UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SILEX
Sistema para a Integração de Ferramentas de Projeto de Circuitos Integrados

por

Gilberto Fernandes Marchioro

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Altamiro Amadeu Suzim
Orientador

Porto Alegre, Abril de 1992.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Marchioro, Gilberto Fernandes

SILEX - Sistema para a Integração de Ferramentas de Projeto de Circuitos Integrados / Gilberto Fernandes Marchioro.—Porto Alegre: CPGCC da UFRGS, 1992.

150 p.: il.

Dissertação (mestrado)—Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1992. Orientador: Suzim, Altamiro Amadeu

Dissertação: Microeletrônica
Ferramentas de CAD, Framework, Ambientes Integrados, Projeto VLSI, PAC para a Microeletrônica

AGRADECIMENTO

Agradeço aos orgãos financiadores CNPq, CAPES, PROPESP e FAPERGS por patrocinarem meus estudos, por investir na educação e pesquisa, pelo incentivo ao desenvolvimento de projetos e pela liberdade no trabalho, permitindo a minha dedicação integral e exclusiva ao aperfeiçoamento.

Agradeço ao orientador, professor e amigo Altamiro Amadeu Suzim pelo exemplo de dedicação à pesquisa, por mostrar de forma paciente o caminho do dever e por provar que a inteligência supera qualquer dificuldade. Agradeço a confiança em mim depositada, por colocar à disposição e oferecer, sem restrições, toda a sabedoria e conhecimentos adquiridos durante anos de estudos, pesquisas e experiências.

Agradeço ao grande amigo e "professor" Luigi Carro que desde o tempo do trabalho escravo (bolsista) se preocupou com minha futura formação profissional. O seu profissionalismo e humildade em se colocar a nível de colega, dando idéias e explicações sobre os mais variados assuntos. Nossas discussões informais e sua paciência em me iniciar no contexto da microeletrônica.

Agradeço ao professor Dante Barone por ter desde o início acreditado no trabalho que eu viria a desenvolver (espero não ter decepcionado), pela confiança e orientação no meu primeiro ano de pesquisa.

Agradeço ao professor Ricardo Reis por sempre estar disponível a discussões, pela forma com que coordenou o CPGCC, facilitando meu trabalho, e por incentivar a exposição dos fractais.

Agradeço ao professor Tiaraju Wagner pela amizade e por sempre estar solícito a ajudar nas mais variadas tarefas, dispondo seu tempo em longos bate-papos. Ao professor Sergio Bampi pelo exemplo de dedicação à pesquisa. Ao Professor Newton Faller pela forma com que me acolheu, durante meu período de aperfeiçomento, junto ao seu ambiente de trabalho.

Agradeço aos demais alunos do Curso de Pós-Graduação em Ciência da Computação pela amizade. Em especial: Carlos Eduardo, colega de carona nas inúmeras viagens à Porto Alegre, durante as quais muito aprendi; Claudia Batistela, que a anos divide a "baia" comigo; Fernando Osório pela amizade e por ajudar a expor nossas imagens fractais; Cesar Marcon pelo bate-papo informal; Alexandro por ajudar com a implementação do Editor Simbólico; Soto com o Gerador de Geradores; Hentz com o Editor de Máscaras; aos amigos Roberto, Flavio, Benhur, Paulo, Petry, Alex, Dinamérico, Pazzini, Moraes, Manuel, Santos, Laerte, Marcos e Fernando que de alguma forma influenciaram neste trabalho. E aos que vierem a trabalhar no projeto.

Agradeço ao pessoal da biblioteca pela eficiência e dedicação ao trabalho.

E agradeço àqueles que com amizade e carinho serviram para mim como exemplo de vida. A dedicação e amor foram a base para o meu crescimento e tornaram claros os caminhos a serem seguidos e os objetivos a serem alcançados.

- *recolhi tudo que me foi dado ...*
- *em mim há uma parte de todos ...*

À minha família pelo apoio e dedicação durante estes anos e aos amigos que não me deixaram estudar nos fins de semana.

"Cada vez mais a tecnologia se torna a grande parceira
de uma humanidade em busca de respostas
para tantos e tão complexos problemas.
Continuará, entretanto, em nossas mãos
a decisão do que fazer com ela"
Rudolf Höhn

SUMÁRIO

CONVENÇÕES DE NOTAÇÃO

- Texto escrito em TIMES ROMAN
- *Termos técnicos escritos em TIMES ITALIC*
- `Código fonte de programas escritos em COURIER`
- [ENTER] - caracteres do teclado escritos em letras maiúsculas e entre colchetes
- [opcional] - Termos opcionais escritos entre colchetes

LISTA DE FIGURAS

GLOSSÁRIO

- Aplicação - um programa ou conjunto de programas que rodam em uma máquina cliente.
- Apontador, ponteiro ou *pointer* - variável de memória que contém um endereço para outra região ou variável.
- *Benchmark* - conjunto de testes padronizados realizados para avaliar características de sistemas.
- *Button* - elemento gráfico lembrando um botão que permite a ativação de comandos pelo usuário.
- *CallBack Functions* - funções associadas aos elementos da interface gráfica que são ativadas quando o usuário realiza alguma interação.
- *Canvas* - janela de desenho de primitivas gráficas e textos. É o tipo mais básico de janela.
- Chamada de Procedimentos Remotos - permite que um processo execute uma chamada de função em outro processo. Este procedimento é executado como um função local ao chamador.
- Cliente - uma máquina ou processo que solicita (consome) recursos.
- *Cursor* - elemento gráfico de reduzidas dimensões que indica a região corrente de posicionamento no vídeo. O formato geralmente é de uma seta, podendo variar de acordo com a função em execução.
- *Design Framework* - é o esqueleto formado por ferramentas servidoras e normas nas quais as ferramentas para o projeto estão organizadas.
- Ferramenta - um simples programa executável capaz de realizar uma função de projeto específica.

- *Framework* - é uma coletânea de programas e interfaces que junto com o Sistema Operacional definem o cenário no qual as ferramentas são desenvolvidas, integradas e operadas.

- Ícone - região gráfica retangular (geralmente 64x64 *pixels*) que representa a janela base no estado "fechada".

- Meta arquivo - arquivo que contém informações sobre outros arquivos de dados do projeto.

- Metodologia - é a especificação de uma seqüencia de tarefas.

- *Mouse* - dispositivo de apontamento gráfico que permite a movimentação do cursor e seleção de objetos.

- *Panel* - janela que contém elementos com os quais o usuário interage, como botões, mensagens, *sliders.*

- Pipe - mecanismo que permite a transferência de dados entre processos, por um caminho dedicado.

- Processo - uma combinação específica de ferramentas que executam funções de projeto.

- Protocolo - conjunto de regras, formato de dados e convenções que regulam a transferência de dados entre os participantes da comunicação.

- *Scrollbar* - barra de movimentação que permite a alteração da região de exibição em janelas do tipo *canvas* e *panel.*

- Servidor - máquina ou processo que provê recursos para a rede ou para o sistema.

- *Slider* - elemento gráfico que permite a entrada de um valor entre um domínio definido.

- *Socket* - forma bidirecional de comunicação entre processos que utiliza a rede.

- *Socketpair* - canal bidirecional de comunicação entre dois processos, semelhante ao *pipe*.
- Tarefa - é uma abstração da função de projeto. Por exemplo: simulação, edição.
- *Toggle* - lista de elementos gráficos que podem assumir um entre dois valores (*true* ou *false*).
- Usuário - pessoa que utiliza uma máquina cliente de recursos.
- Versão - elemento de estado de um objeto. Serve para organizar as mudanças feitas no projeto através do tempo.
- *Widget* - elemento básico de interação com o sistema de janelas, como botões, menus e *sliders*.
- *Window* - zona retangular do dispositivo gráfico (janela) na qual os processos se comunicam com o usuário.

ABREVIATURAS

AF_UNIX - Domínio de comunicação do *socket* (*Address Format*)

ASIC - Circuito integrado de aplicação específica (*Application Specific Integrated Circuit*)

BD - Banco de Dados

CAD - Projeto auxiliado pelo computador (*Computer Aided Design*)

CFI - Iniciativa na definição de um ambiente de CAD (*CAD Framework Initiative*)

CPGCC - Curso de Pós-Graduação em Ciência da Computação

DBMS - Sistema para a gerência de bancos de dados (*Data Base Management System*)

DOS - Sistema operacional em disco (*Disk Operator System*)

EDA - Automação eletrônica do projeto (*Electronic Design Automation*)

EDIF - Formato para a troca de projetos eletrônicos (*Electronic Design Interchange Format*)

Fig. - Figura

GME - Grupo de MicroEletrônica

IPC - Comunicação entre processos (*Inter-Process Comunication*)

PAC - Projeto automatizado pelo computador (*Project Automatized by the Computer*)

PID - Identificador de processos (*Process Identificator*)

RISC - Computador com conjunto de intruções reduzido (*Reduced Instruction Set Computer*)

SO - Sistema Operacional (*Operating System*)

RPC - Chamada de procedimentos remotos (*Remote Procedure Call*)

Tab. - Tabela

TCP - Protocolo de transmissão da comunicação (*Transmission Control Protocol*)

UFRGS - Universidade Federal do Rio Grande do Sul

XDR - Representação externa dos dados (*eXternal Data Representation*)

RESUMO

SILEX é um ambiente aberto e integrado que busca auxiliar a concepção de CIs. O sistema é composto por ferramentas internas (servidoras de recursos) e ferramentas do usuário (clientes de recursos). O usuário interage com o sistema *SILEX* através de uma interface gráfica baseada em janelas, ativando os recursos de forma padronizada e consistente. Sendo um sistema de CAD, *SILEX* é formado por um conjunto de módulos (ferramentas) interdependentes. Cada módulo realiza a sua função e transmite seus resultados. O usuário torna-se cliente de um conjunto de processos que concorrentemente responde às suas requisições. A idéia básica é esconder do usuário os procedimentos que não estão diretamente ligados ao projeto, como: configuração e forma de interação do usuário com as ferramentas; formato, conversão e local de armazenamento dos dados.

A regularidade na utilização é um dos principais objetivo do sistema, tendo em vista as constantes mudanças na forma de integração e utilização das ferramentas. Novos algoritmos, quando disponíveis, são informados aos usuários e estes decidem da inclusão em seus ambientes de trabalho, não necessitando qualquer mudança de código.

O projetista de ferramentas é auxiliado no desenvolvimento e integração pois conta com um conjunto de rotinas, normas de codificação e serviços prestados. As rotinas permitem a integração das ferramentas ao ambiente, enquanto que as normas regulam a utilização dos recursos disponíveis. A utilização dos recursos dá-se pelo envio de requisições ao servidor do sistema.

Os dados gerados pela interação com as ferramentas estão ligados a um projeto, inicialmente definido e cadastrado. Estes são manipulados por uma ferramenta dedicada, que realiza a leitura, escrita e conversão, liberando as ferramentas do usuário destas tarefas. Centralizados, os dados tem controle de acesso, dependência e versão facilitados.

SILEX em sua implementação não se beneficia das facilidades adquiridas com a utilização de um *framework* comercial, visto que foi totalmente construido sobre uma plataforma *OpenWindows*. O objetivo é inicialmente prover soluções simplificadas e eficazes, que permitam a integração de um conjunto de ferramentas e, subseqüentemente, incrementar e expandir a fim de que o *SILEX* tenha todas as características desejadas e ainda nao alcançadas pelos *frameworks* reportados na bibliografia.

Palavras-chave: Ferramentas de CAD, *Framework*, Ambientes Integrados, Projeto VLSI, PAC para a Microeletrônica.

ABSTRACT

SILEX is an open and integrated system built up to aid the design of integrated circuits. The *SILEX* System is composed of internal resources and user tools (clients of the resources). The user has at his disposal a graphic interface based on the use of windows, activating tools in an uniform and consistent way. The *SILEX* CAD system is formed by a set of interdependent modules (tools), each one realizing certain function and transmitting data. The designer is client of a set of processes that answer his/her requests. The main idea of the project is to hide from the final user all tasks which are not directly related to the art of design, like format conversion, data storage and maintenance and user interaction with tools.

One of the goals of the system is the regularity in its use, for there is always the need to integrate new tools. The user can suply new algorithms that may be included in the working environment without any change in the *SILEX* code.

The system helps tool designers by suplying them with a set of routines, coding rules and resources. The set of routines allows integration of the tool with the system, while the coding rules normalize the use of the available resources.

All data generated by the user interaction with the available tools is linked to a Project, previously defined and cataloged. Data is then handled by a dedicated tool performing I/O, responsible for the reading, writing and converting of data among different tools, freeing User Tools from this task. By being centralized, Project Data are controlled regarding access, dependency and versioning.

SILEX is completely built on top of the OpenWindows environment. Its goal is to initially provide simple and efficient solutions that allow the integration of a set of tools. Next tasks will be the enhancement of the system so that *SILEX* acquires all desirable characteristics not yet reached or reported in the literature.

Key-words: CAD tools, Framework, Integrated Environment, VLSI Design, CAD for Microelectronics.

1 INTRODUÇÃO

Atualmente é arriscado fazer uma previsão sobre os futuros sistemas de CAD. Algumas questões fundamentais persistem, como:

1. Os futuros sistemas de CAD serão apenas uma extensão dos atuais ?
2. Serão estes sistemas mais centralizados no projeto ?
3. As indústrias de CAD se direcionarão à prestação de serviços ou ao desenvolvimento de ferramentas ?
4. O *framework* solucionará todos os problemas ?

Estas questões tem estimulado discussões entre projetistas de sistemas eletrônicos e fornecedores de CAD. O consenso a que se chega é que somente com a união de esforços e entendimentos, entre profissionais envolvidos na construção e na utilização de ferramentas, poderão ser construídos sistemas que realmente supram as necessidades dos projetistas. Sem o cumprimento das etapas de especificação e teste dos sistemas de CAD, fica visível a duplicação de esforços, a falta de padronização no método de trabalho e a produção de ferramentas com reduzida perspectiva de utilização e destinadas apenas à solução de problemas imediatos.

Uma proposta que vem facilitar o uso e integração de ferramentas é a utilização de um *framework*. Um *framework* é um ambiente integrado para projeto que agrega e suporta ferramentas para automação de projetos [BHA 90]. Um *framework* também inclui: um *software* que trata da visualização das ferramentas no terminal de vídeo (sistema de janelas); um *software* que gerencia a grande quantidade de dados utilizados em projetos de sistemas VLSI (base de dados [FOO 90]); um gerenciador de projeto que suporta a existência de grupos de projetistas em um projeto.

O conceito *framework* existe desde os anos 80 mas até o momento ainda não há uma definição universalmente aceita do que venha a ser ou quais os módulos o compõe. O primeiro *framework* comercial está disponível no mercado há apenas meia década, tendo sido desenvolvido pela Cadence Design System. Até agora outros tantos estão disponíveis ([CLA 88], [MAL 89], [VAN 89], [DAN 89]). Eles diferem tanto em suas características e capacidades que fica difícil compará-los.

A escolha de um *framework* para a construção de um ambiente de projeto envolve inúmeros testes. Esta tarefa de seleção de sistemas e fornecedores pode ser longa, e deve-se levar em consideração o investimento envolvido na compra deste *software*. Em [COR 91] está registrado o trabalho de seleção e teste para a escolha de um *framework*. Este trabalho envolveu 40 pesquisadores, por mais de um ano, na realização de *benchmarks*.

No que diz respeito ao armazenamento dos dados de projeto, a utilização de DBMS comerciais tem tido limitada aceitação junto aos projetistas de sistemas de CAD. A própria natureza dos dados de projeto, ligada a sua simplicidade e grande quantidade de informações, torna os bancos de dados comerciais extremamente lentos, custosos e ineficientes. Algumas formas de solucionar este problema tem sido:

- desenvolver um DBMS dedicado ao projeto (tarefa que exige grande volume de trabalho);
- alterar algum DBMS existente a fim de adicionar novas características e funções (requer grande conhecimento do DBMS utilizado);
- construir uma casca sobre o DBMS para compensar as deficiências (pode prejudicar o desempenho do sistema, uma vez que esta dificilmente utilizará todos os recursos disponíveis).
- desenvolver uma interface gerenciadora de dados (esta tem aceitação mais imediata, uma vez que é mais direcionada ao problema e pode gradativamente ser

expandida com novas funções).

Cada uma destas alternativas apresenta vantagens e desvantagens; a escolha depende das características do sistema de CAD a ser implementado.

SILEX surge, neste contexto, como um sistema para a integração de ferramentas de projeto de circuitos integrados. A idéia é tornar mais racional a utilização das ferramentas, tanto as desenvolvidas no Grupo de Microeletrônica (GME) da UFRGS, quanto as ferramentas adquiridas de outros centros de pesquisa. Com os algoritmos básicos implementados nas ferramentas, aliados ao sincronismo das operações e fluxo dos dados, obtém-se um ambiente de fácil utilização e atualização.

A figura 1.1 exibe os componentes do sistema *SILEX* , estes são o *framework* e as ferramentas de projeto. O *framework SILEX* é composto por quatro módulos servidores, que são ativados no início do funcionamento e permanecem dispondo seus recursos para o sistema. Estes módulos são: Controle de Processos, Interface Gráfica, Controle de Projetos e Interface com os Dados.

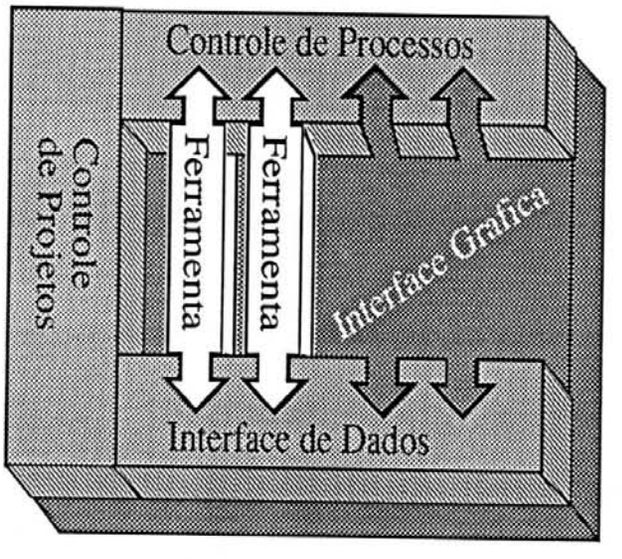

Figura 1.1: Sistema *SILEX*

1.1 Tendências

Qualquer sistema em sua fase inicial de concepção, para acompanhar o rápido crescimento da indústria de informática e eletrônica, deve levar em conta as tendências previstas para o futuro próximo. Torna-se inviável a execução de projetos com longo período de concepção. O conceito de *time-to-market*, descrito em [CAR 91] relaciona o tempo de projeto às necessidades de mercado. A demora na disponibilidade de um produto, facilmente poderá ser aproveitada por empresas concorrentes, gerando grandes perdas na comercialização.

As tendências a curto prazo previstas no contexto da concepção de circuitos são as seguintes:

- Ambientes: Os sistemas evoluirão de um conjunto de ferramentas isoladas para um ambiente de projeto integrado. Este ambiente agregará ferramentas de diferentes procedências, com grande volume de dados e que seguem uma metodologia de projeto.

- *Hardware*: O *hardware* destinado ao projeto será composto por estações de trabalho ligadas em rede. Estes equipamentos possuem grande capacidade de processamento, capacidade gráfica, sistema de janelas e intensiva execução multi-tarefa.

- Padronização: A preocupação com a padronização tem sido constante em todas as áreas da informática. O *hardware* permite que computadores e periféricos de fabricantes diferentes se comuniquem através de uma interface de comunicação padrão. O sistema operacional predominante é o UNIX, a linguagem de programação C, C++ (linguagem orientada a objetos) e *PostScript*, o protocolo de comunicação TCP-IP (*Transmission Control Protocol-Interconect Process*), o sistema de janelas Xwindow e a interface com o usuário padrão OSF/Motif ou OpenLook. Na descrição do projeto dominam as linguagems CIF, GDSII, EDIF e VHDL para a representação e troca de dados entre ferramentas.

- *Framework*: A fim de facilitar, com a padronização, o projeto de equipamentos, a adição, substituição e portabilidade de ferramentas, o gerenciamento de dados do projeto e a implementação de metodologias de projeto ([BHA 90]) foi criado o CFI (*CAD Framework Initiative* [FID 90]). O CFI é um fórum para a padronização, com a participação de empresas e do meio acadêmico, representando mais de 40 companhias envolvidas na produção de CIs. Para o CFI um framework é uma união de programas ou um conjunto crescente de serviços usados no desenvolvimento de sistemas de CAD unificados.

1.2 Motivação

A integração de ferramentas de CAD para a microeletrônica em um ambiente que permita ao projetista descrever e validar circuitos, tem sido objetivo antigo do CPGCC ([SUZ 86], [WAG 88], [MOR 91]). Esta integração era dificultada devido a inexistência de uma plataforma de *hardware* e *software* que suportasse multiprogramação, segurança dos dados, velocidade de processamento e grande capacidade de processamento. Com a chegada das estações de trabalho do tipo *SUN-SPARC*, estes requisitos tornaram-se disponíveis, permitindo uma mudança drástica na forma de implementação das ferramentas que até então utilizavam equipamentos *IBM-PC*.

