente gerar camada sobre wafer. **Gera Máscara** retorna periodicamente o estado da camada que está sendo gerada. Durante a geração de uma máscara sobre o wafer o operador pode requisitar relatórios de gravação de outros wafers ou interromper o processo em andamento. Assim sendo, deve haver uma porta de saída de **Gerencia Execução Comando** para pedido de serviço e outra de entrada independente por onde retorna estado e fim de execução;
- pede serviço a **Gera Máscara**, que deve informar dados de geração de uma ou mais camadas sobre um dado wafer ("relatório_wafer"). Assim sendo, deve haver uma porta de saída com resposta associada;
- pede serviço a **Alinha Wafer** que deve efetuar o alinhamento de wafer. O serviço de alinhar wafer se subdivide em: iniciar alinhamento, ajustar marca nas direções x ou y e confirmar marca. Assim sendo, para cada um desses serviços não concorrentes, deve-se associar uma resposta que recebe a informação de sucesso ou insucesso do comando de alinhamento.

⇒ **Para o subsistema "Gera Máscara"**

- possui as portas necessárias à prestação dos serviços requisitados por **Gerencia Execução de Comando** para geração de máscara;
- pede serviço a **Controle Mesa_xy** e a **Controle Obturador Feixe** para posicionamento da Mesa nas direções x e y e para abertura/fechamento de obturador do feixe respectivamente. Como a execução de gravação sobre wafer deve esperar a execução de cada um desses serviços acima mencionados, as portas por onde são requisitados os serviços devem ser de saída com resposta associada;
- pede serviço a **Monitora Microscópio** para inspeção do microscópio eletrônico antes de iniciar a gravação do wafer e para monitoramento de estado do microscópio eletrônico durante a gravação do wafer. **Gera Máscara** é, portanto, responsável por iniciar o monitoramento e requisitar periodicamente o estado do microscópio. Trata-se de serviço interno ao subsistema **Gera Máscara** que, após cada requisição, espera a resposta que permitirá ou não a continuidade do processo de geração de camada.

⇒ **Para o subsistema "Monitora Microscópio"**

- esse subsistema realiza serviço de inspeção periódica do microscópio eletrônico, serviço este que só necessita estar ativo durante a geração de camada. Essa inspeção é realizada segundo valores fornecidos ao subsistema na ativação do serviço. Portanto, os seguintes serviços não concorrentes podem ser requisitados através da porta de entrada com resposta associada deste subsistema:
  - ♦ iniciar a inspeção periódica do microscopio microscópio eletrônico, verificando se os valores de operação encontram-se conforme padrões especificados na requisição do serviço e informar a situação atual;

- ♦ verificar se valores do microscópio permanecem ou não dentro dos padrões especificados na ativação;
- ♦ desativar o serviço de inspeção periódica.

⇒ **Para o subsistema "Alinha Wafer"**

- ■ possui a porta necessária à prestação dos serviços não concorrentes requisitados por **Gerencia Execução de Comando** para alinhamento de wafer;
- ■ pede serviço a **Controle Mesa_xy** e **Controle Mesa_θ** para movimentação da mesa. Como o processo de alinhamento necessita esperar o término da movimentação, são portas de saída com resposta associada.

⇒ **Para o subsistema "Controle Mesa_xy"**

- ■ possui a porta necessária à prestação do serviço de movimentação da mesa nas direções x e/ou y.
- ■ pede serviço às encarnações x e/ou y de **Controle Mesa** para requisição do prosicionamento de mesa em cada uma das direções.

Quanto aos demais subsistemas básicos, as portas alocadas são as necessárias à prestação de serviços por eles oferecidos. Essas portas devem ser compatíveis com as que fazem a requisição dos serviços comentados precedentemente.

Portas com o ambiente externo são derivadas de agregações de fluxos de entrada/saída presentes no Esquema Transacional do Modelo da Essência (modelagem do contexto). A configuração de portas e subsistemas básicos correspondentes aos serviços prestados pela aplicação se encontram na Figura 5.

2.4 Derivando Organização Hierárquica

Na modelagem da essência, a organização hierárquica do Modelo do Comportamento reflete uma organização baseada nos critérios de baixo acoplamento e alta coesão aplicados a atividades essenciais. Já o Projeto Básico visa primordialmente à descoberta das classes que compõem o sistema, a sua implementação como subsistemas básicos, à configuração de software (conexões entre portas de subsistemas básicos) e à identificação do número de encarnações de cada subsistema.

O Projeto Básico também explicita os serviços que cada subsistema básico executa. Diferentemente dos assuntos presentes no Modelo da Essência, esses serviços podem ser requisitados pelo ambiente externo e por outros subsistemas básicos.

A derivação das classes e conseqüentemente dos subsistemas básicos, é feita, conforme a descrição apresentada na seção 2.3.1, a partir do Modelo do Comportamento. Novamente, um processo de agregação de baixo para cima se faz necessário para um melhor entendimento da composição do sistema em termos de subsistemas. Essa organização hierárquica usa como critérios principais de agregação alta coesão e baixo acoplamento, já empregados na construção da organização hierárquica do Modelo do Comportamento e das seguintes restrições:

- ♦ um serviço só é executado por um subsistema;
- ♦ a agregação deve levar em consideração o reúso de subsistemas de grande porte;

A partir desse instante são anexados alguns subsistemas básicos característicos da alternativa de implementação escolhida. Esses subsistemas dizem respeito à Interface Usuário-Máquina (IUM) e à utilização específica de recursos de aplicativos. No caso particular do COMONLIFE, é adicionado apenas um subsistema básico para tratamento da IUM. Esse subsistema aparece no nível mais alto da organização hierárquica, separado dos subsistemas que tratam da aplicação, conforme os critérios de organização mencionados em [Sanchez95.1].

Diferentemente de outras técnicas de design, considerações de desempenho só serão levadas em conta na definição da arquitetura de hardware e na configuração do software nessa arquitetura.

A restrição que impõe um serviço por subsistema e as diferenças, indicadas precedentemente, existentes entre assuntos e serviços são responsáveis pela geração de uma organização hierárquica do Projeto Básico diferente daquela obtida no Modelo da Essência: a organização hierárquica do Projeto Básico reflete apenas parcialmente a Organização Hierárquica das Atividades gerada no Modelo do Comportamento. Visando aumentar o grau de reúso de subsistemas, bem como a manutenibilidade do sistema, as características relacionadas à implementação da comunicação direta entre o sistema e dispositivos sensores/atuadores específicos devem, sempre que possível, ser agregadas em subsistemas independentes daqueles que implementam os serviços específicos da aplicação. Trata-se de um tratamento análogo ao que deve ser dado à implementação da Interface Usuário-Máquina, onde é sempre possível essa separação. Em resumo, o design deve refletir um tratamento uniforme para todas as interfaces com o ambiente externo, tanto envolvendo usuários quanto envolvendo dispositivos.

Para exemplificar o exposto no parágrafo anterior, considere-se um sistema A responsável pelo tratamento dos dados enviados por um sensor que pode ser fornecido por diferentes fabricantes. As alternativas referentes à escolha do sensor produzido por um ou outro fabricante associam-se a opções diferentes de implementação. O critério acima exposto prescreve encapsular em subsistemas diferentes:

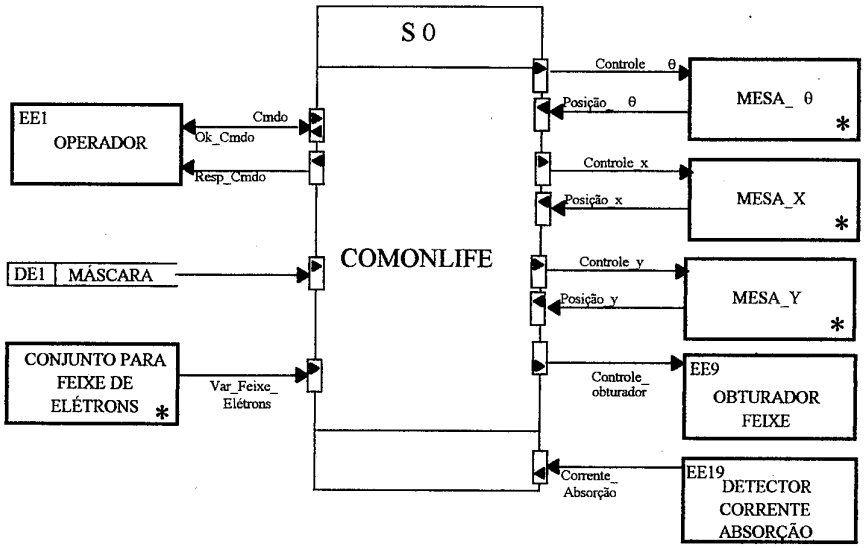
- o tratamento direto do sensor produzido por um fabricante específico (por exemplo, encapsulando esse tratamento no subsistema X);
- o tratamento dos dados, relativos àquela classe de sensores, na forma que interessa à aplicação associada ao sistema A (por exemplo, encapsulando esse tratamento no subsistema Y).

Esse critério facilita o **reúso** do subsistema X no sistema B, cuja aplicação pode usar os dados extraídos do mesmo sensor de forma completamente diferente daquela que interessa à aplicação associada ao sistema A. Também, aumenta a **manutenibilidade** do sistema A: ao ser trocado o sensor inicialmente usado na implementação do sistema A por outro mais moderno, do mesmo ou de outro fabricante, basta substituir o subsistema X pelo subsistema Z que efetua o tratamento direto do novo sensor.

3 Projeto Básico

3.1 Visão de Contexto

Essa seção apresenta o Diagrama de Contexto da implementação do sistema COMONLIFE e as estruturas a ele associadas. Esse diagrama emprega a linguagem de representação dos Diagramas de Estrutura e modela a fronteira existente entre o ambiente externo e o sistema - Entidades Externas que interagem diretamente com o sistema e as mensagens que fluem entre o sistema e essas entidades. Essa visão esquemática do contexto visa facilitar o entendimento das agregações dos Eventos Externos presentes no Modelo da Essência em subconjuntos correspondentes aos serviços a serem prestados pelo sistema.



O asterisco ("*") colocado no canto inferior direito do símbolo de entidade externa indica tratar-se de uma agregação de entidades externas.

Figura 6- Diagrama de Contexto do Design do COMONLIFE

3.1.1 S0 - Comonlife

Serviços a executar

1) Gerenciar a entrada de comandos, interpretando-os.

   (Serviços de IUM acrecentados ao Modelo da Essência)

2) Informar os resultados de execução dos comandos.

   (Serviços de IUM acrecentados ao Modelo da Essência)

3) Executar serviços requisitados através da introdução de comandos pelo operador.

3.1) Iniciar wafer.

3.2) Alinhar wafer.

3.3) Gerar camada Genérica.

3.4) Gerar Camada.

3.4.1) Iniciar a Geração de Camada
(fim de alinhamento e não ocorrência do evento externo 29)

3.4.2) Interromper o processo de geração de camada a pedido do operador
(evento externo 6)

3.4.3) Fornecer um histórico sobre a gravação de determinado wafer.
(eventos externos 7, 8)

4) Atuar/Monitorar os equipamentos necessários à geração de camada.

4.1) Posicionar Mesa xy

4.2) Posicionar Mesa θ

4.3) Controlar Obturador do Feixe.

4.4) Monitorar Microscópio Eletrônico.

Os nomes de serviços não comentados correspondem aqueles relacionados na seção 2.1

3.1.2 Agregações de Entidades

Nome: CONJUNTO PARA FEIXE ELÉTRONS
Composição: EE10_FILAMENTO +
EE11_FONTE_ALTA_TENSÃO +
EE12_LENTE_OBJETIVA +
EE13_LENTE_CONDENSADORA

Nome: MESA_θ
Composição: MOTOR_θ +
DETECTOR_θ

Nome: MESA_X
Composição: MOTOR_X +
INTERFERÔMETRO_ÓTICO_X

Nome: MESA_Y
Composição: MOTOR_Y +
INTERFERÔMETRO_ÓTICO_Y

3.1.3 Agregações de Fluxos

Nome: Cmdo
Composição: FDD_pedido_início_alinhamento |
FDD_pedido_iniciação |
FDD_pedido_relatório_wafer |
S_interrupção_máscara |
S_confirmação_marca |
FDD_ajuste_marca_x |
FDD_ajuste_marca_y

Nome: Resp_Cmdo
Composição: FDD_rejeição_início_alinhamento |
FDD_rejeição_iniciação |
FDD_tensão_e_corrente |
FDD_rejeição_relatório_wafer |
FDD_relatório_wafer |
FDD_rejeição_geração_máscara |
FDD_interrupção_processo |
S_término_processo |
FDD_status

Nome: Var_Feixe_Elétrons
Composição: FDC_variáveis_feixe_elétrons

Nome: Controle_θ
Composição: FDC_controle_θ_motor

Nome: Posição_θ
Composição: FDC_θ_mesa

Nome: Controle_x
Composição: FDC_controle_x_motor

Nome: Posição_x
Composição: FDC_x_mesa

Nome: Controle_y
Composição: FDC_controle_y_motor

Nome: Posição_y
Composição: FDC_y_mesa

Nome: Controle_Obturador
Composição: FDC_controle_obturador

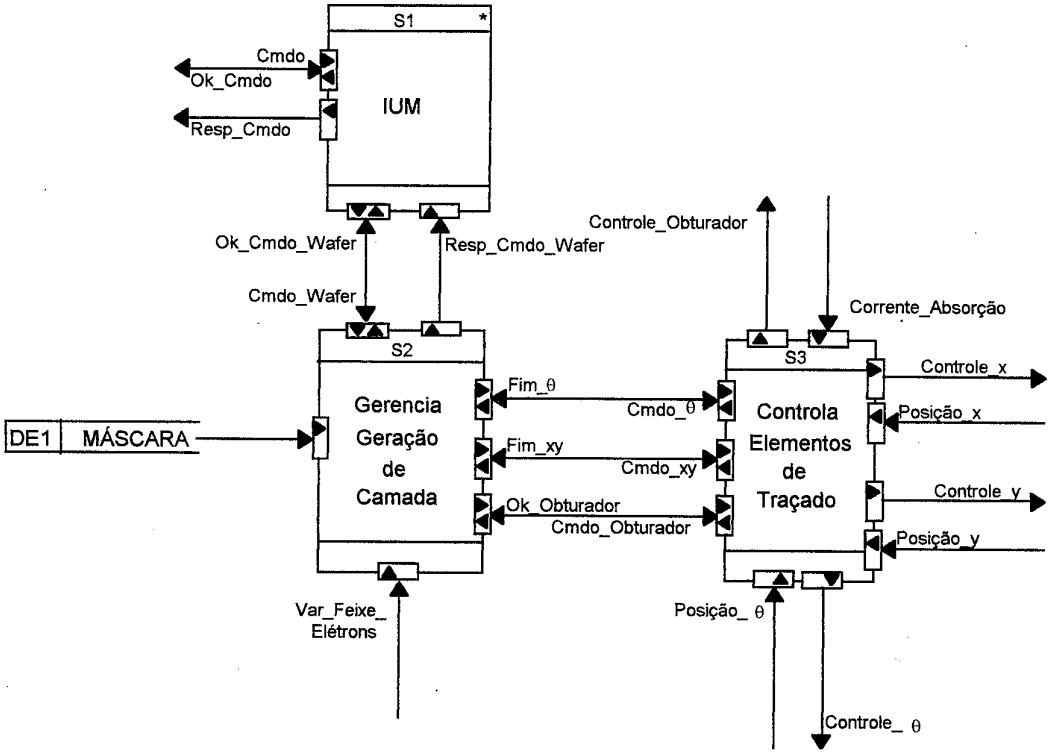
Nome: Corrente_Absorção
Composição: FDC_y_mesa

3.2 Estrutura do Sistema

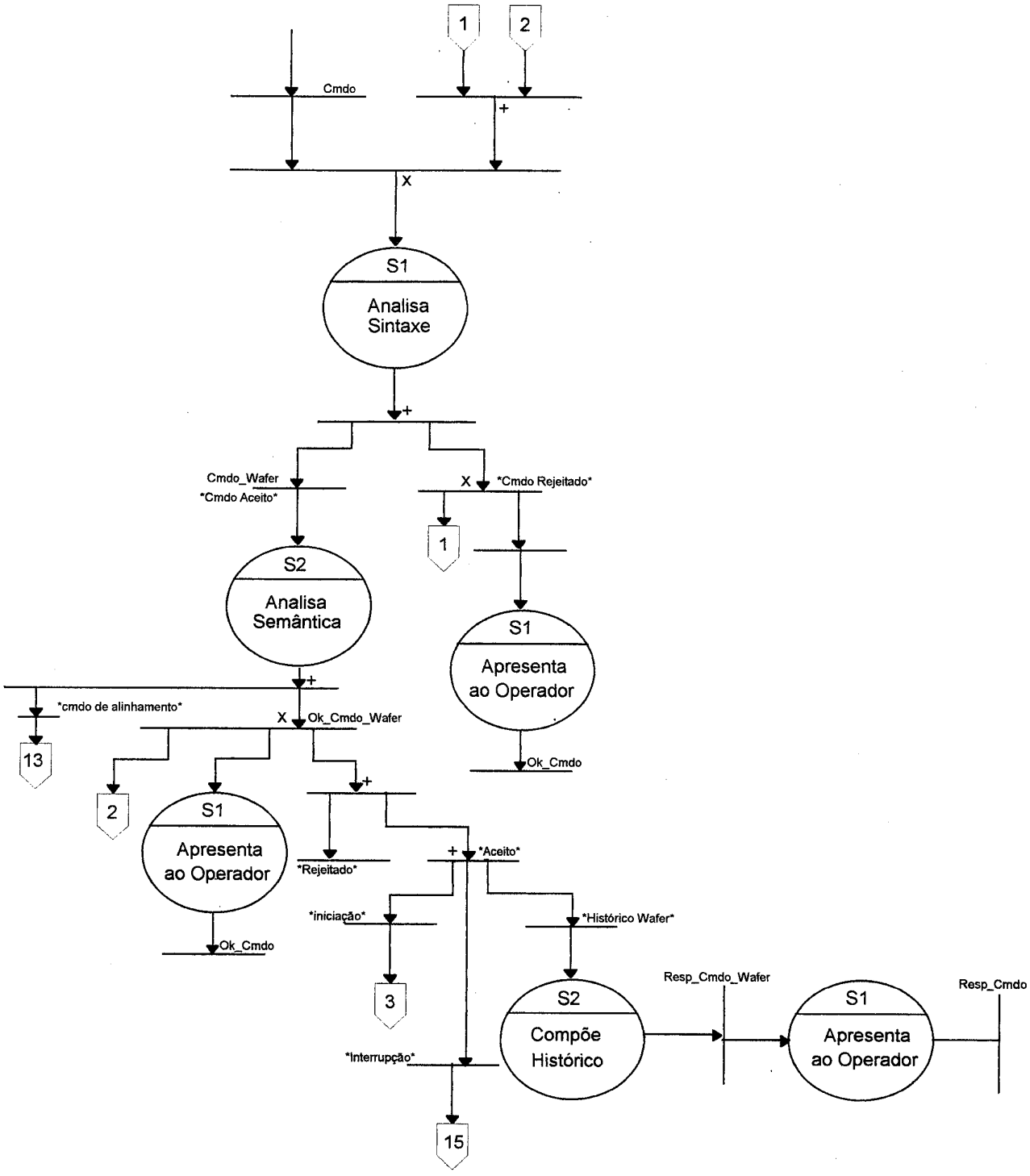
Esta seção apresenta a decomposição do sistema em subsistemas até o nível de subsistemas básicos. Essa decomposição é representada através dos Diagramas de Estrutura e de Comportamento cuja semântica reflete, respectivamente, a relação estrutural entre subsistemas componentes e a relação dinâmica entre os elementos de cada estrutura (seqüência e concorrência existente entre os subsistemas componentes).

3.2.1 S0 - COMONLIFE

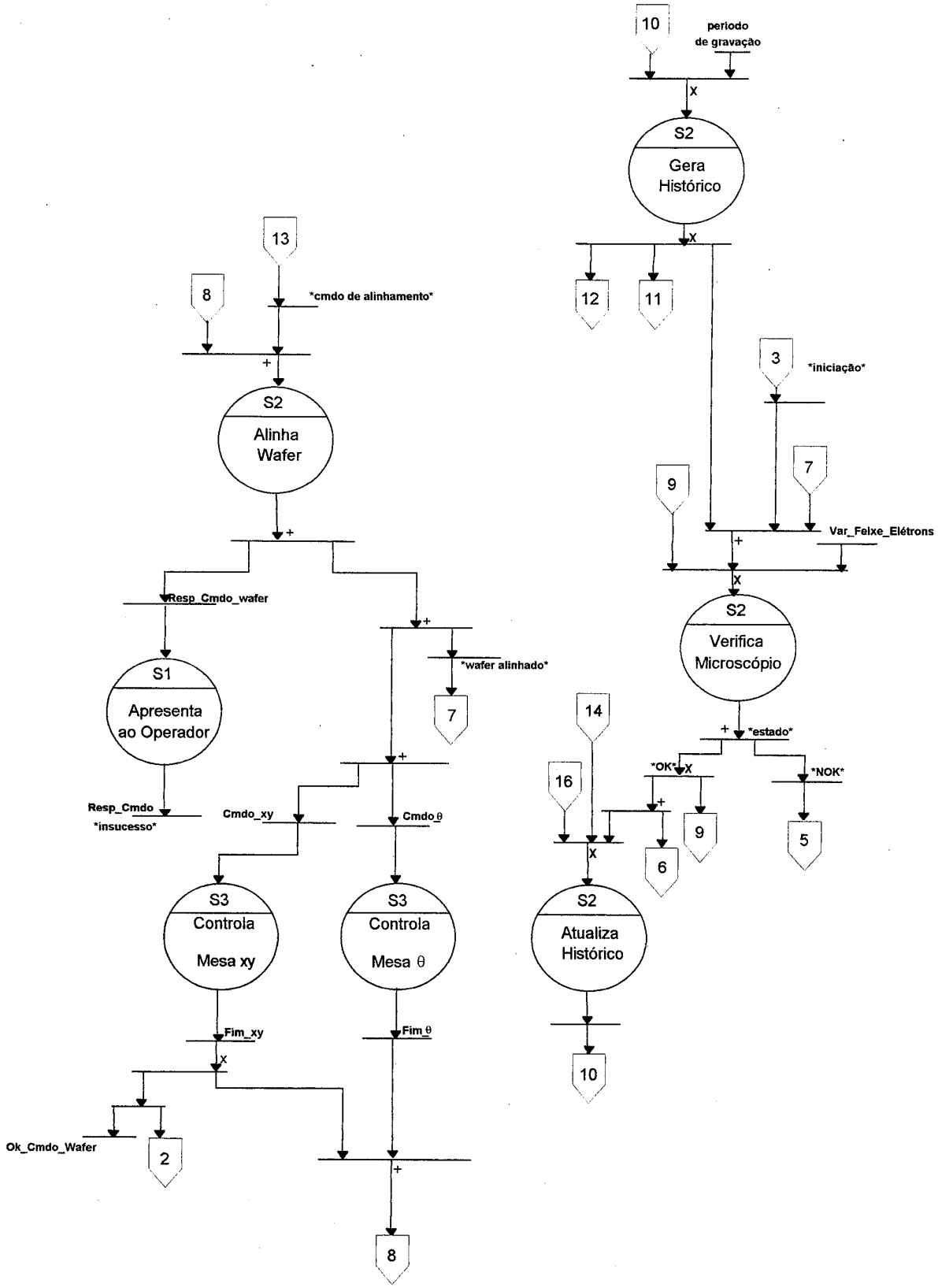
DE /S0 - COMONLIFE



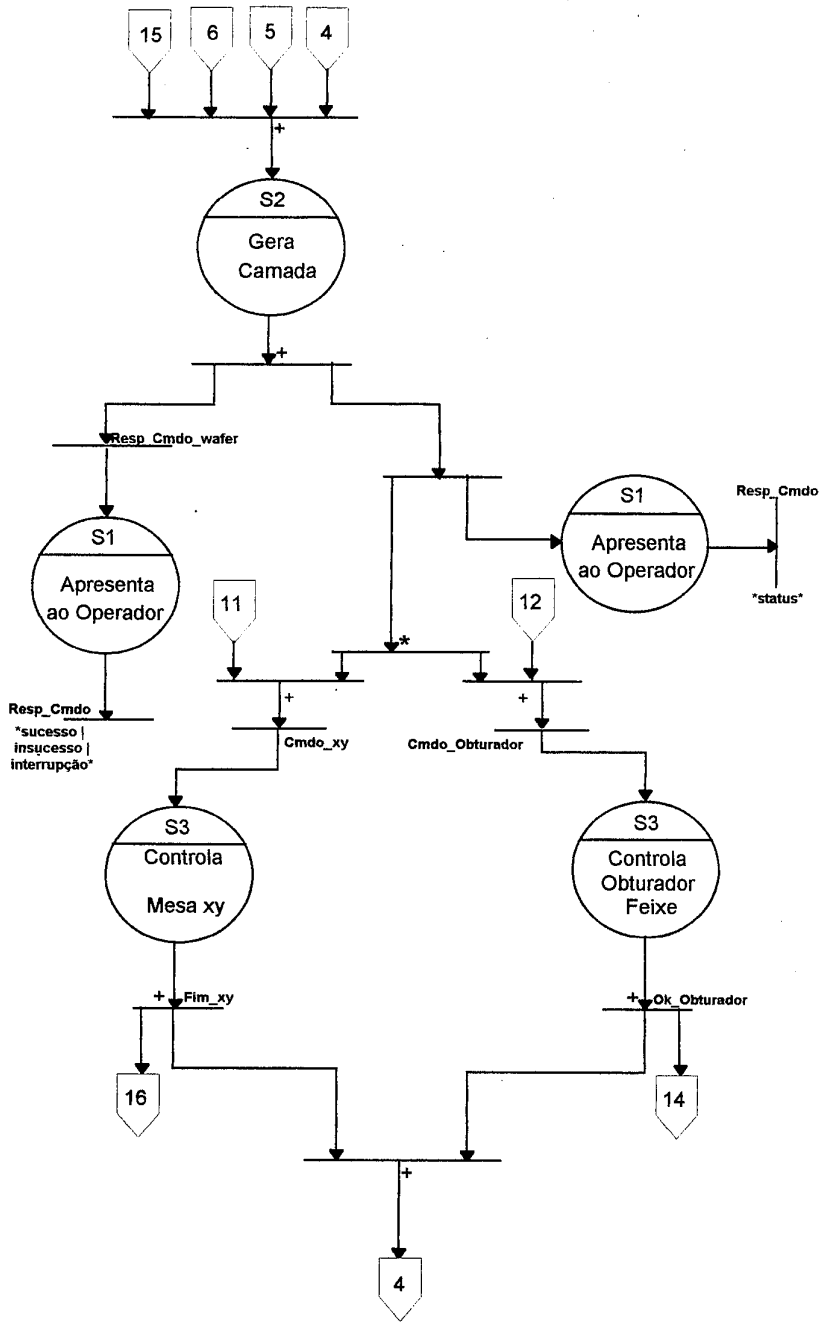
DC - COMONLIFE (1/3)



DC- COMONLIFE (2/3)



DC - COMONLIFE (3/3)



Serviços a Executar para cada Subsistema

S1 - Interface Usuário Máquina (IUM)

1) Gerenciar a entrada de comandos, interpretando-os.