Com os novos equipamentos o fator restritivo passou novamente a ser o *software* (ferramenta), gerando novos desafios aos programadores de aplicações. O desenvolvimento de ferramentas conta, neste novo ambiente, com recursos antes não disponíveis, como por exemplo: processamento concorrente, proteção de acesso á memória, grande capacidade de armazenagem de dados e alta velocidade de processamento. Sobre esta plataforma está sendo implementado o sistema *SILEX* , que objetiva prover o projetista de circuitos com um conjunto de ferramentas básicas devidamente integradas, assim como prover o programador com normas e funções para a inclusão de suas ferramentas. O *SILEX* é, desta forma, um sistema de CAD

encarregado de auxiliar o usuário no projeto de circuitos integrados.

O projeto de um circuito integrado é uma tarefa complexa para ser tratada em uma única operação e por uma única ferramenta, por este motivo necessita a subdivisão em níveis de abstração, hierarquia e descrição ([CAR 87]). Costuma-se dividir o problema em módulos de mais fácil tratamento. Desta forma, cada ferramenta integrada ao sistema fica responsável por solucionar um dado tipo de problema (adaptação), como mostra a figura 1.2 ([SUZ 86]). Ferramentas controladoras estão disponíveis a fim de facilitar a utilização das ferramentas e organizar a armazenagem de dados.

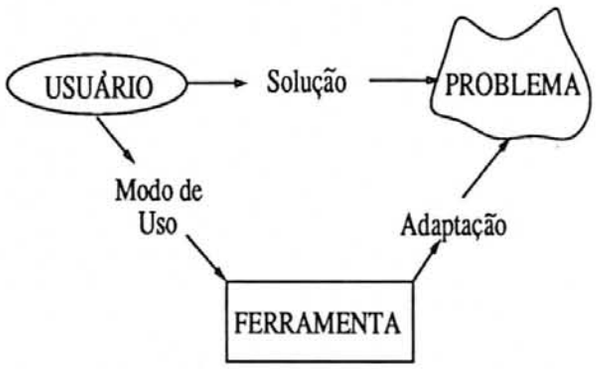

Figura 1.2: Auxílio ao usuário

1.3 Organização do Trabalho

Este trabalho relata o estudo e a experiência adquirida no desenvolvimento de parte do sistema *SILEX* . Conta com 7 capítulos que descrevem as características do sistema, necessidades do grupo de concepção, ambiente de implementação, sugestões e normas para a integração de novas ferramentas.

A introdução relata os motivos que conduziram ao desenvolvimento do *SILEX*. No 2º capítulo são estudados alguns *frameworks* e sistemas de CAD mais significativos para direcionar a tarefa de implementação, onde as características, vantagens e restrições são verificadas. O 3º capítulo é um tutorial sobre o *SILEX*,

abordando cada um dos seus componentes. No 4º capítulo são detalhados os recursos de *hardware* e *software* utilizados na implementação. Uma descrição sobre a utilização e filosofia do sistema de janelas é realizada. O capítulo 5º é destinado aos usuários interessados em integrar sua ferramenta ao sistema. São descritos comandos e normas de integração.

2 FRAMEWORK

2.1 Porque um *Framework* ?

Os fabricantes de CAD (*Computer Aided Design*) não estão mais conseguindo, separadamente, acompanhar o crescimento das necessidades dos sistemas projetados para suportar diferentes metodologias de projeto em diferentes organizações (ambientes de trabalho).

As ferramentas são normalmente adquiridadas de fornecedores especializados ou produzidas no próprio local de trabalho ([BRO 87]). Os sistemas de CAD para estarem atualizados com as novas tendências e mudanças devem permitir a inclusão de novas ferramentas e fontes de dados (bibliotecas), sem a necessidade de modificações no *framework*.

Da mesma forma que um sistema operacional oferece serviços para os programadores, um *framework* deve dispor de funções que permitam a anexação de diferentes ferramentas em um ambiente integrado. Estes sistemas contêm, basicamente, dois tipos diferentes de dados: estáticos e dinânicos. Os dados estáticos representam uma coleção de arquivos de dados. Estes podem ser gerenciados por um banco de dados ou simplesmente utilizar um sistema de arquivos. Os dados dinâmicos representam os programas de aplicação rodando sobre o sistema operacional.

Os sistemas de arquivos convencionais não fornecem uma segurança relativa ao armazenamento dos dados e um controle de versões da forma necessária em um ambiente de projeto. A estrutura de um sistema de arquivos suporta apenas a definição de hierarquias simples. São necessárias facilidades para suportar a hierarquia das estruturas de projetos utilizadas em diferentes metodologias. Deve-se dispor de funções que controlem as várias alterações nos dados (versões) e múltipla

equivalência na representação dos objetos (descrições de *layout* e esquemático representando o mesmo objeto).

Dois fatores básicos levam o projetista de ferramentas à necessidade de utilizar um *framework* para lidar com os problemas acima expostos. Os fatores são: 1) os sistemas operacionais convencionais não oferecem um repertório muito variado de funções para suportar o desenvolvimento de aplicações; 2) os sistemas de banco de dados comerciais dispõem de poucas facilidades, uma vez que são construídos para suportar aplicações comerciais.

Um *framework* é, desta forma, uma coletânea de programas e interfaces que junto com o sistema operacional definem o cenário no qual as ferramentas serão desenvolvidas, integradas e operadas.

2.2 Estudo de *frameworks* disponíveis

2.2.1 EDGE - *Framework*

EDGE é um *framework* desenvolvido pela SDA Systems. Este *framework* contém características básicas nas quais as ferramentas produzidas na SDA são construídas ([CAD 88]). A arquitetura empregada tenta solucionar os problemas envolvidos em um projeto, direcionando o usuário a utilizar os recursos disponíveis. Este é um sistema amplo, que engloba desde a interface com o usuário até a base de dados.

O sistema SDA é portável para diversas estações de trabalho. Possui uma arquitetura aberta, ou seja, ferramentas externas podem ser integradas. Dados de projeto nos formatos EDIF ou GDSII podem ser convertidos para o formato interno. A interface com o usuário é completa e dispõe de funções de alto nível, como: *zoom*, *panning*, menus e janelas. É um sistema configurável pelo usuário e trabalha com

processos concorrentes.

A interface provida pelo sistema dispõe dos seguintes elementos (*widgets*):

- Menus do tipo *pop-up context-sensitive*;
- Múltiplas janelas;
- Editor de texto padrão VI (*Visual text editor*);
- Janela de configuração, permite que se personalize a aparência do sistema;
- *Help on-line.*

Quando este sistema foi concebido não existiam comercialmente gerenciadores de janelas como *X window* ou *Sunview.* Todo este trabalho pesado de programação de interface foi necessário. As próximas evoluções do sistema prometem ser compatíveis com estes gerenciadores.

O sistema dispõe de uma base de dados unificada, ou seja, todas as informações armazenadas no projeto são compatíveis com as ferramentas. Isto significa que, com excessão da entrada e saída de dados do sistema, nenhuma outra conversão é necessária, pois todas as ferramentas trabalham utilizando uma linguagem global e interna, não sendo possíveis extensões.

A forma de identificar um objeto do projeto é dada por três campos:

- Nome do objeto (ULA, registrador, RAM, ...);
- Representação do projeto (*spice*, *layout*, *schematic*, ...);
- Nome da versão (*current*, *slow*, *tall*, ...).

A figura 2.1 exemplifica este formato.

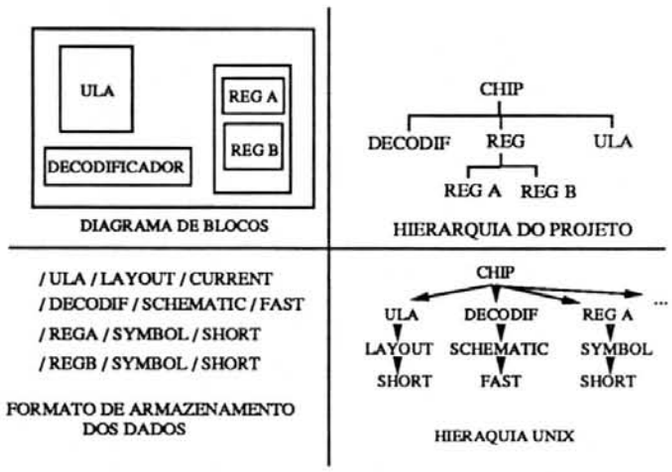

Figura 2.1: Representações de um circuito

O controle de versões disponível permite que se referencie instantaneamente a versão mais estável do projeto ou a que está em uso corrente. Apenas os objetos na versão corrente podem ser editados, outras versões podem apenas ser exibidas ou convertidas para corrente. As operações sobre versões ocorrem recursivamente sobre a hierarquia do circuito.

Um número definido de versões de um objeto, geradas automaticamente, pode ser especificado. Neste caso, após cada operação de escrita, as mudanças realizadas sobre o objeto são salvas. Versões permanentes sobre objetos podem ser criadas, quando se deseja realizar grandes mudanças no projeto. A entrada e saída de dados do projeto é realizada por conversores, que transformam estes dados do formato interno para EDIF, GDSII e vice-versa.

2.2.2 D-BUS - *Framework*

D-BUS é um *framework* desenvolvido pela DIGITAL EQUIPMENT CORPORATION para uso interno na automação de projetos ([VAN 89]), que tenta solucionar a complexidade técnica atingida pelos novos produtos. Os principais objetivos deste *framework* são: permitir a utilização de ferramentas sofisticadas de projeto, lidar com a complexidade e grande quantidade de dados e organizar o trabalho da

numerosa equipe de profissionais envolvida no desenvolvimento de cada produto.

O montante de dados ultrapassa facilmente a cifra de 10 *G bytes* em grandes projetos e o número de profissionais envolvidos pode chegar a uma centena, espalhados em diferentes centros de produção. Sobre este cenário foi desenvolvido o D-BUS, um sistema baseado em uma arquitetura aberta e estável, para a integração de ferramentas. Esta arquitetura possui as seguintes características:

1. Aceita um conjunto variado de implementações, evitando a inclusão de detalhes;
2. Novas funções podem ser adicionadas sem invalidar o trabalho de integração prévio;
3. Provê uma visão consistente dos objetos com os quais trabalha, através da linguagem DIN e;
4. Apresenta bom desempenho da arquitetura implementada.

É utilizada uma linguagem interpretada para a descrição de informações dos projetos, livre de formato e restrições de armazenamento. DIN (*D-BUS Information Notation*) permite a representação de informações com base em uma rede de interconexões (*network*) entre objetos e relações. Um controle sobre os dados de projeto é realizado por um banco de dados que suporta:

- Controle de versões, explicitamente identificado por um nome indicado pelo usuário. Objetos em qualquer nível hierárquico podem conter versões com informações relacionadas. É permitido o acesso a qualquer versão, não apenas à corrente;
- Configuração, relaciona os níveis hierárquicos dos objetos com as versões. Objetos de níveis superiores da hierarquia herdam atributos dos níveis inferiores;
- Restrição de acesso aos dados, é utilizado um meta-arquivo para conter informações sobre os dados armazenados (quem e quando realizou qual operação

sobre que objeto). Permite que vários usuários acessem dados concorrentemente mantendo a consistência da base de dados e;

- Protocolo definido de acesso ao BD.

Este *framework* permite a integração de ferramentas com o BD em três níveis:

1. Sem alteração das ferramentas. Envolve a utilização de uma ferramenta encapsuladora para acessar o BD. O encapsulador converte as requisições de entrada e saída das ferramentas em comandos da linguagem DIN. O BD é utilizado na carga e salvamento de arquivos.

2. Emula arquivos. Nesta configuração, o encapsulador é substituído pela alteração do código de entrada e saída da ferramenta. Não é alterado o algoritmo propriamente dito. O acesso aos arquivos e informações, para a alimentação da estrutura interna das ferramentas, é feito diretamente sobre o BD. Facilidades deste nível incluem o tratamento de conjunto de dados através de listas;

3. Mapeamento com o BD. Neste nível a estrutura de dados da ferramenta é tratada pelo BD, necessitando profundas mudanças nas ferramentas. O código de entrada e saída não é mais necessário, sendo substituído por chamadas ao BD. A ferramenta notifica diretamente o BD para obter o acesso à representação particular de dados. Esta forma de integração também torna possível o uso incremental de dados do projeto.

2.2.3 EDA - *Framework*

EDA é um *framework* desenvolvido pela EDA System Inc. . Este *framework* conta com um conjunto de funções que atuam sobre o sistema operacional e sobre o sistema de arquivos. O objetivo é criar uma plataforma de projeto sobre

a qual várias tecnologias e ferramentas de projeto possam atuar ([BRO 87]). O sistema é incremental, uma vez que ferramentas projetadas por diferentes fabricantes podem ser integradas. Para tanto são fornecidas facilidades e utilitários para auxiliar na tarefa de integração de novas ferramentas.

Este *framework* tem uma arquitetura independente de *hardware* e sistema operacional, podendo rodar em diferentes CPUs. As funções realizadas são basicamente o controle dos dados e das ferramentas, possuindo os seguintes módulos:

- Servidor de dados, é um servidor que contém objetos gerados pelo sistema e constantemente verifica as regras de utilização, como versão, configuração, proteção, composição e representação. Este componente é responsável pelo gerenciamento dos dados e interface com o usuário. Os elementos básicos do projeto são: esquemáticos, redes de conexão (*net-list*) e *layouts*.

  A figura 2.2 exemplifica a forma de armazenagem e operações sobre os dados.

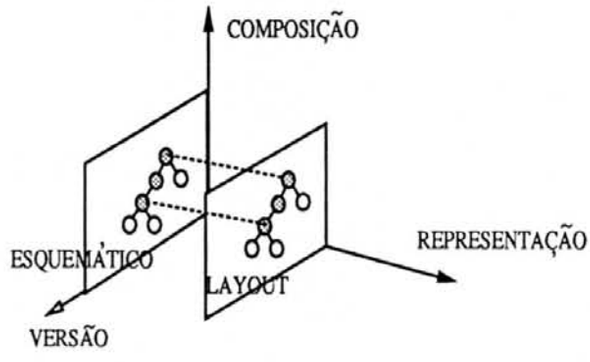

Figura 2.2: Representação de objetos em EDA

Cada objeto possui uma versão corrente e versões secundárias referenciadas por números. Cada versão contém informações adicionais sobre as últimas alterações nos dados. A composição está ligada à possibilidade de um objeto ser composto de outro objeto. Desta forma é criada a hierarquia de projeto. Os objetos podem conter múltiplas representações (esquemático, *netlist*). Algumas destas representações são criadas automaticamente, enquanto que outras apenas podem ser comparadas ou validadas. A dependência relaciona elementos

em diferentes representações.

- Biblioteca de serviços, é um conjunto de rotinas de acesso ao servidor na qual o usuário pode ligar sua ferramenta em tempo de execução. Estas rotinas podem ser extendidas e adaptadas pelo usuário para o seu contexto de utilização. As funções básicas apenas operam sobre arquivos e dados de projeto. Uma política de controle pode ser implementada neste nível.

- Zona de trabalho, é uma janela na qual o projetista interage com o sistema para a visualização e manipulação dos objetos. Esta operação é realizada sobre a interface gráfica ou através de comandos. Comandos existentes podem ser alterados ou novos podem ser criados via interface. Os objetos manipulados incluem: arquivos, diretórios, conjunto de comandos, máquinas. Quando é solicitada a execução de alguma operação sobre objetos, esta operação considera o tipo de objeto a ser manipulado.

2.2.4 FALCON - *Framework*

FALCON é o *framework* desenvolvido pela Mentor Graphics para o projeto concorrente ([MEN 92]). Este abrange vários campos da engenharia, como: elétrico, mecânico, simulação, projeto e documentação. A característica principal deste *framework* é a possibilidade de estimar efeitos de decisões do projeto, facilitando o processo de implementação.

A arquitetura do FALCON, como mostra a figura 2.3, é composta pelos seguintes módulos:

- Interface gráfica com o usuário comum a todas as ferramentas, no padrão Motif.
- Sistema de informação instantânea a partir da ferramenta BOLD, permitindo acesso ON-LINE a toda documentação (utiliza *hipertext*).

Figura 2.3: Módulos do *framework* FALCON

- Gerenciador de projetos utilizando o paradigma de linguagens orientadas a objetos (C++) para organizar os dados e ferramentas do projeto.
- Gerenciador de dados dispondo de um conjunto simples de funções e facilidades no controle de versões, disponível a todas as ferramentas.
- O Suporte a decisões (*decision suport*) ajuda o usuário a analisar as características e o comportamento do produto ainda durante a fase de projeto. O usuário pode monitorar restrições e violações, implementando rapidamente seu próprio método de previsão de como as suas decisões afetam as características finais do produto.

A integração de ferramentas ao FALCON é possível em três níveis, abaixo descritos:

1. No nível mais baixo de integração, a ferramenta é registrada no gerenciador de projetos e esta aparece na forma de ícone para os usuários. As interfaces com o usuário e com os dados permanecem inalteradas.
2. O segundo nível utiliza a interface gráfica disponível pelo *framework* para tornar a aparência e interação do usuário com a ferramenta mais regular. Está disponínel a linguagem AMPLE (*Advanced Multi-Purpose LanguagE*) para facilitar esta adaptação.
3. A completa integração utiliza todos os recursos disponíveis, como o gerente de dados, interface gráfica e gerente de projeto.

2.2.5 FRAMEWORK II

O Framework II foi desenvolvido pela Cadence com o objetivo de unir um grupo de ferramentas individuais em um sistema de projeto de fácil utilização, interação, organização lógica e aberto a extensões.

A fácil utilização do *framework* dá-se pelo fato de todas as ferramentas possuirem a mesma interface com o usuário, padrão Motif-X window. O Framework II é considerado integrado por eliminar as barreiras existentes entre as ferramentas. Desta forma, utilizar uma ferramenta torna-se tão simples quanto selecionar uma opção do menu ou mover o cursor entre janelas. A organização lógica permite que o usuário visualize e manipule os dados gerados pelas ferramentas com a visão de projeto, esquecendo-se da estrutura de diretórios ou da rede de interconexões entre os equipamentos. É especificada graficamente a ordem de utilização das ferramentas para que o usuário saiba onde está no processo de projeto. O *framework* é considerado aberto por conter todos os seus recusos disponíveis e documentados.

A figura 2.4 exibe os módulos que compõem o Framework II.

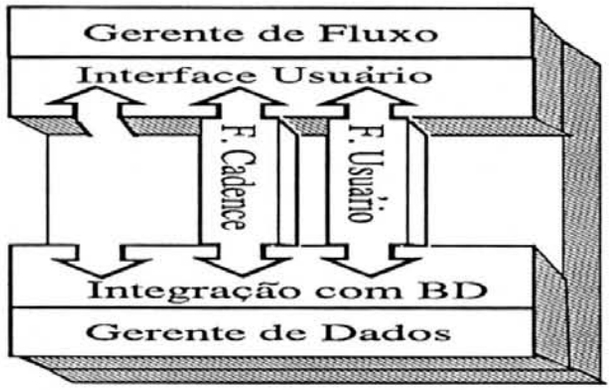

Figura 2.4: Módulos do Framework II

O gerente de fluxo de projeto exibe, no formato de "ferrovia", a seqüencia de passos que o usuário deve executar durante o projeto. Este fluxo automaticamente muda a cor dos objetos para indicar uma nova posição estágio da implementação. O gerente de fluxo pode simplificar e automatizar a utilização das ferramentas. O

usuário define a seqüencia de ferramentas a serem ativadas, seus parâmetros e a transferência de dados e o gerente realiza a operação automaticamente. Na base do *framework* está o banco de dados unificado. Todas as ferramentas possuem um mesmo modelo de dados, permitindo a direta transferência de informações e assegurando a integridade no projeto.

2.3 Estudo de Sistemas de CAD

Um sistema de CAD é uma coleção de ferramentas e de gerenciadores de dados de projetos. As ferramentas são destinadas a criar módulos e validar a sua correção. Os gerenciadores de dados são responsáveis pela estruturação dos módulos e por utilizar esta estrutura na tarefa de manter o projeto consistente.

Nesta seção é relatado um estudo sobre características de sistemas de CAD disponíveis. Este estudo objetiva selecionar características desejáveis e formar uma base para a concepção do *SILEX*. Conhecendo-se bem as características dos sistemas disponíveis, fica facilitada a tarefa de implementação de um novo sistema.

Os sistemas de CAD vão desde versões mais simples que rodam em sistemas operacionais monoprocessos e comunicação por linha de comando, até versões mais completas, com banco de dados, sistema de janelas próprio e comunicação mais complexa entre processos. Cada sistema tem suas características associadas ao seu ambiente de construção e utilização, desta forma deve-se estudar este contexto para avaliação. Por exemplo um esquema de troca de informações entre ferramentas em um SO monoprocesso (exemplo: DOS) geralmente requer mais trabalho de codificação do que em um SO multiprocessos (exemplo: UNIX).

Abaixo podemos visualizar as principais características dos sistemas, ligados ao seu ambiente de desenvolvimento.

- Sistemas fracamente integrados, rodando em computadores pessoais, sistema

operacional monoprocesso e interface gráfica específica;

- Sistemas integrados, rodando em estações de trabalho, sistema operacional multiprocessos, conexão em rede e interface gráfica genérica.

Os sistemas fracamente integrados, como mostra a figura 2.5, possuem um programa seqüenciador de tarefas. Este basicamente contém chamadas para a ativação de ferramentas do sistema. A preparação dos parâmetros de ativação para cada ferramenta, passados por linha de comando, é previamente realizada. Após a ativação, o processo seqüenciador cede lugar à nova ferramenta na memória. Um arquivo especial chamado *meta-arquivo* é utilizado para conter informações sobre o estado atual de utilização dos dados. Como único processo em funcionamento a ferramenta deve assumir a função de cliente e servidor de recursos, dispondo de todas as funções necessárias para o seu funcionamento.