2) Informar resultados da execução de comandos.

S2 - Gerencia Geração de Camada

1) Iniciar wafer.

2) Alinhar wafer.

3) Gerar camada Genérica.

4) Gerar Camada.

4.1) Iniciar a Geração de Camada

(fim de alinhamento e não ocorrência do evento externo29)

4. 2) Interromper o processo de geração de camada a pedido do operador

(evento externo 6)

4.3) Fornecer um histórico sobre a gravação de determinado wafer.

(eventos externos 7,8)

5) Monitorar Microscópio Eletrônico.

S3 - Controla Elementos de Traçado

1) Posicionar Mesa xy

2) Posicionar Mesa θ

3) Controlar Obturador do Feixe.

Justificativa das Decisões de Design

As principais decisões de design, nas quais está baseada a segmentação do sistema, são as seguintes:

- Para a IUM foram alocados os serviços relativos à entrada de comandos e à apresentação de resultados e mensagens para o operador. Os serviços alocados a Gerencia Geração de Camada dizem respeito à análise semântica e à execução dos comandos relativos à gravação de um wafer. O subsistema Controla Elementos de Traçado coordena as funções associadas ao tratamento dos sensores/atuadores relacionados ao traçado de máscaras.

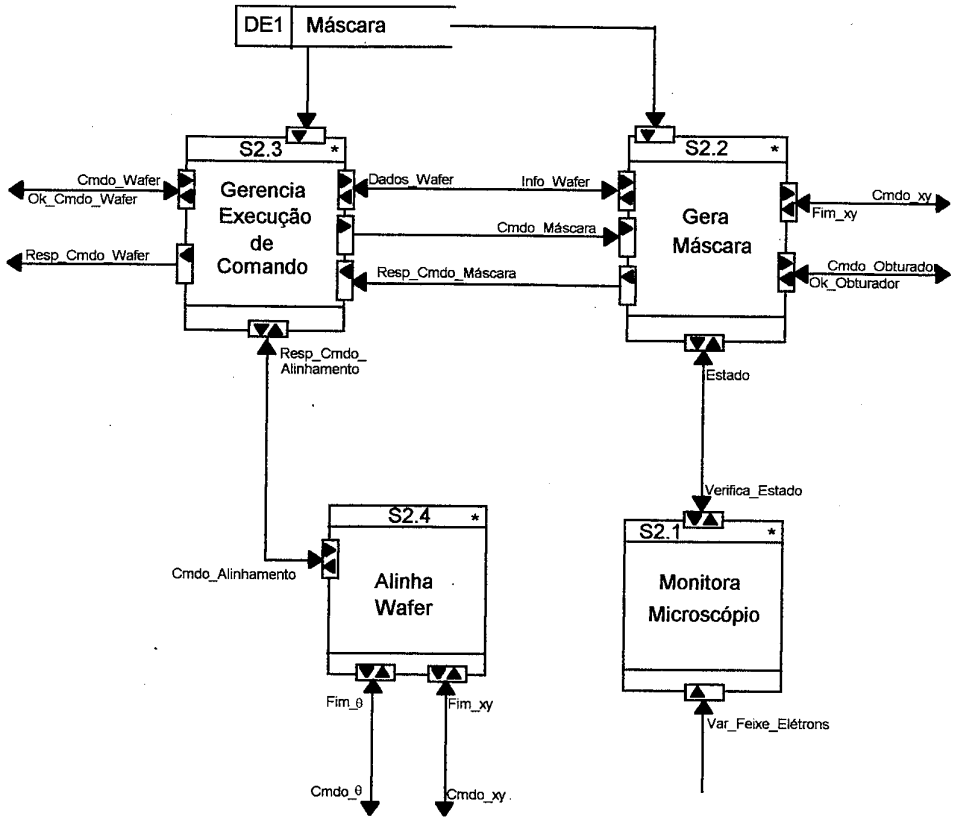
Essa segmentação se justifica a partir dos seguintes critérios:

- Encapsular a IUM em um subsistema específico conforme descrito em [Sanchez95.1], decisão essa baseada em características de reúso comprovadas em [CCCL93, CILS93a, CILS93b, Cowan95];
- Tratar separadamente sensores e atuadores com serviços relacionados ao traçado de máscara. Encapsular separadamente o tratamento de sensores e atuadores, analogamente ao que foi feito para a IUM, aumenta a probabilidade de reúso. No entanto, é importante que o tratamento de sensores/atuadores corresponda a um encapsulamento em subsistemas onde seja respeitado o princípio da coesão máxima entre serviços prestados por esses sensores/atuadores e prestados a outros subsistemas;
- Encapsular em um único subsistema os serviços relacionados ao objetivo principal do sistema (geração de chips);

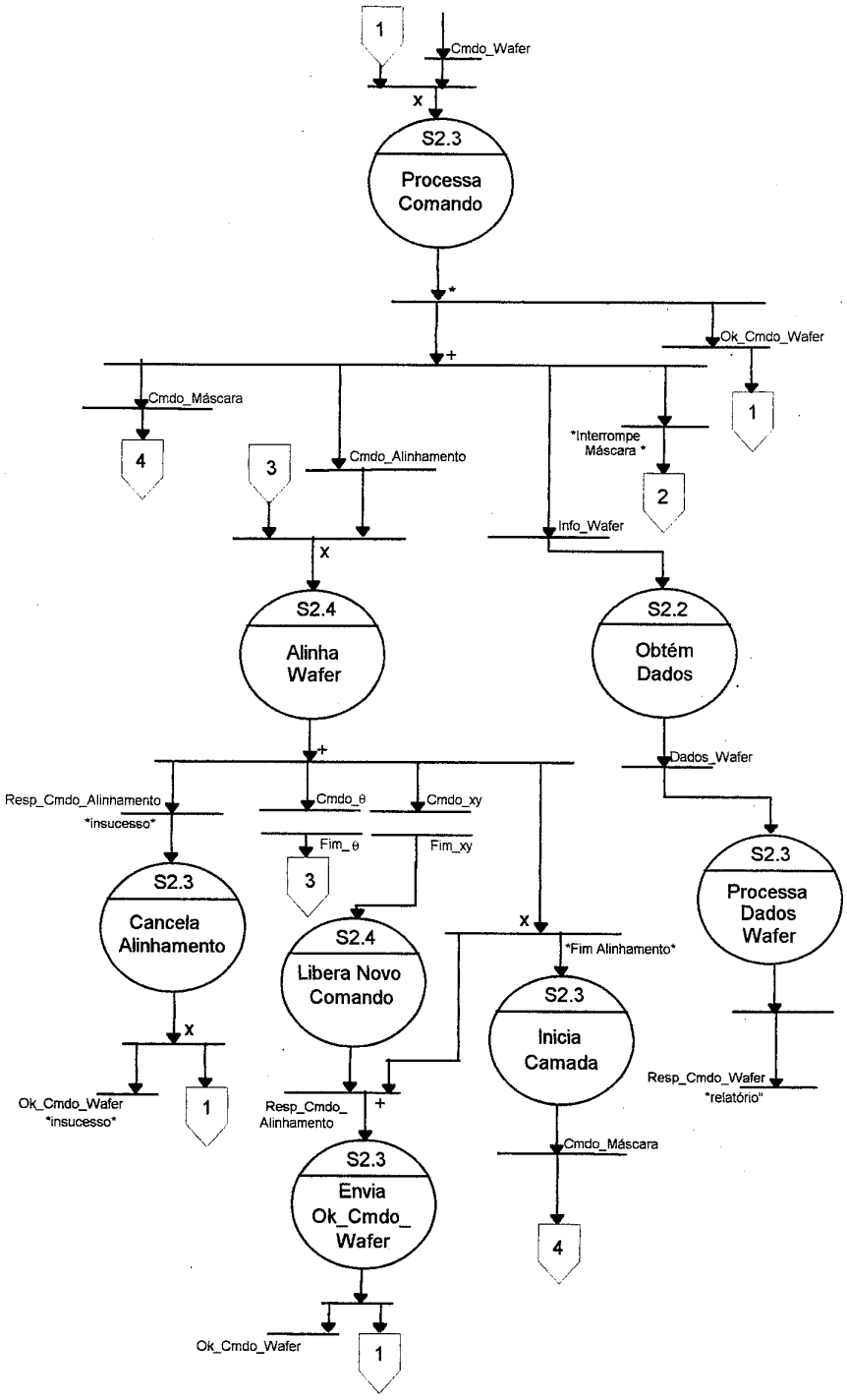
♦ Distorções em relação à organização hierárquica do Modelo da Essência visam aumentar o grau de reúso dos subsistemas, aumentar a independência desses subsistemas e obter a não redundância de código (um serviço só é prestado por um subsistema). Como conseqüência, o grau de manutenibilidade do sistema deve aumentar. A organização hierárquica da modelagem conceitual, apesar de ser a mais apropriada para o entendimento do sistema, permite redundância de atividades em decorrência da hipótese de emprego de tecnologia ideal.

3.2.2 Gerencia Geração de Camada

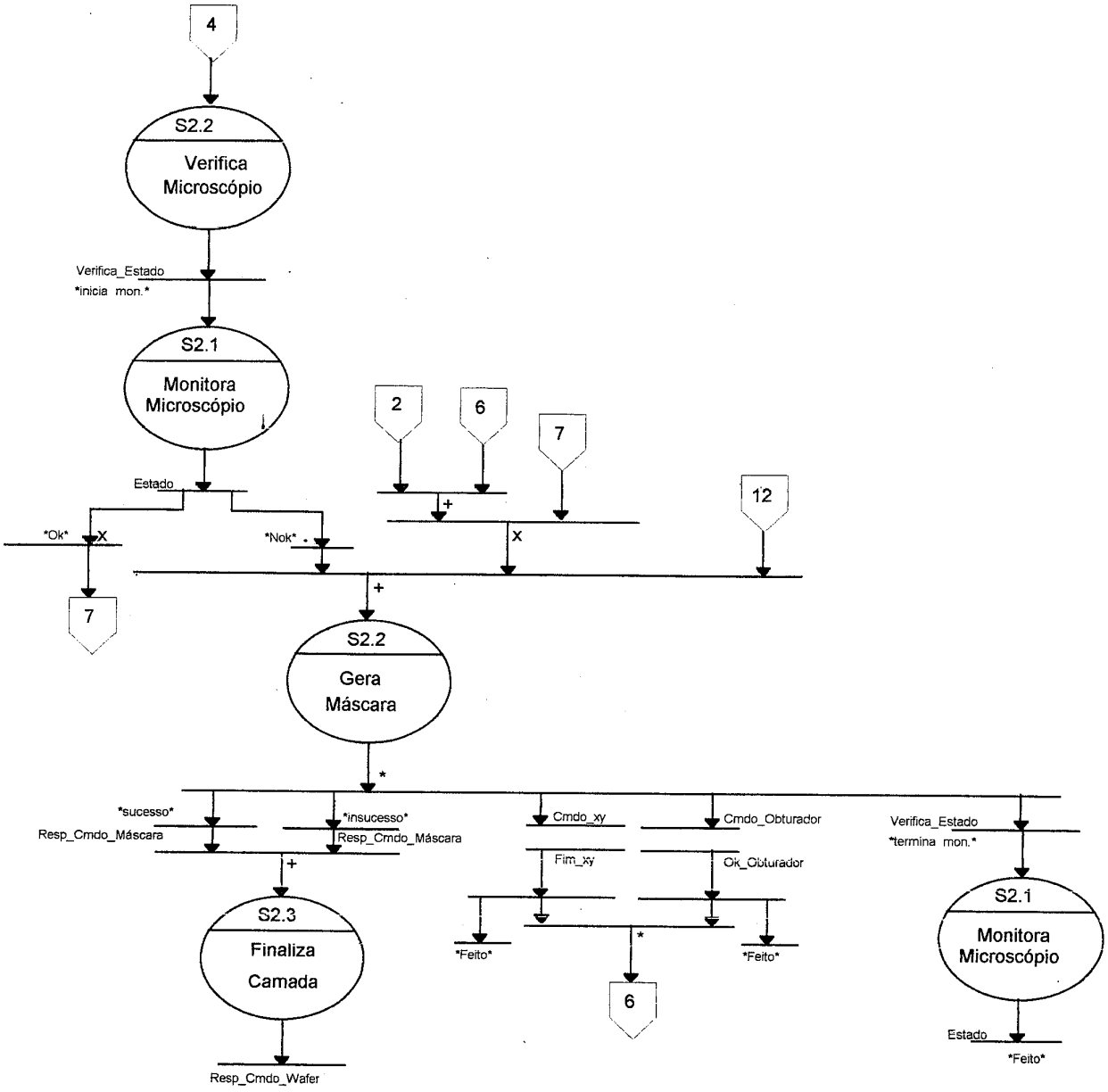
DE / S2 - Gerencia Geração de Camada



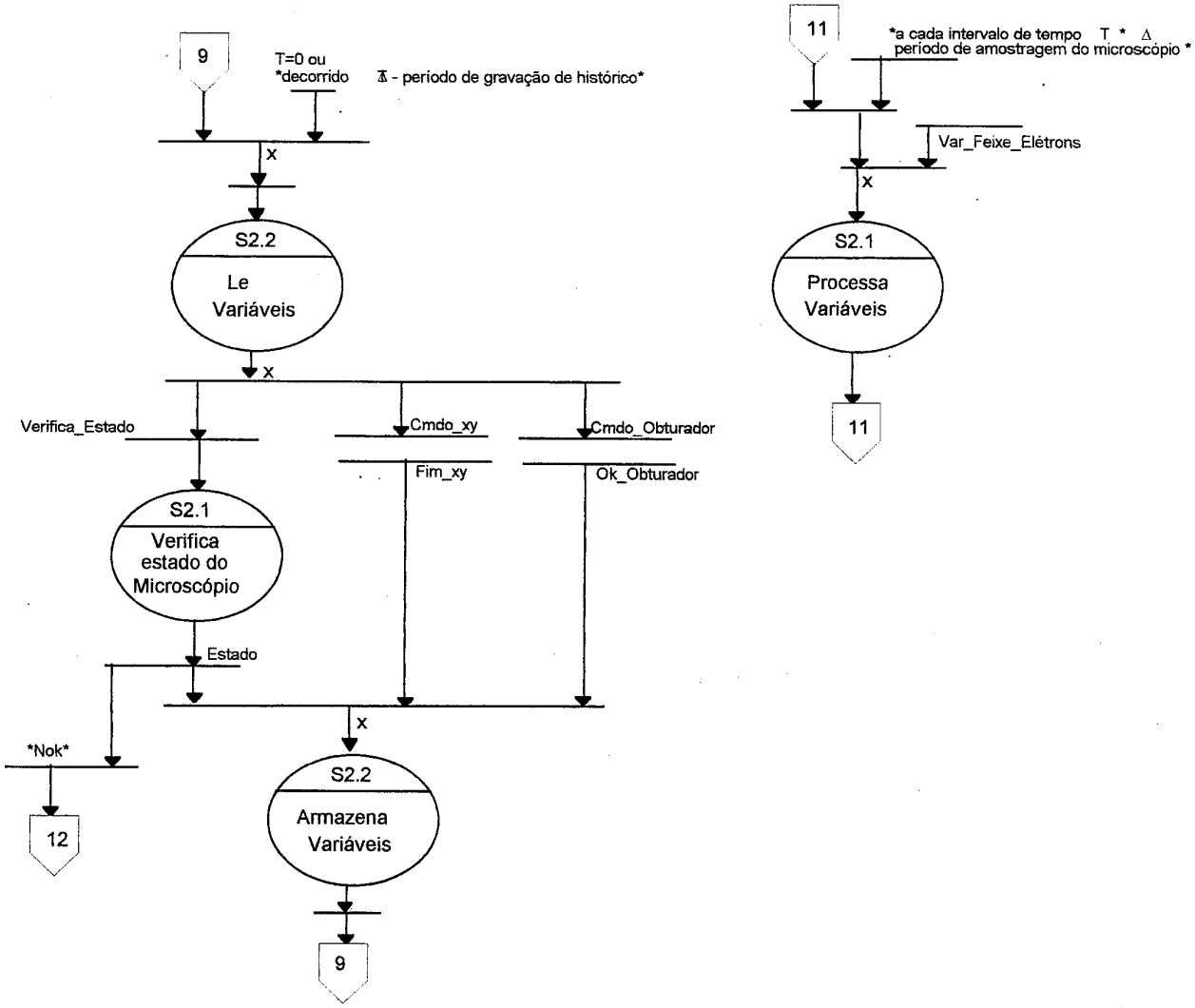
DC - Gerencia Geração de Camada (1/3)



DC - Gerencia Geração de Camada (2/3)



DC - Gerencia Geração de Camada (3/3)



Serviços a Executar para cada Subsistema

S2.1 - Monitora Microscópio

1) Processar as informações provenientes do microscópio eletrônico.

2) Informar se o microscópio eletrônico está operando na faixa aceitável conforme parâmetros de operação do wafer.

S2.2 - Gera Máscara

1) Fornecer um histórico sobre a gravação de um wafer.

2) Gerar uma camada num wafer.

S2.3 - Gerencia Execução de Comando

1) Analisar semanticamente os comandos, gerenciando as ações necessárias para sua execução.

1.1) Informar o estágio atual da geração do wafer

1.2) Processar as respostas das várias etapas de execução de um comando.

2) Processar as informações provenientes do depósito de máscaras DE1_MÁSCARA.

S2.4 - Alinha Wafer

1) Efetuar o alinhamento de um wafer.

2) Processar os comandos relativos ao alinhamento do wafer.

2.1) Confirmar marca de alinhamento

2.2) Ajustar marca de alinhamento.

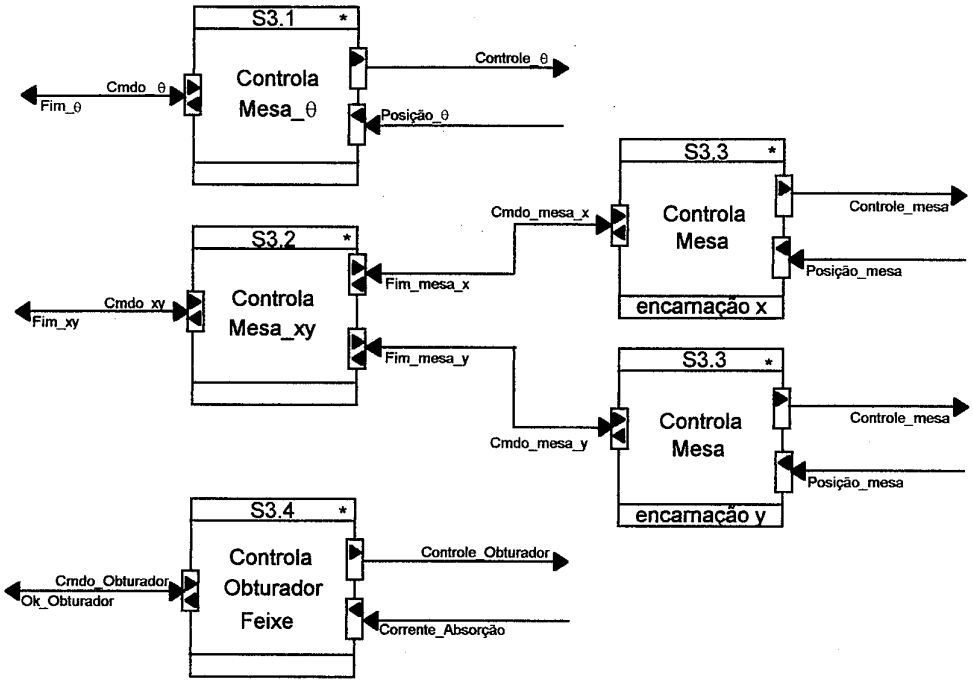
Justificativa das Decisões de Design

As principais decisões de design nas quais está baseada essa segmentação são:

- A decomposição em subsistemas procurou atender primordialmente ao critério do reúso. Em particular, isso acarreta a independência entre os subsistemas que efetuam tratamento de sensores e os subsistemas relativos ao tratamento das ações específicas da aplicação. Em Monitora Microscópio, é alocado o tratamento dos sensores que compõem o Conjunto Para Feixe Elétrons.
- Os subsistemas Gera Máscara e Alinha Wafer executam serviços bastante distintos do sistema. O primeiro gerencia a reprodução de máscaras sobre o wafer, enquanto o segundo garante o posicionamento do wafer em relação ao eixo de coordenadas absolutas. O subsistema Gerencia Execução de Comando é responsável pela análise semântica e coordenação da execução de comandos.

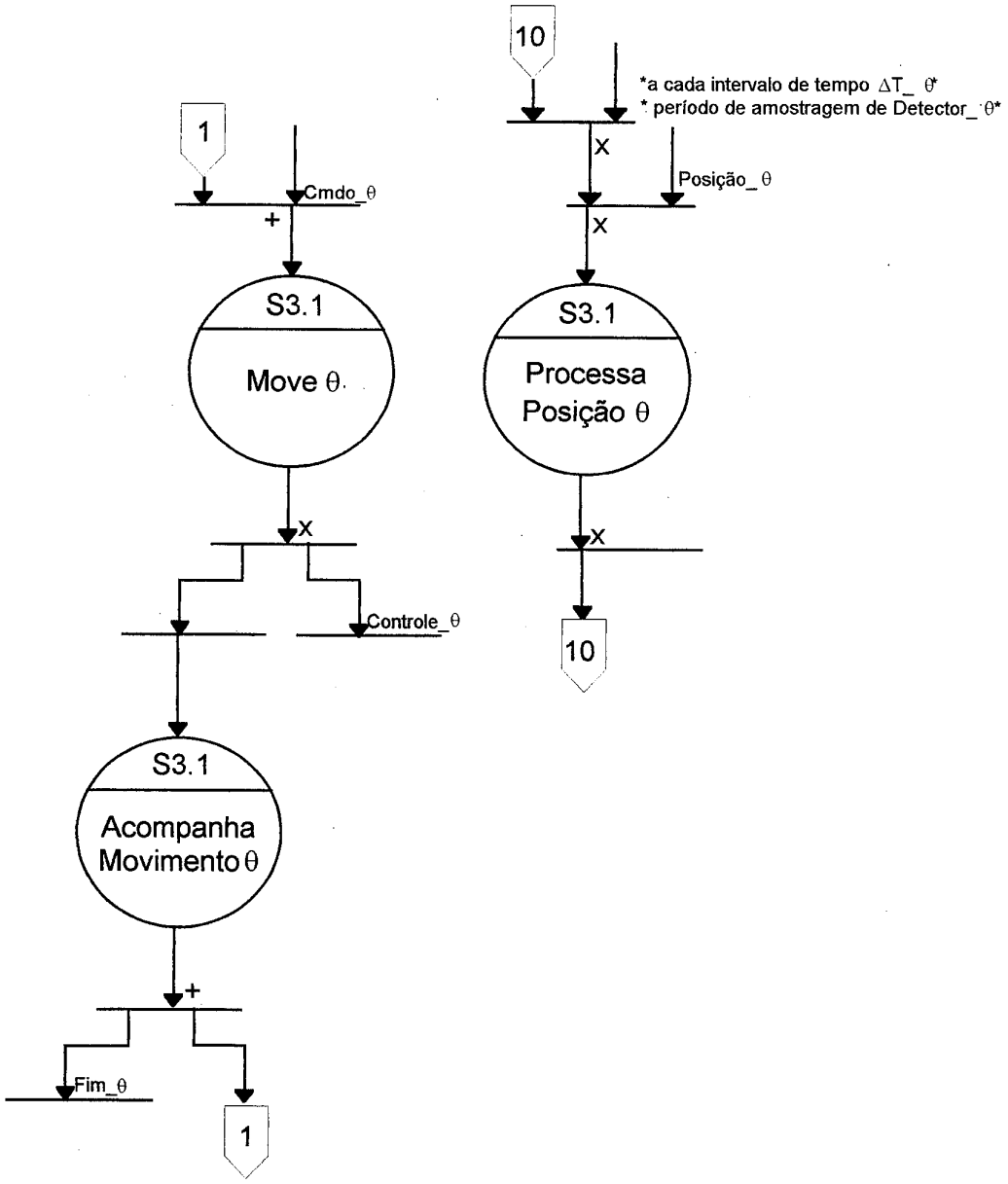
3.2.3 Controla Elementos de Traçado

DE / S3 - Controla Elementos de Traçado

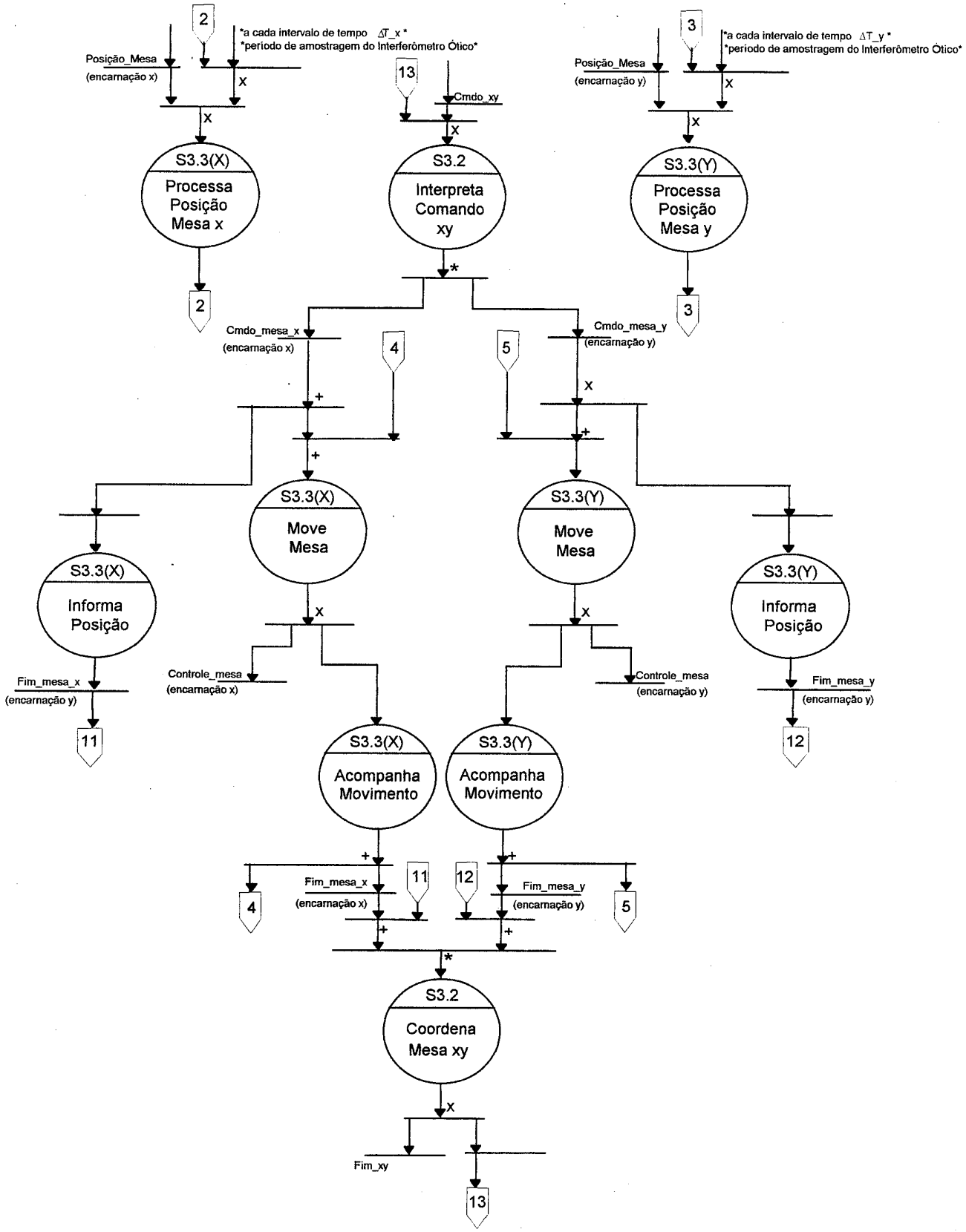


Controle_x = Controle _mesa emitido pela encarnação x de S3.3
Posição_x = Posição _mesa recebido pela encarnação x de S3.3
Controle_y = Controle _mesa emitido pela encarnação x de S3.3
Posição_y = Posição _mesa recebido pela encarnação x de S3.3

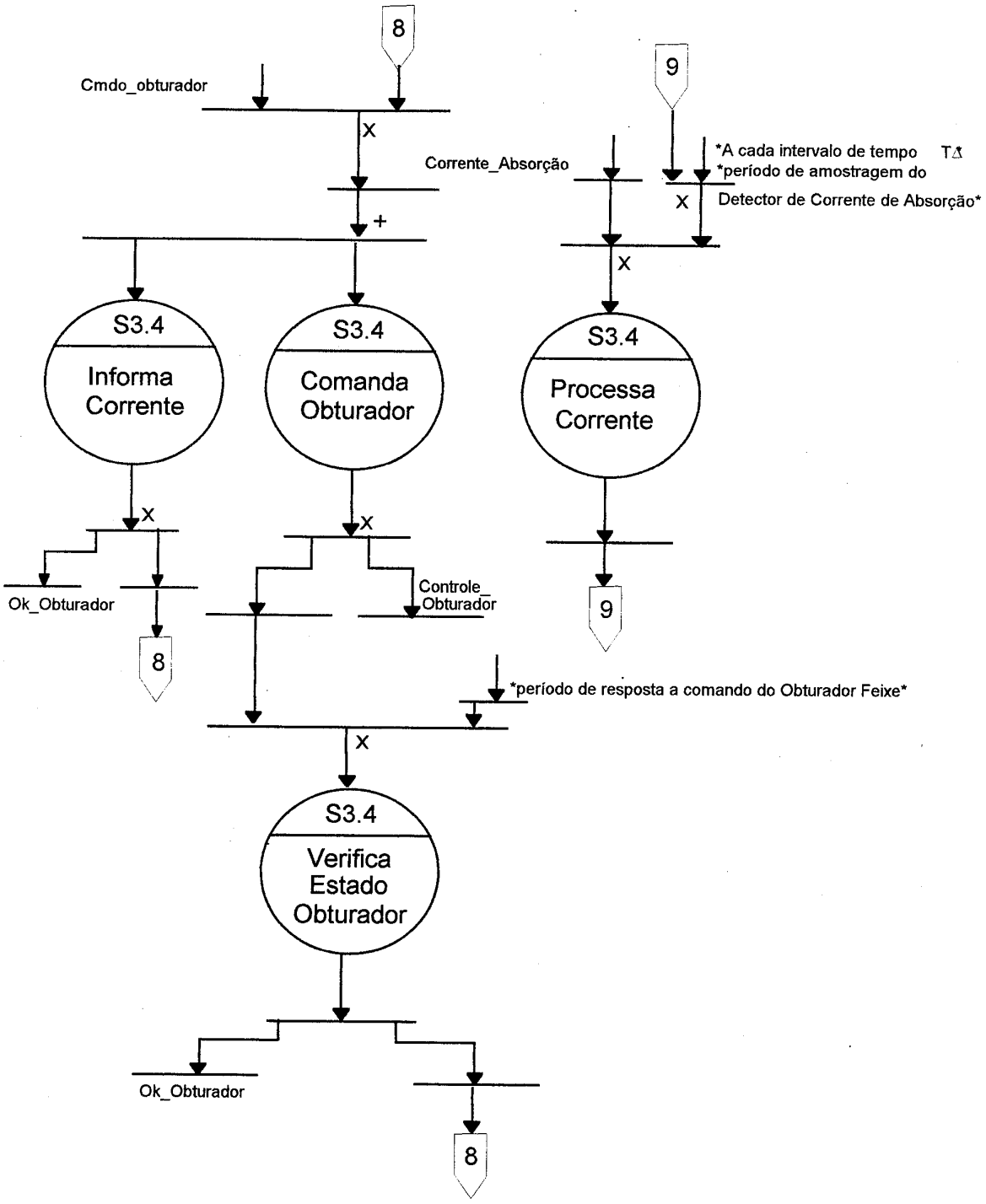
DC - Controla Elementos de Traçado (1/3)



DC - Controla Elementos de Traçado (2/3)



DC - Controla Elementos de Traçado (3/3)



Serviços a Executar para cada Subsistema

S3.1 - Controla Mesa_θ

1) Processar as informações referentes ao deslocamento angular θ da mesa.

2) Efetuar deslocamento angular θ da mesa.

S3.2 - Controla Mesa_xy

1) Processar as informações referentes ao deslocamento simultâneo da mesa nas direções x e y.

2) Efetuar o deslocamento da mesa nas direções x e y, simultaneamente, mantendo essa movimentação sobre uma diagonal.

S3.3 - Controla Mesa

1) Processar as informações referentes ao deslocamento da mesa em uma direção.

2) Efetuar deslocamento linear da mesa, mantendo sua movimentação dentro de uma faixa de tolerância de erro.

S3.4 - Controla Obturador Feixe

1) Processar as informações do detector de corrente de absorção, que indicam o estado do obturador de feixe.

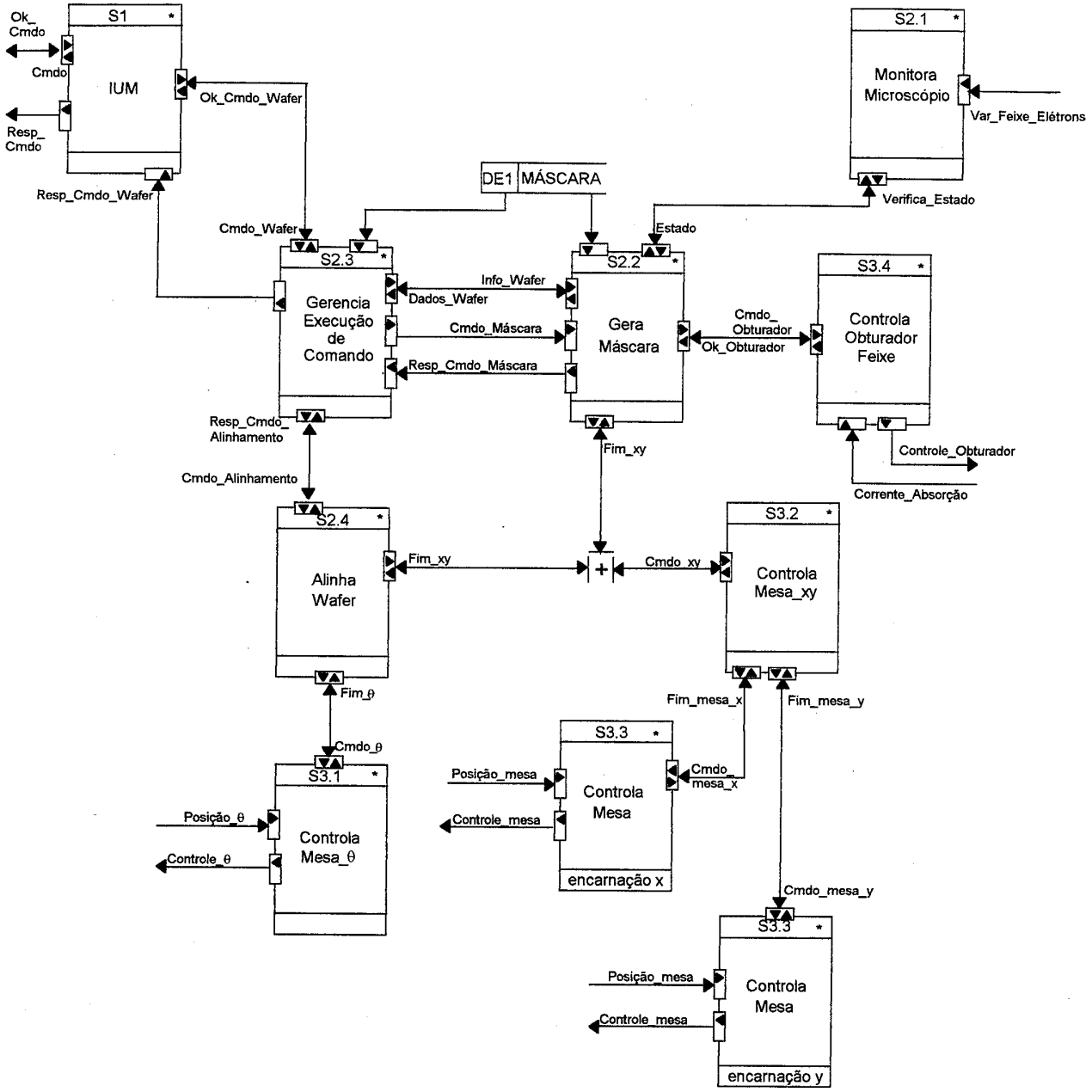
2) Abrir/fechar o obturador de feixe.

Justificativa da Decisões de Design

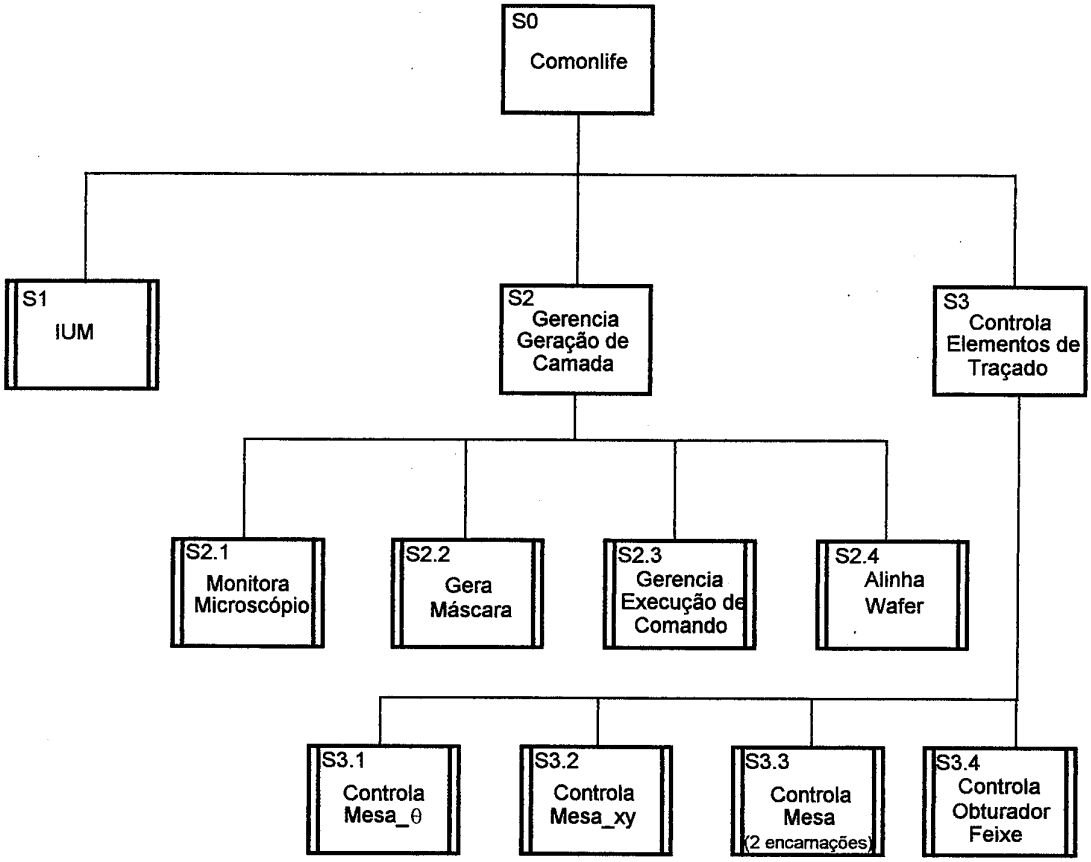
As principal decisão de design na qual está baseada essa segmentação é:

- A segmentação empregada segue rigorosamente o critério da reúso pois encapsula em subsistemas diferentes os grupos de sensores/atuadores usados em conjunto. O controle dos deslocamentos simultâneos da mesa nas direções x e y foi agrupado em Controla Mesa_xy pois deve haver sincronismo nesse processo. Como a característica da movimentação em uma ou outra direção independe de qual seja a direção específica, são utilizadas duas encarnações do mesmo subsistema Controla Mesa para efetuar esse controle.

3.3 Diagrama dos Subsistemas Básicos



3.4 Árvore de Subsistemas



4 Conclusão

O objetivo básico deste trabalho foi realizar um experimento controlado envolvendo a utilização das ferramentas conceituais e técnicas de modelagem da implementação descritas em [Sanchez95.1], onde está definido o método de design empregado. Procurou-se dessa forma evidenciar a adequação desse método para a geração do design do software de aplicação embutido na arquitetura de sistemas processo-controle de grande porte e em ambiente distribuído. A experimentação realizada beneficiou a própria definição do método pois determinou a revisão das partes que se mostraram inadequadas.

Conforme enfatizado em [Glass94], é de fundamental importância o emprego do método científico clássico na pesquisa relacionada ao desenvolvimento de software e isso significa fortalecer a vertente experimental desse método. Tradicionalmente, a pesquisa em engenharia vem sendo realizada, explícita ou implicitamente, com base em um paradigma de ciência experimental: dado um problema, analisam-se as soluções existentes - aí incluindo-se a busca de analogias, para o problema e para as soluções -, propõem-se melhores soluções, realizam-se construções/ desenvolvimentos com base nessas propostas e, repetitivamente, efetua-se a análise comparativa qualitativa/quantitativa dos resultados obtidos, até que não seja mais possível melhorar as soluções propostas. É recomendável e urgente que um paradigma análogo, talvez o mesmo, seja também utilizado em Engenharia de Software.

Dentro desse contexto, foram propostas alterações em métodos existentes para a modelagem de implementação [Ward85, Rumbaugh91, Booch94], com o objetivo específico de enfatizar quatro métricas de qualidade consideradas primordiais para o design de sistemas para controle de processos em tempo-real: Modularidade, Manutenibilidade, Portatilidade e Reúso. O presente trabalho corresponde ao resultado da pesquisa que visou demonstrar a adequação do método proposto ao objetivo enunciado. A pesquisa evoluiu experimentalmente, através da contrução/desenvolvimento do modelo da implementação de um sistema de controle realista. Essa experimentação deu origem ao refinamento e à depuração de ferramentas conceituais e técnicas de modelagem e gerou novo formato para o método de design inicialmente empregado. Esse novo formato, reportado em [Sanchez95.1], resultou em melhor solução para o problema proposto, relacionado à otimização das métricas de qualidade investigadas.

Este trabalho apresenta o Modelo da Implementação de um sistema realista (Sistema para Controle e Monitoramento de Litografia por Feixe de Elétrons - COMONLIFE), possuindo complexidade intrínseca e porte suficientes para testar o método de design a ser validado e compatibilidade, em termos de prazo e custo, com um processo de denvolvimento a ser efetuado em ambiente de pesquisa acadêmica. Essa apresentação, analogamente ao que ocorreu com o Modelo da Essência do COMONLIFE [Sanchez95.2], viola largamente as restrições de espaço impostas pelos meios tradicionais de divulgação técnico-científica. Entretanto, o espaço utilizado é compatível com o porte do sistema escolhido para o experimento realizado e com o propósito da pesquisa. Não é recomendável reduzir o porte do sistema pois é importante que o resultado obtido nessa experiência refira-se a um caso real de desenvolvimento que permita uma avaliação expressiva das ferramentas e técnicas empregadas e analisadas. Por outro lado, o resultado da experimentação deve ser reportado integralmente para que eventuais interessados possam exercer toda sua capacidade crítica, em amplitude e em detalhe.

Como subproduto desse experimento, o próprio modelo obtido, descrito neste trabalho, serve como base para a utilização do método pois constitui uma exemplificação bastante completa de como empregá-lo. A avaliação dessa exepriência baseada no COMONLIFE encontra-se descrita a seguir.

A técnica de design proposta em [Sanchez95.1] e empregada neste trabalho, permitiu uma transição suave do Modelo da Essência do COMONLIFE, apresentado em [Sanchez95.2], para um design do sistema orientado para encapsulamento de dados e operações em subsistemas autônomos baseados em classes. A implementação dessas classes através de subsistemas básicos não difere substancialmente de uma implementação orientada para objetos. No entanto, a utilização

- do conceito de portas, inserido em um contexto de troca de mensagens, e
- de um procedimento orientado para configuração independente de subsistemas (ao invés da ligação entre subsistemas através de invocações de serviços inseridas diretamente no código)

permite maior independência e implica maior capacidade de reúso, sem qualquer necessidade de introduzir alterações nos subsistemas reusáveis [Werner93].

As ferramentas de design aqui empregadas permitem focalizar o sistema de dois modos distintos e complementares. O uso de Diagramas de Estrutura (DE), gerando uma representação do relacionamento estático entre os subsistemas, permitiu uma visão funcional macro do sistema e dos serviços que ele presta. A modelagem da dinâmica do sistema através de Diagramas de Comportamento (DC) foi capaz de representar precisamente, em cada nível da hierarquia de decomposição funcional, o paralelismo (a concorrência) existente entre serviços alocados a um mesmo subsistema ou a subsistemas distintos. Nos DC's, a forma de explicitar a relação entre serviços de subsistemas facilita a avaliação, conseqüentemente a especificação, dos aspectos temporais associados a esses subsistemas visando atender aos requisitos de desempenho impostos ao sistema. Dessa forma, garante-se a possibilidade de avaliar a correta transposição das relações temporais existente entre Atividades Essenciais.

A técnica de associar a esses diagramas (DE's e DC's) as justificativas das decisões de design permitiu um controle conceitual que garante um processo de segmentação em subsistemas obedecendo a critérios explícitos de encapsulamento de informação e operações. Esses critérios, oriundos de conceitos sistêmicos básicos que prescrevem maximizar a coesão funcional de subsistemas e minimizar o acoplamento entre subsistemas funcionalmente coesos, visaram:

- à maior autonomia possível de cada subsistema;
- à melhor forma possível de sincronismo das ações do sistema através do uso de conexões, entre subsistemas, implementando trocas de mensagens de tipo bem definido;
- ao maior potencial de reúso de cada subsistema.

A técnica de design aqui empregada favoreceu a concepção de subsistemas básicos de elevado nível de abstração, refletindo basicamente a implementação dos requisitos essenciais especificados na modelagem conceitual. Claramente, esse tipo de procedimento, que garante uma transição suave entre a modelagem da essência e a modelagem da implementação, proporciona alta modularidade e manutenibilidade aos subsistemas básicos. Mais importante ainda, trata-se de procedimento que amplia significativamente o potencial de reúso: através de maior disciplina imposta pela gerência do processo de desenvolvimento, é possível obter-se, não somente reúso de código, também o reúso dos produtos gerados nas fases de design e de modelagem conceitual. Com proveito análogo, essa técnica de design permite isolar as dependências em relação à arquitetura de hardware, à arquitetura de software básico e aos requisitos não-essenciais de tolerância a falhas e desempenho. Possíveis impactos relacionados a essas dependências ficam restritos à configuração do sistema e à Interface de Compatibilização. Isso facilitará significativamente a tomada de decisão relativa ao reúso de subsistemas concebidos no processo de desenvolvimento de outros sistemas: desde que o mesmo método tenha sido empregado, o potencial de reúso pode ser avaliado diretamente através da análise comparativa das Listas de Eventos Externos do sistema antigo e do novo sistema, sem, obrigatoriamente, exigir-se a revisão de código fonte.

O emprego das ferramentas conceituais e técnicas de modelagem propostas em [Sanchez95.1] permitiu a geração de um design totalmente independente da arquitetura de hardware onde o código será executado. Na realidade, em nenhum momento essa arquitetura foi mencionada. Isso indica que qualquer mudança na arquitetura de hardware implicará apenas a revisão do software da unidade de implementação responsável pela configuração, o que garante portatilidade para a camada de aplicação.

A documentação associada ao Modelo da Implementação do sistema foi gerada durante o processo de desenvolvimento. Assim sendo, todas as características de segmentação associadas à técnica de modelagem estão refletidas na documentação do design do sistema. Sua divisão em dois níveis de abstração (um, associado ao Projeto Básico, contendo uma visão global do sistema e outro, correspondendo à Especificação Detalhada para cada Subsistema Básico, apresentando uma visão específica de cada componente) tornou direto o acesso a pontos específicos do modelo. A Especificação Detalhada dos Subsistemas Básicos, tratando-se de documento independente gerado em uma segunda etapa do processo de design, favoreceu o adiamento da especificação de algumas características da implementação do sistema (omissão de detalhes associados a escolha de equipamentos).

O isolamento da IUM, em subsistema (independente da aplicação) de elevado nível de abstração, permitiu que o processo design fosse dirigido inicialmente apenas para a aplicação (cujos SB's foram derivados do Modelo da Essência). Posteriormente, foi gerado o design da IUM. Dispondo-se de uma equipe suficientemente numerosa, é possível minimizar esse seqüenciamento e introduzir um paralelismo mais produtivo nos tratamentos independentes da aplicação e de sua interface. Beneficia-se dessa forma o processo de desenvolvimento pois, no início de um projeto, raramente são conhecidos todos os detalhes referentes à interface, principalmente quanto à interligação com sensores. Nessas condições, o design da IUM pode ser tratado em paralelo com o design da aplicação, enquanto aguarda-se o conhecimento completo de todos os detalhes referentes à interface com dispositivos sensores/atuadores.

Sumarizando a percepção atual dos autores, é possível concluir que a experimentação controlada relativa ao desenvolvimento do COMONLIFE, baseada no emprego de ferramentas conceituais e técnicas de modelagem propostas em [Sanchez95.1], deu origem a um design com elevado nível de consistência em relação às métricas de qualidade (manutenibilidade, portatilidade, modularidade e reúso) consideradas relevantes para o desenvolvimento de sistemas de tempo-real para controle de processos. Em particular, considerando as regras de derivação atualmente existentes e partindo de um Modelo da Essência totalmente concebido, o que a rigor não seria necessário, percebe-se também que o custo de concepção do Projeto Básico é bastante reduzido. Isso resulta sem prejuízo do poder de comunicação deste instrumento de modelagem, qualidade que garante uma boa comunicação da planta da implementação, fator determinante para a manutenibilidade do sistema.

Bibliografia

[Booch94] Booch G.; Object-Oriented Analysis and Design with Apllications; The Benjamin/Cummings Publishing Company, INC. , Second Edition; 1994

[CCCL93] Carneiro L.M.F., Coffin M.H., Cowan D.D., Lucena C.J.P.; User Interface High-Order Architectural Models; Technical Report 93-14, Computer Science Departament, University of Waterloo, Waterloo, Ontario, Canada; 1993.

[CILS93a] Cowan D.D., Ierusalimschy R., Lucena C.J.P., Stepien T.M.; Abstract Data Views; Structured Programming, 14(1):1-13; janeiro de 1993.

[CILS93b] Cowan D.D., Ierusalimschy R., Lucena C.J.P., Stepien T.M.; Abstract Data Views; Constructing Composite Applications from Interactive Components; Software Practice and Experience, 23(3):255-276; março de 1993.

[Cowan95] Cowan D.D., Lucena C.J.P.;Abstract Data Views: An Interafce Specification Concept to Enhance Design for Reuse; IEEE Transactions on Software Engineering, vol21, nº3, 1995.

[Glass94] Glass R. L.; The Software-Research Crisis; IEEE Software; nov. 1994.

[Maffeo92] Maffeo B.; Engenharia de Software e Especificação de Sistemas; Editora CAMPUS, 1992.

[Mcmenamin84] McMenamin M. e Palmer J. F.; Essential Systems Analysis; Yourdon Press; 1984.

[Poulin94] Poulin J.S.; Measuring Software Reusability; Proceedings of Third International Conference on Software Reuse - Advances in Software Reusability; novembro de 1994.

[Rumbaugh91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W.; Object-Oriented Modeling and Design; Prentice-Hall; 1991.

[Sanchez95.1] Sanchez M. L., Maffeo B; Sistema de Tempo-Real para Controle de Processos - Design Orientado a Encapsulamento de Dados e a Troca de Mensagens entre Subsistemas Autônomos; Série Monografias em Ciência da Computação 13/95, Departamento de Informática, PUC-Rio; 1995.

[Sanchez95.2] Sanchez M. L., Maffeo B; Modelagem da Essência de um Sistema de Litografia por Feixe de Elétrons - COMONLIFE; Série Monografias em Ciência da Computação xx/95, Departamento de Informática, PUC-Rio; 1995.

[Sanchez94] Sanchez M. L., Maffeo B., Leite J. C. S. P. ; Ferramentas e Técnicas para a Modelagem da Essência de Sistemas de Tempo-Real para Controle e Monitoramento de Processos; Anais do 10º Congresso Brasileiro de Automática e do 6º Congresso Latino Americano de Controle Automático, Rio de Janeiro; setembro de 1994.

[Sanchez93] Sanchez M. L., Maffeo B. ; Ferramentas e Técnicas para a Modelagem da Essência de Sistemas de Tempo-Real para Controle e Monitoramento de Processos; Série Monografias em Ciência da Computação 14/93, Departamento de Informática, PUC-Rio; 1993.

[Veldwiljk94] Veldwijk R.J. e Boogaard M.; Assessing the Software Crisis: Why Information Systems are Beyond Control; Information Sciences 81; 1994.

[Ward85] Ward P.T. e Mellor S.J.; Structured Development for Real-Time Systems; Yourdon Press; 1985.

[Werner93] Werner J.A.V. e Loques Fº O.G.;Ambiente RIO: Metodologia e Suporte para Sistemas Configuráveis; XX SEMISH - Seminário Integrado de Software e Hardware, Florianópolis; setembro 1993.