Após o término de execução da operação, a ferramenta deve chamar o processo seqüenciador para este assumir a continuação do trabalho. O seqüenciador recebe informações, através do código de retorno sobre o sucesso ou não do processamento. O seqüenciador examina os dados gerados e se prepara para uma nova interação. Com este tipo de sistema dificilmente consegue-se trabalhar com o conceito de projetos, onde um grupo de projetistas estão envolvidos e o contexto dos dados é variável.

Os sistemas integrados não apenas auxiliam o projetista a tratar com a grande quantidade de dados do projeto, mas também são capazes de avaliar o estado e as operações realizadas sobre os mesmos. O usuário perde o acesso direto aos dados, pois não mais interessa o local ou formato de armazenamento. É fundamental a função de visualização do estado atual do projeto, módulos construídos, validados, simulações e dependências.

Para suportar as características acima citadas necessita-se da cooperação de vários processos do sistema, cada qual realizando sua função e mais um processo

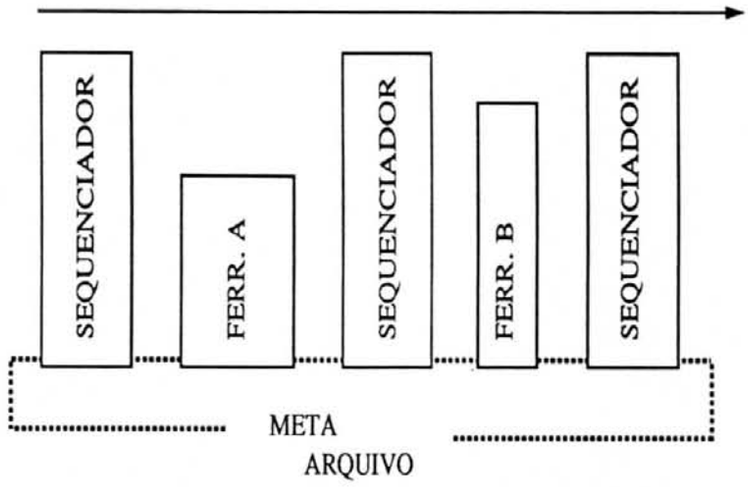

Figura 2.5: Sistemas fracamente integrados

controlador. A arquitetura básica é mostrada na figura 2.6. É utilizada alguma técnica de comunicação entre processos, tal como *pipes*, *sockets* ou RPC. A interface utiliza múltiplas janelas. Por exemplo: é possível visualizar a forma de onda de um simulador enquanto se edita o esquemático.

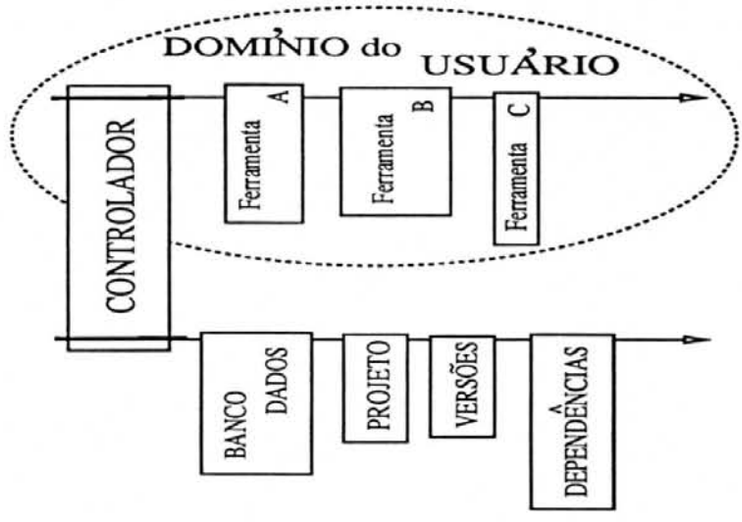

Figura 2.6: Sistemas integrados

2.3.1 Sistema Navigator

Navigator é um sistema de CAD desenvolvido pela PHILIPS para o projeto de circuitos integrados ([KOK 91]). O projeto foi iniciado em 1988 e utiliza

o *framework* EDA-DIGITAL (ver seção 2.2.3) para as tarefas de controle de projeto. O sistema utiliza um encapsulador em todas as suas ferramentas. Existe um controlador principal, como mostra a figura 2.7, que contém a interface gráfica do sistema.

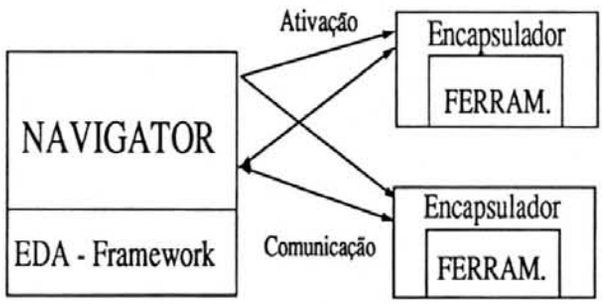

Figura 2.7: Arquitetura do sistema Navigator

O encapsulador é ativado pelo controlador como um processo filho (via comando `fork`). Após a ativação, a comunicação entre estes processos define os parâmetros de chamada da ferramenta encapsulada. A ferramenta é ativada por linha de comando contendo os parâmetros necessários. O usuário pode encerrar o processo controlador sem afetar o funcionamento das ferramentas.

Após o encerramento da execução da ferramenta o encapsulador verifica o código de retorno. Este valor indica se a execução foi com sucesso ou não. A conversão dos dados para o formato do banco de dados, se necessária, é realizada neste momento. Uma mensagem de término é enviada à interface do sistema, para que esta se atualize. Enquanto o processo principal está desativo, as mensagens são empilhadas para posterior processamento.

As variáveis que descrevem o estado de utilização do sistema são armazenadas e recuperadas de um arquivo. Desta forma, ao ativar o sistema, o usuário encontra a mesma configuração que deixou na última interação. Um arquivo de instalação é utilizado para gerenciar as variáveis do ambiente, como por exemplo os *path_names* das ferramentas e o conteúdo dos menus dinâmicos. Com isto consegue-se evitar a compilação do sistema após a alteração de algum parâmetro, como a inclusão de uma nova ferramenta.

A interface com o usuário é composta por três zonas:

1. Zona de Fluxo, é uma janela gráfica com retângulos representando as ferramentas e setas, representando o fluxo de dados entre as ferramentas. Este fluxograma combina as diversas ferramentas com os arquivos de entrada e saída. Também é mostrada a seqüência na qual as ferramentas devem ser utilizadas.

2. Zona de Menus, contém menus que permitem a seleção de opções das ferramentas, a criação ou edição de arquivos de dados e a visualização do estado do sistema. Todos os comandos do sistema são ativados por esta zona.

3. Zona de Visualização, realiza a procura, exibição e visualização dos dados de projeto contidos no banco de dados.

2.3.2 Sistema PACE

O sistema PACE foi desenvolvido no IST/INEST em Portugal ([SAR 91]). Este adota o modelo cliente-servidor indicado para o ambiente de implementação utilizado: estações de trabalho, sistema de janelas Xwindow, linguagem de programação C++ e sistema operacional UNIX. O que motivou o desenvolvimento deste sistema, como uma evolução do sistema IMAGE, foi o surgimento e afirmação de algumas normas e recursos, como GKS para a manipulação gráfica independente de terminal, linguagem orientada a objetos, conexão em rede e monitores *bitmaps* de alta resolução.

A arquitetura do sistema está centrada no servidor de projetos. Este coordena a interação com as ferramentas e mantém em memória central as informações necessárias para o funcionamento destas. O servidor fica encarregado de alimentar as ferramentas com dados e organizar o acesso ao banco de dados. É utilizado um protocolo de alto nível para a comunicação entre as ferramentas.

Na transmissão de dados é utilizado um modelo baseado em objetos para

abstrair detalhes necessários à comunicação. É utilizado um canal de comunicação baseado em *sockets*, para a comunicação das ferramentas com o servidor (Plib), como mostra a figura 2.8. Este formato de transmissão (*byte stream*) necessita a conversão dos objetos a serem transferidos em uma lista de *bytes* e, após a transmissão, a operação inversa de decodificação.

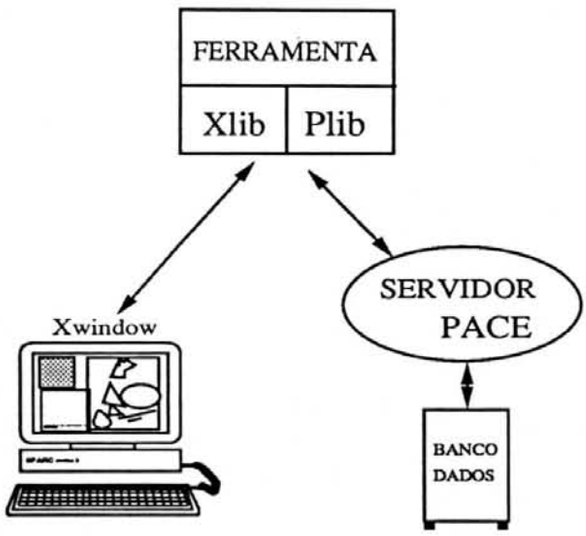

Figura 2.8: Arquitetura do sistema PACE

2.3.3 Sistema TENTOS

O sistema TENTOS foi desenvolvido no GME-UFRGS ([MOR 91]). Este é um gerenciador de ferramentas de PAC para a microeletrônica, dotado de uma interface gráfica responsável por controlar a execução dos diversos programas e manipular os diversos formatos de arquivos. O TENTOS roda em sistema operacional monoprocessos e equipamentos compatíveis com o padrão IBM-PC. É utilizado um programa sequenciador com a função de: ativar as ferramentas, gerenciar os diretórios e controlar o formato dos arquivos.

A interface gráfica do programa sequenciador utiliza o *mouse* e as ferramentas estão disponíveis em uma lista para a ativação. É disponível *Help on-line* para auxiliar na interação do usuário com o sistema TENTOS. Estão integradas 27

ferramentas que realizam edição, simulação, extração e verificação a nível de *layout.*

2.3.4 Sistema OLYMPUS

O sistema OLYMPUS foi desenvolvido na Universidade de Stanford para o projeto de circuitos digitais ([MIC 90]). Este conta com um conjunto de ferramentas integradas que auxiliam o projetista na concepção, e ferramentas automáticas de sintese multi-nível (lógica, estrutural e comportamental) e simulação.

Algumas características importantes do sistema OLYMPUS são citadas abaixo:

- O sistema utiliza uma linguagem textual chamada HardwareC para descrever o *hardware* orientado para a síntese. HardwareC é uma linguagem com sintaxe semelhante à linguagem C ([KER 86], mas tem sua própria semântica.
- A implementação obedece as restrições de *timing* e recursos especificados na descrição de alto nível.
- Suporta tanto a síntese automática, que converte a descrição em HardwareC diretamente para sua implementação lógica, quanto a síntese semi-automática, nesta o usuário acompanha o processo, podendo intervir e tirar decisões baseadas em resultados parciais.
- Suporta *technology mapping* que mapeia a representação lógica com os objetos pré-definidos na biblioteca.

Fazem parte do sistema ferramentas como o Hercules, Ariadne, Hebe, Mercury e Ceres, que realizam, respectivamente, síntese comportamental, simulação comportamental, síntese estrutural, síntese lógica e *technology mapping.* São utilizadas as linguagens intermediárias SIF (*Sequencing Intermediate Format*) que é uma linguagem de descrição comportamental do *hardware* baseada em grafos e a linguagem SLIF (*Structural/Logic Intermediate Format*) que descreve a implementação a nível lógico.

As ferramentas do sistema Olympus utilizam a mesma interface com o usuário. Esta interface é modelada diretamente sobre o sistema operacional UNIX, possuindo três níveis de interação com o usuário: noviço, avançado e manutenção.

2.4 Conclusão

A observação das características, composição e funcionamento de sistemas de CAD e *frameworks* permitiu a avaliação e um estudo crítico do sistema a ser implementado. Um conjunto de idéias e sugestões que ha algum tempo ([SUZ 86]) foram formalizadas, poderam ser consistidas, atualizadas e expandidas à nível de controle de projetos, formas de armazenamento de dados, técnicas de comunicação entre ferramentas e interface com o usuário.

SILEX não se beneficia das facilidades adquiridas com a utilização de um *framework* comercial, como alguns dos sistemas anteriormente descritos. O objetivo é inicialmente prover soluções simplificadas e eficazes, que permitam a integração de um conjunto de ferramentas e, subsequentemente, incrementar e expandir afim de o *SILEX* tenha todas as características de um *framework*. A utilização de um *framework* comercial simplificaria a tarefa de integração das ferramentas, mas não foi efetivada, não apenas pela indisponibilidade de um *framework* no momento do início do trabalho de dissertação, mas por acreditar-se que um esforço direcionado à implementação, consiente do contexto de utilização e total domínio do problema, poderia suprir esta necessidade.

3 SISTEMA *SILEX*

3.1 Arquitetura básica do sistema

SILEX, sendo um sistema de CAD, é formado por um conjunto de módulos (ferramentas) interdependentes. Cada módulo do sistema realiza a sua função e transmite seus resultados. O usuário torna-se cliente de um conjunto de processos que concorrentemente responde às suas requisições. A idéia básica é esconder do usuário os procedimentos que não estão diretamente ligados ao projeto. Quanto maior for a integração, mais o usuário pode concentrar-se no projeto, sendo auxiliado pelo trabalho de suporte realizado.

A arquitetura do sistema é esquematizada na figura 3.1, onde são exibidas as ferramentas servidoras e clientes de recursos. Esta arquitetura tenta resolver os problemas normalmente encontrados no projeto de CIs, tais como:

- os projetistas interagem com diversas ferramentas que produzem enorme quantidade de dados de projeto;
- dificuldade de saber em um dado momento o estado atual do projeto e;
- os projetistas gastam muito tempo resolvendo problemas não diretamente ligados ao projeto, como conversão de dados e configuração das ferramentas.

3.2 Ferramentas do usuário

Estas ferramentas possuem código fonte passível de alteração, permitindo sua adaptação aos recursos do *SILEX* . A quantidade de código a ser reescrita ou alterada é reduzida quando a ferramenta já obedece a uma modularização ou

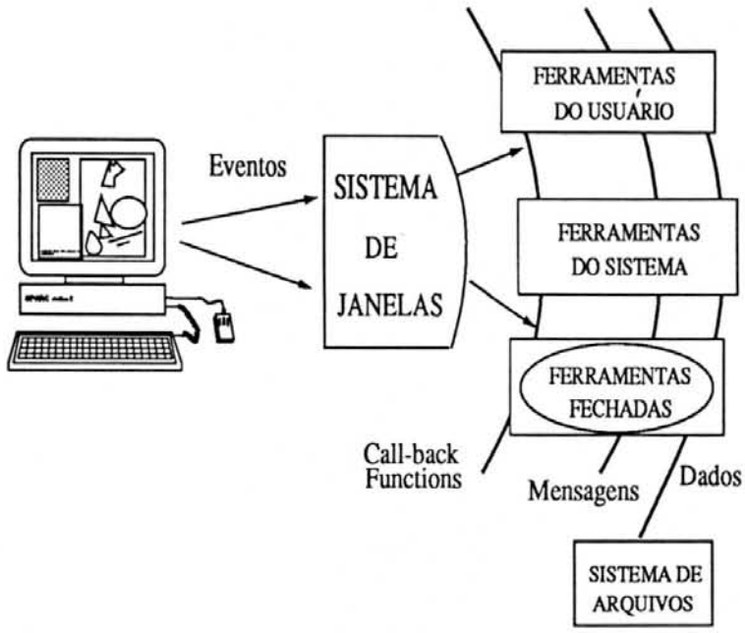

Figura 3.1: Arquitetura do *SILEX*

formalismo na escrita do código. Um trabalho maior de integração é necessário quando a ferramenta em questão está codificada para um ambiente diferente do utilizado pelo sistema.

Na integração de uma ferramenta, o usuário deve ter em mente quais os recursos disponíveis e algumas normas de construção do código. Os recursos disponíveis são os serviços prestados pelas ferramentas do sistema. Estes são ativados pelo envio e recebimento de mensagens. Por exemplo, enviando-se a mensagem "CARGA_ARQUIVO" à interface de dados, obtém-se o conteúdo deste no formato codificado, pronto para leitura.

As normas de codificação regulam a utilização destes recursos. Abaixo são descritas algumas destas normas:

- toda a operação de carga e salvamento de arquivos deve utilizar a interface de dados. Existem três formas de leitura (e escrita), estas são:
  - CARGA_ARQUIVO: executa a leitura do arquivo na interface de dados pelo *parser* apropriado. O *parser* é definido pela linguagem na qual o

arquivo está codificado, por exemplo CIF, LDS. É realizada uma conversão para o formato codificado (definido no anexo A-4) e enviado pelo canal de troca de dados.

- ABRE_FLUXO: realiza a conexão do arquivo com o canal de troca de dados. As operações de leitura/escrita dar-se-ão diretamente sobre os dados armazenados. Não há conversão de formatos.
- MOVIMENTA_ARQUIVO: faz uma cópia dos dados armazenados na interface de dados, em um diretório temporário. Com isto, é preservada a versão original dos dados. Esta operação é realizada principalmente para alimentar de dados as ferramentas fechadas.

- cada linguagem utilizada deve conter um (e apenas um) arquivo que a descreve no formato codificado. Com isto consegue-se uma uniformidade da descrição, evitando o uso de conversores. Este formato codificado armazena as informações da linguagem em uma estrutura (na linguagem C) de fácil tratamento pelos programas.

- para cada nova linguagem um *parser* de leitura e escrita deve ser codificado e integrado à interface de dados. Esta operação é facilitada com o uso das ferramentas *LEX - LEXical analizer* e *YACC - Yet Another Compiler Compiler* ([SUN 88c]).

- As ferramentas devem processar mensagens padrão do sistema, tais como:

  - ARQUIVO: recebe a informação do arquivo, descrição e versão especificados para uso corrente.
  - TECNOLOGIA: indica a tecnologia escolhida para operação.
  - SOLICITA_FIM: questiona sobre o estado de utilização da ferramenta. O sistema necessita saber se é possível finalizar a ferramenta, ou seja, se não existem arquivos sendo editados.
  - RECEBE_PIPE_DADOS: prepara para o recebimento de dados pelo canal indicado

– ATIVA_PIPE_DADOS: prepara o canal para a comunicação.

- As ferramentas com interface gráfica devem suportar um sistema de janelas compatível com o utilizado pelo *SILEX* .
- Ferramentas com interface gráfica devem estar preparadas para o recebimento de mensagens assíncronas vindas do controle de processos.
- Ferramentas sem interface gráfica ficam à espera de mensagens vindas do controle de processos.

A figura 3.2 esquematiza os dois tipos de ferramentas do usuário.

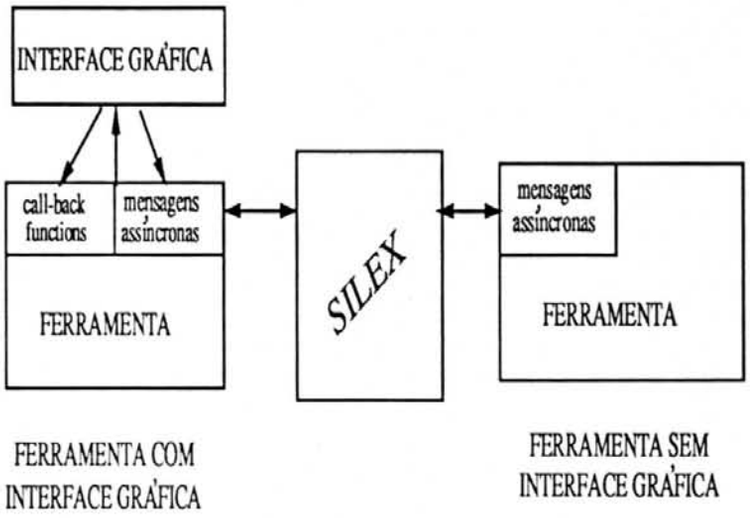

Figura 3.2: Ferramentas do Usuário

As ferramentas com interface possuem uma região de interpretação da interação chamada *call-back function*. Esta zona contém uma função associada a cada elemento da interface (*widget*). Por exemplo: um botão na interface e a função de interpretação de sua ativação. A região de recebimento de mensagens assíncronas é ativada quando informações tornam-se disponíveis no canal de recepção de mensagens. As ferramentas sem interface gráfica apenas possuem a região de recepção de mensagens.

3.3 Ferramentas Fechadas

Ferramenta fechada é aquela que não pode ter o seu código alterado, seja porque o código fonte não está disponível para efetiva integração, ou porque a integração seria custosa. As ferramentas fechadas recebem um reduzido número de informações pela linha de comando. Na ativação, as informações necessárias já devem ser codificadas na forma de parâmetros. Após a ativação, a ferramenta procede independente dos recursos do sistema no qual a mesma está integrada. As ferramentas fechadas geralmente possuem reduzida interação e grande volume de processamento. Enquadram-se neste grupo as ferramentas provenientes de diferentes distribuidores ou desenvolvidas em outros centros de pesquisa, não estando disponível o código fonte.

A leitura e escrita dos arquivos de dados é realizada em um diretório temporário. Antes da ativação da ferramenta, um processo chamado "encapsulador" é responsável pela preparação dos dados neste diretório. Os arquivos podem necessitar conversão para serem compatíveis com os formatos da ferramenta e com os formatos do *SILEX* . Neste caso o encapsulador fica responsável pela ativação do programa codificador.

O encapsulador é um programa escrito em linguagem C, responsável pela adaptação da nova ferramenta ao sistema, como mostra a figura 3.3.

Figura 3.3: Ferramentas fechadas

Quando solicitada a ativação de uma ferramenta fechada, o encapsulador entra em operação para realizar o interfaceamento. Primeiramente o encapsulador comunica-se com o módulo de controle de processos para obter os parâmetros de ativação. Os parâmetros normalmente são os arquivos de entrada e saída, diretório de trabalho, tecnologia e *flags*. Os *flags* de ativação são obtidos através da interface gráfica do encapsulador.

O código de retorno de execução é testado com o término da ferramenta. Caso a operação tenha sucesso, os dados gerados, se existirem, são passados da área temporária para a interface de dados.

A ativação da ferramenta dá-se via comandos *fork* e *exec*, como descrito na seção 4.

3.4 Ferramentas do sistema SILEX

Fazem parte do *SILEX* basicamente quatro ferramentas servidoras. Ativadas no início do funcionamento, permanecem dispondo seus recursos para o sistema *SILEX* . Estas ferramentas, como mostra a figura 3.4, são as seguintes:

1. Controle de processos (CPROC)
2. Interface gráfica (IG)
3. Controle de projetos (CPROJ)
4. Interface de dados (ID)

A comunicação entre estas ferramentas obedece ao formato de mensagens. Todas as mensagens que fluem no sistema passam pelo CPROC e este a envia ao seu destino. Isto permite um sincronismo na comunicação e um controle por parte do usuário das operações realizadas pelos diversos componentes do sistema. Por

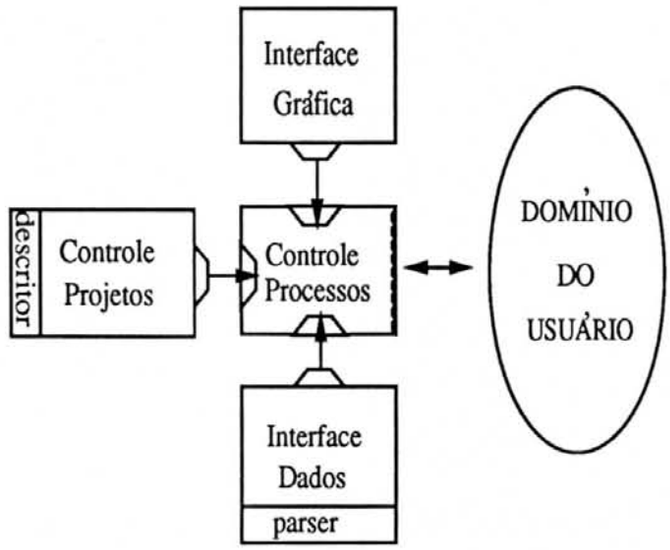

Figura 3.4: Ferramentas do Sistema

exemplo, se uma ferramenta realiza o salvamento de um arquivo, esta informação é exibida textualmente na janela do *SILEX* .

Esta configuração permite a substituição de qualquer uma das ferramentas do sistema sem interferência nas demais. Por exemplo, a interface com os dados do sistema pode ser substituída por uma estrutura mais completa, porém esta mudança deve respeitar as normas de comunicação utilizadas.

3.4.1 Controle de processos

O controle de processos é responsável pela ativação e comunicação das ferramentas. Este módulo armazena em um descritor informações relevantes a cada ferramenta. O formato deste descritor é listado no anexo A-8. Juntamente com a ativação das ferramentas são criados os descritores. Eles indicam quais ferramentas estão ativas e o estado no qual as mensagens se encontram. No mínimo três descritores referentes à IG, CPROJ e ID estão ativos durante o funcionamento do sistema *SILEX* .

O CPROC fica constantemente examinando o canal de comunicação dos

descritores. Quando é feita alguma requisição de mensagem, esta é tratada e em um primeiro instante é verificado qual o servidor que irá atender a requisição. Caso este não seja o CPROC, a mensagem é transmitida ao seu destino. Quando a execução da requisição implicar algum retorno (*feed-back*) ao usuário, a mensagem é convertida para o formato textual e enviada à IG.

3.4.2 Interface Gráfica

A interface gráfica é a ferramenta que permite a entrada de comandos globais e a exibição de resultados ([MAR 90a]). Um croqui da janela inicial é mostrado na figura 3.5, onde distinguem-se quatro regiões:

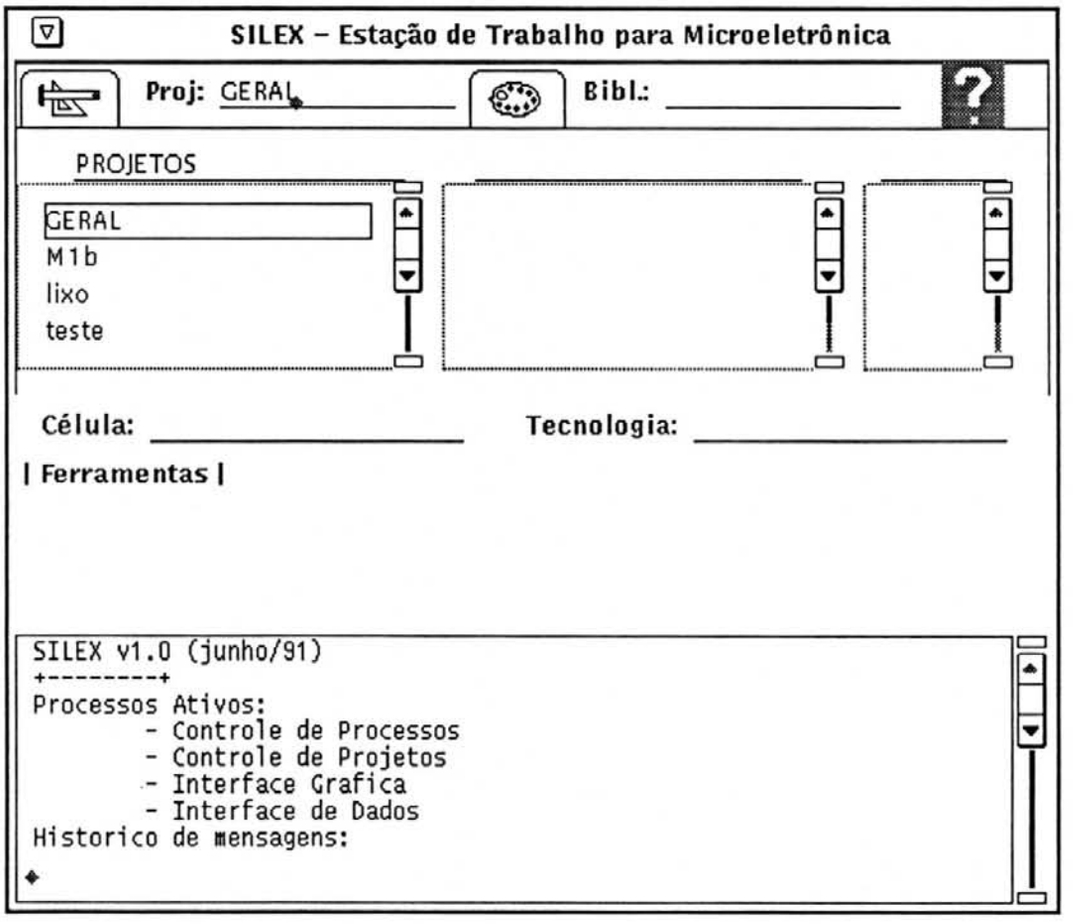

Figura 3.5: Interface Gráfica do *SILEX*

1. A região de iniciação (superior) permite a indicação do projeto e da biblioteca de trabalho. O projeto indicado deve ser previamente cadastrado e ter acesso permitido ao usuário. O projeto com denominação "GERAL" tem acesso permitido a todos os usuários e é utilizado nas operações temporárias sobre o sistema.

2. A região de visualização (segunda região) informa os projetos cadastrados e as bibliotecas, ambos com seus conteúdos. A seleção dos objetos visualizados dá-se por apontamento. Pode ser indicado o objeto, a linguagem e a versão de trabalho. Primeiramente são exibidos os projetos e as bibliotecas, após, com a seleção destes campos, são exibidos os diversos objetos que os compõem. A figura 3.6 mostra um croqui da janela após a operação de seleção do projeto. A interface se configura para esta seleção.

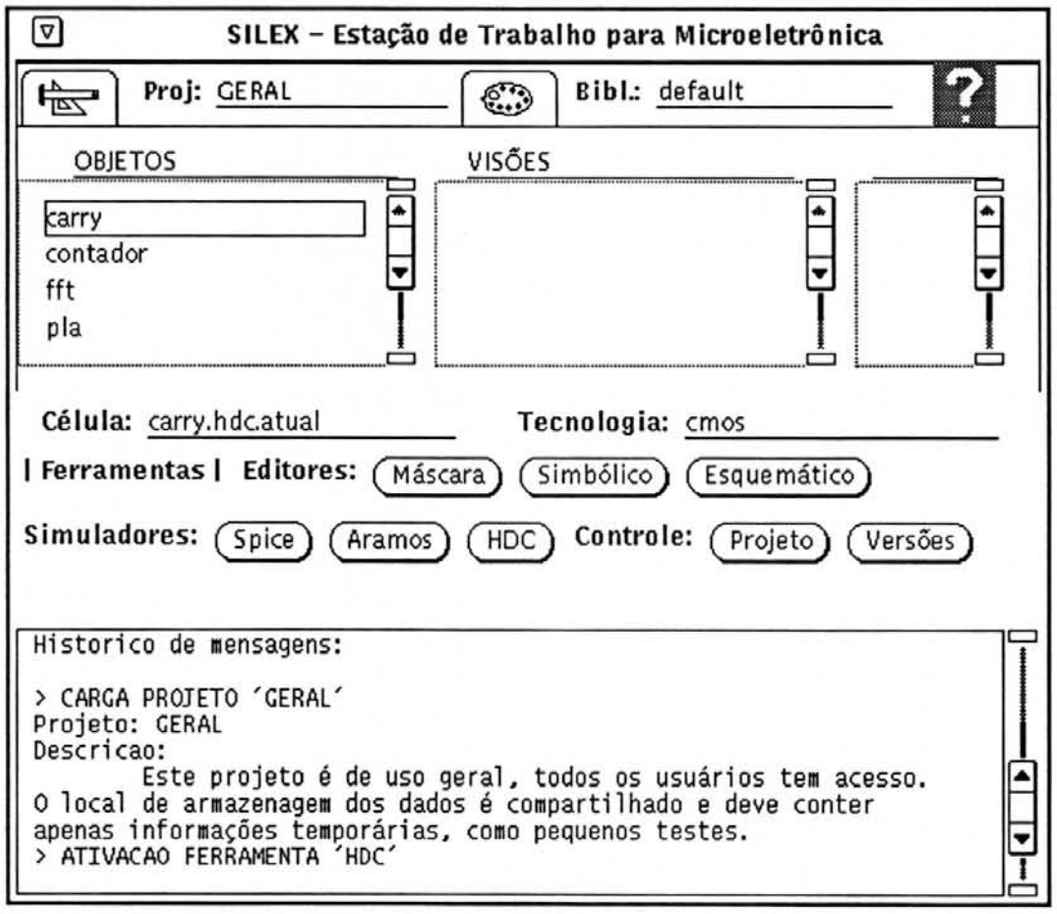

Figura 3.6: Interface pós-carga de projeto

3. A região de ferramentas (terceira região) exibe através de botões as várias ferramentas disponíveis para o projeto. Esta região é dinâmica, uma vez que novas ferramentas podem ser acopladas ao sistema sem a necessidade de recompilação de código. É utilizado um arquivo textual que descreve o conteúdo desta região, como mostra o anexo A-2. Cada projeto contém a sua versão deste arquivo. As alterações realizadas ficam automaticamente disponíveis ao usuário.

4. A região de mensagens (quarta região) lista as mensagens relevantes que chegam ao módulo de controle de processos. O usuário, desta forma, toma conhecimento das operações realizadas pelo sistema. É listado Abaixo o conteúdo desta região com o início da ativação do sistema.

```
SILEX v1.0 (junho/91)
+--------+
Processos Ativos:
- Controle de Processos
- Controle de Projetos
- Interface Grafica
- Interface de Dados
Historico de mensagens:
```

3.4.3 Controle de Projetos

É uma ferramenta que controla a utilização do sistema, com base nas informações contidas no descritor do projeto. A ativação de uma ferramenta e uso dos dados de um projeto são controlados. O formato do descritor é mostrado no anexo A-8. O descritor contém as seguintes informações:

1. Nome do projeto na forma extendida. Por exemplo "Projeto de um Microprocessador de 1 Bit";

2. Os usuários com permissão de acesso. Este campo contém uma lista de nomes (*login-name*) ou o *flag* TODOS indicando nenhuma restrição de acesso.

3. Campo confidencial, quando indicado significa restringir não apenas a alteração, mas também a leitura das informações por usuários sem permissão.
4. Mensagem, contém um breve resumo do projeto, apenas documentacional.
5. Diretório raiz dos dados do projeto.
6. Tecnologia utilizada.

Quando especificado o projeto de trabalho, o arquivo descritor é lido e variáveis de *status* são inicializadas. O cadastro de um projeto, com a geração do arquivo descritor, é feito automaticamente com o uso do programa CAD_PROJ. Este programa tem seu acesso permitido aos usuários do sistema.

Um controle sobre os dados existentes nos projetos é realizado com o auxílio do descritor. O CPROJ gerencia um meta-arquivo contendo informações sobre os dados, as linguagens utilizadas, as versões e as dependências. Este gerenciamento objetiva situar melhor o usuário no contexto do projeto, respondendo questões como: o que existe?; o que falta fazer?; é necessária tal operação?

O formato deste arquivo é listado no anexo A-2 e contém as seguintes informações:

- Lista dos objetos
  - Identificação do objeto
  - Nome do objeto
  - Linguagem de descrição
  - Versão
  - Data de criação
- Lista de dependências
  - Identificador do objeto criado
  - Ferramenta geradora
  - Identificação dos objetos fontes

3.5 Interface com os dados

A Interface com os Dados é responsável pela organização dos dados de projeto. Esta solicita a ativação dos *parsers* de leitura e escrita das linguagens para o formato codificado.

Os dados do projeto são armazenados em um diretório reservado no sistema de arquivos. Somente o usuário com *login_name SILEX* tem acesso a estes dados, utilizando a interface. A forma de armazenamento destes dados obedece à seguinte estrutura:

`Diretório_raiz_dados / Objeto / Linguagem / Versão / Arquivos`

por exemplo:

`/home/sb01/giba/chip.dir / contador / hdc / atual / contador.c`

O `Diretório_raiz_dados` é especificado no descritor do projeto. O `Objeto` é um nome (sem extensão) que identifica os arquivos contidos nos subdiretórios (por exemplo: pla, ram, inverter). A `Linguagem` indica a descrição na qual o objeto se encontra ou a ferramenta geradora. As versões recebem nomes para diferenciá-las, sendo que "atual" refere-se à versão de uso corrente. Por último encontram-se os arquivos que compõem o objeto na linguagem e versão especificados.

A estrutura, à primeira vista pode parecer longa ou de difícil implementação, mas engloba facilidades no tratamento e direto acesso aos arquivos. Por exemplo, a atualização da versão de trabalho basicamente envolve a mudança do nome do diretório.

A segurança dos dados é obtida utilizando os recursos de permissão de acesso do sistema UNIX. Todos os arquivos tem como usuário de criação o *SILEX* e somente este tem permissão de leitura e escrita, os demais usuários apenas acessam

os dados via interface de dados. É atribuída execução privilegiada ao *SILEX* para este controlar as permissões de acesso internas.

3.5.1 *Parsers* de leitura e escrita

A movimentação de dados pelas ferramentas que compõem o sistema é realizada no formato codificado das linguagens de descrição. Este formato tem direta recepção e envio pois está na forma de estruturas (*struct* na linguagem de programação C).

Para a entrada e saída dos dados do sistema, ou seja, a leitura para o formato codificado e o salvamento para o formato de descrição são utilizados *parsers* que são considerados ferramentas com as funções de: formar o canal de transmissão, realizar a movimentação dos dados e encerrar o canal.

O *parser* de leitura executa os seguintes passos:

1. Ativa o canal para troca de dados
2. Acessa o arquivo para leitura
3. Envia comandos de sincronismo no canal
4. Para cada comando do arquivo:
   - Converte para o formato codificado
   - Envia no canal de dados
5. Envia o comando de finalização
6. Encerra o canal de transmissão.

O *parser* de escrita contém os seguintes passos:

1. Ativa o canal para troca de dados
2. Envia comandos de sincronismo no canal
3. Enquanto recebe comando da linguagem
   - Recebe do canal de dados
   - Converte para o formato da linguagem
   - Escreve no arquivo de dados
4. Recebe comando de finalização
5. Fecha o arquivo de dados criado
6. Encerra o canal de transmissão.

Com isto, quando uma nova linguagem passa a fazer parte do sistema, apenas os seus *parsers* de leitura e escrita devem ser construídos e integrados. Pequenas alterações ou extensões das linguagens apenas alteram os *parsers* das mesmas. Uma terceira vantagem desta forma de tratamento é que mais de uma linguagem pode utilizar a mesma codificação interna, como é o caso das linguagens CIF e RS.

3.6 Comunicação no *SILEX*

Para a comunicação entre as ferramentas, optou-se pelo uso dos *sockets*. O *socket*, como mostra a seção 4.6 é um caminho de fluxo de informações entre processos que possuem características desejáveis, como:

1. fluxo bidirecional. Quando um *socket* é ativado, tanto operações de leitura quanto de escrita tornam-se disponíveis em ambos os processos comunicantes;
2. o domínio da comunicação pode ser restrito ao equipamento (AF_UNIX) ou utilizar, de forma transparente aos processos, a rede de comunicação AF_INET, no padrão TCP-IP. O *SILEX* utiliza o domínio AF_UNIX até o momento;
3. sincronismo no envio e recebimento de mensagens no formato SOCKET_STREAM;

4. utiliza a memória na transmissão de dados, sem armazenamento local, acelerando a transmissão;

5. disponibilidade de uso do comando "select" que facilita a utilização do canal de transmissão e permite verificar a disponibilidade de dados para a leitura, escrita ou condições especiais em vários canais simultaneamente;

6. possibilidade de ativação automática de funções do usuário quando dados tornam-se disponíveis nos canais.

3.6.1 Comunicação por mensagens

As mensagens que fluem entre as ferramentas são as que efetivamente integram o sistema. As ferramentas possuem um canal exclusivo para a troca de mensagens com o controle de processos e por este canal são feitas as requisições de recursos para as ferramentas servidoras e o envio de mensagens às ferramentas do usuário.

Uma lista com todas as mensagens utilizadas na comunicação está contida no anexo A-6. Caso uma ferramenta queira acessar um recurso do sistema, basta verificar a mensagem apropriada para a ativação e enviar ao seu destino. Juntamente com o código da mensagem é enviada uma estrutura contendo informações adicionais para certas mensagens. Por exemplo a mensagem VERIFICA_PROJETO envia a identificação do projeto a ser verificado no campo apropriado (ENV_TXT).

As funções para a troca de mensagens estão listadas no anexo A-1. Todas as mensagens possuem o mesmo formato, com o seguinte conteúdo:

- Código da mensagem
- Ferramenta origem
- Ferramenta destino

- Conteúdo da mensagem

O conteúdo da mensagem é composto por um dos três elementos:

1. Vetor de 80 caracteres - TXT
2. Vetor de 10 números inteiros - VAL
3. Identificador do Objeto - OBJ; Linguagem - LIN; Versão - VER

Os descritores das ferramentas possuem duas zonas de mensagens: uma estrutura para o envio e outra para a recepção, acessadas pelas definições ENV e REC respectivamente. Por exemplo, REC_VAL[0] indica o primeiro número inteiro recebido pela última mensagem.

3.6.2 Comunicação de Dados

Quando um conjunto de informações de tamanho e formato variado deve ser enviado às ferramentas, utiliza-se um caminho traçado para esta operação. Este caminho é um *socket* que permanece ativo apenas durante a operação de envio dos dados. Após a conclusão da operação o *socket* é descontinuado. A troca de dados pode dar-se entre quaisquer ferramentas do sistema. A ativação e desativação do fluxo de dados dá-se pelos comandos abaixo descritos, onde "formato" indica a codificação na qual os dados serão enviados.

```
ENV_VAL[0] = ID_FERR_ORIGEM;
ENV_VAL[1] = ID_FERR_DESTINO;
ATIVA_PIPE_DADOS();
... comunica ...
DESATIVA_PIPE_DADOS();
```

O envio dos dados pode dar-se em dois formatos distintos:

1. dados com formato livre e;

2. dados com formato definido.

Os dados com formato livre, como o próprio nome indica, não definem um formato para a transmissão, e são tratados de forma binária. Este formato é utilizado, por exemplo, na transmissão de textos e códigos objetos, onde não é viável a definição de um protocolo e a criação de *parsers* de leitura e escrita.

Os dados com formato definido são utilizados basicamente para a transmissão de informações codificadas das linguagens de descrição de circuitos (como CIF ou SPICE). Neste caso, a codificação é definida na forma de estruturas, uma para cada linguagem.

Quando se deseja realizar o envio de dados, primeiro deve-se alimentar a união e após chamar a função de envio. Estas funções fazem com que todos os comandos da linguagem contenham estruturas de igual tamanho, facilitando a transmissão.