Apêndices

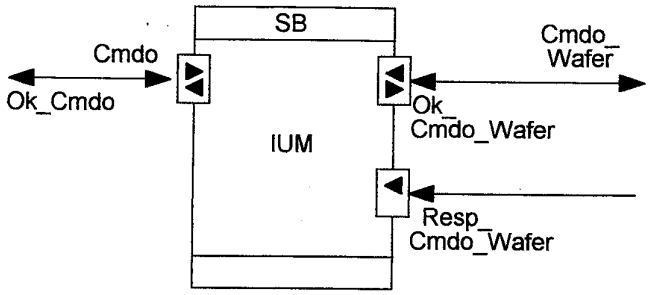
A. Subsistema Básico - Interface Usuário Máquina

A.1 Especificação

A.1.1 Descrição

Estabelece interação com operador, através de um console (terminal alfanumérico, teclado, mouse e impressora) permitindo entrada de comandos e apresentação de resultados do sistema COMONLIFE.

A.1.2 Diagrama de Contexto



A.1.3 Descrição dos Serviços

A.1.3.1 Serviços Externos

- Gerenciar entrada de comando
Entradas: Cmdo /* pedido de serviço */
Saídas: Ok_Cmdo

Serviço responsável por identificar um comando introduzido no sistema pelo operador. Esse serviço é requisitado por Cmdo que representa a interface entre o sistema e o operador.

A IUM utiliza um sistema de janelas e menus. A introdução de um comando é efetuada através de menus e janelas para selecionar valores de parâmetros. O susbsistema é responsável por tratar toda a dinâmica de interação com o operador, incluindo executar comandos caracteríticos da IUM (como cancelamento do comando sendo selecionado, fim de execução, etc.) e analisar sintaticamente comandos relativos ao sistema COMONLIFE. Caso exista erro de sintaxe, o subsistema envia mensagem de erro em Ok_Cmdo e libera teclado/mouse para correção ou introdução de outro comando. Caso contrário, requisita a execução do comando através da mensagem Cmdo_Wafer e espera o recebimento de Ok_Cmdo_Wafer, contendo:

- o resultado da analise semântica;
- o novo estado do sistema caso o comando tenha sido aceito.

Após receber Ok_Cmdo_Wafer é determinado os comandos diponíveis ao operador, em função do estado do sistema, e enviado Ok_Cmdo contendo informação de aceitação ou erro e liberação à introdução de novos comandos.

```
{* variáveis locais *}
int estado_execução, estado_ant_execução
{  =0  equivale  a  menu  principal;  -1  a  menu  de  finalização  de  entrada  de
parâmetros e > 0 aos outros menus da aplicação }
boolean parm
{ verdadeiro - entrando parâmetros, falso - selecionando comando }

início
estado_execução = 0 { principal }
parm = falso
Apresenta Menu equivalente a estado_ execucao
enquanto Verdadeiro faça
início
            Espera Cmdo
            Analisa tipo de CMDO
            Caso tipo
            IUM :    { comando executado pela IUM }

            { executa comando de IUM }
            {apresenta menu, trata digitação de parâmetros, help, fim de
             execução, cancelamento do comando ou confirmação do comando
             OBS: A IUM só apresenta comandos possíveis de serem  escolhidos  em
                  função do estado_execução }

                  Caso código do cmdo
                        APRESENTA MENU :
                              Apresenta menu correspondente a estado_execução
                        TERMINA PROGRAMA:
                              Finaliza execução
                        DIGITA_PARÂMETRO:
                              Trata parâmetro digitado
                        CANCELA_COMANDO:
                              se parm
                                    fecha janela de parâmetros
                                    parm = falso
                              Envia Ok_Cmdo
                              estado_execução = estado_ant_execução
                        CONFIRMA_COMANDO:
                              se parm
                                    analisa parâmetros
                              se (parâmetros completos }
                              então
                                    parm = falso
                                    fecha tela de parâmetros
                                    Envia Cmdo_Wafer
                                    Espera Ok_Cmdo_Wafer
                                    Envia Ok_Cmdo
                                    { envia mensagem de aceitação/ou não  do
                                    cmdo}
                                    estado_execução = estado_execução
                                    proveniente em Ok_Cmdo_Wafer
                              senão
                                    Envia Ok_Cmdo com mensagem de erro nos
                                    parâmetros
                                    { continua apresentando parâmetros }

                              fim
            WAFER : { comando do sistema }
                  se Cmdo é não terminal
                  então
                        parm = verdadeiro
                        Envia Ok_Cmdo equivalente a
                        apresentar menu correspondente a estado_execucao
                  senão
                  {trata comando terminal com parâmetros}
                  se possui parâmetros
                  então
                        estado_comando = -1 { estado de confirmação }
```

```
                        parm = verdadeiro
                        Apresenta janela com parâmetros
                  senão
                        Envia Cmdo_Wafer
                        Espera Ok_Cmdo_Wafer
                        Envia Ok_Cmdo
                        { envia mensagem de aceitação/ou não  do cmdo}
                        estado_execução = estado_execução proveniente em
                        Ok_Cmdo_Wafer
            fim

            fim

      fim
```

- Tratar Resposta a Comando Wafer
Entradas: Resp_Cmdo_Wafer
Saídas: Resp_Cmdo
Descrição:

Atualiza o estado de execução do sistema e formata a informação contida em uma resposta de comando, apresentando-a no terminal de vídeo ou impressora de acordo com as características da resposta recebida.

```
{variáveis locais}
int estado_execução

início
    enquanto verdadeiro faça
    início
          Espera Resp_Cmdo_Wafer
          se estado_execução diferente de estado_execução em Resp_Cmdo_Wafer
          então
                  atualiza estado_execucao

          formata informação de Resp_Cmdo_Wafer e identifica dispositivo a
          apresentar.
          prepara cadeia de comandos a enviar para apresentação em dispositivo
          { para vídeo ⇒ janela de apresentação, cadeia de comandos com texto a
          apresentar }
          { para impressora ⇒ cadeia de comandos com texto a imprimir }
          envia Resp_Cmdo
    fim
fim
```

A.1.3.2 Serviços Internos

Nenhum.

A.1.3.3 Serviços Requisitados

- Executar o comando sintaticamente correto digitado pelo operador
Saída: Cmdo_Wafer
Entrada: Ok_Cmdo_Wafer

Serviço requisitado para executar comando sintaticamente correto introduzido pelo operador. Retorna em Ok_Cmdo_Wafer a informação de comando semanticamente correto mas em função da característica do serviço requisitado pelo comando, esse retorno indica apenas início ou o término da execução do serviço.

A.1.4 Descrição Detalhada dos Dados

A.1.4.1 Definições de Tipos

```
type tipo_parâmetros = record
      begin
            tensão_fonte                  : real
            corrente_objetiva             : real
            corrente_condensadora         : real
            corrente_emissão_filamento    : real
            dose                          : real
            tipo                          : {resolução, diâmetro}
            case tipo
                  begin
                        resolução         : real
                        diâmetro_feixe    : real
                  end
      end
{define os parâmetros de operação do microscópio eletrônico}
(*) diâmetro_feixe pode armazenar também a resolução do mesmo.

type tipo_posição_mesa = record
      begin
            pos_x : real
            pos_y : real
      end
{define uma posição da mesa}

type tipo_info_mic = record
      begin
            corrente_objetiva: real
            corrente_condensadora: real
            tensão_fonte: real
            filamento: real
      end
{ define informação sobre o microscópio eletrônico }

type tipo_info_obturador = record
      begin
            corrente_absorção: real
      end
{ define informação sobre o microscópio eletrônico }

type tipo_iniciação_wafer = record
      begin
            id_wafer                : integer
            id_chip                 : integer
            num_réplicas            : integer
            tempo_registro          : real
            Delta_vértice           : tipo_posição
            distância_entre_réplica : tipo_posição

      end

type tipo_projeto_chip = record
      begin
            id_chip          : integer
            máscaras_a_fazer : integer
            id_marcas        : string
            operação_marca   : tipo_parâmetro
            dados_camada     : array[1..máscaras_a_fazer] of record
                  begin
                        id_máscara        : string
                        nível             : integer
```

```
                        parâmetro_operação: tipo_parâmetro
                    end
        end

type tipo_histórico = record
        begin
              dados_iniciação_wafer    : tipo_iniciação_wafer
              projeto_chip             : tipo_projeto_chip
              nome_arquivo             : string
              {contém nome de arquivo cujo registro é no formato de tipo_medidas,
              sequencial e com medidas por camada }

        end

type medidas = record
        begin
              camada             : inteiro
              info_mesa          : tipo_posição_mesa
              info_mic           : tipo_info_mic
              info_obturador     : tipo_info_obturador
              tempo_medição      : real
        end
{tipo de dados que contém uma medição efetuada no equipamento - formato do
registro do arquivo de histórico wafer}

type tipo_status_geração = record
        begin
              tempo_previsto     : real
              tempo_gasto        : real
        end
```

A.1.4.2 Estruturas de Dados Encapsuladas

Nenhuma.

A.1.4.3 Estrutura das Mensagens

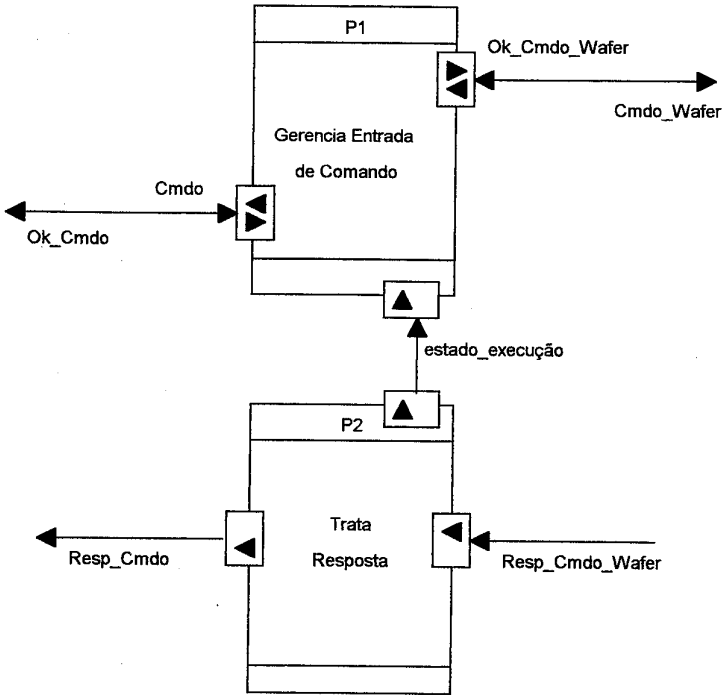
```
var Cmdo_Wafer : record
    begin
      código : {DEF_CHIP /INICIA / ALINHA / CONFIRMA / AJUSTE / INTERROMPE /
      HISTÓRICO}
      case código
      DEF_CHIP
            projeto_chip : tipo_projeto_chip

      INICIA :
            dados_iniciação_wafer : tipo_iniciação_wafer
      ALINHA :
            id_wafer : integer
      AJUSTE :
            deslocamento : tipo_posição_mesa
      HISTÓRICO :
            id_wafer     : integer
    end
{modela um comando sobre um wafer}

var Ok_Cmdo_Wafer: record
      begin
            estado_execução    : inteiro
            aceito             : boolean
            motivo             : string
      end

var Resp_Cmdo_Wafer: record
    begin
            estado_execução : inteiro
            tipo_resp: {HISTÓRICO / STATUS / INTERROMPE}
```

```
            caso tipo_resp seja
            HISTÓRICO:
                  histórico    : tipo_histórico
            STATUS:
                  status       : tipo_status_geração
            INTERROMPE:
                  OK           : boolean
                  mensagem     : string
                  {* mensagem com motivo da interrupção *}
    end
{modela a resposta a um comando sobre um wafer}
```

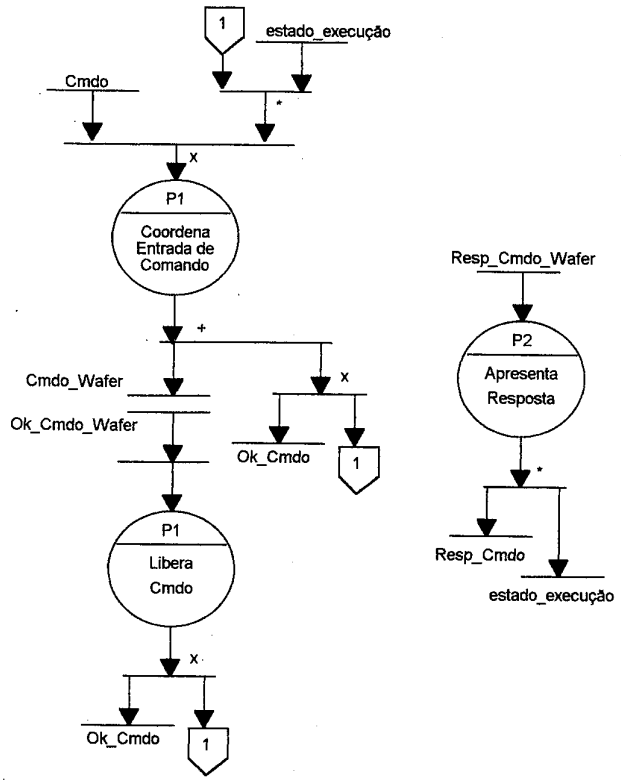
Obs: Cmdo e OK_Cmdo correspondeme a portas de comunicação com monitor e teclado. As mensagens que fluem nessass portas bem como a necessidade de sua definição dependem de utilitários para o desenvolvimento de interface usuário-máquina do ambiente de desenvolvimento.

A.2 Design do Subsistema Básico

A.2.1 Diagrama de Estrutura



A.2.2 Diagrama de Comportamento



A.2.3 Descrição dos Processos

- P1 - Gerencia Entrada de Comando
Entradas: Cmdo /* pedido de serviço */
Saídas: Ok_Cmdo

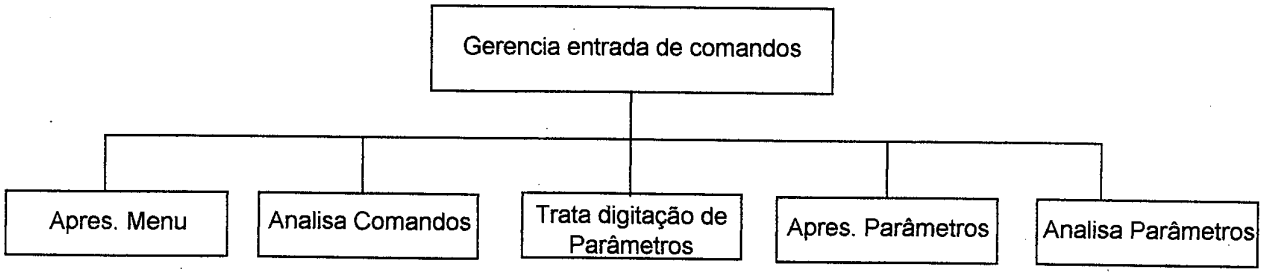
Serviço responsável por identificar um comando introduzido no sistema pelo operador e para comandos com parâmetros, requisitar e controlar a seleção de parâmetros.

OBS: A sintaxe de comandos do COMONLIFE, comandos disponíveis em função do estado do sistema, parâmetros associados a comandos devem estar definidos em tabelas para facilitar o reúso caixa-branca desse subsistema para outros sistemas.

```
{* variáveis locais *}
int estado_execução, estado_ant_execução
{ =0 equivale a menu principal; -1 a menu de finalização de entrada de
parâmetros e > 0 aos outros menus da aplicação }
boolean parm
{ verdadeiro - entrando parâmetros, falso - selecionando comando }

início
estado_execução = 0 { principal }
parm = falso
Apresenta Menu equivalente a estado_ execucao
enquanto Verdadeiro faça
início
            Espera Cmdo
            Analisa tipo de CMDO
            Caso tipo
            IUM :    { comando executado pela IUM }
```

```
{ executa comando de IUM }
{apresenta menu, trata digitação de parâmetros, help, fim de
 execução, cancelamento do comando ou confirmação do comando
 OBS: A IUM só apresenta comandos possíveis de serem  escolhidos  em
      função do estado_execução }

      Caso código do cmdo
            APRESENTA MENU :
                  Consulta porta com estado_execução
                  Apresenta menu correspondente a estado_execução
            TERMINA PROGRAMA:
                  Finaliza execução
            DIGITA_PARÂMETRO:
                  Trata parâmetro digitado
            CANCELA_COMANDO:
                  se parm
                        fecha janela de parâmetros
                        parm = falso
                  Envia Ok_Cmdo
                  estado_execução = estado_ant_execução
            CONFIRMA_COMANDO:
                  se parm
                        analisa parâmetros
                  se (parâmetros completos }
                  então
                        parm = falso
                        fecha tela de parâmetros
                        Envia Cmdo_Wafer
                        Espera Ok_Cmdo_Wafer
                        Envia Ok_Cmdo
                        { envia mensagem de aceitação/ou não  do
                        cmdo}
                        estado_execução = estado_execução
                        proveniente em Ok_Cmdo_Wafer
                  senão
                        Envia Ok_Cmdo com mensagem de erro nos
                        parâmetros
                        { continua apresentando parâmetros }

                  fim
WAFER : { comando do sistema }
      se Cmdo é não terminal
      então
            parm = verdadeiro
      Envia Ok_Cmdo equivalente a
            apresentar menu correspondente a estado_execucao
      senão
      {trata comando terminal com parâmetros}
      se possui parâmetros
      então
            estado_comando = -1 { estado de confirmação }
            parm = verdadeiro
            Apresenta janela com parâmetros
      senão
            Envia Cmdo_Wafer
            Espera Ok_Cmdo_Wafer
            Envia Ok_Cmdo
            { envia mensagem de aceitação/ou não  do cmdo}
            estado_execução = estado_execução proveniente em
            Ok_Cmdo_Wafer
fim

fim

fim
```

Diagrama Hierárquico de Módulos



Objetivos dos Módulos

- **Apres. Menu** - Preparar apresentação em vídeo do menu correspondente a estado_comando.
- **Analisa Comandos** - Identificar se o comando selecionado é da IUM ou da aplicação.
- **Trata digitação de Parâmetros** - Identificar parâmetro digitado, atualiza seu valor e maca-o como digitado.
- **Apres. Parâmetros** - Preparar apresentação em vídeo de janela correspondente a entrada de parâmetros para o comando selecionado. Inclui a apresentação dos valores pré-definidos para parâmetros que os possuem.
- **Analisa Parâmetros**- verificar se todos os parâmetros obrigatórios foram digitados.

- P2 - Trata Resposta
Entradas: Resp_Cmdo_Wafer
Saídas: Resp_Cmdo
Descrição:

Atualiza o estado de execução do sistema e formata a informação contida em uma resposta de comando, apresentando-a no terminal de vídeo ou impressora de acordo com as características da resposta recebida.

OBS: a relação Resp_Cmdo_Wafer, dispositivo e forma de apresentação deve estar em tabelas de forma a facilitar o reúso caixa-branca desse subsistema em outros sistemas.

```
{variáveis locais}
int estado_execução

início
    enquanto verdadeiro faça
    início
          Espera Resp_Cmdo_Wafer
          se estado_execução diferente de estado_execução em Resp_Cmdo_Wafer
          então
                  atualiza estado_execucao
                  Envia Estado_execução
          formata informação de Resp_Cmdo_Wafer e identifica dispositivo a
          apresentar.
          prepara cadeia de comandos a enviar para apresentação em dispositivo
```

```
            { para vídeo ⇒ janela de apresentação, cadeia de comandos com texto a
            apresentar }
            { para impressora ⇒ cadeia de comandos com texto a imprimir }
            Envia Resp_Cmdo
    fim
fim
```

A.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S1.1

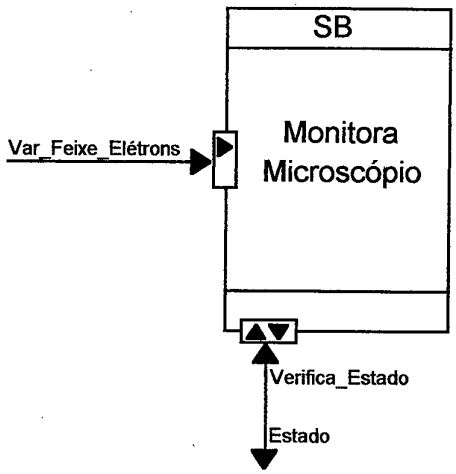
B. Subsistema Básico - Monitora Microscópio

B.1 Especificação

B.1.1 Descrição

Acompanhar o estado do microscópio eletrônico, obtido através da leitura de dados da corrente do filamento, fonte de alta tensão, lente objetiva e lente condensadora e sua comparação com os valores desejados para uma dada camada sendo gerada.

B.1.2 Diagrama de Contexto



B.1.3 Descrição dos Serviços

B.1.3.1 Serviços Externos

- Monitorar Dados do Microscópio
Entradas: Verifica_Estado
Saídas: Estado

Este serviço indica se o microscópio eletrônico, para um padrão de operação requisitado no início do monitoramento, está operando com suas variáveis dentro da faixa desejada. Para isso, antes de iniciar o monitoramento do microscópio, deve receber os parâmetros relativos que determinam o estado do microscópio. A cada pedido de verificação de estado, analisa e informa o estado em função das variáveis medidas.

```
{* variáveis locais: *}
var estado: tipo_estado
var parâmetros_camada: tipo_info_mic
{representa o valor esperado para as variáveis feixe elétrons}
var camada = {EM_GERAÇÃO, NÃO_GERAÇÃO}

início
    enquanto Verdadeiro faça
    início
          Espera Verifica_Estado
          caso Verifica_Estado.cmdo seja
             INICIA_CAMADA:
```

```
                    camada = EM_GERAÇÃO
                    {armazena parâmetros da camada}
                    parâmetros_camada = Verifica_Estado.parâmetros
                    estado = determina_estado_mic (parâmetros_camada)
                    Envia estado
              TERMINA_CAMADA:
                    camada = NÃO_GERAÇÃO
                    estado.ok = OK
                    Envia estado
              INFORMA_ESTADO:
                    Envia estado
        fim
fim

função determina_estado_mic (parâmetros) : tipo_estado
início
      estado.ok = OK
      se módulo(parâmetros.lente_condensadora -
      LENTE_CONDENSADORA_ESPERADA) > FAIXA_LENTE_CONDENSADORA
      então
            determina_estado_mic.ok = NOK
      se módulo(parâmetros.lente_objetiva - LENTE_OBJETIVA_ESPERADA) >
      FAIXA_LENTE_OBJETIVA
      então
            determina_estado_mic.ok = NOK
      senão se módulo(parâmetros.tensão_fonte - TENSÃO_FONTE_ESPERADA) >
      FAIXA_TENSÃO_FONTE
      então
            determina_estado_mic.ok = NOK
      senão se módulo(parâmetros.filamento - FILAMENTO_ESPERADO) >
      FAIXA_FILAMENTO
      então
            determina_estado_mic.ok = NOK
      senão se Estado.ok = OK
      então
            estado.valores_mic = parâmetros
      retorna determina_estado_mic
fim
```

B.1.3.2 Serviços Internos

- Integrar Variáveis do Microscópio continuamente no tempo
Entradas : Var_Feixe_Elétrons
Saídas : nenhuma

Periodicamente, o sistema lê o valor de operação da lente condensadora, da lente objetiva, da fonte de tensão e do filamento do microscópio eletrônico, integrando os valores lidos e avaliando o estado do microscópio.

```
início
      lê Var_Feixe_Elétrons
      integra Var_Feixe_Elétrons e armazena em Variáveis_Microscópio
      se estado = OK
      então
            estado = determina_estado_mic(parâmetros_camada)
fim

*Obs: integra = armazena dados e determina o comportamento da variável no tempo
```

B.1.3.3 Serviços Requisitados

Nenhum.

B.1.4 Descrição Detalhada dos Dados

B.1.4.1 Definições de Tipos

```
type tipo_info_mic: record
     begin
            lente_condensadora: real
            lente_objetiva: real
            tensão_fonte: real
            filamento: real
     end
{tipo de dados que contém as variáveis associadas ao microscópio}

type  tipo_estado: record
      begin
            ok: boolean {true = microscópio com estado ok}
            valores_mic: tipo_info_mic
      end
{tipo de dados que contém o estado do microscópio}
```

B.1.4.2 Estruturas de Dados Encapsuladas

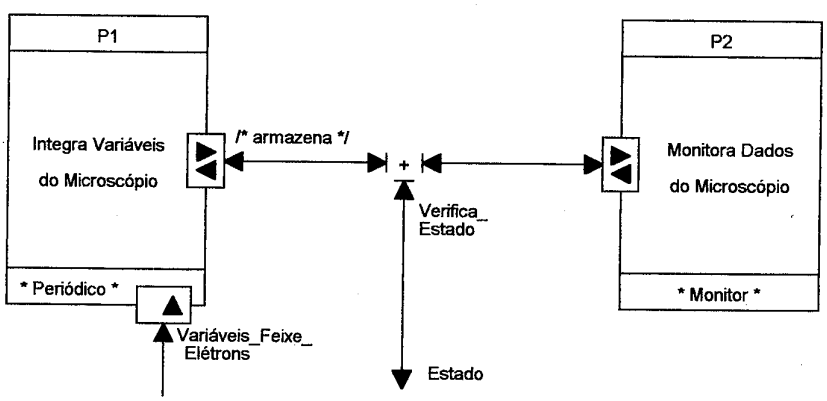
```
var Variáveis_Microscópio: tipo_info_mic
{representa o valor atual das variáveis feixe elétrons}
```

B.1.4.3 Estrutura das Mensagens

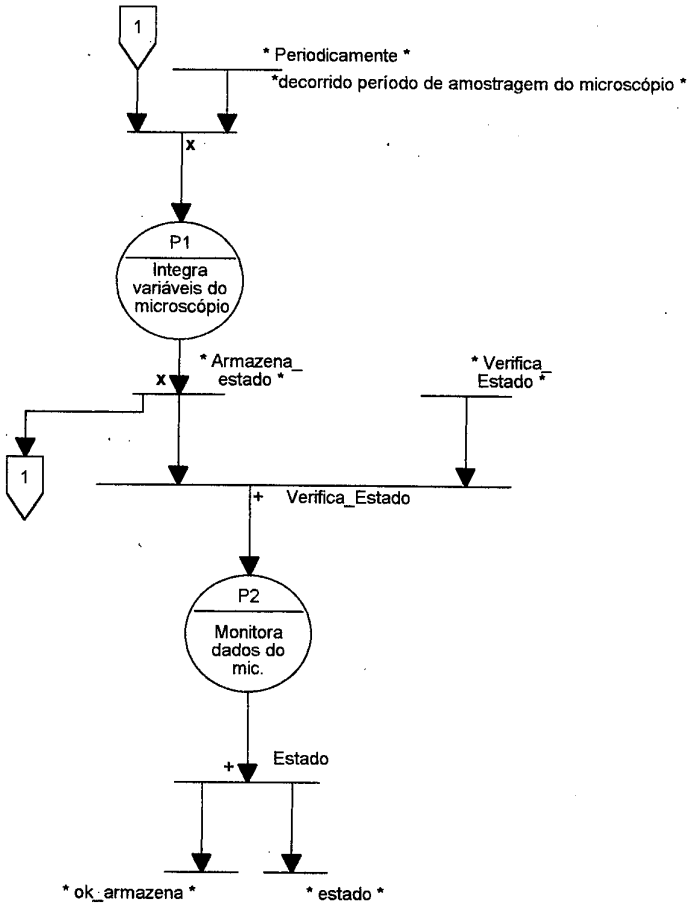
```
var Var_Feixe_Elétrons: tipo_info_mic

var Verifica_Estado: record
      begin
            cmdo:   {INICIA_CAMADA   /   TERMINA_CAMADA   /   INFORMA_ESTADO   /
            ARMAZENA_ESTADO}
            caso tipo seja
            ARMAZENA_ESTADO:
            INICIA_CAMADA:
                  parâmetros_camada: tipo_info_mic
      end
{modela a resposta do subsistema a um pedido de verificação do microscópio
eletrônico}

var Estado: tipo_estado
```

B.2 Design do Subsistema Básico

B.2.1 Diagrama de Estrutura



B.2.2 Diagrama de Comportamento



B.2.3 Descrição dos Processos

- P1 - Integra Variáveis do Microscópio
Entradas : Var_Feixe_Elétrons
Saídas : nenhuma
Serviços Requisitados : armazena_info_mic, com resposta associada, ok_armazena
Objetivo :

Ler da interface com o microscópio as variáveis feixe elétrons (valor da corrente da lente condensadora, lente objetiva, tensão da fonte e corrente do filamento); integrar esses valores segundo algoritmo que considere a periodicidade dos valores lidos.

```
início
      lê Var_Feixe_Elétrons
      integra Var_Feixe_Elétrons
      pede serviço Verifica_estado com AMAZENA_ESTADO e Var_Feixe_Elétrons
      Espera ok_armazena
fim
```

- P2 - Monitora dados do microscópio
Entradas: cmdo_info_mic
Saídas: ok_cmdo_mic
Serviços Requisitados: nenhum
Objetivo:

Monitorar o acesso às variáveis feixe elétrons, determinar e informar quando requisitado o estado do microscópio eletrônico.

```
{* variáveis locais: *}
var estado: tipo_estado
var parâmetros_camada: tipo_variáveis_microscópio
       {representa o valor Esperado para as variáveis feixe elétrons}

início
    enquanto Verdadeiro faça
    início
          Espera Verifica_Estado
          caso Verifica_Estado.tipo seja
                INICIA_CAMADA:
                      camada = EM_GERAÇÃO
                      {armazena parâmetros da camada}
                      parâmetros_camada = Verifica_Estado.parâmetros
                      Estado = determina_estado_mic (parâmetros_camada)
                      Envia Estado
                TERMINA_CAMADA:
                      camada = NÃO_GERAÇÃO
                      Estado = OK
                      Envia Estado
                INFORMA_ESTADO:
                      Envia Estado
                ARMAZENA_ESTADO:
                      se Estado = OK e camada = EM_GERAÇÃO
                      então
                      início
                            armazena valores_mic
                            Estado = determina_estado_mic (parâmetros_camada)
                      fim
                      se camada = NÃO_GERAÇÃO
                      então
                      início
                            armazena valores_mic
                      fim
                      Envia ok_armazena
    fim
fim
```

OBS:

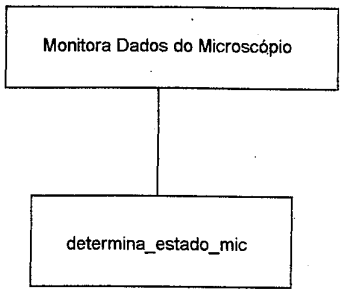
1. O procedimento:

   - determina_estado_mic(parâmetros)

   é igual ao definido na especificação do serviço

2. *A necessidade de Verifica_Estado.tipo = ARMAZENA_ESTADO surgiu no design do SB. A alteração do tipo de dado da porta (Verifica_estado e Estado) ficou refletida na Estrutura das Mensagens (ítem 3) já que altera a interface do SB.*

Diagrama Hierárquico de Módulos



Objetivos dos Módulos

- **determina_estado_mic:** verificar se variáveis do feixe de elétrons estão com valores dentro das faixas estabelecidas nos parâmetros de operação.

B.3 Sistemas onde é empregado

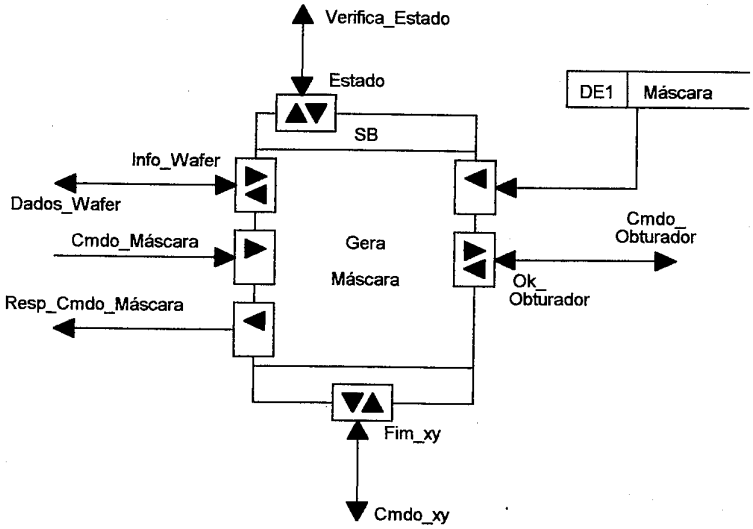
1. COMONLIFE - Subsistema Básico S2.1

C. Subsistema Básico: Gera Máscara

C.1 Descrição

Coordena a geração de uma camada, analisando e decompondo a máscara a ser traçada, composta de linhas e sólidos, em componentes mais simples (linhas de traçado). Cada linha de traçado é gravada no wafer com a movimentação da mesa nas direções x e y e a abertura/fechamento do obturador. Também armazena dados do histórico de geração de cada camada. O processo de geração de camada é interrompido em caso de defeito no equipamento ou a pedido do operador.

C.1.1 Diagrama de Contexto



C.1.2 Descrição dos Serviços

C.1.2.1 Serviços Externos

- Atender a Pedido de Dados Sobre Wafer
Entradas: Info_Wafer
Saídas: Dados_Wafer

Retorna nome de arquivo em disco com dados armazenados sobre a gravação de um Wafer específico para camadas já geradas.

```
início
      prepara histórico de wafer em arquivo
      Envia Dados_Wafer com nome de arquivo contendo  histórico_wafer de
      Info_Wafer.Id_Wafer
fim
```

- Gerar Máscara
Entradas: Cmdo_Máscara
Saídas: Resp_Cmdo_Máscara

Responsável pela geração de uma camada sobre o wafer. Interpreta o arquivo C.A.D. correspondente à máscara que está sendo tratada, calculando as linhas de traçado correspondentes. Coordena a movimentação da mesa nas direções x e y e a abertura/fechamento do obturador de

feixe, requisitando serviços, necessários ao traçado da máscara. Interrompe a geração de camada em caso de erro nos serviços requisitados ou a pedido do operador.

```
{* variáveis locais: *}


var linha : record
      begin
            pos_inicial : tipo_posição_mesa;
            pos_final : tipo_posição_mesa;
      end;
{ armazena uma linha que forma a máscara a ser traçada, com as posições iniciais
e finais de traçado }

var situação := {ok, NOK}
{ armazena situação de funcionamento dos diversos equuipamentos controlados para
geração de camada }

var obturador: {ABERTO / FECHADO};
{ armazena o estado do obturador }

var linhas_traçado = record
      begin
            num_linhas : integer;
            { armazena a quantidade de linhas a serem traçadas no wafer para o
              traçado da máscara pedida }
            linhas_traçado: array of tipo_linha
      end

var num_réplicas : integer;
{ armazena a quantidade de réplicas a serem traçadas no wafer }

início
      caso Cmdo_Máscara.código

            INICIA:
                  cria arquivo para armazenamento de histórico
                  gera_camada(Cmdo_Mascara.num_réplicas, Cmdo_Máscara.id_camada,
                  nível, Cmdo_Máscara.Delta_vértice,
                  Cmdo_Máscara.distância_entre_réplicas)
                  termina armazenamento de dados de histórico wafer


            GERA:
                  abre arquivo de armazenamento de histórico para id_wafer
                  gera_camada(Cmdo_Mascara.num_réplicas, Cmdo_Máscara.id_camada,
                  nível, Cmdo_Máscara.Delta_vértice,
                  Cmdo_Máscara.distância_entre_réplicas)
                  termina armazenamento de dados de histórico wafer

            INTERROMPE:
                  { Não é feito nada - só quando está geranda máscara }
fim


gera_camada (num_réplicas, Máscara_Corrente, nível, Delta_vértice,
distância_entre_réplicas)
início
{ verifica primeiro estado do micrscópio }
pede serviço Verifica_Estado com INICIA_CAMADA e parâmetros_operação
Espera Estado
se Estado.ok = NOK
então
      Envia Resp_Cmdo_Máscara com "Camada não Iniciada por Instabilidade"
      situação = NOK
      retorna
fim
```

```
senão
      {inicia geração de camada}
      Envia Resp_Cmdo_Máscara com "Início de Geração de Camada"
      situação = OK
      pede serviço Cmdo_xy com REPOUSO
      Espera Fim_xy
      se Fim_xy = NOK
      então
            situação = NOK
      senão
         posição_atual = POSIÇÃO_REPOUSO
         Faça enquanto (situação = OK e Existe elemento máscara a processar e
                        geração camada não interrompida)
            analisa Máscara_Corrente[nível].AD_elemento_máscara para elemento
            máscara a ser traçado gerando linhas de traçado e num_linhas
            le próxima linha de traçado
            atualiza posição inicial e final para traçar dentro da 1ª réplica
            faz de 1 até num_réplicas, para cada linha de traçado e para
            situação = OK
                  pede serviço Cmdo_xy com MOVE, posição_inicial e
                  VELOCIDADE_MÁXIMA
                  Espera Fim_xy
                  se Fim_xy = NOK
                  então
                        situação = NOK
                  senão
                        pede serviço Cmdo_Obturador com ABRE
                        calcula deslocamento entre posição_atual e
                        linha.pos_final
                        calcula velocidade em função de parâmetros de operação
                        recebidos em Cmdo_Máscara
                        Espera Ok_Obturador
                        se Ok_Obturador = NOK
                        então
                              situação = NOK
                        senão
                              obturador = ABERTO
                              pede serviço Cmdo_xy com MOVE, posição_final e
                              velocidade
                              Espera Fim_xy
                              se Fim_xy = NOK
                              então
                                    situação = NOK
                              senão
                                    pede serviço Cmdo_Obturador com FECHA
                                    Espera Ok_Obturador
                                    se Ok_Obturador = NOK
                                    então
                                          situação = NOK
                                    senão
                                          obturador = FECHADO
                                    fim
                        fim
                  fim
                  atualiza posição inicial e final para traçar dentro da próxima
                  réplica
                   Consulta porta Cmdo_Máscara
                  se existe mensagem na porta Cmdo_Máscara
                  então
                  se Cmdo_Máscara.código = INTERROMPE
                  então
                        interrompe_processo
                        Envia Resp_Cmdo_Máscara com "Interrompida Geração de
                        Camada"
            fim {réplicas }

            Envia Resp_Cmdo_Mascara com Status de geração
```

```
        fim { num_linhas }

        se situação = NOK
        então
              interrompe_processo
              Envia Resp_Cmdo_Máscara com "Falha de Geração de Camada"
              retorna situação
        senão
              Envia Resp_Cmdo_Máscara com "Fim de Geração de Camada"
              retorna situação
        fim
 fim

interrompe_processo
início
      pede serviço Cmdo_xy com PARA
      se obturador = ABERTO
      então
            pede serviço Cmdo_Obturador com FECHA
      fim
      Envia Resp_Cmdo_Máscara com Insucesso
fim
```

C.1.2.2 Serviços Internos

- Armazenar Variáveis do Processo
Entrada : variáveis do processo
Saída : Nenhuma

Este serviço armazena, periodicamente, durante a geração de uma camada, as variáveis do processo no histórico wafer (posição x, posição y, lente objetiva, lente condensadora, tensão na fonte, filamento, corrente de absorção e tempo de medição). Estando o microscópio com valores fora da faixa desejada, o processo de geração de máscara é interrompido.

```
início
          pede serviço Verifica_Estado com INFORMA_ESTADO
          pede serviço Cmdo_xy com INFORMA
          pede serviço Cmdo_Obturador com INFORMA
          Espera Estado
          se Estado.Ok = NOK
          então
                {microscópio com valores fora da faixa desejada}
                interrompe_processo
          senão
                Espera Fim_xy
                Espera Ok_Obturador
                Armazena em histórico_wafer para id_wafer
                medições.num_camada = camada sendo gerada
                { obtida de Cmdo_Máscara }
                medições.info_mesa = Fim_xy.info_mesa
                medições.info_obturador = Ok_Obturador.info_obturador
                medições.info_mic = Estado.Variáveis_Microscópio
                medições.tempo_medida = tempo corrente

          fim
    fim
fim
```

C.1.2.3 Serviços Requisitados

- Movimentar Mesa xy
Saída: Cmdo_xy
Entrada: Fim_xy

Serviço requisitado para mover a mesa nas direções x e/ou y ou para informar a posição atual da mesa. O serviço retorna a condição de sucesso (OK) ou insucesso (NOK), no caso de pedido de movimentação, e retorna a posição da mesa, para o pedido de informação.

- Abrir/Fechar Obturador
Saída: Cmdo_Obturador
Entrada: Ok_Obturador

Serviço requisitado para abrir/fechar o obturador de feixe de elétrons ou para informar a corrente de absorção atual. O serviço retorna a condição de sucesso (OK) ou insucesso (NOK) no caso de pedido de abertura / fechamento, e o valor da corrente, no caso de pedido de informação.

- Verificar Estado do Microscópio
Saída : Verifica_Estado
Entrada : Estado

Serviço requisitado para checar o estado do microscópio eletrônico. O serviço retorna o estado do microscópio (OK) ou (NOK) e as variáveis associadas de lente objetiva, lente condensadora, fonte de alta tensão e corrente de filamento.

C.1.3 Descrição Detalhada dos Dados

C.1.3.1 Definições de Tipos

```
type tipo_estado_wafer = { EM_GERAÇÃO, SUCESSSO, INSUCESSO, INTERROMPIDO }

type tipo_posição_mesa = record
      begin
            pos_x : real
            pos_y : real
      end;
{ define uma posição da mesa }

type tipo_info_mic = record
      begin
            corrente_objetiva: real
            corrente_condensadora: real
            tensão_fonte: real
            filamento: real
      end;
{ define informação sobre o microscópio eletrônico }

type tipo_info_obturador = record
      begin
            corrente_absorção: real
      end
{ define informação sobre o obturador do feixe }

type medidas = record
      begin
            num_camada          : integer
            info_mesa           : tipo_info_mesa
            info_mic            : tipo_info_mic
            info_obturador      : tipo_info_obturador
            tempo_medição       : real
      end;
{ tipo de dados que contém uma medição efetuada no equipamento }

type tipo_parâmetros = record
      begin
            tensão_fonte                  : real
```

```
            corrente_objetiva              : real
            corrente_condensadora          : real
            corrente_emissão_filamento     : real
            dose                           : real
            tipo : {resolução, diâmetro}
            case tipo
            begin
                  resolução                : real
                  diâmetro_feixe           : real
            end
      end
{ define os parâmetros de operação do microscópio eletrônico }

type tipo_estado = record
      begin
            ok: boolean; {verdadeiro = microscópio com estado ok}
            valores_mic: tipo_info_mic
      end;
{ tipo de dados que contém o estado do microscópio }


type tipo_velocidade = record
      begin
            velocidade_x : real
            velocidade_y : real
      end
{ define a velocidade de deslocamento da mesa }

type tipo_status_gravação = record
      begin
            tempo_previsto : real
            tempo_gasto    : real
      end
{ define a velocidade de deslocamento da mesa }

type tipo_nome_máscara = record
  begin
      nome_máscara                 : string
      nível                        : integer
      parâmetros_máscara           : tipo_parâmetros
  end

type tipo_iniciação_wafer = record
  begin
      id_wafer                     : integer
      id_chip                      : integer
      numero_réplicas              : integer
      tempo_registro               : real
      Delta_vértice                : real
      distância_entre_réplicas     : real
  end

type tipo_parâmetros_camada = record
  begin
      máscara_corrente             : tipo_nome_máscara
      num_camada                   : integer
      posição_origem               : tipo_posição_mesa
      origem                       : tipo_posição_mesa
  end
```

C.1.3.2 Estruturas de Dados Encapsuladas

```
var histórico_wafer : array of medidas;
{ arquivo contendo registros do tipo medidas }
```

Estrutura Encapsulada Armazenada em Meio Magnético

As informações armazenadas pela estrutura histórico_wafer não podem ser perdidas entre execuções diferentes do sistema. Assim sendo, essa estrutura ficará armazenadas em arquivo.

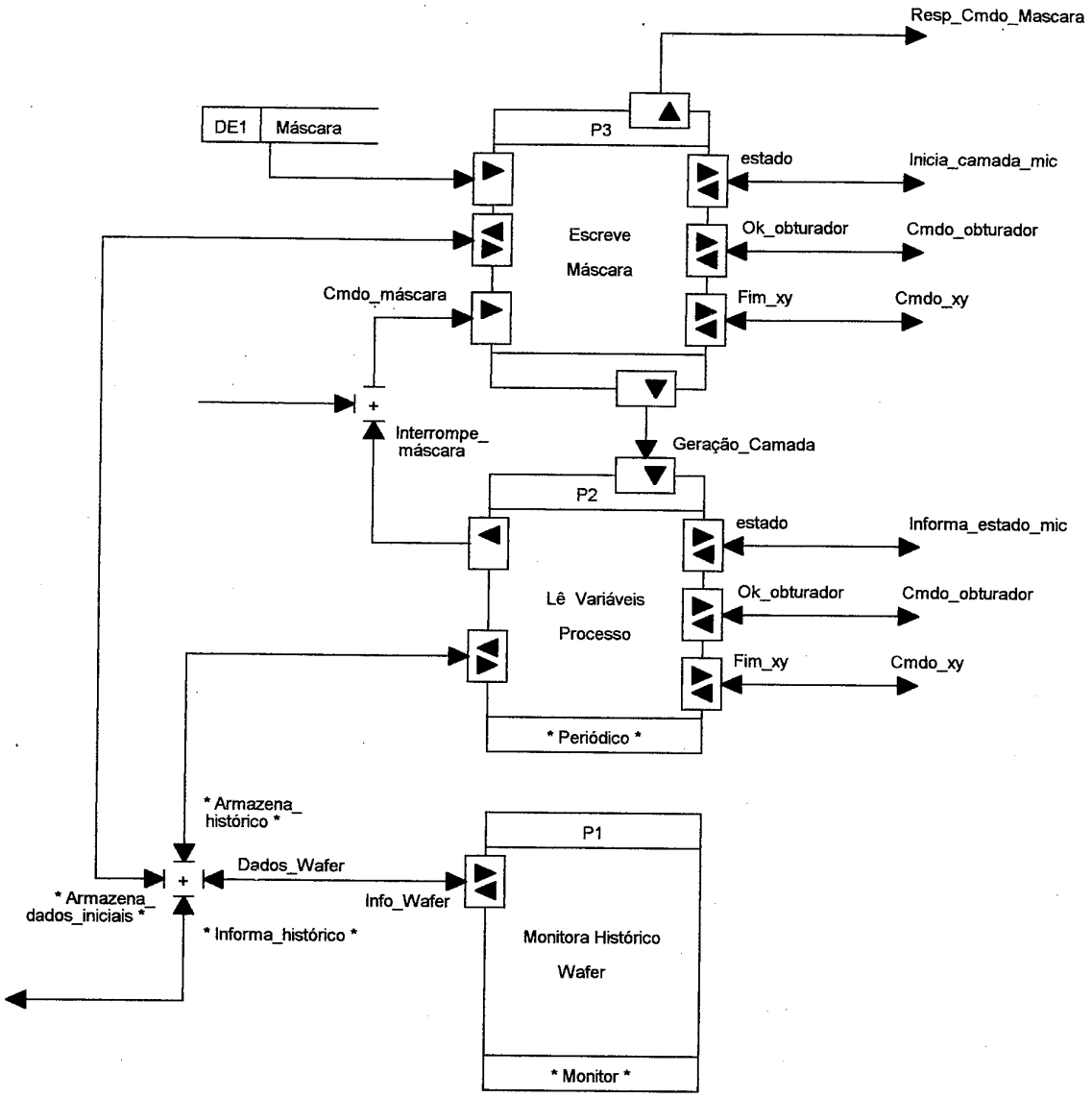
C.1.3.3 Estrutura das Mensagens

```
var Verifica_Estado: record
      begin
            cmdo:   {INICIA_CAMADA   /   TERMINA_CAMADA   /   INFORMA_ESTADO   /
            ARMAZENA_INFO_MIC}
            caso tipo seja
            ARMAZENA_INFO_MIC:
            INICIA_CAMADA:
                  parâmetros_camada: tipo_info_mic
      end
{modela a resposta do subsistema a um pedido de verificação do microscópio
eletrônico}

var Estado: tipo_estado
{modela a resposta a um pedido de verificação do microscópio eletrônico}

var Cmdo_Máscara : record
begin
      código : {INTERROMPE/ INICIA/ GERA }
      caso código seja
      GERA|INICIA : record
      begin
            dados_iniciação_wafer    : tipo_iniciação_wafer
            parâmetros_corrente      : tipo_parâmetros_camada
      end
end
{modela um comando de geração de máscara}

var Resp_Cmdo_Máscara : record
begin
      tipo:{STATUS/ FIM/ FALHA/ INTERROMPIDA/ INSTABILIDADE}
      status : tipo_status_geração
      mensagem : string
end
{modela a resposta a um comando de geração de máscara}

var Info_Wafer: record
      begin
            operação: {CRIA/ ARMAZENA_FIM/ INFORMA / ARMAZENA_DADOS_INICIAIS /
            ARMAZENA_HISTÓRICO}
            case operação
                  INFORMA:
                        id_wafer : integer
                  CRIA:
                  ARMAZENA_DADOS_INICIAIS:
                        id_wafer : interger
                        camada   : integer
                  ARMAZENA_HISTÓRICO:
                        medições : tipo_medidas

      end
{modela um pedido de serviço sobre o histórico de um wafer.}
      end

var Dados_Wafer: record
      begin
            ok_operação : boolean
            Arq_histórico: string
```

```
            { nome de arquivo sequencial contendo informações de gravação das
            camadas cujo registro é do tipo_medidas }
      end

var Cmdo_Obturador : {ABRE / FECHA / INFORMA}
{ modela um pedido de abertura/fechamento do obturador de feixe de elétrons }

var Ok_Obturador: record
    begin
            Ok: boolean
            info: tipo_info_obturador
    end

var Cmdo_xy : record
    begin
      código : {INTERROMPE/MOVE/INFORMA/REPOUSO}
      case código
      MOVE : record
      begin
            velocidade          : tipo_velocidade
            posição_destino     : tipo_posição_mesa
            {contém para posições x e y o deslocamento a ser empreendido à mesa}
      end
    end
{ modela um pedido de movimentação da mesa nas direções x e/ou y }

var Fim_xy : record
    begin
            ok_mesa: boolean
            posição: tipo_posição_mesa
    end
{ modela a resposta a um pedido de movimentação da mesa nas direções x e/ou y e
a um pedido de informação da posição da mesa }
```

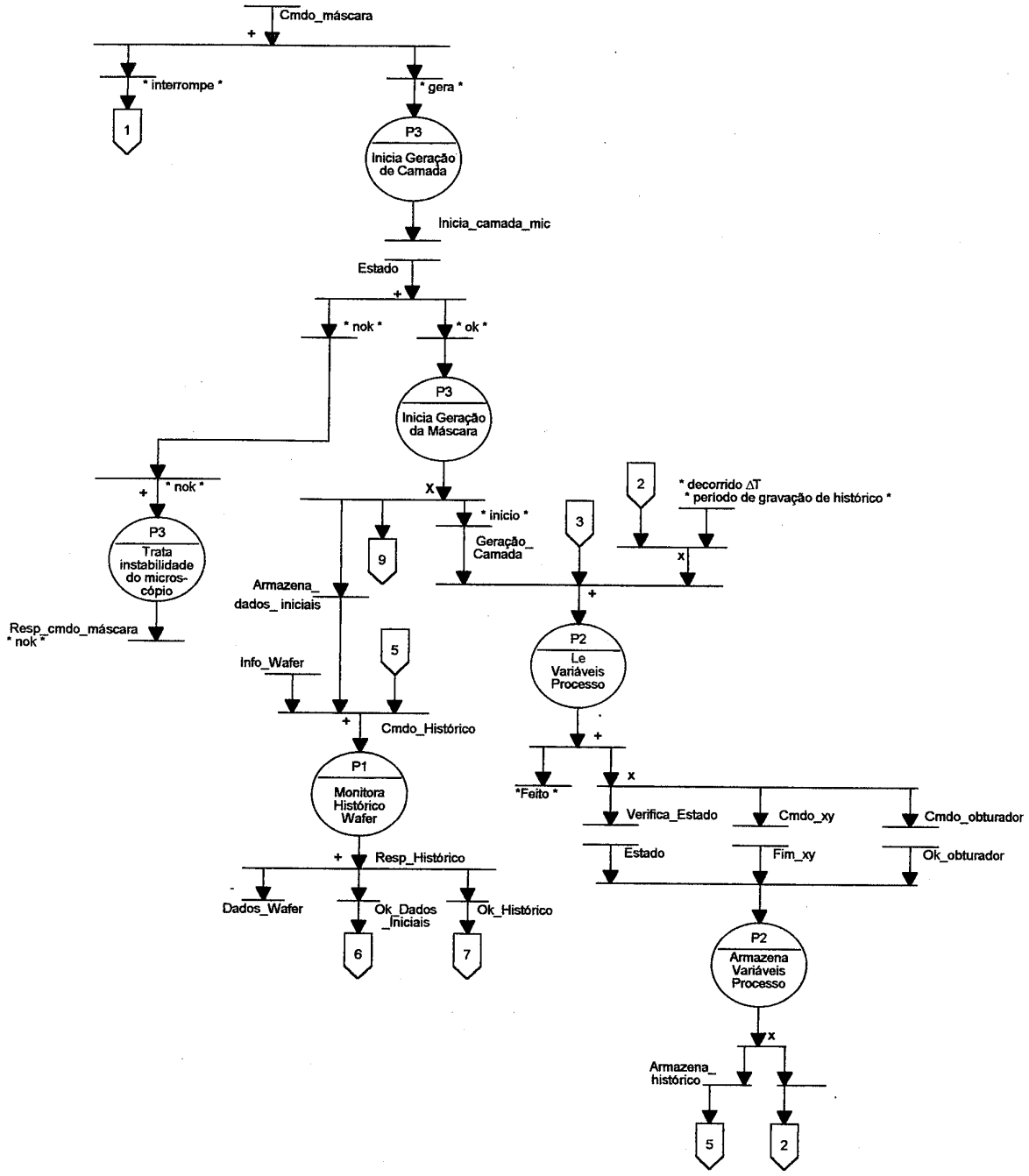
C.2 Design do Subsistema Básico

C.2.1 Diagrama de Estrutura

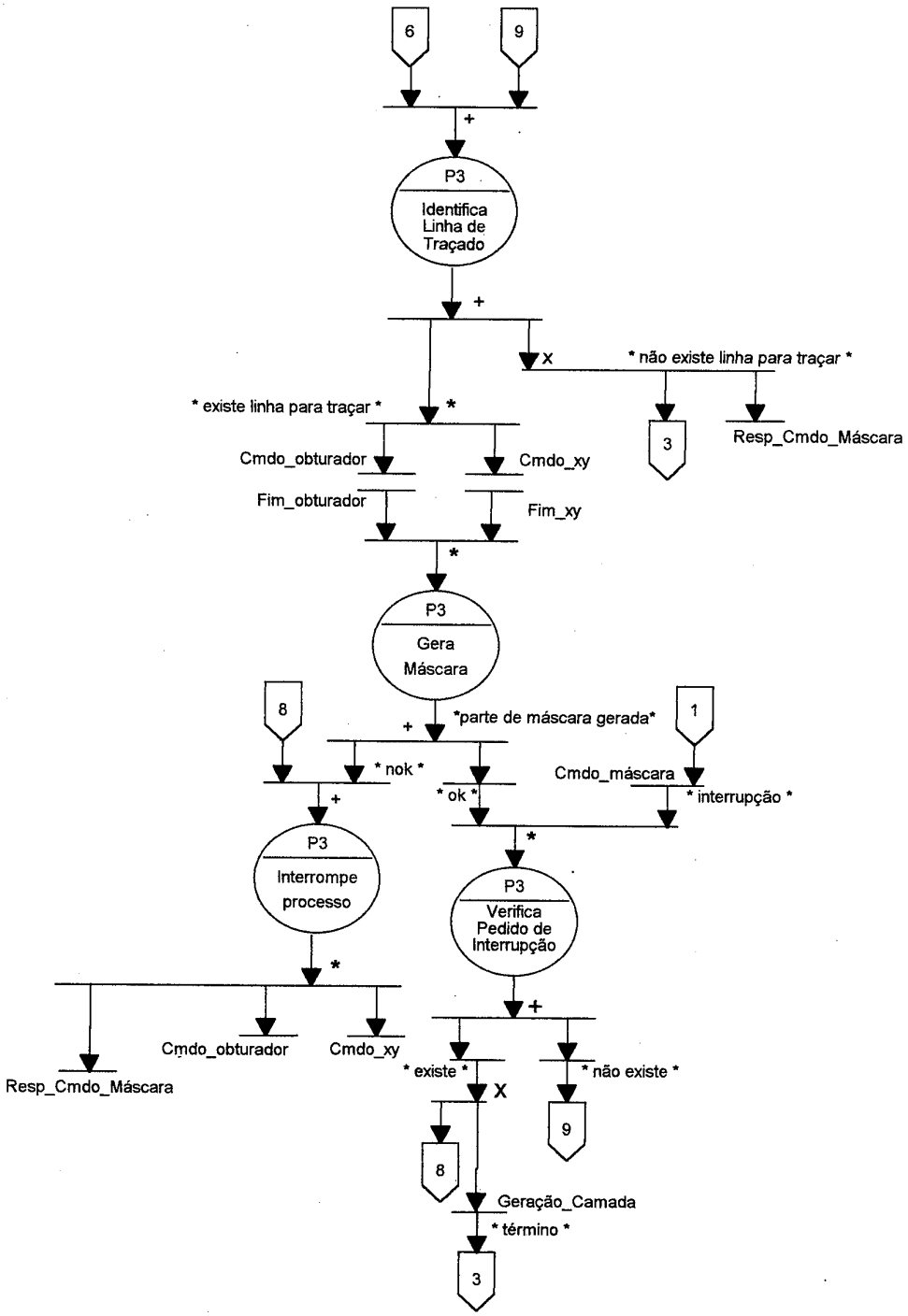


C.2.2 Diagrama de Comportamento

(1/2)