Os campos textuais contidos nos comandos são tratados de forma diferenciada, uma vez que não cabe restringi-los a um tamanho máximo. Assim, primeiro é enviado o comando pelo canal, e após os textos em número e tamanho variado. A operação é realizada de forma transparente ao programador.

Os comandos para a troca de dados são listados no anexo A-1.

3.7 Organização dos diretórios

A organização dos diretórios pertencentes ao *SILEX* é mostrada na figura 3.7 e obedece ao seguinte formato:

BIN - Diretório contendo os códigos executáveis

PROJ - Diretório contendo os descritores dos projetos cadastrados

BIB - Diretório contendo os descritores das bibliotecas de objetos

TMP - Diretório de arquivos temporários

DOC - Diretório contendo a documentação do sistema

silex - Script de ativação do *SILEX*

cad_proj - Script de ativação do cadastro de projetos

leia_me - Texto com informações básicas.

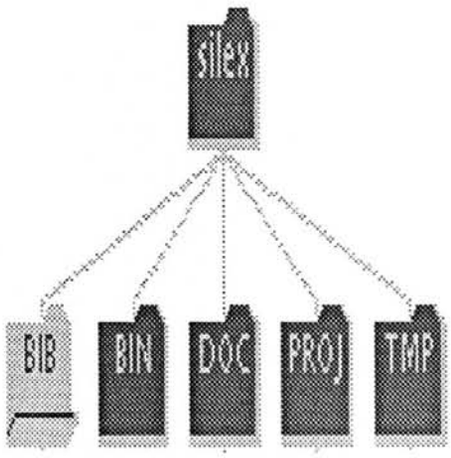

Figura 3.7: Organização dos diretórios

4 COMUNICAÇÃO ENTRE PROCESSOS

A comunicação entre processos em um sistema de CAD indica o grau de integração entre as ferramentas, e permite a distribuição das tarefas entre os diversos componentes do sistema. Cada processo executa uma função específica e compartilha os resultados ou consultas.

Com a definição de linguagens e protocolos de comunicação, as ferramentas tornam-se modulares e simplificadas. Um sistema realmente integrado é aquele no qual cada ferramenta executa basicamente apenas o algoritmo que lhe é próprio, sendo o restante do processamento executado por processos do sistema.

Convém lembrar que o complexo cérebro humano é formado por bilhões de células (neurônios) que realizam funções extremamente simples. Cabe à rede de interconexões destas células a função de integrá-las em um conjunto capaz de realizar funções "inteligentes". Quando se afirma que "*o todo é mais que apenas a soma das partes*" refere-se à forma de comunicação.

Basicamente existem dois tipos de comunicações:

- a comunicação na ativação e término dos processos;
- a comunicação durante o funcionamento dos processos.

O sistema operacional UNIX dispõe de oito formas de comunicação entre processos. A forma que mais se adapta à finalidade desejada deve ser estudada a fim de viabilizar o constante fluxo de informações existente entre as ferramentas. As formas de comunicação estão listadas abaixo e descritas nas próximas seções.

1. Semáforos
2. Sinais

3. Memória Compartilhada
4. *Pipes*
5. *Socketpairs*
6. *Sockets*
7. Chamada de Procedimentos Remotos
8. *Lightweight processes*

Antes de detalhar as formas de comunicação é necessário que se entenda o funcionamento dos processos no sistema UNIX. Um processo pode ser pensado como uma simples lista de controle em um programa, ou seja, o processamento dá-se em apenas uma posição do programa a cada instante. Quando é necessária a execução de mais de uma função concorrentemente, convém dividir o processo em duas linhas independentes (Fig. 4.1). A divisão é executada por uma rotina da linguagem utilizada. Na linguagem C a rotina chama-se `fork()`. O processo resultante da divisão é chamado processo filho.

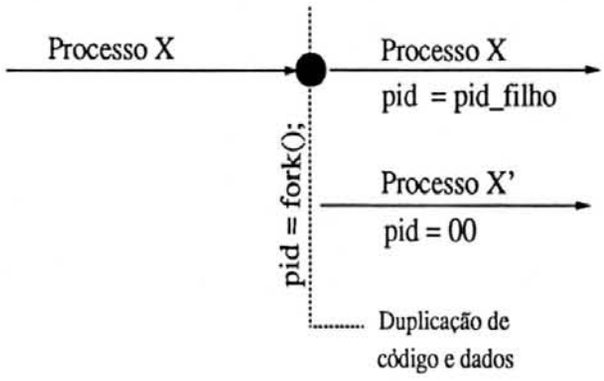

Figura 4.1: Linha de controle em processos

O processo filho passa a ser executado independentemente do processo ativador (processo pai) e as linhas de controle não se juntam mais. O resultado é que os dois processos passam a executar o mesmo código, com os mesmos dados existentes na memória antes do comando `fork`, deste modo, subsequentemente cada versão pode alterar a sua cópia das variáveis. A única diferença entre os dois processos é o valor retornado pelo `fork`. O processo pai recebe o `PID` (*Process Identification*) do

processo filho, enquanto o processo filho recebe o valor zero. A chamada do comando de duplicação geralmente está incluida em uma condição, listada abaixo.

```
/* ... Instrucoes do processo pai */
pidf = fork();        /* Duplicacao do processo */
if (pidf == 0) {
        /* Caso Process-Id seja igual a zero */
        /* o processo em execucao e' o filho */
        /* ... Instrucoes do processo filho */
        /* ou exec() ... */
}
/* Continuacao do processamento ... */
```

Os comandos duplicam o processo e dividem o fluxo de controle. Caso `pidf` seja diferente de zero, referindo ao `pid` do processo filho, o processo pai continua a execução. Caso `pidf` for igual a zero, é a vez do processo filho, neste caso geralmente é executado o comando `exec(<programa>)`. Este faz com que o processo filho passe a executar um programa diferente do chamador.

Um dado importante é que a tabela de descritores é mantida, ou seja, os arquivos em uso continuam disponíveis. Esta característica é essencial em alguns mecanismos de comunicação entre processos que veremos a seguir. Um cuidado que deve ser tomado no término de execução destes processos, é executar o comando `_exit()`, para resguardar a tabela de descritores intacta.

O que vem a ser uma tabela de descritores? Basicamente, representa os arquivos de entrada e saída, *pipes* e *sockets*. É a forma com que os processos visualizam o resto do sistema. A tabela contém índices numéricos referentes a descritores de comunicação. Os primeiros três descritores são conhecidos pelos nomes *stdin*, *stdout* e *stderr*, representando respectivamente a entrada padrão (*standard input*), a saída padrão (*standard output*) e a saída de erros (*standard error*).

4.1 Semáforos

Semáforo é um mecanismo de comunicação no qual processos podem enfileirar ou alterar informações de *status* do sistema operacional. São geralmente utilizados no controle da disponibilidade de recursos, como acesso a arquivos e memória compartilhada.

O acesso aos recursos pode ser controlado através de operações de bloqueio e liberação de semáforos associados. Por exemplo, antes de executar a leitura de um arquivo é conveniente que se bloqueie a escrita por outros processos.

Antes de utilizar um semáforo o processo deve inicializam sua estrutura interna através da função `semget()`. Basicamente três operações são executadas pelos semáforos, estas são:

- incrementar o contador do semáforo de um valor específico,
- decrementar o contador do semáforo de um valor específico,
- esperar até que o contador do semáforo atinja o valor zero.

Estas operações são realizadas através de uma única função, chamada `semop()`. Dependendo do valor passado por parâmetro (positivo, negativo ou zero) é selecionada a operação. Os passos executados na alocação de um recurso ou proteção de acesso, são:

1. aguardar a disponibilidade do recurso indicada pelo semáforo associado (função `semop(0)`);
2. registrar a alocação do recurso (função `semop(1)`) e;
3. após a utilização, liberar o recurso (função `semop(-1)`).

4.2 Sinais

O mecanismo de sinais envia uma pequena quantidade de informação a um processo. O processo sinalizado apenas recebe o tipo de sinal e não a identificação do processo que o enviou. O número de sinais possíveis é relativamente pequeno. A semântica dos sinais, detalhada abaixo, limita a flexibilidade deste mecanismo de comunicação entre processos.

Um conjunto definido de sinais pode ser enviado aos processos, vindos do *hardware*, sistema operacional ou sistema de janelas. Os sinais enviados geralmente causam o bloqueio do processo sinalizado, o salvamento do contexto de processamento e a execução de uma operação indicada para o sinal, como mostra a figura 4.2. Após o término desta operação o contexto é restaurado e o processo continua a execução. A diferença entre o processo antes e depois da interrupção está no valor de suas variáveis globais.

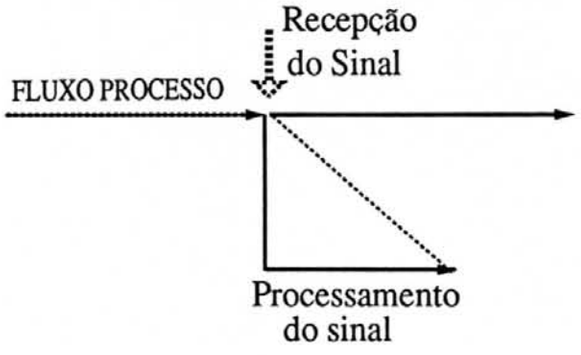

Figura 4.2: Recepção de sinal

Os processos podem indicar a aceitação ou não da ocorrência de certos sinais, sendo que alguns tipos quando não aceitos causam o término do processo (por exemplo *SIGILL* na execução de intrução ilegal pela CPU).

Todos os sinais tem a mesma prioridade e são armazenados quando da sua recepção. A ordem em que os sinais são recuperados não é definida. Basicamente existem cinco tipos de sinais, que possuem o seguinte significado:

- Sinais de hardware indicam condições de execução não convencionais que ocorrem durante o processamento. Por exemplo: *SIGFPE* é acionado em operações ilegais com ponto flutuante; *SIGSEGV* na tentativa de acesso fora do espaço de memória reservado para o processo;

- Sinais de software são interrupções requisitadas pelo usuário, por exemplo *SIGTERM* no término de um processo; *SIGKILL* na finalização de um processo sem aviso. Os programas podem definir seus próprios sinais usando *SIGURG*;

- Operações de entrada e saída; a notificação é feita quando operações de leitura ou escrita podem ser realizadas em um descritor, por exemplo *SIGIO*;

- Controle de processos permite a alteração do fluxo normal de um programa, por exemplo: *SIGSTOP* interrompe a execução e *SIGCONT* indica continuação da execução após a interrupção;

- Controle de recursos indica o uso de algum recurso do computador além da capacidade permitida, por exemplo: *SIGXFSZ* quando um arquivo em disco ultrapassa o tamanho máximo.

Para enviar um sinal utiliza-se a função abaixo. Note que os sinais são enviados apenas a processos de um mesmo usuário.

```
kill(<processo_id>,<sinal>);
```

Para proteger o código de algum sinal indesejado utiliza-se a função:

```
sigblock(<mascara>);
```

4.3 Memória Compartilhada

O compartilhamento da memória permite que mais de um processo, ao mesmo tempo, esteja ligado a um segmento do espaço de endereçamento da máquina.

O acesso a este espaço, nas operações de escrita, deve ser controlado por semáforos para evitar conflitos.

O tamanho do segmento de memória a ser alocado é definido pelo comando `shmget()`, onde é especificado o tamanho, em *bytes*. Processos podem conectar e desconectar estes segmentos de memória à sua zona de endereçamento, através das funções `shmat()` e `shmdt()`. Quando a operação de conexão do segmento de memória a um processo obtém sucesso, é retornado um ponteiro para o início deste segmento. O processo deve tomar o cuidado de não violar este segmento da memória.

4.4 Comunicação via *pipes*

O *pipe* é um mecanismo de comunicação que permite a transmissão de dados, de um processo para outro, por um caminho de dados. Estes processos devem estar rodando em uma mesma máquina e serem ativados por um processo pai comum. Os dados são escritos em uma ponta do canal e lidos na outra.

O uso de *pipe* mais comum no sistema UNIX é indicado pelo sinal "||" na linha de comando (por exemplo:*SILEX % prog1 || prog2*). Isto faz com que a saída de um programa seja conectada à entrada de outro.

Pipes podem, com funcionamento semelhante, serem criados dinamicamente por programas. Neste caso, os programas controlam a utilização do canal. Um *pipe* é geralmente utilizado na comunicação entre processo pai e processo filho ou dois processos filhos. A seguir são listados alguns comandos que exemplificam a criação e uso de um *pipe*. Inicialmente é criado um *pipe* e duplicado o processo pelo comando *fork*. O processo resultante da duplicação (processo filho) escreve no pipe pela entrada *porta[1]*, enquanto que o processo principal (pai) lê na *porta[0]*.

```
#include <stdio.h>                    /* Biblioteca padrao da comunicacao */
```

```
main()
{
   int          porta[2];       /* Canal Leit/Escr do pipe */
   if (pipe(porta)) {           /* Ativacao do canal de comunicacao */
      perror("Abertura do canal de comunicacao");
      exit(1);
   }
    if (fork()) {               /* Duplicacao do processo principal */
      char          buf[80];    /* Processo principal */
      close(porta[1]);
      read(porta[0], buf, 100);    /* Operacao de leitura no canal */
      printf(buf);
      close(porta[0]);
   } else {
      close(porta[0]);          /* Processo filho */
      write(porta[1], "Troca de mensagens OK pelo canal", 32);
                  /* Operacao de escrita */
      close(porta[1]);
   }
   exit(0);
}
```

O processo executa a chamada da rotina `pipe()` do sistema criando o *pipe* e colocando seus descritores na tabela do processo. O vetor passado como parâmetro é inicializado. O índice retornado no primeiro elemento do vetor é aberto para leitura e o segundo o é para escrita.

A figura 4.3 exemplifica o efeito do programa listado acima, exibindo a tabela dos descritores após cada operação.

Sendo o *pipe* um mecanismo de comunicação em um sentido, os processos pai e filho devem concordar na escolha de um dos dois sentidos possíveis para a comunicação. Caso for necessária a comunicação nos dois sentidos, dois *pipes* devem ser ativados.

O comando *select* está disponível nas versões mais recentes do sistema UNIX (a partir da versão 4.2). Este comando permite que se verifique o estado da leitura e escrita do canal antes de ativar uma operação.

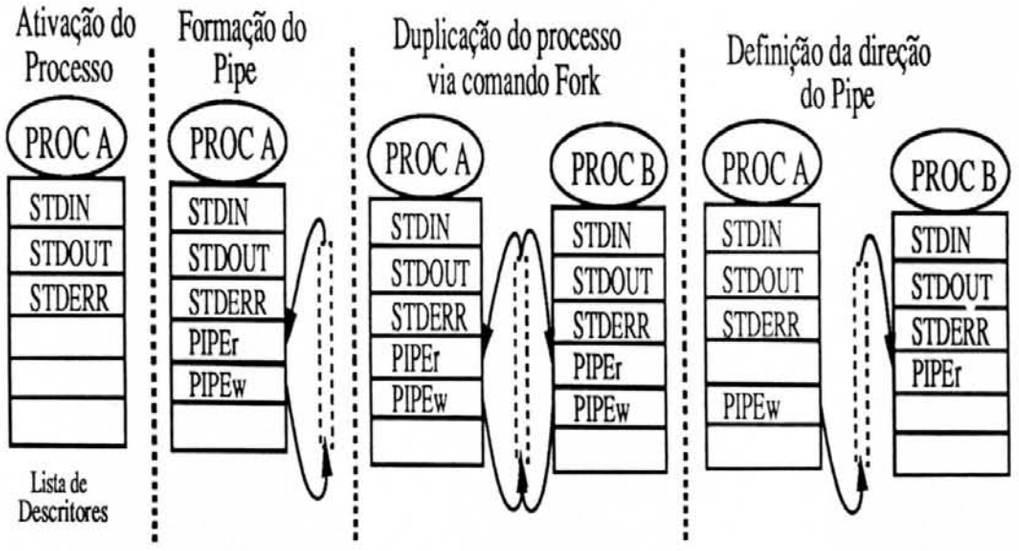

Figura 4.3: Utilização do *pipe* na comunicação

As mensagens são enfileiradas no canal. Quando é solicitada a leitura de um número de bytes, são retornados tantos quanto estiverem disponíveis até momento.

4.5 Comunicação via *socket pairs*

Socket pairs é uma generalização do *pipe*, que utiliza dois canais para a comunicação entre processos. Este mecanismo é restrito a comunicação entre processos em uma mesma máquina, e não é utilizado o armazenamento local.

A função que ativa o *socket pair* é listada abaixo, onde "domínio de comunicação" define a sua abrangência (na versão atual do UNIX é restrito a AF_UNIX). Este valor define um conjunto de convenções para a manipulação do nome do canal. "Estilo" define a forma de comunicação (SOCK_STREAM, SOCK_RAW).

```
socketpair(<domínio>,<estilo>,<protocolo>,<caminho>)
```

4.6 Comunicação via *socket*

O *socket* é um mecanismo de comunicação que possui um nome associado ao canal de troca de dados. Cada *socket* em uso tem um tipo e um ou mais processos conectados. Os *sockets* possuem um domínio de comunicação que é uma abstração contendo uma estrutura de endereçamento e um conjunto de protocolos que implementam os vários tipos de *sockets* da familia. Existem vários domínios disponíveis, mas apenas dois são os mais utilizados, identificados pelos nomes AF_UNIX e AF_INET (*Address Format*).

A estrutura de endereçamento difere para cada tipo de endereçamento. Para o domínio UNIX, o nome do *socket* é um caminho no diretório até um arquivo (por exemplo "/tmp/hdc_5"). Um nodo é criado para que outros processos possam ser referenciados. AF_UNIX é utilizado na comunicação de processos em uma mesma máquina que usem um mesmo nome de arquivo (*path_name*) para a comunicação. O domínio AF_INET utiliza a rede na comunicação entre processos, com o protocolo DARPA. Estes processos podem estar rodando em máquinas diferentes. O início da conexão consiste na atribuição de um identificador de rede e de um número diferenciador.

Os tipos de *sockets* existentes são: *DATAGRAM*, *STREAM* e *RAW*. Cada qual possui propriedades que os distinguem e estão ligadas à semântica da comunicação.

1. DATAGRAM, *socket* que suporta fluxo bidirecional de dados, não garante sequenciamento, segurança e unicidade na recepção. As mensagens podem chegar duplicadas ou fora da ordem de envio. Uma importante característica é que as informações, como o endereço de destino, são adicionados aos registros. Não é realizada conexão entre os processos comunicantes, cada mensagem é tratada individualmente. Esta forma é utilizada na troca de pacotes de dados de uma rede Ethernet.

2. STREAM, *socket* que suporta fluxo bidirecional, sequenciamento, segurança, não duplicidade e fluxo de dados sem informações adicionais nos registros. A filosofia é semelhante a do *pipe* . Necessita a conexão entre os processos para efetivar a comunicação.

3. RAW, tipo de *socket* que permite acesso a um nível mais baixo do protocolo de comunicação que o *socket* suporta. Não é direcionado para o uso geral, mas é utilizado por pessoas interessadas em desenvolver novos protocolos de comunicação.

Cada tipo de *socket* e domínio utilizados tem um protocolo de comunicação. O protocolo é um conjunto de regras, formato de dados e convenções que regulam a transferência de dados entre os participantes da comunicação.

4.7 Chamada de Procedimentos Remotos

A Chamada de Procedimentos Remotos é uma biblioteca de funções que permite a um processo ativar uma função pertencente a um outro processo. Esta função é executada como se pertencesse ao processo chamador, ou seja, em seu espaço de endereçamento. Estes processos não necessitam estar na mesma máquina, pois utilizam a rede na comunicação. O processo chamador envia um pacote de dados ao processo servidor. Quando o pacote chega, a rotina requisitada é ativada com base nos dados recebidos. O resultado é enviado ao processo chamador.

Para a execução do procedimento acima descrito, pode-se utilizar RPC em três níveis:

- Alto: Modo transparente de sistema operacional, máquina ou rede. É disponível uma biblioteca de funções que é utilizada na ativação da comunicação.
- Médio: Neste nível o usuário não necessita entrar em detalhes sobre o sistema

UNIX, *sockets* ou forma de implementação. Simplesmente estão disponíveis rotinas que permitem a ativação de procedimentos remotos. Este nível de ativação é utilizado pela maioria das aplicações por ser de uso simples.

- Baixo: recomendado quando se necessita um maior controle das operações. Programas utilizando este nível de ativação de processos remotos são mais eficientes.

A figura 4.4 ilustra este mecanismo de comunicação.

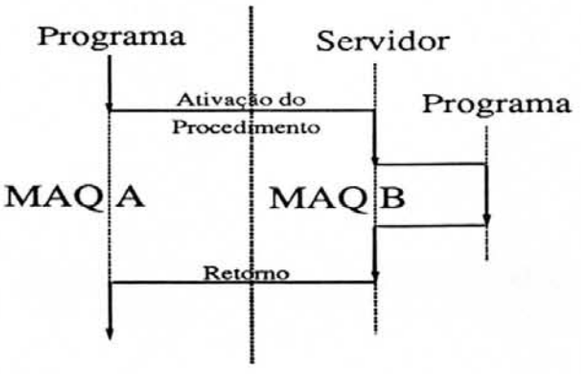

Figura 4.4: Chamada de Procedimentos Remotos

É utilizada a rede para a transmissão dos dados. O problema surge quando as máquinas comunicantes não são compatíveis, quanto à representação interna dos dados, por exemplo: a ordem dos *bytes* na representação de um número inteiro. Para solucionar este problema foi criada a representação externa de dados (XDR - *eXternal Data Representation*). Neste contexto, os dados antes de serem enviados são convertidos a um formato padrão e, após o envio, são novamente convertidos para o formato da máquina receptora.