(2/2)



C.2.3 Descrição dos Processos

- P1 - Monitora Histórico Wafer
Entradas : Cmdo_Histórico
Saídas : Resp_Histórico

Monitora a estrutura de dados relativa aos Históricos de wafers já gerados.

```
início
```

```
caso Info_Wafer.tipo seja
      INFORMA_HISTÓRICO:
            Envia Resp_Histórico com nome do arquivo contendo
            histórico_wafer de Cmdo_Histórico.Id_Wafer
      ARMAZENA_HISTÓRICO:
            armazena medições em histórico_wafer
            Envia Resp_Histórico com OK
      ARMAZENA_DADOS_INICIAIS:
            abre arquivo para dados referentes a histórico de id_wafer e
            camada sendo gerada
            Envia Resp_Histórico com OK
      CRIA:
            cria arquivo para dados referentes a histórico de id_wafer e
            camada sendo gerada
            Envia Resp_Histórico com OK
      ARMAZENA_DADOS_INICIAIS:
            fecha arquivo para dados referentes a histórico de id_wafer e
            camada sendo gerada
            Envia Resp_Histórico com OK

fim
```

Implementação de Histórico_Wafer em Arquivo:

- no início da gravação de cada camada é criado um arquivo em disco onde cada registro é do tipo medida e com nome da forma:

WXXXXXXX.Cn

- onde, **XXXXXXX** equivale aos caracteres correspondentes ao inteiro que identifica o wafer (id_wafer) e **n** é o caracter correspondente ao numero da camada(W[id_wafer].Cn);

- P2 - Lê Variáveis Processo

Entradas : Nenhuma (periódico)

Saídas : Armazena_Histórico, Interrompe_Máscara

Serviços Requisitados : Informa_estado_mic, Cmdo_obturador, Cmdo_xy, com respostas associadas: Estado, Ok_obturador e Fim_xy, respectivamente.

Periodicamente, a partir de Geração_Máscara com INICIA, lê as variáveis do processo e pede seu armazenamento no histórico. Termina o processo de gravação de máscara através de Interrompe_máscara quando o estado do microscópio estiver Não-OK. Para a execução quando receber Geração_Máscara com INTERROMPE.

```
início
Espera Geração_Camada.código = INICIA
faça enquanto (não ocorrência de mensagem ao efetuar consulta em Geração_Camada)
            pede serviço Informa_estado_mic com INFORMA_ESTADO
            pede serviço Cmdo_xy com INFORMA
            pede serviço Cmdo_Obturador com INFORMA
            Espera Estado
            se Estado.Ok = NOK
            então
                  {microscópio com valores fora da faixa desejada}
                  pede serviço Cmdo_Máscara com Interrompe_Máscara
            senão
                  Espera Fim_xy
                  Espera Ok_Obturador
                  pede serviço Info_Wafer com Armazena_Histórico com
                  Fim_xy.info_mesa,                Ok_Obturador.info_obturador,
                  Estado.Variáveis_Microscópio, tempo de medição
                  Espera Dados_Wafer
            fim
      fim
```

```
se Espera Geração_Camada.código <> TERMINA então erro em que requisita esses
serviços.
fim
```

- P3 - Escreve Máscara
Entradas : Cmdo_Máscara
Saídas : Resp_Cmdo_Máscara
Serviços Requisitados : Geração_Camada

Controla a geração de camada sobre o chip, gravando uma máscara sobre cada réplica do chip sendo gerado. A geração de uma camada é iniciada com a chegada de uma mensagem na porta Cmdo_Máscara com tipo INICIA. Envia mensagem Geração_Camada com INICIA pedindo serviço de armazenamento de histórico. O processo transforma o arquivo CAD correspondente à máscara em linhas de traçado e para cada linha:

- move a mesa para a posição inicial de traçado;
- abre obturador;
- move a mesa para posição final de traçado;
- fecha obturador.

Em caso de erros em quaisquer destes dispositivos, interrompe o processo de geração de camada, enviando Geração_Camada com TERMINA.

Enquanto gerando camada, o processo consulta a porta Cmdo_Máscara verificando a chegada de mensagem com tipo INTERROMPE. Caso ocorra, o processo de geração de camada envia Geração_Camada com TERMINA e é interrompido.

```
var linha : record
      begin
            pos_inicial : tipo_posição_mesa;
            pos_final : tipo_posição_mesa;
      end;
{ armazena uma linha que forma a máscara a ser traçada, com as posições iniciais
e finais de traçado }

var situação := {ok, NOK}
{ armazena situação de funcionamento dos diversos equuipamentos controlados para
geração de camada }

var obturador: {ABERTO / FECHADO};
{ armazena o estado do obturador }

var linhas_traçado = record
      begin
            num_linhas : integer;
            { armazena a quantidade de linhas a serem traçadas no wafer para o
              traçado da máscara pedida }
            linhas_traçado: array of tipo_linha
      end

var num_réplicas : integer;
{ armazena a quantidade de réplicas a serem traçadas no wafer }

início
      caso Cmdo_Máscara.código

            INICIA:
                  Envia Info_Wafer com Armazena_Dados_Iniciais para armazenar
                  info. incial de camada sendo gerada para wafer
                  Espera Dados_Wafer
                  gera_camada(Cmdo_Mascara.num_réplicas, Cmdo_Máscara.id_camada,
                  nível, Cmdo_Máscara.Delta_vértice,
                  Cmdo_Máscara.distância_entre_réplicas)
```

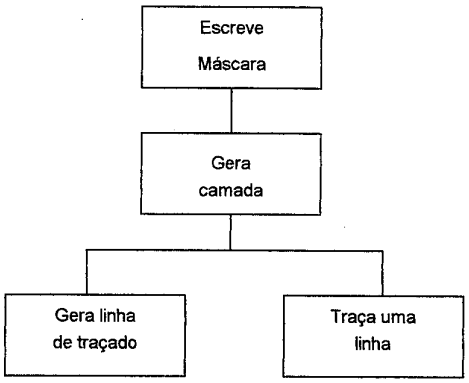
```
            termina armazenamento de dados de histórico wafer

      GERA:
            Envia Info_Wafer com Armazena_Dados_Iniciais para armazenar
            info. incial de camada sendo gerada para wafer
            Espera Dados_Wafer
            gera_camada(Cmdo_Mascara.num_réplicas, Cmdo_Máscara.id_camada,
            nível, Cmdo_Máscara.Delta_vértice,
            Cmdo_Máscara.distância_entre_réplicas)
            termina armazenamento de dados de histórico wafer

      INTERROMPE:
            { Não é feito nada - só quando está geranda máscara }
fim
```

OBS: Os procedimentos:

- gera_camada (num_réplicas, Máscara_Corrente, nível, Delta_vértice, distância_entre_réplicas)
- interrompe_processo

são iguais aos definidos na especificação do serviço

Diagrama Hierárquico de Módulos



Objetivos dos Módulos

- **Gera camada** - Coordena a leitura do arquivo CAD e o traçado de linhas.
- **Gera linha de traçado** - transforma uma figura lida do CAD em um conjunto equivalente de linhas de traçado.
- **Gera camada** - Coordena o traçado de uma linha (posicionamento da mesa em início, abertura do obturador, posicionamento da mesa no fim, fechamento do obturador).

C.2.4 Descrição dos Dados Introduzidos

```
var Geração_Camada: {INICIA / TERMINA}
```

C.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S2.2

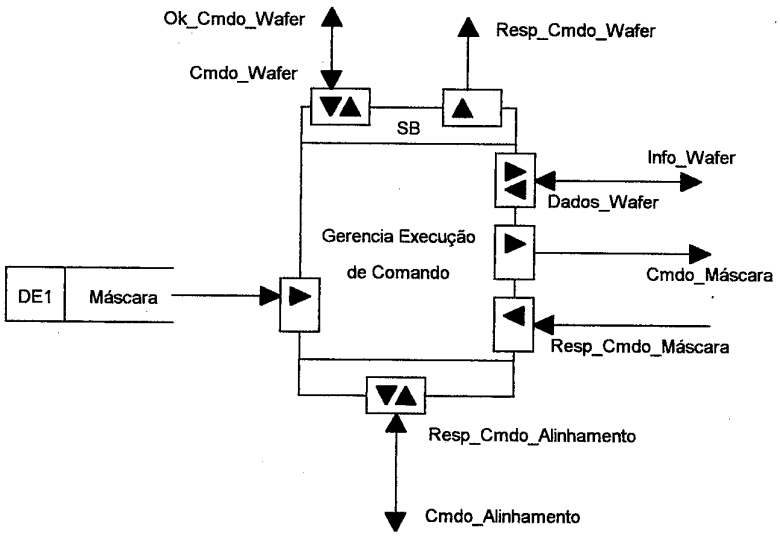
D. Subsistema Básico - Gerencia Execução de Comando

D.1 Especificação

D.1.1 Descrição

Faz a análise semântica de comandos sintaticamente corretos que dizem respeito à geração de wafers. Para isso, controla quais wafers já foram iniciados e o estado de geração de cada um deles. Gerencia a execução dos comandos, o que inclui gerência de alinhamento, gerência de geração de máscara e pedido/análise de dados sobre wafers.

D.1.2 Diagrama de Contexto



D.1.3 Descrição dos Serviços

D.1.3.1 Serviços Externos

- Gerenciar Execução de Comando
Entradas : Cmdo_Wafer
Saídas : Ok_Cmdo_Wafer e Resp_Cmdo_Wafer

Este serviço gerencia a execução de comandos, o que corresponde a:

- Iniciar Wafer
- Alinha Wafer
- Emitir Histórico Wafer
- Interromper Processo
- Confirmar Marca de Alinhamento
- Ajustar Marca nas direções X ou Y

OBS: Como vários wafers podem ser gerados com um mesmo projeto de chip (máscaras e parâmetros de operação) foi introduzido um comando (Definir Chip) que define um projeto de chip referenciado no comando de Iniciar Wafer.

```
{* variáveis locais *}

var estado: {OCIOSO / GERANDO / ALINHANDO}
{armazena o estado do processo}

var estado_wafer: tipo_estado_wafer
      {armazena o estado do wafer
      (EM_GERACAO/SUCESSO/INTERROMPIDO/INSUCESSO)}

var id_wafer: integer
 {armazena o identificador do wafer que está sendo tratado}

var parâmetros_correntes : tipo_parâmetros_camada

var dados_iniciação_wafer : tipo_iniciação_wafer

início
se Cmdo_Wafer.código <> INTERROMPE
então
   caso Cmdo_Wafer.código seja
   DEF_CHIP:
            se id_chip existe em projeto_chip
            então
            envia Ok_Cmdo_Wafer com NOK e motivo
            senão
            início {#3}
            analisa parâmetros
            armazena em projeto_chip
            fim {#3}
      fim {#2}

   INICIA:
      se estado = OCIOSO
      então
      armazena Cmdo_Wafer.Id_Wafer em id_wafer
      início {#2}
            se (id_wafer existe em wafer) ou
            (não existe id_chip em projeto_chip) ou
            (não existe pelo menos uma  máscara_a_traçar em
            DE1_MÁSCARA)
            então
            Envia Ok_Cmdo_Wafer com NOK e motivo
            senão
            início {#3}
                  armazena em dados_iniciação_wafer os valores dos dados
                  em  Cmdo_wafer
                  analisa parâmetros
                  armazena    em    parâmetros_correntes    os    parâmetros
                  correspondentes a geração de máscara de marcas
                  pede serviço Cmdo_Máscara com:
                  INICIA,dados_iniciação_wafer e parâmetros_correntes
                  estado = GERANDO
                  Envia Cmdo_Máscara para Máscara de marcas
            fim {#3}
      fim {#2}
   ALINHA:
      se (id_wafer inexistente) ou (id_chip inexistente)
      (estado = TRAÇANDO ou ALINHANDO) ou
      (estado_wafer <> SUCESSO)
      então
            envia Ok_Cmdo_Wafer com NOK
      senão
      início {#2}
            se    (não existe máscara a gerar) ou
            (arquivo de máscara não existe) ou
            (estado_wafer <> SUCESSO}
            então
```

```
                envia OK_Cmdo_Wafer com NOK e motivo
          senão
                armazena em parâmetros_correntes os parâmetros
                correspondentes a máscara a gerar
                Envia Cmdo_Alinhamento com INICIAR, número de marcas e
                posição de marcas
                estado = ALINHANDO
                estado_wafer = EM_GERAÇÃO
                Espera Resp_Cmdo_Alinhamento
                se Resp_Cmdo_Alinhamento = NOK
                então
                início
                      estado_wafer = SUCESSO
                      {erro no alinhamento mantém SUCESSO da última
                      camada gerada}
                      Envia Ok_Cmdo_Wafer com NOK e motivo
                fim
          fim {#2}
HISTÓRICO:
    se (id_wafer inexistente) ou
       (id_wafer = ao de id_wafer com camada sendo gerada)
    então
          Envia Ok_Cmdo_Wafer com NOK e motivo
    senão
    início {#2}
          Envia Ok_Cmdo_Wafer com OK
          Envia Info_Wafer com id_wafer
          Espera Dados_Wafer
          {Dados_Wafer contém nome de arquivo com Dados do Wafer}
          Resp_Cmdo_Wafer.código = HISTÓRICO
          envia Resp_Cmdo_Wafer com Dados_Wafer
    fim {#2}
INTERROMPE:
    se estado = TRAÇANDO
    então
    início {#2}
          Envia Ok_Cmdo_Wafer com OK e motivo
          Envia Cmdo_Máscara com INTERROMPE
          estado = INATIVO
          estado_wafer = INSUCESSO
    fim {#2}
    senão
          Envia Ok_Cmdo_Wafer com NOK e motivo
CONFIRMA:
    se estado = ALINHANDO
    então
    início {#2}
          Envia Cmdo_Alinhamento com CONFIRMA_MARCA
          Espera Resp_Cmdo_Alinhamento
    se ok {* Próxima marca *} e Resp_Cmdo_Alinhamento =
    FIM_ALINHAMENTO
          Envia Ok_Cmdo_Wafer com OK e motivo
    então
    início {#3}
                estado = TRAÇANDO
                Envia Cmdo_Máscara com GERA, dados_iniciação_wafer e
                parâmetros_correntes para camada a gerar
    fim {#3}
    senão
          Envia Ok_Cmdo_Wafer com NOK e motivo
    fim {#2}
    senão
          Envia Ok_Cmdo_Wafer com NOK
AJUSTE:
    se estado = ALINHANDO
    então
    início {#2}
          Envia Cmdo_Alinhamento com AJUSTE_MARCA e ajuste de Cmdo_Wafer
```

```
                Espera Resp_Cmdo_Alinhamento
                se Resp_Cmdo_Alinhamento = NOK
                então
                Envia Ok_Cmdo_Wafer com NOK e motivo
            senão
                Envia Ok_Cmdo_Wafer com OK e motivo
            fim
            fim {#2}
            senão
                Envia Ok_Cmdo_Wafer com NOK e motivo
fim
```

D.1.3.2 Serviços Internos

- Tratar resposta de comando máscara
Entradas: Resp_Cmdo_Máscara
Saídas: Resp_Cmdo_Wafer

Espera resposta assíncrona ao envio de Cmdo_Máscara, atualiza estado e envia resposta a Cmdo_Wafer de acordo com o resultado obtido.

```
início
    enquanto verdadeiro faça
      início
          Espera Resp_Cmdo_Máscara
          CASO Resp_Cmdo_Máscara.tipo_resp
          FIM_WAFER:
             estado_wafer = SUCESSO
             Envia Resp_Cmdo_Wafer
          FALHA_WAFER:
             estado_wafer = INSUCESSO
             Envia Resp_Cmdo_Wafer
          HISTÓRICO:
             Envia Resp_Cmdo_Máscara
          STATUS:
             Envia Resp_Cmdo_Máscara
        fim
fim
```

D.1.3.3 Serviços Requisitados

- Atender a Pedido de Histórico Wafer
Saída : Info_Wafer
Entrada : Dados_Wafer

Serviço requisitado para emissão de um histórico sobre a gravação de um wafer. Como resposta envia todos os dados sobre esse wafer.

- Gerar Máscara
Saída : Cmdo_Máscara
Entrada : nenhuma

Serviço requisitado para gerar uma máscara em um wafer, conforme a máscara que lhe é enviada.

- Alinhar Wafer
Saída : Cmdo_Alinhamento
Entrada : Resp_Cmdo_Alinhamento

Serviço requisitado para alinhar um wafer segundo marcas de alinhamento traçadas na iniciação. O serviço retorna um status relativo à execução do comando de alinhamento (OK / NOK).

D.1.4 Descrição Detalhada dos Dados

D.1.4.1 Definições de Tipos

```
type tipo_estado_wafer = {EM_GERAÇÃO / SUCESSO / INSUCESSO }

type tipo_parâmetros = record
      begin
            tensão_fonte                  : real
            corrente_objetiva             : real
            corrente_condensadora         : real
            corrente_emissão_filamento    : real
            dose                          : real
            tipo : {resolução, diâmetro}
            case tipo
            begin
                  resolução               : real
                  diâmetro_feixe          : real
            end
      end
{define os parâmetros de operação do microscópio eletrônico}
(*) diâmetro_feixe pode armazenar também a resolução do mesmo.

type tipo_posição_mesa = record
      begin
            pos_x : real
            pos_y : real
      end
{define uma posição da mesa}

type tipo_iniciação_wafer = record
      begin
            id_wafer                : integer
            id_chip                 : integer
            num_réplicas            : integer
            tempo_registro          : real
            Delta_vértice           : tipo_posição
            distância_entre_réplica : tipo_posição

      end

type tipo_dados_geração = record
      begin
            tipo_estado             : tipo_estado_wafer
            numumero_de_marcas      : integer
            posição_marcas          : array of tipo_posição
            máscaras_feitas         : integer
      end

type tipo_wafer = record
      begin
            dados_gerais            : dados_iniciação_wafer
            dados_geração           : tipo_dados_geração
            posição_origem          : tipo_posição_mesa
            origem                  : tipo_posição_mesa
      end

type tipo_máscara = record
      begin
            id_máscara        : string
            nível             : integer
            parâmetro_operação: tipo_parâmetro
      end
```

```
type tipo_projeto_chip = record
      begin
                  id_chip           : integer
                  máscaras_a_fazer  : integer
                  id_marcas         : string
                  operação_marca    : tipo_parâmetro
                  dados_camada      : array[1..máscaras_a_fazer] of
                                      tipo_máscara
      end

type tipo_info_mic = record
      begin
            corrente_objetiva       : real
            corrente_condensadora   : real
            tensão_fonte            : real
            filamento               : real
      end
{define informação sobre o microscópio eletrônico}

type tipo_info_obturador = record
      begin
            corrente_absorção       : real
      end
{define informação sobre o obturador do feixe}

type tipo_histórico = record
      begin
            dados_gerais      : dados_iniciação_wafer
            projeto_chip      : tipo_projeto_chip
            nome_arquivo      : string
            {contém nome de arquivo cujo registro é no formato de tipo_medidas,
            sequencial e com medidas por camada }

      end

type medidas = record
      begin
            info_mesa         : tipo_posição_mesa
            info_mic          : tipo_info_mic
            info_obturador    : tipo_info_obturador
            tempo_medição     : real
      end
{tipo de dados que contém uma medição efetuada no equipamento}

type status_geração = record
      begin
            tempo_previsto    : real
            tempo_gasto       : real
      end

type tipo_parâmetros_camada = record
  begin
      máscara_corrente              : tipo_nome_máscara
      num_camada                    : integer
      posição_origem                : tipo_posição_mesa
      origem                        : tipo_posição_mesa
  end
```

D.1.4.2 Estruturas de Dados Encapsuladas

```
var wafer : array of tipo_wafer
{essa estrutura mantém as informações gerais de todos os wafers gerados pelo
sistema}

var projeto_chip : array of tipo_projeto_chip
```

{essa estrutura mantém a informação de todos os projetos chips usados na geração de wafers pelo sistema}

Estruturas Encapsuladas Armazenadas em Meio Magnético

As informações armazenadas pelas estruturas wafer e projeto_chip não podem ser perdidas entre execuções diferentes do sistema portanto, essas estruturas ficarão armazenadas em arquivo.