O tempo dispendido na conversão da representação da máquina para o padrão, é insignificante em aplicações que utilizam a rede. A maior parte do tempo é gasto na conversão da estrutura de dados para o formato passível de ser transmitido.

4.8 Processos Leves

Os processos leves (*Lightweight Process*) representam uma linha de controle onde diversas rotinas (*threads*) compartilham da mesma área de endereçamento. Estas rotinas operam de forma semelhante a de processos do sistema, mas tem sua comunicação muito facilitada pelo compartilhamento de memória.

Estas *threads* processam concorrentemente, obedecendo a um esquema de prioridades. As *threads*, por ocuparem a mesma área de endereçamento, são de fácil criação e comunicação. Programas que necessitam de processamento concorrente, como servidores e simuladores, obtém vantagens com a utilização destes processos.

A idéia é criar uma abstração do processo. As *threads* são um tipo de dado que representam um fluxo de controle. O processo que contém as várias *threads* compartilha o seu tempo de execução com as mesmas. Estas não recebem tempo de execução dedicado do sistema operacional.

Uma biblioteca de funções para a manipulação dos processos é disponível, esta realiza:

- Criação das *threads*, finalização, verificação de estado, controle da execução (*scheduling*, bloqueio e execução);
- Compartilhamento da fatia de tempo de execução (*timeslice*) e;
- Sincronismo das *threads*.

Um esquema simples de prioridades de execução é disponível, sendo que podem ser criados outros a partir destas funções. Uma *thread* é uma rotina do programa que recebe propriedades especiais de execução quando executada a função de inicialização. A troca de mensagens entre as *threads* utiliza a chamada de procedimentos remotos, obtendo vantagens no sincronismo e no domínio de comunicação.

Primitivas para o envio, recepção e confirmação de recepção de mensagens são disponíveis.

4.9 Conclusões

No sistema operacional UNIX temos a comunicação entre processos ligada à entrada e saída (via arquivos e descritores). Este tratamento provou ser de grande valia. As formas mais simples de comunicação (sinais, *pipes* e *socket pairs*) restringem-se a processos em uma mesma máquina, não utilizando a rede de conexões.

Em versões mais recentes do sistema UNIX (*Berkeley Unix System* v4.2) tornou-se possível a comunicação entre máquinas. Esta expansão necessitou mudanças na forma de como os descritores eram criados. Adicionalmente novas possibilidades para as operações de entrada e saída tornaram-se disponíveis. Inicialmente a semântica destes canais era simples e se restringia a: após o comando de escrita o dado é enviado; a leitura fica bloqueada até o dado chegar.

Com base no estudo acima relatado de comunicação entre processos, chegou-se a algumas conclusões quanto à possível utilização destas técnicas no contexto do *SILEX* . O mecanismo de comunicação que melhor se adaptou às necessidades do *SILEX* na troca de mensagens e dados entre as ferramentas foi *socket* . Abaixo encontram-se características dos mecanismos de comunicação estudados.

- A utilização do *pipe* na comunicação é muito restrita, porque os processos devem ter um ancestral comum. Esta restrição pode não parecer muito grande no contexto do *SILEX*, uma vez que existe um único processo controlador, mas torna mais complexa a comunicação entre processos. Com a utilização do *pipe*, antes da ativação de ferramentas, o controlador cria um canal de comunicação para a leitura, outro para a escrita e o atribui ao processo filho.

- O problema surge quando se deseja efetuar a troca de dados entre ferramentas, sem a interferência do controlador. Para realizar esta função de troca, é necessário que o controlador, antes da ativação de qualquer ferramenta, crie e gerencie uma lista de canais disponíveis.

- Em um futuro próximo a expansão do *SILEX* para um ambiente distribuído tornar-se-á necessária. A semântica do *pipe* torna impossível esta configuração.

- A diferença no desempenho entre os tipos de comunicação DATAGRAM e STREAM do *socket* é, em geral, menos importante que a diferença em semântica. O desempenho ganho com a utilização do DATAGRAM aumenta a complexidade da programação em aplicações que necessitam segurança na recepção das mensagens.

- Os semáforos e os sinais são de fácil utilização, mas apresentam reduzida possibilidade de uso pelas ferramentas do *SILEX*. As operações de seqüenciamento e interrupção realizadas por estes mecanismos podem ser substituídas pelo envio e recepção de mensagens.

- A memória compartilhada entre processos para a troca de mensagens e dados acelera a comunicação, mas aumenta a complexidade dos algoritmos que tratam do sincronismo entre os processos. O mecanismo de memória compartilhada pode ser substituído pelos canais de comunicação sem perda significativa de eficiência.

- Os processos leves, com a criação das *threads*, obtém vantagens na criação e troca de mensagens. Caracterizam-se por juntar várias linhas de execução em um único processo UNIX. Este mecanismo é recomendado para processos de reduzidas dimensões, como rotinas. As ferramentas do sistema ultrapassam este limite.

- O uso de socket do tipo STREAM mostra-se de grande valia como meio de interligar as ferramentas. Este, como foi visto, permite o sincronismo e armazenamento de mensagens em tamanho e formato definido pelo usuário. O domínio

AF_UNIX para comunicação entre processos, rodando na mesma máquina, pode ser utilizado na primeira versão do sistema. A mudança de domínio para AF_INET requer apenas alguns cuidados na formação do canal de comunicação.

5 AMBIENTE DE IMPLEMENTAÇÃO DO SILEX

O sistema *SILEX* está implementado em estações de trabalho do tipo *SUN-SPARC*, sistema de janelas XVIEW, linguagem de programação C e sistema operacional UNIX. Este ambiente de trabalho é ideal para o projeto de circuitos e desenvolvimento de ferramentas de CAD, pois engloba diversas características desejáveis, como:

- Alta velocidade de processamento do processador RISC;
- Processador gráfico avançado;
- Sistema de janelas poderoso, compatível com Xwindow, e pacotes gráficos em vários níveis ([SWI 88]);
- Linguagem de programação C padrão;
- Robusto sistema operacional UNIX com dezenas de utilitários, como (*YACC*, *MAKE* [SUN 88c]);
- Proteção de acesso à memória e arquivos ([SUN 88a]);
- Variadas técnicas de comunicação entre processos ([SUN 88b]);
- Comunicação transparente em rede ([FUL 88]).

Estas características facilitam a tarefa de integração, tornando possível a escolha de uma entre diversas opções de arquitetura para novos sistemas.

5.1 Estações *SUN-SPARC*

A arquitetura SPARC foi introduzida em 1987 nas estações de trabalho do tipo SUN-4. Estas estações utilizam um processador RISC. Estações anteriores

baseavam-se na família *motorola 68000*, considerada várias vezes mais complexa em relação ao número de instruções. SPARC passou de uma arquitetura aberta para tornar-se uma familia de processadores, com inúmeros fabricantes utilizando-os em seus novos produtos. Estes incluem: Fujitsu, Texas Instruments, Philips entre outros.

O alto desempenho atingido por estes computadores deve-se à filosofia RISC utilizada no processador, com *pipeline* implementado em *hardware* e escalonamento de janelas de registradores. A arquitetura RISC foi escolhida, ao invés de circuitos com conjunto de instruções cada vez mais complexo (CISC), pois um conjunto reduzido e simplificado de instruções possibilita a construção de uma arquitetura mais veloz ([JUN 92]). Verificou-se que as instruções sofisticadas de circuitos complexos são geralmente pouco utilizadas. Os compiladores, na sua maioria, utilizam apenas 30% das instruções disponíveis na geração de código para um processador CISC.

A conexão em rede de estações é transparente para o usuário e para oprogramador de ferramentas. O sistema operacional, aliado ao sistema de janelas, permite que processos se comuniquem utilizando a rede no padrão ETHERNET, com computadores e sistemas operacionais diferentes. Desta forma, terminais gráficos de relativo baixo custo podem utilizar todo o potencial da rede ([YAG 91]).

A configuração da rede de computadores deve ser devidamente estudada para não degradar o sistema. O fluxo de informações na rede pode deteriorar a interação com o usuário, pois cada operação, como movimentação do *mouse*, traçado de linha são enviados ao processador remoto, passando por vários passos de codificação, como mostra a figura 5.1.

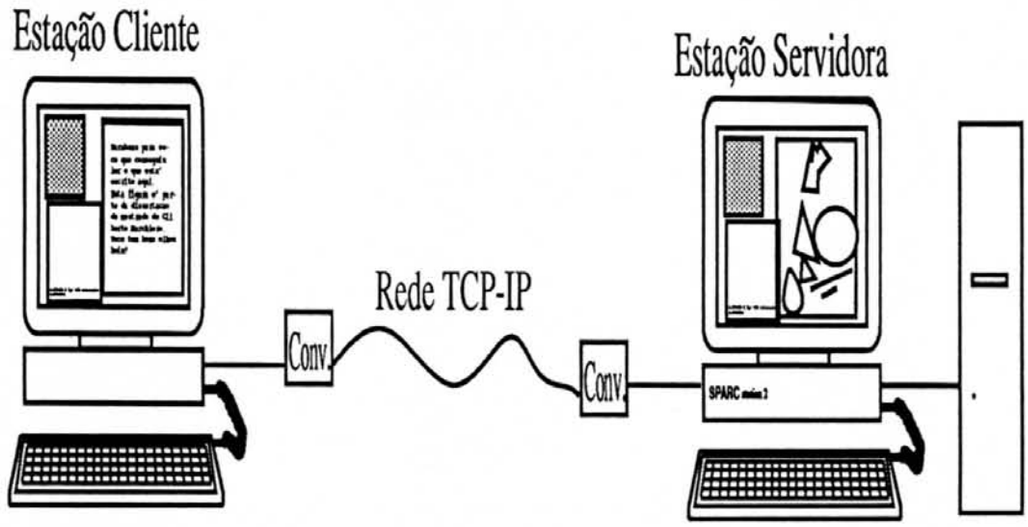

Figura 5.1: Estações de trabalho

5.2 Sistema de Janelas

O *SILEX* utiliza como interface gráfica o sistema de janelas OpenWindows. O OpenWindows é um conjunto de recursos para a interface com o usuário que suporta aplicações interativas e gráficas utilizando janelas. A interação com o usuário é realizada por objetos básicos agrupados. Estes objetos são: botões (*buttons*), janelas de desenho (*canvas*), painéis de controle (*panels*), e outros.

A idéia é permitir uma rápida visualização das funções disponíveis e regular a sua utilização nas ferramentas. Cada ferramenta possui uma janela base (*base frame*), na qual são ligadas as sub-janelas, e um conjunto de objetos de interface com suas funções associadas.

A figura 5.2 mostra a hierarquia de utilização das janelas para a montagem da interface.

A janela base une os objetos da interface. Esta possui basicamente uma zona superior de exibição de mensagens (geralmente exibe o nome da ferramenta) e uma zona inferior na qual são posicionadas as sub-janelas. Associada a esta janela se encontra um icone, que é um objeto gráfico retangular que representa o estado

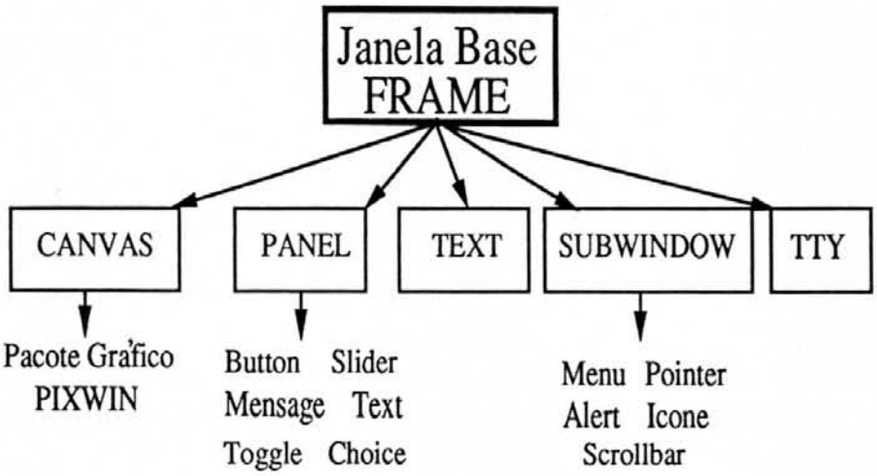

Figura 5.2: Hierarquia de janelas

"fechado" da janela, ou seja, a ferramenta continua processando, apenas não é visível graficamente.

Em um terminal pode ser exibido um número variado de ferramentas, em diversas janelas. Para gerenciar a interação e entrada de dados é utilizado um controlador, que é baseado em um notificador central que distribui a interação às aplicações. Um gerente de janelas fica responsável pela sobreposição e correta exibição dos elementos gráficos.

5.3 Elementos Básicos de uma Interface

Os elementos básicos de construção de uma interface (*widgets*) estão listados abaixo e são exibidos graficamente na figura 5.3.

- Janela Base
- Janela de desenho
- Painel de controle
  - Mensagem
  - Botão

  - Entrada de texto
  - *Slider*
- Barra de movimentação
- *Pop-up sub-window*
- Sub-janela de textos
- Icone
- Mensagem de alerta
- Cursor

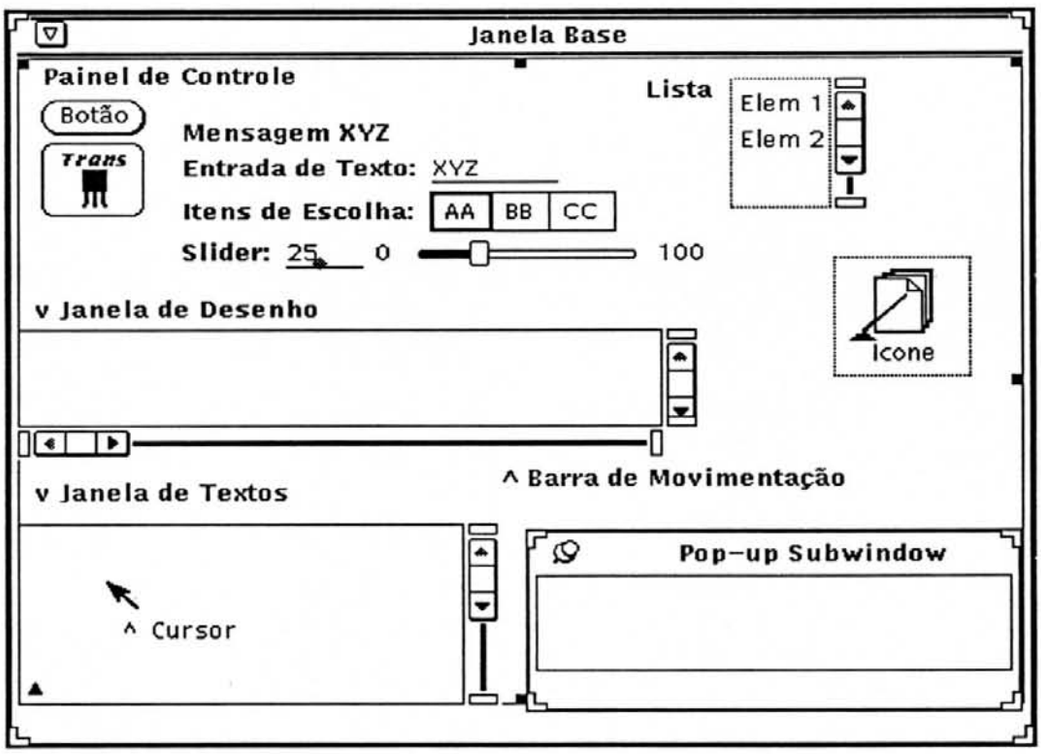

Figura 5.3: Elementos de uma interface gráfica

5.3.1 Janela de desenho

O *canvas* é uma janela onde é realizado o desenho de primitivas gráficas e a escrita de textos. Este é o mais básico tipo de sub-janela.

O *canvas* em sua criação deve estar associado à janela principal ou a uma sub-janela do tipo *pop-up*. Para desenhar no *canvas* pode se utilizar o pacote gráfico padrão PIXWIN, ou qualquer outro disponível no sistema de janelas OpenWindows. PIXWIN trabalha no nível mais baixo, com primitivas de traçado de linhas, círculos, polígonos, entre outras. Pacotes de mais alto nível que trabalham com objetos e realizam funções de câmera sintética são os seguintes: SunCGI, SunCORE e SunGKS.

O traçado de primitivas utilizando o pacote Pixwin, demanda a criação prévia de um descritor, que contém um conjunto de valores utilizados nas diversas primitivas, como por exemplo: cor de traçado, espessura da linha e fonte de texto. O descritor é passado como primeiro parâmetro em quase todas as chamadas de primitivas.

O *canvas* possui uma largura, altura e profundidade. A profundidade está ligada ao número de cores utilizadas no desenho. Cada *canvas* pode definir um segmento particular do conjunto de cores (*palette*) para uso próprio. Isto faz com que a alteração das cores pertencentes ao *canvas* não interfira nas demais cores da interface.

A imagem desenhada no *canvas* pode ser maior que a exibida pela janela. Para alterar a zona de visualização no domínio do desenho são utilizadas as barras de movimentação (*scrollbars*).

A função de redesenho (*repaint function*) é utilizada na recuperação do conteúdo de um *canvas*. É uma função criada pelo usuário e ativada pelo gerente de janelas quando este detecta a necessidade de redesenhar parte ou toda a janela. Esta operação ocorre quando uma janela é aberta ou alguma região sobreposta passa a ser visível.

A interação com o *canvas* geralmente dá-se com a movimentação do *mouse* e com o pressionar de seus botões. O usuário cria uma função de notificação que é ati-

vada quando eventos são recebidos pelo *canvas*. Estes eventos são: a movimentação do *mouse* (LOC_MOVE), a entrada ou saída do cursor no *canvas* (LOC_WINENTER e LOC_WINEXIT) ou o pressionar de botões do *mouse* (MS_LEFT, MS_RIGHT e MS_MIDDLE).

5.3.2 Painel de Controle

O *panel* é uma janela que contém elementos com os quais o usuário interage. O *panel* permite que se modele uma variedade de funções, como: entrada textual, exibição de mensagens, seleção de itens, menus e listas.

O *panel* pode utilizar a barra de movimentação e contém basicamente os seguintes elementos: mensagens, botões, itens de escolha, *toggles*, textos e *sliders*. Os elementos podem ser formados por um ou mais componentes. O componente básico encontrado em todos os elementos é o *label*. Este pode ser um texto ou um gráfico. Com excessão das mensagens, os demais elementos possuem um menu associado.

Os elementos contidos no painel são dinâmicos, ou seja, podem ser criados ou destruídos durante o funcionamento do processo. Abaixo são descritos estes elementos.

- Mensagens exibem um texto ou um gráfico. Tem utilidade na formação de comentários, títulos, e figuras;
- Botões permitem a ativação de comandos pelo usuário. Tem função semelhante às mensagens, pois possuem texto e função notificadora. A diferença é que botões exibem visualmente a confirmação da seleção pela mudança de sua cor ou de sua forma;
- Itens de Escolha permitem a seleção de um item de uma lista. O formato de exibição dos itens varia muito, dependendo da opção escolhida;

- *Toggles* são semelhantes aos itens de escolha, sendo que nestes não há relação entre as diversas opções. Cada item pode estar ativo ou inativo;
- Itens Textuais permitem a entrada de seqüencias de caracteres, possui *label* e rotina de notificação e;
- *Sliders* permitem a entrada gráfica ou textual de um valor pertencente a um domínio;

5.3.3 Janela de Edição

A janela de edição permite a exibição e entrada de textos. Estes podem estar armazenados na memória principal ou em disco. Um variado conjunto de funções de manipulação estão disponíveis, podendo ser ativadas pela interação com o usuário ou por rotinas da ferramenta.

5.4 Forma de Construção da Interface

A criação da interface é feita através de chamadas a funções. Existem funções globais como `window_create()`, `window_set()` e `window_get()`, que realizam variadas operações, dependendo dos valores passados como parâmetro. Estas funções recebem uma lista com tamanho variado de parâmetros e para cada um deles o seu tipo e valor.

O anexo A-9 exemplifica a definição da interface exibida na figura 5.4.

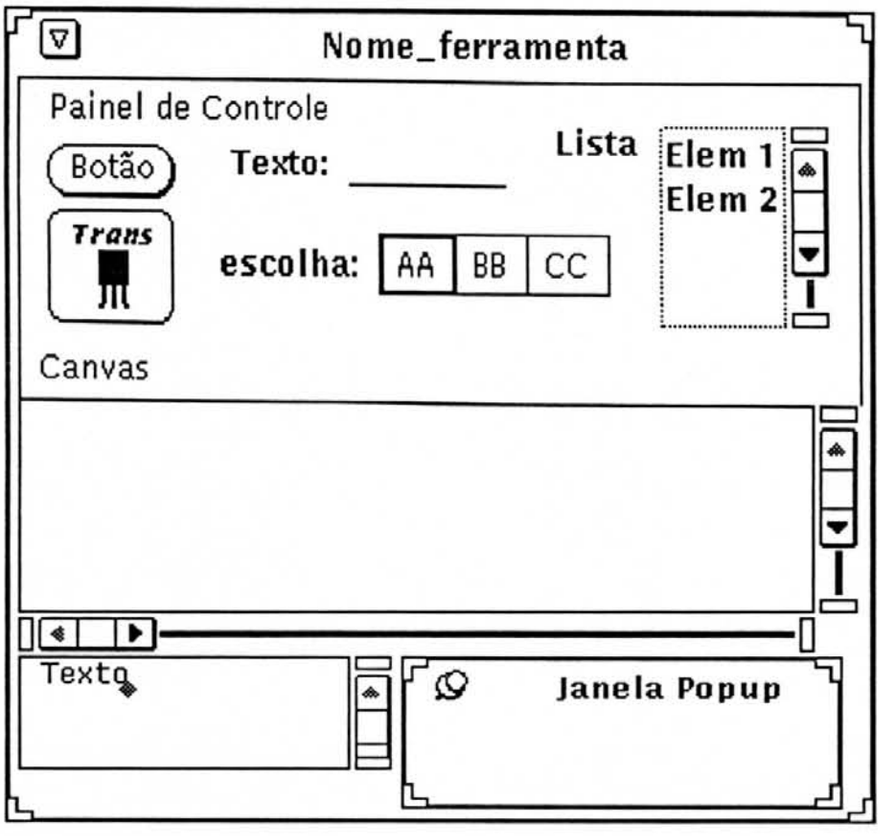

Figura 5.4: Croqui de uma Interface Exemplo

5.5 Ferramentas de Auxílio

A tarefa de projeto de uma interface é muito facilitada pelos gerenciadores disponíveis. Estes oferecem um conjunto consistente de funções, definições e normas de codificação. Qualquer usuário pode, com base nestas informações, definir a sua interface.