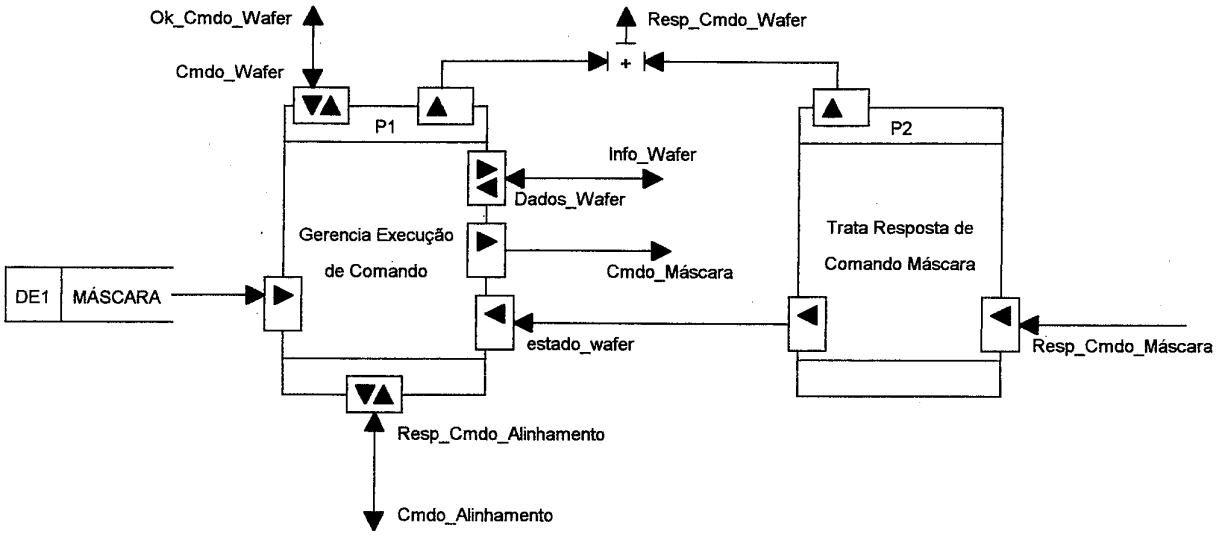
D.1.4.3 Estrutura das Mensagens

```
var Cmdo_Wafer : record
    begin
      código : {DEF_CHIP /INICIA / ALINHA / CONFIRMA / AJUSTE / INTERROMPE /
      HISTÓRICO}
      case código
      DEF_CHIP:
            projeto_chip : tipo_projeto_chip

      INICIA :
            dados_iniciação_wafer : tipo_iniciação_wafer
      ALINHA :
            id_wafer : integer
      AJUSTE :
            deslocamento : tipo_posição_mesa
      HISTÓRICO :
            id_wafer    : integer
    end
{modela um comando sobre um wafer}

var Ok_Cmdo_Wafer: record
      begin
            estado_execução   : inteiro
            aceito            : boolean
            motivo            : string
      end

var Resp_Cmdo_Wafer: record
    begin
            estado_execução : inteiro
            tipo_resp: {HISTÓRICO / STATUS / INTERROMPE}
            caso tipo_resp seja
            HISTÓRICO:
                  histórico   : tipo_histórico
            STATUS:
                  status      : tipo_status_geração
            INTERROMPE:
                  OK          : boolean
                  mensagem    : string
                  {* mensagem com motivo da interrupção *}
    end
{modela a resposta a um comando sobre um wafer}

var Info_Wafer : record
      begin
            operação:  {INFORMA/  ARMAZENA_DADOS_INICIAIS/  ARMAZENA_HISTÓRICO/
            CRIA/ Armazena_FIM}
            case operação
                  INFORMA:
                        id_wafer : integer
                  CRIA:
                  ARMAZENA_DADOS_INICIAIS:
                        id_wafer : integer
                        camada   : integer
                  ARMAZENA_HISTÓRICO:
                        medições : tipo_medidas
```

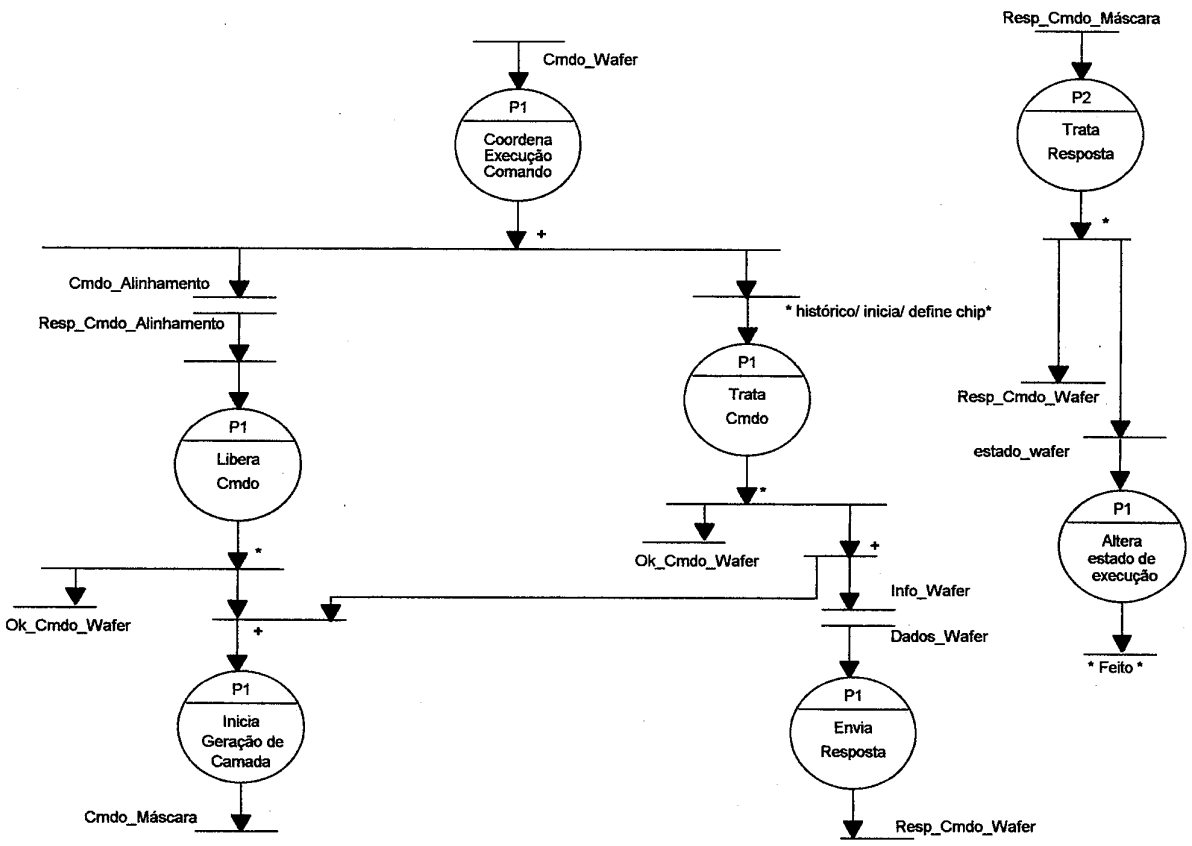
```
      end
{modela um pedido de serviço sobre o histórico de um wafer.}

var Dados_Wafer : record
    begin
            caso seja
            ARMAZENA | INICIA_ARM:
                  ok : boolean
            INFORMA:
                  nome: string
                  {nome de arquivo contendo dados de histórico wafer}
    end
{modela a resposta a um pedido de dados sobre um wafer}

var Cmdo_Máscara : record
      begin
            código : {INTERROMPE/ GERA/ INICIA }
            caso código seja
            GERA : record
            begin
                  id_wafer                  : integer
                  num_réplicas              : integer
                  tempo_registro            : real
                  Delta_vértice             : tipo_posição
                  distância_entre_réplica   : tipo_posição
                  id_máscara                : string
                  nível                     : integer
                  parâmetro_operação        : tipo_parâmetro

            end
      end
{modela um comando de geração de máscara}

var Resp_Cmdo_Máscara : {FALHA_WAFER/ FIM_WAFER/ HISTÓRICO/ STATUS}
{modela a resposta a um comando de geração de máscara}

var Cmdo_Alinhamento : record
      begin
            código : {CONFIRMAR/ AJUSTAR/ INICIAR}
            case código
            AJUSTAR:
                  deslocamento : tipo_posição
      end
{modela um comando de alinhamento}

var Resp_Cmdo_Alinhamento : record
        begin
              tipo: {OK / NOK / FIM_ALINHAMENTO };
              caso tipo seja
                    FIM_ALINHAMENTO:
                          pos_origem_x: real;
                          pos_origem_y: real;
                          {em relação ao eixo de coordenadas absolutas}
        end;
{modela a resposta a um comando de alinhamento}
```

D.2 Design do Subsistema Básico

D.2.1 Diagrama de Estrutura



D.2.2 Diagrama de Comportamento

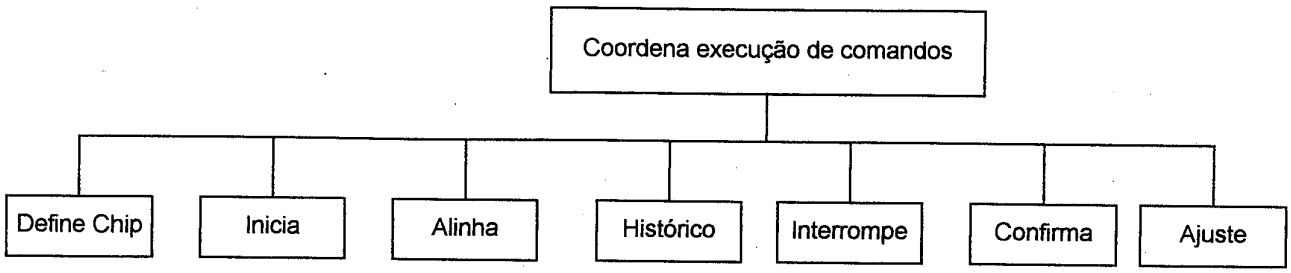


D.2.3 Descrição dos Processos

Equivalente à descrição dos serviços Gerenciar Execução de Comando e Tratar Resposta de Comando Máscara.

- P1 - Coordena Execução de Comandos

Diagrama Hierárquico de Módulos



Objetivos dos Módulos

- **Define Chip** - Responder ao comando de definição de um projeto de chip. Verifica a existência prévia do projeto de chip e, se existir, retorna mensagem de erro. Caso contrário, armazena dados do projeto.

- **Inicia** - Responder ao comando de inicialização de wafer. Verifica a existência prévia do wafer e se existir, retorna mensagem de erro. Verifica a existência prévia do projeto de chip associado e se não existir, retorna mensagem de erro. Retorna erro também caso a operação de traçado já esteja sendo efetuada. Caso contrário, pede serviço de geração de camada com dados de camada de marcas..

- **Alinha** - Responder ao comando de alinhamento de wafer. Retorna erro se o wafer não existir ou se a operação de traçado já estiver sendo efetuada. Caso contrário, pede serviço de início de alinhamento e espera resposta.

- **Histórico** - Responder ao comando de pedido de histórico. Envia o nome do arquivo contendo os dados de histórico de um wafer. Retorna mensagem de erro se o wafer não existir ou wafer sendo gravado.

- **Interrompe** - Responder ao comando de interrupção de traçado. Pede serviço de interrupção.

- **Confirma** - Responder ao comando de confirmação de uma marca de alinhamento. Começa a traçar a máscara caso a marca tenha sido a última.

- **Ajuste** - Responder ao comando de ajuste de marca. Pede serviço de ajuste, caso esteja em estado de ajuste. Caso contrário, retorna mensagem de erro.

D.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S2.3

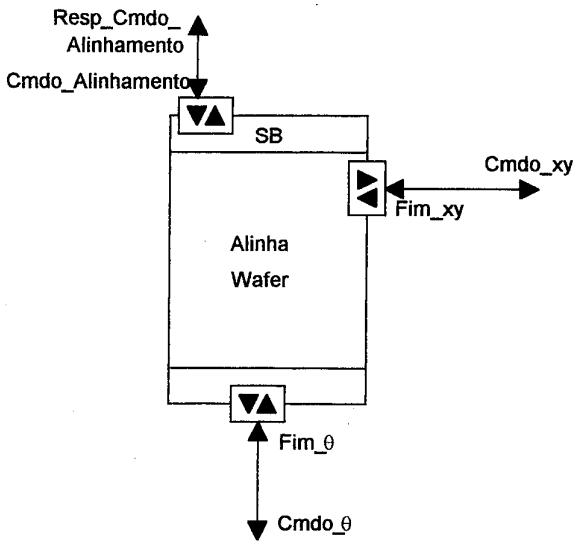
E. Subsistema Básico - Alinha Wafer

E.1 Especificação

E.1.1 Descrição

Controla o alinhamento do wafer segundo marcas de alinhamento traçadas na iniciação. Para isso, pede serviços de movimentação de mesa nas direções x, y e θ. Executa os comandos relativos ao processo de alinhamento (ajuste nas direções x e y e confirmação de marca).

E.1.2 Diagrama de Contexto



E.1.3 Descrição dos Serviços

E.1.3.1 Serviços Externos

- Interpretar Comandos de Alinhamento
Entradas : Cmdo_Alinhamento
Saídas : Resp_Alinhamento
Este serviço executa os comandos relativos ao alinhamento. Para tal, possui subserviços, que são:

- Iniciar Alinhamento
- Ajustar Marca X e/ou Y
- Confirmar Marca de Alinhamento

```
{* variáveis locais *}

var a_confirmar : integer
      {armazena o número da confirmação de marca de alinhamento seguinte}

var ajuste : tipo_posição_mesa
{armazena o valor de movimentação da mesa nas direções x e y, tanto para o
ajuste de uma marca, quanto para a movimentação inicial}
```

```
var ajuste_θ : real
      {armazena a rotação a ser aplicada na mesa}

início
      enquanto Verdadeiro faça
            Espera Cmdo_Alinhamento com num_marcas e posição_marcas
            caso Cmdo_Alinhamento seja
            INICIAR:
                  Envia Cmdo_xy com REPOUSO
                  calcula ajuste para primeira marca
                  Espera Fim_xy
                  se Fim_xy = NOK
                        então
                        Envia Resp_Cmdo_Alinhamento com NOK
                  senão
                  início {#2}
                        Envia Cmdo_xy com MOVE, ajuste e VELOCIDADE_MÁXIMA
                        Espera Fim_xy
                        a_confirmar = 0 {* primeira marca *}
                        Envia Resp_Cmdo_Alinhamento com Fim_xy
                  fim {#2}
            AJUSTAR:
                  Envia    Cmdo_xy    com    MOVE,    Cmdo_Alinhamento.ajuste    e
                  VELOCIDADE_MÁXIMA
                  Espera Fim_xy
                  Envia Resp_Cmdo_Alinhamento com Fim_xy
            CONFIRMAR:
                  incrementa a_confirmar
                  caso a_confirmar seja
                  2: {1a. confirmação 2a. marca}
                        calcula ajuste_θ
                        Envia Cmdo_θ com ajuste_θ
                        Espera Fim_θ
                        Envia Resp_Cmdo_Alinhamento com Fim_θ
                  3: {2a. confirmação 2a. marca}
                        calcula posição de origem
                        Envia Cmdo_xy com MOVE, ajuste e VELOCIDADE_MÁXIMA
                        Espera Fim_xy
                        Envia Resp_Cmdo_Alinhamento com Fim_xy
                        se a_confirmar = num_marcas + 1: {confirmou todas as
                        marcas e a 2a. duas vezes*}
                        então
                        Envia Resp_Cmdo_Alinhamento com FIM_ALINHAMENTO
                  outras:
                        se a_confirmar = num_marcas + 1: {confirmou todas as
                        marcas e a 2a. duas vezes*}
                        então
                        Envia Resp_Cmdo_Alinhamento com FIM_ALINHAMENTO
                        senão
                              calcula ajuste para próxima marca
                              Envia Cmdo_xy com MOVE, ajuste e VELOCIDADE_MÁXIMA
                              Espera Fim_xy
                              Envia Resp_Cmdo_Alinhamento com Fim_xy
                        fim
fim
```

E.1.3.2 Serviços Internos

Nenhum.

E.1.3.3 Serviços Requisitados

- Movimentar Mesa θ
Saída : Cmdo_θ
Entrada : Fim_θ

Serviço requisitado para mover a mesa na direção θ. O serviço retorna a condição de sucesso (OK) ou insucesso (NOK).

- Movimentar Mesa xy
Saída : Cmdo_xy
Entrada : Fim_xy

Serviço requisitado para mover a mesa nas direções x e y, ou para informar a posição atual da mesa. Retorna a condição de sucesso (OK) ou insucesso (NOK), no caso de um pedido de movimentação, e a posição atual da mesa, no caso do pedido de informação.

E.1.4 Descrição Detalhada dos Dados

E.1.4.1 Definições de Tipos

```
type tipo_posição_mesa = record
      begin
            x_mesa: real;
            y_mesa: real;
      end;
{define uma posição da mesa}

type velocidade = record
      begin
            velocidade_x : real;
            velocidade_y : real;
      end;
{define a velocidade de deslocamento da mesa}
```

E.1.4.2 Estruturas de Dados Encapsuladas

Nenhuma.

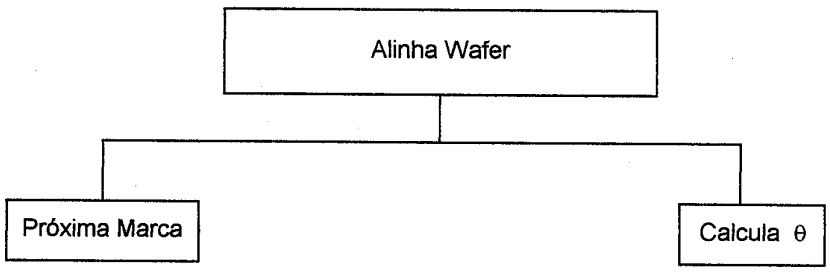
E.1.4.3 Estrutura das Mensagens

```
var Cmdo_Alinhamento : record
      begin
            código : {CONFIRMAR / AJUSTAR / INICIAR}
            case código
            INICAR: record
            begin
                  num_marcas        : integer
                  posição_marcas    : array of tipo_posição_mesa
            end
            AJUSTAR:
            ajuste : tipo_posição_mesa
      end
{modela um comando de alinhamento}

var Resp_Cmdo_Alinhamento : record
      begin
            tipo: {OK / NOK / FIM_ALINHAMENTO };
            caso tipo seja
```

```
                    FIM_ALINHAMENTO:
                          pos_origem_x: real;
                          pos_origem_y: real;
                          {em relação ao eixo de coordenadas absolutas}
      end;
{modela a resposta a um comando de alinhamento}

var Cmdo_xy : record
    begin
      código : {INTERROMPE / MOVE / INFORMA / REPOUSO}
      case código
      MOVE : record
      begin
      velocidade: tipo_velocidade
      posição_destino: tipo_posição_mesa
            {contém para posições x e y o deslocamento a ser empreendido à
      mesa}
      end
    end
{modela um pedido de movimentação da mesa nas direções x e/ou y}

var Fim_xy : record
    begin
            caso seja
            MOVE | REPOUSO:
                  ok_mesa: boolean
            INFORMA:
                  posição: tipo_posição_mesa
    end
{modela a respota a um pedido de movimentação da mesa nas direções x e/ou y e a
um pedido de informação da posição da mesa}

var Cmdo_θ : record
    begin
            código: {INTERROMPE/MOVE/INFORMA/REPOUSO}
            ângulo_destino: real
    end
{modela um pedido de rotação da mesa}

var   Fim_θ: record
       begin
      ok_mesa: boolean
      ângulo: real
      end
{modela a resposta a um pedido de rotação da mesa}
```

E.2 Design do Subsistema Básico

Diagrama Hierárquica de Módulos



- **Próxima Marca** - Comandar o movimento da mesa para a posição provável da próxima marca.
- **Calcula** θ - Calcular ângulo necessário para alinhamento angular da mesa.

E.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S2.4

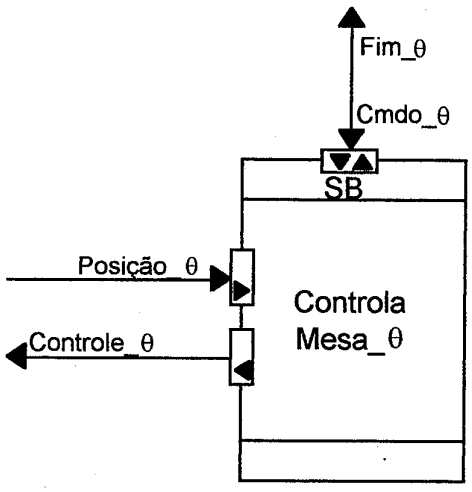
F. Subsistema Básico - Controla Mesa_θ

F.1 Especificação

F.1.1 Descrição

Controla a movimentação da mesa θ para a posição requisitada no pedido de serviço "Cmdo_θ". Lê periodicamente, durante a movimentação da mesa, sua posição, e avalia se o movimento está dentro do esperado. Caso não esteja, corrige desvios ou, em caso de insucesso, determina a falha no equipamento Mesa_θ.

F.1.2 Diagrama de Contexto



F.1.3 Descrição dos Serviços

F.1.3.1 Serviços Externos

- Movimentar Mesa θ
Entradas : Cmdo_θ
Saídas : Fim_θ

Este serviço efetua a movimentação da mesa θ, recebendo um comando de movimentação (i.e. uma rotação) e retornando um status de OK ou NOK. O serviço também deverá efetuar um acompanhamento para saber se houve algum desvio na trajetória, corrigindo-o, quando possível. Caso essa correção não seja possível por falha no equipamento, deverá ser indicada a condição de insucesso.

```
{* variáveis locais *}
var sit_mesa: {TÉRMINO, FALHA}

var pos_estimada : real
      {armazena a posição estimada atual da mesa}

var count_nok: integer
      {armazena o número de vezes que a mesa sai da posição estimada}

início
      Espera Cmdo_θ
      calcula tempo para atingir Δθ com velocidade máxima
      Envia controle_θ
      acompanha_mov_mesa_θ (sit_mesa)
      se sit_mesa = TÉRMINO
      então
            Envia Fim_θ com OK
      senão
            Envia Fim_θ com NOK
      fim

acompanha_mov_mesa_θ (sit_mesa)
início
   faça enquanto verdadeiro
      mesa_θ_estado = OK
      espera Δt {período de amostragem para detector θ }
      calcula pos_estimada_θ
      lê posição atual da mesa
      integra valor de θ_mesa
      se θ_mesa = fim ± faixa
      então
            retorna  TÉRMINO
      se módulo(θ_mesa - pos_estimada_θ) > faixa {* fora da faixa *}
      então
      início
            se mesa_θ_estado = NOK
            então
            início
                  incrementa count_nok
                  se count_nok > num_máximo
                  então
                  início
                        retorna FALHA
                        Envia Controle_θ com velocidade 0 {* parar mesa *}
                  fim
            fim
            senão
            início
                  recalcula tempo e velocidade
                  Envia novo Controle_θ
                  se mesa_θ_estado = OK
                  então
                  início
                        {inicia contagem de tempo para erro_mesa}
                        mesa_θ_estado = NOK
                        count_nok = 0
                  fim
            fim
      fim
      senão
      início
      se mesa_θ_estado = NOK
      então
      início
```

```
            para contagem de tempo
            mesa_θ_estado = OK
        fim
fim
```

F.1.3.2 Serviços Internos

Nenhum.

F.1.3.3 Serviços Requisitados

Nenhum.

F.1.4 Descrição Detalhada dos Dados

F.1.4.1 Definições de Tipos

Nenhum.

F.1.4.2 Estruturas de Dados Encapsuladas

```
var mesa_θ_estado: boolean {OK / NOK}
{armazena o estado da mesa θ}

var θ_mesa: (*)
{armazena o ângulo θ da mesa}
```

*Obs: O tipo da variável θ_mesa depende totalmente do hardware, não estando ainda perfeitamente definido.

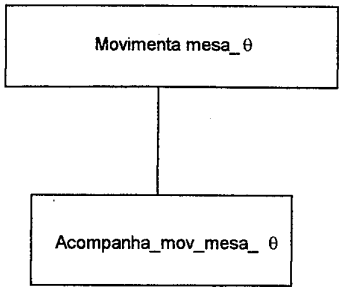
F.1.4.3 Estrutura das Mensagens

```
var Cmdo_θ : record
    begin
            código: {INTERROMPE/MOVE/INFORMA/REPOUSO}
            ângulo_destino: real
    end
{modela um pedido de rotação da mesa}

var   Fim_θ : record
      begin
            ok_mesa: boolean
            ângulo: real
      end
{modela a resposta a um pedido de rotação da mesa}

var Posição_θ : (*)
{modela a informação, proveniente dos sensores, sobre o ângulo da mesa}

var Controle_θ: (*)
```

*Obs: O tipo das variáveis Posição_θ e Controle_θ dependem totalmente do hardware, não estando ainda perfeitamente definido.

F.2 Design do Subsistema Básico

Diagrama Hierárquico de Módulos



Objetivos dos Módulos

- **Acompanha_mov_mesa_θ** - Movimentar a mesa e, a cada Δt, verificar se a mesa está se movimentando dentro da faixa. Corrigir a movimentação caso necessário. Se a movimentação da mesa ultrapassar a faixa desejada e não puder ser corrigida dentro de um período de tempo predeterminado, indicar insucesso.

F.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S3.1

G. Subsistema Básico - Controla Mesa_xy

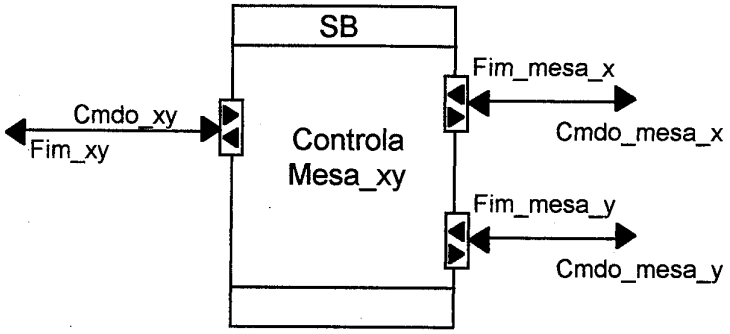
G.1 Especificação

G.1.1 Descrição

O objetivo deste subsistema é controlar a movimentação da mesa nas direções x e y. Para isso, pede serviço de movimentação de mesa em cada uma dessas direções, sincronizando o resultado.

O subsistema deverá indicar, também, qualquer tipo de falha que ocorra na movimentação dessas mesas e sincronizar o resultado.

G.1.2 Diagrama de Contexto do Subsistema Básico



G.1.3 Descrição dos Serviços

G.1.3.1 Serviços Externos

- Controlar Mesa xy
Entradas : Cmdo_xy
Saídas : Fim_xy

Este serviço coordena a movimentação simultânea da mesa nas direções x e y. Essa movimentação é efetuada por dois motores independentes (motor x e motor y).

```
{* variáveis locais *}
var posição_destino_x: real
      {armazena a posição destino da mesa x}

var posição_destino_y: real
      {armazena a posição destino da mesa y}

var velocidade_x: real
      {armazena a velocidade de deslocamento da mesa x}

var velocidade_y: real
      {armazena a velocidade de deslocamento da mesa y}

var fim_x: boolean
      {indica se a mesa x terminou seu deslocamento ou se ocorreu algum erro}

var fim_y : boolean
      {indica se a mesa y terminou seu deslocamento ou se ocorreu algum erro}

var x_está_movendo: boolean
```

```
      {indica se a mesa x está em movimento}

var y_está_movendo:  boolean
      {indica se a mesa y esta em movimento}

início
      caso Cmdo_xy.código seja
      INTERROMPE:
            se x_esta_movendo
            então
                  Envia Cmdo_Mesa_x com INTERROMPE
                  x_esta_movendo = false
            se y_esta_movendo
            então
                  Envia Cmdo_Mesa_y com INTERROMPE
                  y_esta_movendo = false
      MOVE:
            se Cmdo_xy.posição_destino_x <> 0
            então
            início {#2}
                  armazena Cmdo_xy.posicao_destino_x em posicao_destino_x
                  armazena Cmdo_xy.velocidade_x em velocidade_x
                  x_esta_movendo = true
            fim {#2}
            se Cmdo_xy. posição_destino_y <> 0
            então
            início {#2}
                  armazena Cmdo_xy. posição_destino_y em posição_destino_y
                  armazena Cmdo_xy.velocidade_y em velocidade_y
                  y_esta_movendo = true
            fim {#2}
            se x_esta_movendo
            então
                  Envia Cmdo_Mesa_x com MOVE, posição_destino_x e velocidade_x
            se y_está_movendo
            então
                  Envia Cmdo_Mesa_y com MOVE, posição_destino_y e velocidade_y
                  Espera Fim_Mesa_x
            fim_x = Fim_Mesa
            x_esta_movendo = false
            Espera Fim_Mesa_y
            fim_y = Fim_Mesa
            y_esta_movendo = false
            se fim_x = OK e fim_y = OK
            então
                  Fim_xy.ok_mesa = OK
            senão
                  Fim_xy.ok_mesa = NOK
            fim
            Envia Fim_xy
      INFORMA:
            Envia Cmdo_mesa_x com INFORMA
            Envia Cmdo_mesa_y com
            Espera Fim_Mesa_x
            Fim_xy.x_mesa = Fim_Mesa
            Espera Fim_Mesa_y
            Fim_xy.y_mesa = Fim_Mesa
            Envia Fim_xy
      REPOUSO:
            Envia Cmdo_mesa_x com REPOUSO
            Envia Cmdo_mesa_y com REPOUSO
            Espera Fim_Mesa_x
                  Fim_xy.ok_mesa = Fim_Mesa.ok_mesa
                  Fim_xy.posicao.pos_x = Fim_Mesa.posicao
                  Espera Fim_Mesa_y
                  if Fim_xy.ok_mesa = verdadeiro
                        Fim_xy.ok_mesa = Fim_Mesa.ok_mesa
```

```
                    Fim_xy.posicao.pos_y = Fim_Mesa.posicao
                    Envia Fim_xy com Fim_Mesa
fim
```

G.1.3.2 Serviços Internos

Nenhuma.

G.1.3.3 Serviços Requisitados

- Movimentar a Mesa em uma Direção
Saída : Cmdo_mesa
Entrada : Fim_mesa

Requisitado para mover uma mesa em uma direção (x ou y) e para pedir informações sobre a posição atual da mesa. Retorna um status de sucesso (OK) ou insucesso (NOK), quando do pedido de movimentação, e retorna a posição da mesa, quando do pedido de informação.

G.1.4 Descrição Detalhada dos Dados

G.1.4.1 Definições de Tipos

tipo_posição_mesa = record
    begin
        pos_x : real
        pos_y: real
    end

G.1.4.2 Estruturas de Dados Encapsuladas

Nenhuma.

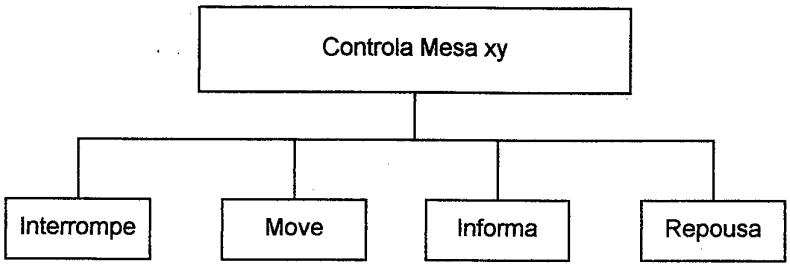
G.1.4.3 Estrutura das Mensagens

```
var Cmdo_xy : record
    begin
      código : {INTERROMPE/MOVE/INFORMA/REPOUSO}
      case código
      MOVE : record
      begin
            velocidade: tipo_velocidade
            posição_destino: tipo_posição_mesa
            {contém para posições x e y o deslocamento a ser empreendido à
            mesa}
      end
    end
{modela um pedido de movimentação da mesa nas direções x e/ou y}

var Fim_xy : record
    begin
            ok_mesa: boolean
            posição: tipo_posição_mesa
    end
{modela a respota a um pedido de movimentação da mesa nas direções x e/ou y e a
um pedido de informação da posição da mesa}

var Cmdo_Mesa : record
    begin
            código : {INTERROMPE/MOVE/INFORMA/REPOUSO}
            caso código seja
            MOVE : record
            begin
```

```
                velocidade : real
                posição_destino : real
            end
        end
{modela um pedido de movimentação da mesa em uma direção (linearmente)}

var Fim_Mesa :  record
    begin
            ok_mesa: boolean
            posição: real
    end
{modela a resposta a um pedido de movimentação da mesa ou informação sobre a
posição da mesa}
```

G.2 Design do Subsistema Básico

G.2.1 Árvore de módulos



G.2.2 Objetivos dos Módulos

- **Interrompe** - Pedir serviços de interrupção de movimentação para as direções x e y, caso necessário.
- **Move** - Pedir serviços de movimentação de mesa nas direções x e y. Esperar até o fim da movimentação e retornar condição de sucesso ou insucesso.
- **Informa** - Pedir serviços de informação de estado às encarnações x e y de S3.3. Esperar por ambas as respostas e retorná-las como resposta.
- **Repousa** - Pedir serviços de movimentação de mesa para a posição de repouso, nas direções x e y. Esperar até o fim da movimentação e retornar condição de sucesso ou insucesso.

G.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S3.2

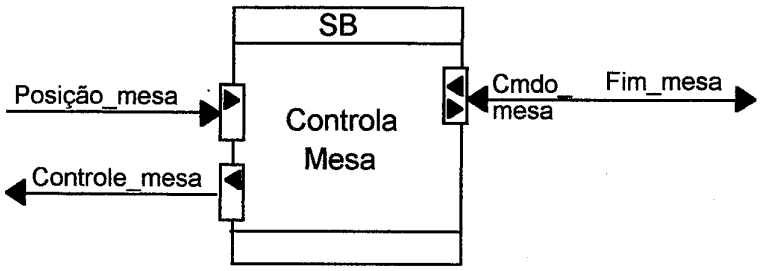
H. Subsistema Básico - Controla Mesa

H.1 Especificação

H.1.1 Descrição

O objetivo deste subsistema é controlar uma mesa linear, movendo-a para uma posição dada, e com uma velocidade específica. O subsistema deverá tentar corrigir qualquer tipo de desvio na movimentação que ocorra na mesa a ele ligada, indicando um erro quando não for possível a correção.

H.1.2 Diagrama de Contexto



H.1.3 Descrição dos Serviços

H.1.3.1 Serviços Externos

- Movimentar Mesa em uma Direção
Entradas : Cmdo_Mesa
Saídas : Fim_Mesa

Este serviço efetua a movimentação da mesa em uma direção (x ou y), recebendo um comando de movimentação (i.e. um deslocamento) e retornando um status de OK ou NOK. O serviço também deverá efetuar um acompanhamento para saber se houve algum desvio na trajetória, corrigindo-o, quando possível.

```
{* variáveis locais *}
var pos_estimada : real
{armazena a posição estimada atual da mesa}

var pos_mesa_atual : real
{armazena a posição atual medida da mesa}

var pos_inicial : real
{armazena a posição da mesa nates de inicia movimento}

var mesa_estado: { NOK/ OK }

var count_nok: integer
{armazena o número de vezes que a mesa sai da posição estimada}

var sit_mesa:{ PARADA/ MOVENDO/ REPOUSO/ FALHA }
```

```
início
      sit_mesa = PARADA
      Espera Cmdo_mesa ou
      Espera Δt
      { intervalo necessário a verificação de correção da movimentação }
      se (sit_mesa = MOVENDO ou sit_mesa = REPOUSO)
      então
            posição_mesa_atual = posição_mesa_atual + deslocamento
            verifica_mov_mesa (sit_mesa)
      fim
      caso sit_mesa seja
            PARADA:
                  Envia Fim_mesa com OK
            FALHA:
                  Envia Fim_mesa com NOK
            fim
      senão {saiu por comando}
            Caso Cmdo_mesa.código seja
            MOVE:
                  calcula tempo para mesa atingir Cmdo_mesa.posição desejada com
                  Cmdo_mesa.velocidade
                  Envia Controle_mesa
                  sit_mesa = MOVENDO
            REPOUSO:
                  Envia Controle_mesa com velocidade máxima e tempo negativo
                  { atinge posição não com tempo zero e sim com deslocamento 0 }
                  sit_mesa = REPOUSO
            PARA:
                  Envia controle_mesa com velocidade zero
                  sit_mesa = PARADA
            INFORMA:
                  Envia Fim_mesa com pos_mesa_atual
      fim
fim

verifica_mov_mesa (sit_mesa)
início
      mesa_estado = OK
      le Posição_mesa com deslocamento
      se deslocamento = 0
      então
            { atingiu posiçao de repouso }
            se sit_mesa = repouso
            então
                  { terminou movimento }
                  sit_mesa = PARADA
                  retorna
            senão
                  { envia mensagem para informar que posição destino não pode
                  ser alcançada }
                  sit_mesa = PARADA
                  retorna
            fim
      senão
            se pos_mesa_atual = fim ± faixa
            então
                  sit_mesa = PARADA
                  retorna
            senão
                  se módulo(pos_mesa_atual - pos_estimada) > faixa {* fora da
                  faixa *}
                  então
                        se mesa_estado = NOK
                        então
                              incrementa count_nok
                              se count_nok > num_máximo
                              então
```

```
                                    sit_mesa = FALHA
                                    retorna
                                    Envia Controle_mesa com velocidade zero {*
                                    parar mesa *}
                              senão
                                    recalcula tempo e velocidade
                                    Envia novo Controle_mesa
                                    se mesa_estado = OK
                                    então
                                          {* inicia contagem para erro_mesa *}
                                          mesa_estado = NOK
                                          count_nok = 0
                              fim
                        senão { dentro da faixa }
                              se mesa_estado = NOK
                              então
                                    mesa_estado = OK
                        fim
                  fim
            fim
fim
```

H.1.3.2 Serviços Internos

- Integrar Deslocamentos em uma direção efetuados pela Mesa continuamente no tempo calculando uma posição estimada
Entradas : Posição_mesa
Saídas : nenhuma

Periodicamente, o sistema lê o valor de deslocamento da mesa, integrando os valores lidos.

```
início
      lê Posição_Mesa
      integra Posição_Mesa
      Calcula posição atual (em função da posição anterior + deslocamento)
fim

*Obs: integra = armazena dados e determina o comportamento da variável no tempo
      Considera sentido do deslocamento.
```

H.1.3.3 Serviços Requisitados

Nenhum.

H.1.4 Descrição Detalhada dos Dados

H.1.4.1 Definições de Tipos

Nenhuma.

H.1.4.2 Estruturas de Dados Encapsuladas

Nenhuma.

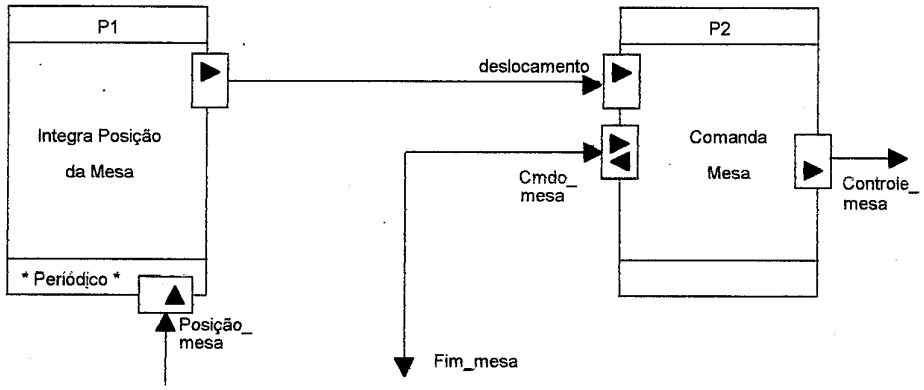
H.1.4.3 Estrutura das Mensagens

```
var Cmdo_Mesa : record
    begin
            código : {INTERROMPE / MOVE /INFORMA / REPOUSO }
            caso código seja
                  velocidade : real
                  posição_destino : real
            end
```

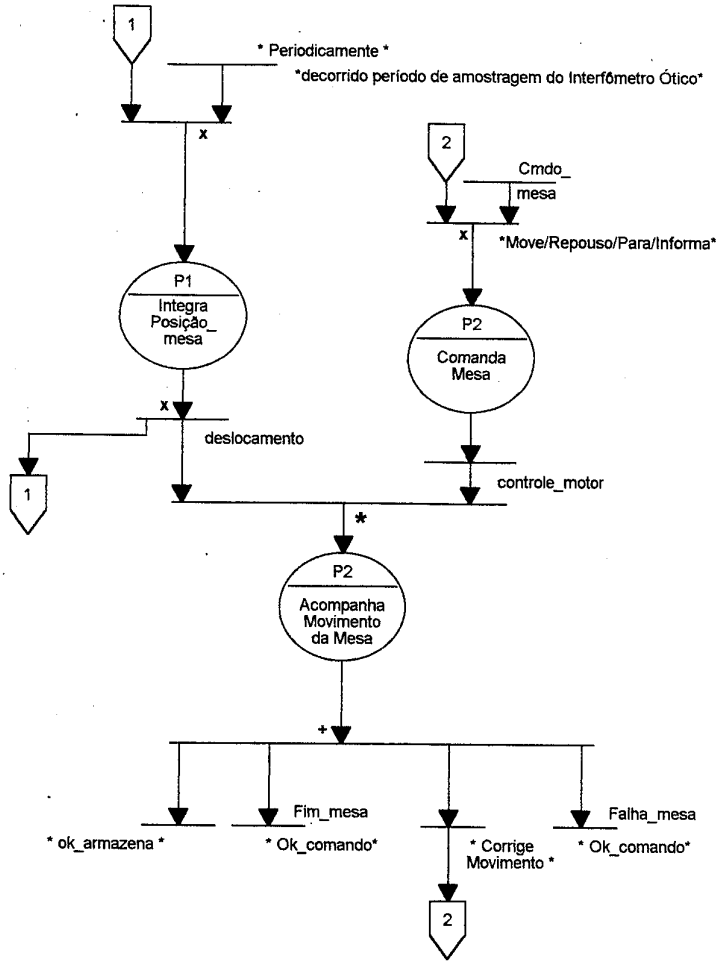
```
    end
{modela um pedido de movimentação da mesa em uma direção (linearmente)}

var Fim_Mesa :  record
    begin
            caso seja
                  ok_mesa: boolean
                  posição: real
    end
{modela a resposta a um pedido de movimentação da mesa ou informação sobre a
posição da mesa}

var Posição_Mesa: (*)
{modela a informação, proveniente dos sensores, sobre o deslocamento da mesa}

var Controle_Mesa: (*)
```

*Obs: O tipo das variáveis Posição_Mesa e Controle_Mesa dependem totalmente do hardware, não estando ainda perfeitamente definido.

H.2 Design do Subsistema Básico



H.2.1 Diagrama de Comportamento



H.2.2 Descrição dos Processos

P1- Integra Posição da Mesa

```
início
      lê Posição_Mesa
      integra Posição_Mesa e armazena em pos_mesa_atual
      Envia deslocamento
      { deslocamento da mesa efetuado no período }
fim
```

P2-Comanda Mesa

```
{* variáveis locais *}
var pos_estimada : real
{armazena a posição estimada atual da mesa}

var pos_mesa_atual : real
{armazena a posição atual medida da mesa}

var pos_inicial : real
{armazena a posição da mesa nates de inicia movimento}
```

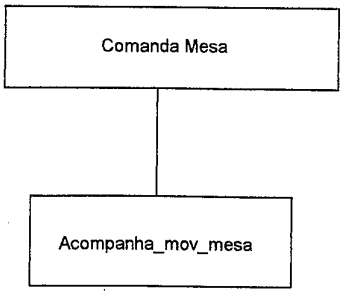
```
var mesa_estado: { NOK/ OK }

var count_nok: integer
{armazena o número de vezes que a mesa sai da posição estimada}

var sit_mesa:{ PARADA/ MOVENDO/ REPOUSO/ FALHA }

início
      sit_mesa = PARADA
      Espera Cmdo_mesa ou
      Espera Δt
      { intervalo necessário a verificação de correção da movimentação }
      se (sit_mesa = MOVENDO ou sit_mesa = REPOUSO)
      então
            consulta deslocamento
            se existir
            então
                  pos_mesa_atual = pos_mesa_atual + deslocamento
            fim
            verifica_mov_mesa (sit_mesa)
      fim
      caso sit_mesa seja
            PARADA:
                  Envia Fim_mesa com OK
            FALHA:
                  Envia Fim_mesa com NOK
            fim
      senão {saiu por comando}
            Caso Cmdo_mesa.código seja
            MOVE:
                  calcula tempo para mesa atingir Cmdo_mesa.posição desejada com
                  Cmdo_mesa.velocidade
                  Envia Controle_mesa
                  sit_mesa = MOVENDO
            REPOUSO:
                  Envia Controle_mesa com velocidade máxima e tempo negativo
                  { atinge posição não com tempo zero e sim com deslocamento 0 }
                  sit_mesa = REPOUSO
            PARA:
                  Envia controle_mesa com velocidade zero
                  sit_mesa = PARADA
            INFORMA:
                  Envia Fim_mesa com pos_mesa_atual
      fim
fim
```

OBS: O procedimento:

- verifica_mov_mesa (sit_mesa)

é igual ao definidos na especificação do serviço
OBS: A necessidade de Cmdo_mesa.código = ARMAZENA surgiu no design do SB. A alteração do tipo de dado da porta (Cmdo_mesa e Fim_mesa) ficou refletida na Estrutura das Mensagens (ítem 3) já que altera a interface do SB.

Diagrama Hierárquica de Módulos



Objetivos dos Módulos

- **Acompanha_mov_mesa** - Verificar a cada Δt se a mesa está se movimentando dentro da faixa. Corrigir a movimentação caso necessário. Se a movimentação da mesa ultrapassar a faixa desejada, indicar insucesso.

H.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S3.3

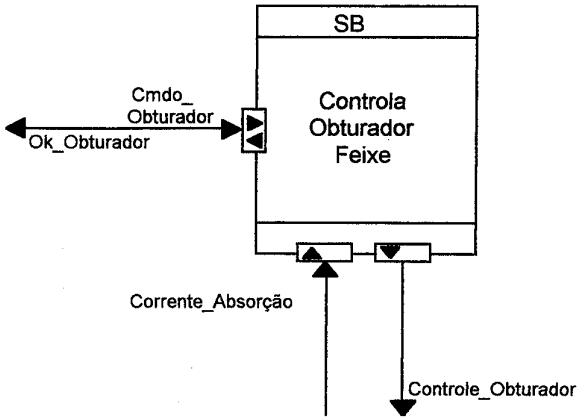
I. Subsistema Básico - Controla Obturador Feixe

I.1 Especificação

I.1.1 Descrição

O objetivo deste subsistema é controlar o obturador de feixe. Ele fornece serviços de abertura/fechamento do obturador. O subsistema deverá, também, indicar qualquer tipo de falha no obturador.

I.1.2 Diagrama de Contexto



I.1.3 Descrição dos Serviços

I.1.3.1 Serviços Externos

- Controlar Obturador Feixe
Entradas: **Cmdo_Obturador**
Saídas: **Ok_Obturador**

Este serviço efetua a abertura/fechamento do obturador de feixe, conforme comando de entrada. Acompanha o estado do obturador para informar a ocorrência de alguma falha.

```
início
Enquanto Verdadeiro faça
      Espera Cmdo_Obturador
            caso Cmdo_Obturador seja
            ABRE:
                  Envia Controle_Obturador com ABRE
                  inicia contagem de tempo necessário para estabilização
                  Ok_Obturador.tipo = ABERTO
                  decorrido tempo de resposta do Obturador_Feixe
                  então
                  {verifica estado do obturador }
                  se corrente_absorção_atual = VALOR_ABERTO
                  então
                        estado = ABERTO
                        Ok_Obturador.Ok = TRUE
                  senão
                        Ok_Obturador.Ok = FALSE
```

```
            FECHA:
                  Envia Controle_Obturador com FECHA
                  inicia contagem de tempo necessário para estabilização
                  estado_cmdo = FECHADO
                  Ok_Obturador.tipo = FECHADO
                  decorrido tempo de resposta do Obturador_Feixe
                  então
                  {verifica estado do obturador }
                  se corrente_absorção_atual = VALOR_FECHADO
                  então
                        estado = FECHADO
                        Ok_Obturador.Ok = TRUE
                  senão
                        Ok_Obturador.Ok = FALSE


            INFORMA:
                  Ok_Obturador.tipo = INFORMA
                  Ok_Obturador.info.corrente_absorção = corrente_absorção_atual

            fim

            envia Ok_Obturador
fim
```

I.1.3.2 Serviços Internos

- Integrar Corrente de Absorção continuamente no tempo
Entradas : Corrente_Absorção
Saídas : nenhuma

Periodicamente, o sistema lê o valor da corrente de absorção, integrando os valores lidos.

```
início
      lê Corrente_Absorção
      integra Corrente_Absorção e armazena em corrente_absorção_atual
fim

*Obs: integra = armazena dados e determina o comportamento da variável no tempo
```

I.1.3.3 Serviços Requisitados

Nenhum.

I.1.4 Descrição Detalhada dos Dados

I.1.4.1 Definições de Tipos

```
type tipo_info_obturador = record
      begin
            corrente_absorção: real;
      end;
{define informação sobre o obturador do feixe}
```

I.1.4.2 Estruturas de Dados Encapsuladas

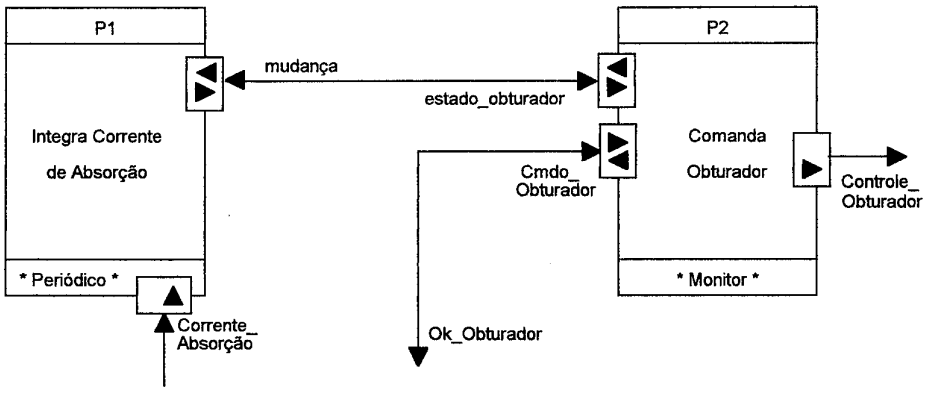
```
var info_obturador : tipo_info_obturador
{armazena o valor atual da corrente de absorção}

var estado: { ABERTO / FECHADO }
{armazena o estado atual do obturador, medido pela corrente de absorção}
```

I.1.4.3 Estrutura das Mensagens

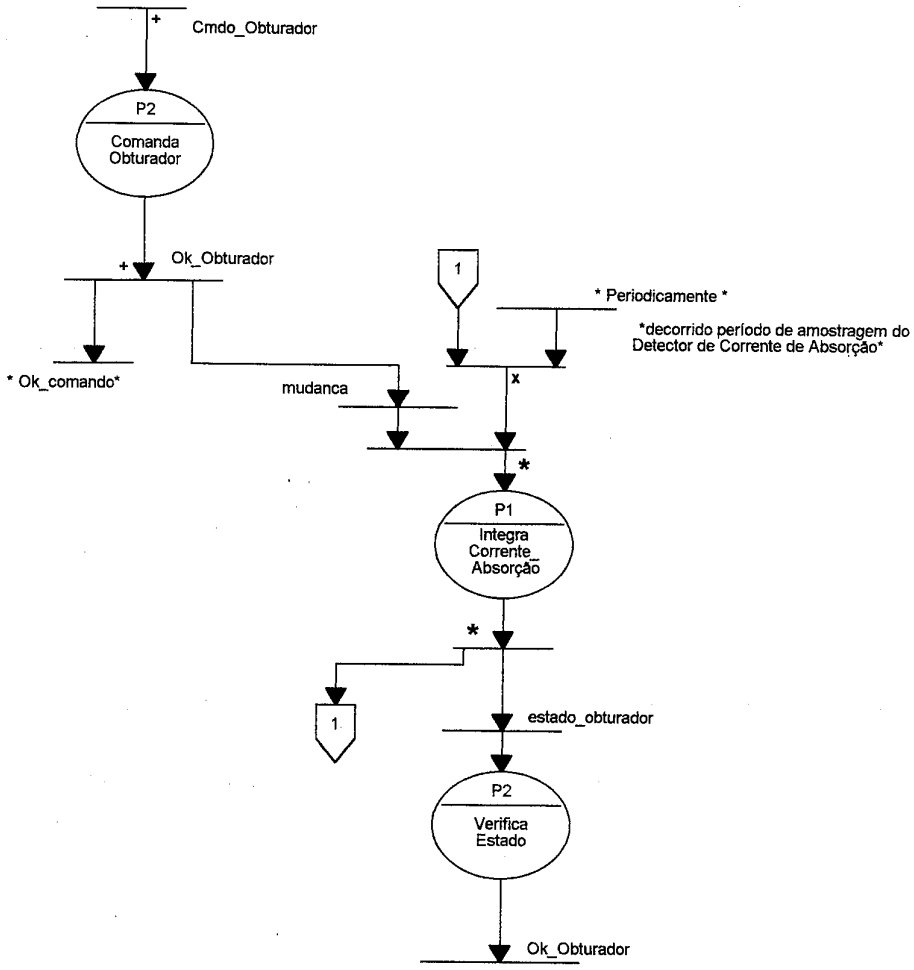
```
var Cmdo_Obturador : {ABRE / FECHA / INFORMA}
{modela um pedido de abertura/fechamento do obturador de feixe de elétrons}

var Ok_Obturador: record
    begin
      Ok: boolean
      info: tipo_info_obturador
    end
{modela a resposta a um pedido de abertura/fechamento do obturador de feixe de
elétrons e a um pedido de informação sobre a corrente de absorção

var Corrente_Absorção : (*)
{modela a informação, proveniente de sensores, sobre o estado (aberto/fechado)
do obturador}

var Controle_Obturador : (*)
{modela a informação de controle para o sensor contendo comando de obturador
aberto/fechado}


*Obs: O tipo da variável Corrente_Absorção depende totalmente do hardware, não
estando ainda perfeitamente definido.
```

I.2 Design do Subsistema Básico

I.2.1 Diagrama de Estrutura



I.2.2 Diagrama de Comportamento



I.2.3 Descrição dos Processos

- P1 - Integra Corrente Absorção
Entradas : Corrente_Absorção
Saídas : nenhuma
Serviços Requisitados : Cmdo_Obturador com armazena e resposta associada, ok_armazena
Objetivo :

Ler da interface com o detector de corrente de absorção o valor da corrente; integrar esses valores segundo algoritmo que considere a periodicidade dos valores lidos.

```
int cont = 0
boolean mudando = false
início
      lê Corrente_Absorção
      cont = cont +1
      integra Corrente_Absorção e atualiza estado _obturador
      se mudando e cont > (tempo de estabilizacao do obturador / tempo de
         amostragem)
         envia estado_obturador
         cont = 0
         mudando = false
```

```
        fim

        se mudando = false
              consulta se existe pedido de serviço em mudança
              caso mudança
              MUDA:
                    cont = 0
                    mudando = true
              INFORMA:
                    envia estado_obturador
        fim
       { enquanto esta mudando nao informa valor transitorio }
fim
```

- P2 - Comanda_Obturador
Entradas: cmdo_info_mic
Saídas: ok_cmdo_mic
Serviços Requisitados: nenhum
Objetivo:

Comandar a abertura/fechamento do obturador e informar quando requisitado o valor da corrente de absorção.

```
início
   estado_cmdo = VAZIO
   Enquanto Verdadeiro faça
      espera Cmdo_Obturador
            caso Cmdo_Obturador.tipo seja
            ABRE:
                  pede serviço Controle_Obturador com ABRE
                  inicia contagem de tempo necessário para estabilização
                  Ok_Obturador.tipo = ABERTO
                  envia mudança com MUDA
                  espera estado_obturador
                  se estado_obturador = VALOR_ABERTO
                  então
                        estado = ABERTO
                        Ok_Obturador.Ok = TRUE
                  senão
                        Ok_Obturador.Ok = FALSE

            FECHA:
                  pede serviço Controle_Obturador com FECHA
                  inicia contagem de tempo necessário para estabilização
                  Ok_Obturador.tipo = FECHADO
                  envia mudança com MUDA
                  espera estado_obturador
                  se estado_obturador = VALOR_FECHADO
                  então
                        estado = FECHADO
                        Ok_Obturador.Ok = TRUE
                  senão
                        Ok_Obturador.Ok = FALSE


            INFORMA:
                  Ok_Obturador.tipo = INFORMA
                  Ok_Obturador.info.corrente_absorção = corrente_absorção_atual
                  envia mudança com INFORMA
                  espera estado_obturador

            fim
            envia Ok_Obturador
fim
```

I.2.4 Descrição dos Dados Introduzidos

var mudança = { MUDA, INFORMA }

var estado_obturador : real
{ contem valor da corrente de absorcao }

I.3 Sistemas onde é empregado

1. COMONLIFE - Subsistema Básico S3.4