O problema está na quantidade de informações necessárias para um usuário iniciante começar a programar sobre estes novos recursos. Os manuais destes sistemas geralmente estendem-se por centenas de páginas de pesado conteúdo. Uma fase inicial, relativamente longa de aprendizado, pode tornar-se necessária.

Para solucionar o problema acima exposto, estão disponíveis as ferramentas de auxílio à criação de interfaces com o usuário. Com estas pode-se interativamente e graficamente criar janelas, posicionar elementos e simular o funcionamento da interface. Pronta a definição, o passo seguinte é a geração do código em linguagem

C que implemente esta interface. Com isto, toda a fase inicial de aprendizado pode ser substituída por um breve estudo da funcionalidade dos diversos componentes da interface.

Gerado e consistente o código da interface com o usuário, com rotinas de notificação e eventos definidos, basta alimentar o "esqueleto" de código com o algoritmo da ferramenta.

Um exemplo de ferramenta geradora de interface com o usuário, utilizada no *SILEX* , é o GUIDE (*Graphical User Interace Design Environment*) em conjunto com o GXV. O GUIDE permite a manipulação automática dos elementos do sistema de janelas, verificando normas de construção. A figura 5.5 mostra a janela principal desta ferramenta. Juntamente com o elemento, acessado via *mouse*, está um conjunto de propriedades. O correto posicionamento dos elementos e a escolha das propriedades é verificado em tempo de execução pelo programa. Por exemplo: se o usuário tentar posicionar um botão sobre uma janela diferente do painel de controle, receberá a mensagem de operação ilegal.

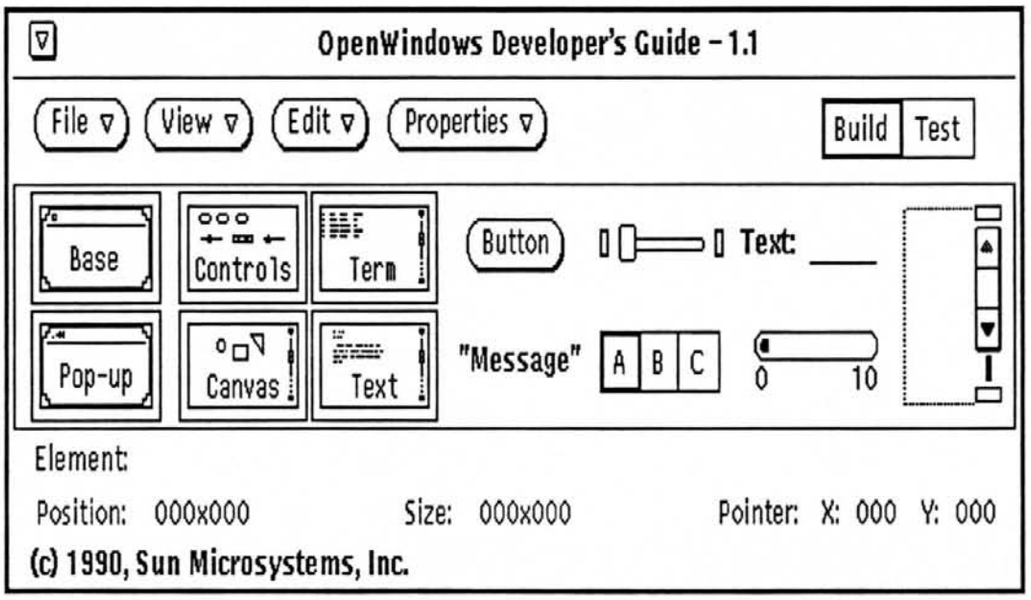

Figura 5.5: Ferramenta de Auxílio - GUIDE

6 INTEGRAÇÃO DE FERRAMENTAS

Uma vez disponível e consistente a primeira versão do sistema *SILEX*, passa-se à integração de ferramentas. A variedade e qualidade das ferramentas é o que dá potencialidade ao sistema. O futuro usuário será motivado a utilizar o *SILEX* ao invés de outros sistemas, pelo conjunto de ferramentas que vier a dispor. Por outro lado, a tarefa de integração deve ser regular e obedecer a normas bem definidas, para que o programador de ferramentas, com base nestas normas, possa anexar seu algoritmo ao sistema sem dificuldades ou restrições.

O conjunto base de ferramentas para tornar o *SILEX* utilizável profissionalmente é composto por, em um primeiro passo, quatro classes de ferramentas: edição, extração/verificação, simulação e síntese. Ferramentas pertencentes a cada uma destas classes já existem no contexto do GME (Grupo de MicroEletrônica) da UFRGS, e são as seguintes:

- Editores
  - Máscaras - SELA [CAS 91]
  - Esquemáticos - ESQUELETO [MOR 90a]
  - Simbólico - CHARRUA [MAR 89]
- Extratores/Verificadores
  - Extrator Elétrico - EXTRIBO [STE 89]
  - Extrator Lógico - EXTRALO [MOR 90b]
  - DRC - DARC [GOM 88]
  - Comparador de *Netlist* - COTONET
- Simuladores
  - Elétrico - SPICE [VLA 81], ARAMOS
  - Funcional - HDC [MAR 91]
  - Lógico - SPLICE

- Síntese
  - Multinível - GPO [CAR 89]
  - Gerador de ROM - GROM [CAR 88a]
  - Gerador de PLA - GALOPA
  - Geradores de Lógica Aleatória - TRAGO [MOR 90c], TRAMO

A quantidade de trabalho necessário à introdução das ferramentas, depende decisivamente da natureza e organização da codificação. Quanto menor for a necessidade de interação da ferramenta com o usuário, mais fácil será a tarefa de integração. Desta forma, editores geralmente necessitam maior trabalho na integração que simuladores. A quantidade de trabalho envolvida nas diversas classes de ferramentas obedece à seguinte ordem crescente: extratores, simuladores, síntese de *layout* e editores.

A organização na codificação, ou seja, o cuidado dispensado na programação, recebe grande importância quando se necessita alterar o código. Modularidade na codificação, documentação de rotinas e obediência dos preceitos da engenharia de software são requisitos indispensáveis para a integração. Dois fatores têm sido observados durante a integração que prejudicam esta tarefa: o programador de ferramentas tem total domínio do código implementado, mas geralmente não é a pessoa encarregada da integração; os programadores são, na maioria das vezes, engenheiros eletricistas ou eletrônicos sem curso formal em programação de sistemas (no contexto do GME).

6.1 Passos na Integração

Conhecendo a organização do sistema relatada no capítulo anterior, podemos detalhar a forma de integração. O fator inicial consiste em entender o funcionamento da ferramenta a ser integrada e separar os diversos módulos que a compõem. Caso não seja possível ou não se deseje alterar o código da ferramenta, deve-se criar

um encapsulador, como mostra a seção 6.2. As ferramentas com código alterável, podem conter ou não interface com o usuário. As ferramentas sem interface tem integração direta.

6.1.1 Ferramentas Sem Interface Gráfica

Este tipo de ferramenta comunica-se com os sistemas para a determinação dos parâmetros de execução. O seu funcionamento muitas vezes não é percebido pelo usuário, pois ocorre em *background.*

O passo inicial consiste na inclusão das estruturas, definições e funções necessárias para a interface da ferramenta com o *SILEX.* Isto dá-se pelo comando:

```
#include "silex.h"
```

O passo seguinte consiste na formação do canal de comunicação de mensagens entre a ferramenta e o controle de processos. A operação é ativada pela função abaixo, incluída logo após o início da ferramenta.

```
INICIA_FERRAM( NULL , argc , argv );
```

O primeiro parâmetro da função (`janela_base = NULL`) indica que a ferramenta não contém interface com o usuário. Os demais parâmetros repassam para a rotina os valores enviados pelo ativador (Controle de Processos). Através destes comandos, qualquer processo passa a rodar no *SILEX* e pode fazer uso das mensagens e recursos do sistema. O formato de envio e recepção das mensagens é mostrado no anexo A-1.

As ferramentas sem interface geralmente são servidoras de recursos, ficando à espera de requisições. Esta operação pode dar-se pela seqüência de comandos abaixo:

```
for ( ; ; ) {
```

```
   switch ( RECEBE_M( ) )
   {
        case ARQUIVO :
            /* recebe indicacao do objeto de trabalho */
            break;
        case TECNOLOGIA :
            /* recebe indicacao da tecnologia de trabalho */
            break;
        case PROCESSA :
            /* comando ativador do processamento */
            break;
        case SOLICITA_FINALIZACAO :
            if ( /* verifica se a ferramenta pode ser finalizada */ )
               RESPONDE_M( SOLICITA_FINALIZACAO , OP_OK , REC_FON
);
            else
              RESPONDE_M( SOLICITA_FINALIZACAO , OP_BUG , REC_FON
);
            break;
        case FIM :
            _exit(0);
   }
}
```

O programa fica em ciclo, processando as mensagens recebidas e enviando resultados na forma de mensagens ou dados. Caso a ferramenta necessite enviar um conjunto de informações (dados) a outra ferramenta, deve anteriormente formar um canal através dos seguintes comandos:

```
ENV_VAL[0] = ID_FERRAM_DESTINO;
if ( PERGUNTA_M( INICIA_COMUNICACAO ) == OP_OK )
   /* sucesso na formacao do canal */
else
   /* falha na formacao do canal */
```

Caso a formação do canal tenha obtido sucesso, a ferramenta pode iniciar o envio dos dados. Falha na formação pode dar-se quando a ferramenta destino não se encontra mais em execução ou está envolvida em um processamento demorado.

6.1.2 Ferramentas Com Interface Gráfica

A ferramenta com interface gráfica, geralmente é cliente de recursos e difere na integração pelo controle de execução estar com o sistema de janelas. Este indica a função a ser executada pela ferramenta através de eventos, funções de chamada (*call-back functions*) e interposições.

Inicialmente devem ser especificados os eventos desejados para os diversos objetos da interface (*widgets*), por exemplo: a movimentação do *mouse* em uma janela de desenho (*canvas*); o pressionar dos botões do *mouse*; a abertura de uma janela. A ativação dos eventos é dada pela função abaixo, onde `objeto` é a *widget* na qual o `evento` está associado.

```
window_set(objeto, WIN_CONSUME_EVENTS, event, 0);
```

As *call-back functions* são funções ativadas pela interação com a interface, por exemplo: o pressionar de um "botão" de uma janela de controle (*panel*) ou; o redesenhar de um *canvas*. A atribuição de uma função de chamada a um objeto da interface é dada pelo comando abaixo.

```
window_set(objeto, PANEL_NOTIFY_PROC, funcao, 0);
```

A interposição consiste em incluir funções do usuário entre a ocorrência do evento e a função padrão do sistema, por exemplo: *Notify_resize_function* implica na chamada da função do usuário sempre que uma operação de mudança do formato da janela é executada. Dependendo do valor retornado pela função do usuário é ativada a função do sistema. Este comando é mostrado abaixo.

```
notify_interpose_resize_func(frame_base, funcao);
```

As ferramentas com interface necessitam, da mesma forma, a inclusão do arquivo "`silex.h`" e a chamada da função "`INICIA_FERRAM`", como é mostrado abaixo. A diferença está no parâmetro da função de iniciação que contém um apon-

tador para o descritor da janela base do sistema.

```
#include "silex.h"
int main ( argc , argv )
int   argc;
char * argv [ ];
{
  JAN_BASE = inicia_interface(); /* formacao da interface */
  INICIA_FERRAM( JAN_BASE , argc , argv );
  X_main_loop( JAN_BASE );
}
```

A função "X_main_loop" passa o controle de execução ao sistema de janelas. O programa apenas é acessado pelas notificações acima descritas. Desta forma o programa passa a conter uma função principal responsável pela iniciação (`main`) e um conjunto de funções de notificação. A forma de codificação mostra-se muito eficiente por forçar a modularidade do código.

6.2 Ferramentas Encapsuladas

O módulo encapsulador é composto por três funções:

- aquisição e preparação dos dados;
- ativação da ferramenta;
- interpretação do resultado.

A aquisição dos dados pode dar-se pela interface gráfica, onde o usuário entra com valores, ou por requisições ao controle de projetos por valores contidos em suas variáveis globais, como objeto de trabalho, tecnologia e *flags*.

O local de leitura e escrita dos arquivos da ferramenta encapsulada é um diretório temporário ($TMPHOME). Neste local é que devem estar disponíveis as informações a serem lidas e onde serão acessados os resultados.

A ativação dá-se pelos comandos:

```
/* Formacao do comando de ativacao da ferramenta */
sprintf( comando , " ferr %s %s ...", par1 , par2 , ... );
if ( ATIVA_FERRAM( comando , TMPHOME ) == OP_OK )
  /* operacao bem sucedida */
else
  /* operacao sem sucesso */
```

O valor retornado pelo comando de ativação indica o sucesso ou falha na execução. Em caso de sucesso, os dados gerados devem ser convertidos para o formato interno do *SILEX* e enviados à ID.

6.3 Estudo de caso

6.3.1 Integração do HDC

HDC é um simulador funcional de *hardware* descrito na linguagem de programação C ([MAR 91]). Foi escolhido como a primeira ferramenta a integrar o *SILEX* por possuir reduzida interação com o usuário, código regular, utilizar diversos recursos do sistema e; ter grande aceitação (no contexto do CPGCC-UFRGS).

É uma ferramenta relativamente nova e que aos poucos adquire facilidades de uso. O ambiente de programação inicial foi IBM-PC, sistema operacional DOS, compilador "Turbo-C" e interface gráfica dedicada. A arquitetura do simulador, neste ambiente de trabalho, não permite muita flexibilidade. O sistema operacional monoprocesso obriga a união da interface às descrições em simulação.

Os circuitos necessitam ser compilados e ligados ao simulador, isto faz com que cada circuito em simulação contenha todas as rotinas da interface. A figura 6.1 esquematiza esta arquitetura, onde pode ser vista a quantidade de código necessária para a simulação de cada circuito.

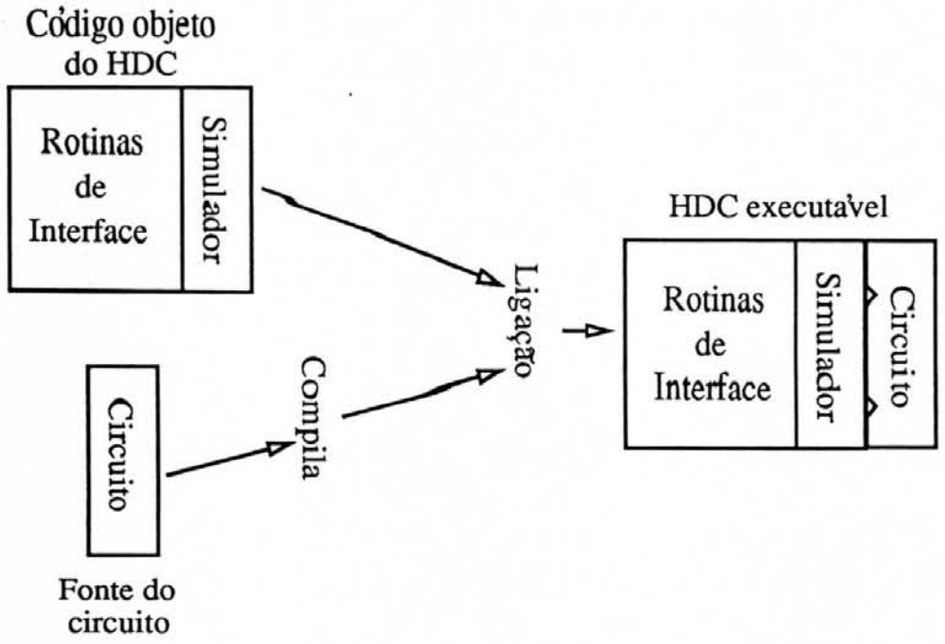

Figura 6.1: Arquitetura do HDC no PC

A arquitetura proposta para a ferramenta integrada no *SILEX* é mostrada na figura 6.2. A arquitetura traz facilidades de uso e utiliza de forma regular os recursos disponíveis pelo *SILEX* .

Com esta arquitetura consegue-se simular mais de um circuito utilizando uma mesma interface. O usuário indica qual o circuito a ser simulado e este passa a ser um novo processo em execução, comunicando-se com a interface por caminhos de mensagens e dados.

A figura 6.3 mostra um croqui da tela da ferramenta simulando um contador descrito em C. O anexo A-7 lista esta descrição, onde podem ser examinadas três regiões:

1. Região de inicialização: inicia as variáveis globais do simulador e a estrutura do circuito, através das funções `Inicializacao_parametros()` e `Inicial()`;

2. Região de descrição: permite a descrição do algoritmo de simulação em C, através da função `Descricao_Circuito()` e funções do usuário e;

3. Região de interface: define-se as variáveis e ciclos em exibição, através da função

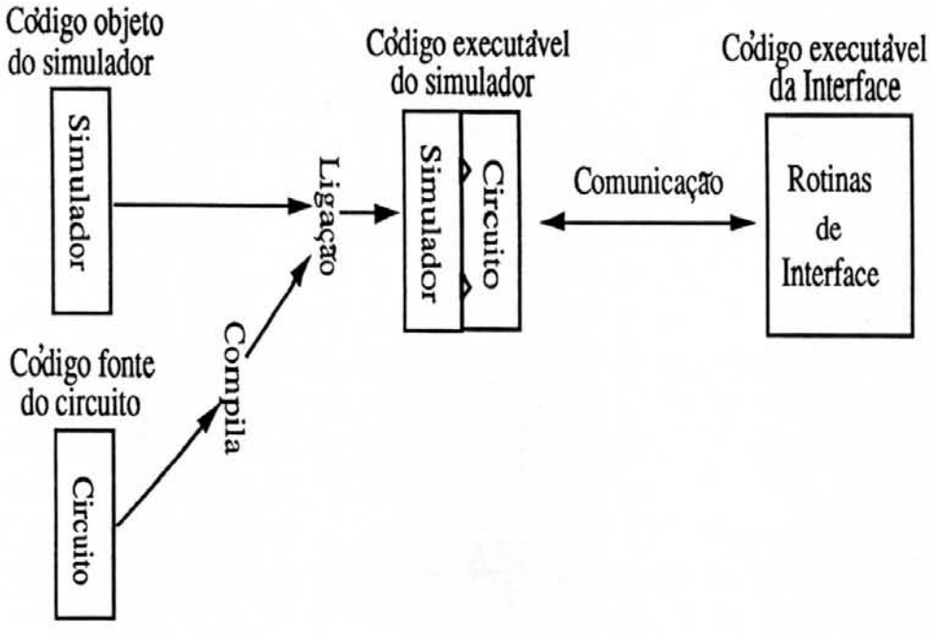

Figura 6.2: Arquitetura do HDC no *SILEX*

```
Exibicao().
```

Quando o circuito não está disponível para execução, a interface através de uma sub-janela, permite a edição da descrição e compilação automáticas. A figura 6.4 exibe esta sub-janela de edição. Caso sejam encontrados erros na descrição, estes são mostrados ao usuário. Caso o código fonte do circuito não exista, um esqueleto deste torna-se disponível. O usuário com base neste descreve o funcionamento do seu circuito. O formato deste esqueleto é listado abaixo:

Estrutura base do arquivo "circuito.c"

```
/*****************************************************************************
 * <arquivo>.C
 *****************************************************************************/

#include "<arquivo>.h"
#include "circuito.h"

/* inicializacao dos parametros globais */
void Inicializacao_parametros() { }

/* nodos que sao alterados durante a execucao da simulacao */
```

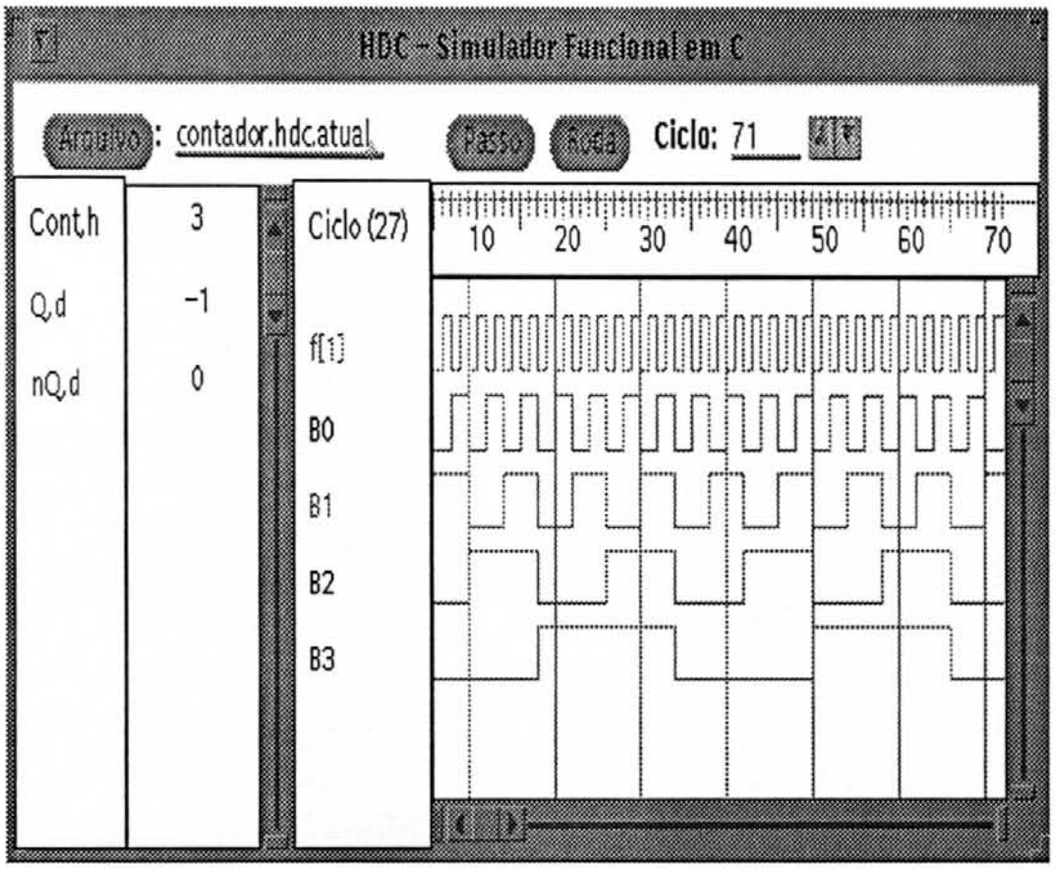

Figura 6.3: Croqui da janela do simulador HDC

```
void Altera_nodos() { }

/* inicializacao das variaveis */
void Inicial() { }

/* descricao do circuito */
void Circuito() { }

/* exibicao grafica das variaveis */
void Exibicao() { }
```

Estrutura base do arquivo "circuito.h"

```
/*****************************************************************************
 * <arquivo>.H
 *****************************************************************************/

typedef struct
{ /* variaveis de definicao do circuito */
```

} circuito;

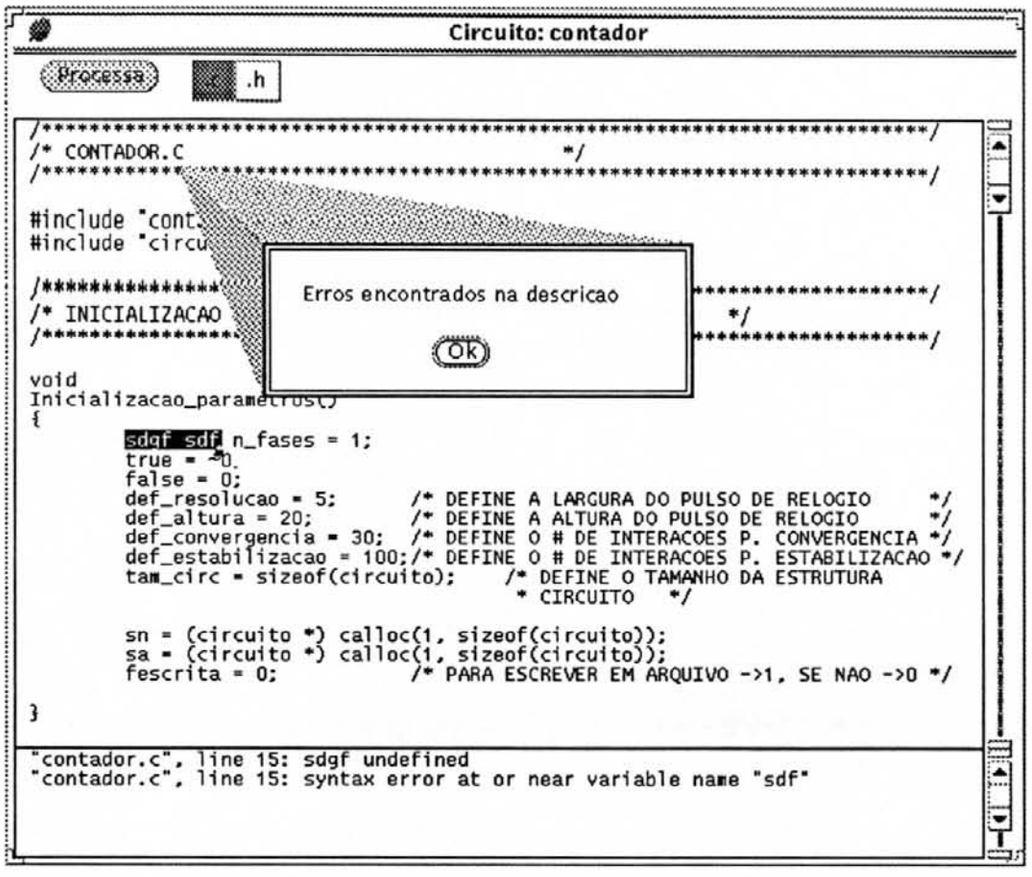

Figura 6.4: Janela de edição do circuito

As mensagens que fluem entre os processos na simulação da descrição de um contador são as seguintes:

1. ...
   { o usuário interage com o *SILEX* solicitando a ativação do HDC }
2. IG -> CPROC = SOLICITA_ATIVAÇÃO (HDC)
   < ativação da ferramenta HDC >
3. CPROC <-> HDC = ATIVAÇÃO_OK
   < teste do canal de comunicação entre HDC e CPROC >
4. IG <- CPROC = OP_OK
   < retorno de operação bem sucedida à IG >
   { o usuário interage com o HDC solicitando a ativação do CONTADOR }

5. HDC -> CPROC = SOLICITA_ATIVAÇÃO (CONTADOR)
   < ativação do circuito CONTADOR >

6. CPROC <-> CONTADOR = ATIVAÇÃO_OK
   < teste do canal de comunicação entre CONTADOR e CPROC >

7. HDC <- CPROC = OP_OK
   < retorno de operação bem sucedida ao HDC >

8. HDC -> CPROC = CANAL_DADOS (CONTADOR)
   < HDC solicita a comunicação de dados com o CONTADOR >

9. CPROC -> CONTADOR = CRIA_CANAL_DADOS
   < CONTADOR recebe a notificação e criação do canal de dados >

10. CPROC -> HDC = RECEBE_CANAL_DADOS
    < HDC conecta-se ao canal formado >

11. CPROC <- CONTADOR = OP_OK
    < retorno de operação bem sucedida ao CPROC >

12. CPROC <- HDC = OP_OK
    < retorno de operação bem sucedida ao CPROC >

13. HDC <-> CONTADOR = SETA_PARAMENTROS_INICIAIS
    < HDC e CONTADOR trocam parâmetros iniciais de funcionamento >

14. HDC -> CONTADOR = PASSO
    < pela interação o usuário solicita a simulação de um ciclo >
    < a simulação é realizada e os valores são enviados pelo canal de dados >

15. HDC <- CONTADOR = OP_OK
    < retorno de operação bem sucedida ao HDC >

16. HDC -> CONTADOR = RODA (número de ciclos)
    < pela interação o usuário solicita a simulação de um conjunto de ciclos >
    < a simulação é realizada e os valores são enviados pelo canal de dados >

17. HDC <- CONTADOR = OP_OK
    < retorno de operação bem sucedida ao HDC >

18. IG -> CPROC = SOLICITA_FIM (HDC)
    { pela iteração o usuário solicita finalização do HDC }

19. CPROC -> HDC = FIM
    < HDC recebe comando de finalização >

20. HDC -> CPROC = SOLICITA_FIM (CONTADOR)
    < o HDC solicita finalização do circuito CONTADOR >

21. CPROC -> CONTADOR = FIM
    < término de execução do CONTADOR >

22. HDC <- CONTADOR = OP_OK
    < retorno de operação bem sucedida ao HDC >

23. IG <- HDC = OP_OK
    < retorno de operação bem sucedida ao HDC >

6.3.2 Integração do Editor Simbólico CHARRUA

CHARRUA é uma ferramenta para a síntese de *layout* que recebe uma descrição simbólica do circuito e a converte para a descrição geométrica ([MAR 89]). Esta conversão é realizada obedecendo a um conjunto de regras de expansão independentes de tecnologia de fabricação (na família CMOS). Para tanto a ferramenta é composta por quatro módulos, abaixo esquematizados.

Os módulos, como mostra a figura 6.5, são os seguintes:

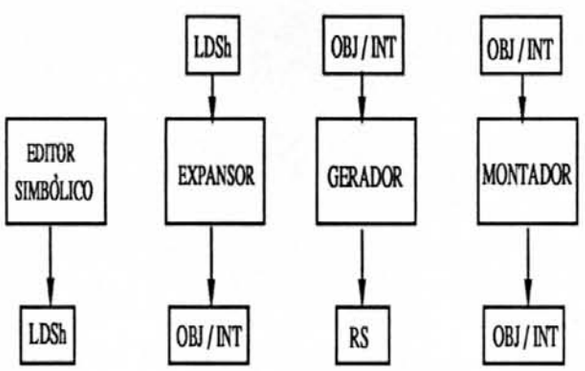

Figura 6.5: Arquitetura do Editor CHARRUA em PC

- O Editor Simbólico é uma ferramenta gráfica e interativa que permite a especificação de células de maneira simbólica, ou seja, os elementos geométricos padrões como transistores, contatos e fios são substituídos por símbolos. Os símbolos e suas interconexões, obedecendo a um conjunto de regras, formam o circuito.

- O Expansor calcula a separação mínima necessária entre os elementos simbólicos para que não ocorram violações sobre as regras de projeto. Existe um conjunto de regras para cada tecnologia.
- O Montador é encarregado da união das diversas células e módulos do circuito. Este recebe a informação de conectividade das interfaces das células e gera o módulo.
- O Gerador CIF/RS converte a descrição simbólica, somada com informações vindas do expansor e montador, para uma descrição de *layout*.

Um primeiro passo para a integração consiste em verificar a seqüência na qual as ferramentas são ativadas e os arquivos de entrada e saída nas diversas opções de funcionamento. A figura 6.6 ilustra este fluxo.

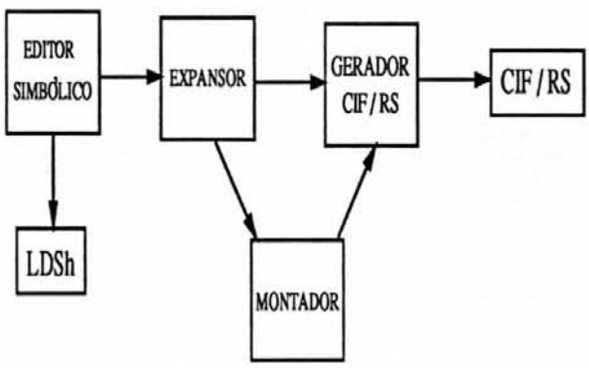

Figura 6.6: Editor CHARRUA no ambiente *SILEX*

A linguagem LDSh ([MAR 90b]) é utilizada para armazenar objetos simbólicos. Esta é textual e regular. Para facilitar a leitura da descrição pelas ferramentas processadoras, optou-se em utilizar uma descrição intermediária (representada por `INT` na figura 6.5), que tem direta leitura e interpretação.

No *SILEX* esta descrição intermediária é substituída pelo formato codificado da linguagem LDSh, como mostra o anexo A-4. Esta operação reduz a quantidade de dados armazenados, porque não é mais necessário salvar e carregar descrições entre a ativação dos processos.

A ferramenta central, pela visão do usuário, é o editor simbólico. Este possui interface gráfica e inicia a ativação dos demais processos que rodam sem

iteração com o usuário (*background*). Caso a descrição seja hierárquica é solicitada a ativação da ferramenta montadora.

7 CONCLUSÃO

A primeira versão do sistema está disponível em estações de trabalho SUN-SPARC, sistema operacional UNIX e gerenciador de janelas OpenWindows. Conta com 50 arquivos, sendo 20 códigos fontes na linguagem C. Estes arquivos foram codificados pelo autor, totalizando $5k$ linhas de código. Existem até o momento 11 programas executáveis que realizam as mais variadas funções do sistema, sendo que apenas dois são ferramentas do usuário.

Apesar do ambiente de implementação ser indicado para o desenvolvimento e utilização de sistemas de CAD, o domínio de suas características, facilidades e forma de utilização era relativamente reduzido por parte dos pesquisadores do GME-UFRGS no início do projeto. Uma fase de descobertas e desafios foi necessária, pois era grande a quantidade de utilitários, aplicativos e documentação.

O primeiro passo na concepção do *SILEX* foi a escolha do equipamento e sistema operacional necessários e disponíveis para suportar as características pensadas para o sistema. No início do projeto dispunha-se de computadores compatíveis com o padrão IBM-PC e sistema operacional DOS. Claramente estes recusos não são indicados para um ambiente integrado, como se propunha para o *SILEX* . Este permite apenas um baixo grau de integração, com o uso de programas seqüenciadores. Uma tentativa de integração para o *SILEX* neste ambiente está relatada em [PER 90], onde as principais características eram a interface gráfica unificada e a estrutura de dados global às ferramentas a nível de *layout.* O aparecimento de sistemas operacionais compatíveis com o UNIX em equipamentos IBM-PC trouxeram novas perspectivas para a implementação.

Um estudo sobre o sistema operacional XENIX (UNIX para PC) foi realizado, mostrando-se de grande utilidade. A idéia no momento era converter ferramentas para este ambiente, construir um gerenciador de janelas próprio (uma vez que não haviam disponíveis até então) e codificar as ferramentas servidoras, como

controle de processo, projeto e interface de dados.

A notícia da existência de gerenciadores de janelas compatíveis com padrão Xwindow para XENIX alteraram os planos de construção do gerenciador para a utilização deste. Passados aproximadamente um ano do início do projeto ainda não se dispunha do sistema operacional XENIX em um equipamento IBM-PC para o projeto. Passou-se então a cogitar a utilização de estações de trabalho do tipo SUN-SPARC, recém chegadas ao grupo de microeletrônica da universidade.

A dúvida inicial, neste novo contexto, se deu quanto ao sistema de janelas a ser utilizado. Xwindow por ser um padrão? Mas não dispunha de documentação apropriada nem primitivas de mais alto nível (*widgets*) para o tratamento de janelas. Sunview por estar muito bem documentado e instalado? Mas restringe a utilização a equipamentos do tipo SUN e não dispõe de mecanismos para a execução de processos distribuídos.

A este momento o autor realizou um estudo junto ao Núcleo de Computação e Eletrônica da Universidade Federal do Rio de Janeiro (NCE-UFRJ) a fim de aprimorar seu conhecimento sobre o sistema Xwindow. Este trabalho foi coordenado pelo professor Newton Faller que, no momento, convertia o sistema de janelas para a estação de trabalho (PEGASUS) projetada pelo seu grupo. Juntamente foi realizado um estudo de sistemas com características semelhantes às até então pensadas. Este estudo foi prejudicado pela escassez de documentação no nível de detalhamento necessário. A documentação encontrada relata resultados, conclusões e abstrai detalhes fundamentais de implementação.

O projeto *SILEX* foi iniciado baseado no sistema Sunview, devido à impossibilidade de utilização da versão disponível do Xwindow. Após alguns meses de início do projeto, surgiu o gerenciador OpenWindows. Imediatamente, partiu-se para a utilização deste, pois engloba as vantagens de ambos os sistemas, como a padronização, perfeita instalação e documentação.

Solucionado este problema, passou-se à definição da forma de integração e comunicação. Os *pipes* foram inicialmente utilizados obtendo sucesso na comunicação através de mensagens. Entre cada ferramenta com o controle de processos eram criados dois *pipes*, um para leitura e outro para escrita. A comunicação pela troca de dados, que pode dar-se entre ferramentas quaisquer, inviabilizou a utilização deste mecanismo, uma vez que os *pipes* comunicam "processos parentes" e devem estar presentes antes da ativação dos processos comunicantes. Passou-se ao estudo das alternativas disponíveis e chegou-se aos *sockets*. Estes superam as deficiências do modelo anterior e mostraram-se ideais para suportar o estado atual e futuras extensões do sistema.

Definida a base, passou-se à implementação das rotinas de comunicação, tratamento dos dados, conversão para o formato codificado, interface gráfica do *SILEX* , interação e construção das ferramentas. Montado o esqueleto do sistema levou-se em paralelo a integração do simulador funcional HDC, com a introdução de novas funções.

O primeiro problema encontrado foi a falta de sincronismo na recepção de mensagens. A definição inicial não previa que diversas ferramentas utilizassem o mesmo canal para o envio e recepção de mensagens. Para solucionar este problema foi implementado um esquema de fila auxiliar na troca de mensagens.

Solucionados os problemas até então encontrados, conseguiu-se uma versão estável. Iniciou-se a integração da ferramenta Editor Simbólico que vinha sendo convertida para o ambiente de janelas. A versão anterior desta ferramenta possuía interface gráfica dedicada. As ferramentas HDC e Editor Simbólico utilizam muitos dos recursos disponíveis.

Os passos a seguir são a formação de um conjunto mínimo de ferramentas para tornar o *SILEX* utilizável no projeto de circuitos. Em paralelo à implementação do *SILEX* , um conjunto de ferramentas de CAD tem sido convertida e adaptada para o sistema de janelas e estações de trabalho. Este trabalho é coordenado pelo

engenheiro (MSc Computação) Luigi Carro e conta com aproximadamente uma dezena de ferramentas. Estas são: GGMOD ([SOT 91]), CHARRUA ([OLI 91]), SELA ([CAS 91]), ESQUELETO ([VAS 91]), STX ([FRA 91]), entre outras. Um conjunto de ferramentas (aproximadamente 15) pertencentes ao sistema TENTOS ([MOR 91]) estão sendo convertidas para estações de trabalho, uma vez que inicialmente implementadas em microcomputadores (do tipo IBM-PC) e passarão a integrar o *SILEX* .

Após a construção deste ambiente torna-se necessário a inclusão de um efetivo controle de projetos. Este trará um valioso retorno ao usuário, permitindo a visualização do estado do projeto. A distribuição da execução de processos na rede pode ser necessária ([ROS 91], [FRE 91]), neste momento, e não traz dificuldades maiores, uma vez que o sistema operacional e os *sockets* cooperam para tal.

Espera-se que os usuários, com base no que foi até então exposto, motivem-se a utilizar e complementar o sistema com novos algoritmos. De pouco vale um sistema integrado que dispõe de um pobre conjunto de ferramentas. São estas que vão indicar o real valor do sistema. Com a possibilidade de integração de novas ferramentas e oferecendo os recursos da interface gráfica, controle de processos e interface de dados, o *SILEX* pode tornar-se um ambiente muito atraente tanto para os especialistas em CAD ,quanto para os especialistas em projeto de de circuitos integrados.

ANEXO A-1 FUNÇÕES DISPONíVEIS NO SISTEMA

Um conjunto de funções para as mais variadas finalidades estão disponíveis para utilização. Estas funções estão contidas no arquivo (*~/src/include/pipe.h*), e são detalhadas abaixo.

TROCA DE MENSAGENS ENTRE PROCESSOS:

As mensagens são registros de tamanho definido. São escritas em um lado do canal de comunicação e lidas na outra extremidade. O intervalo de tempo no qual esta operação deve se efetivar é definido pelo administrador do sistema. Este valor é expresso em segundos e tem o seguinte formato:

`TEMPO_ESPERA` = < segundos > (*default* = 20)

As funções encarregadas do envio e recepção de mensagens são as seguintes:

1. ENVIA_M(MENS), envia a mensagem "MENS" para o controle de processos.
2. ENVIA_MD(MENS,DEST), envia a mensagem "MENS" para a ferramenta identificada pelo seu descritor "DEST".
3. RECEBE_M(), espera por uma mensagem, retornando o código da mesma. Caso a mensagem não tornar-se disponível no período de tempo especificado, é retornado o valor "TEMPO_ESTOURADO".
4. PERGUNTA_M(MENS), envia a mensagem "MENS" para o controle de processos e aguarda resposta. Esta função, enquanto aguarda a resposta, armazena outras mensagens que não a esperada. Com isto consegue-se um sincronismo entre as mensagens.

5. PERGUNTA_MD(MENS,DEST), semelhante a função anterior, sendo que realiza a pergunta a uma ferramenta qualquer.

6. RESPONDE_M(PERG,MENS), responde à pergunta "PERG", enviando a mensagem "MENS".

7. RESPONDE_MD(PERG,MENS,DEST), semelhante a função anterior, sendo que responde à pergunta vinda de uma ferramenta diferente do controle de processos.

8. ESPERA_M(PERG), espera a resposta para a pergunta "PERG".

Juntamente com o código das mensagens é enviada uma estrutura com informações auxiliares. Esta estrutura (u_mens) é listada no anexo A-6. Para facilitar o acesso às informações armazenadas são definidos os seguintes comandos:

- REC, estrutura (u_mens) que contém a mensagem recebida
- REC_VAL, vetor de números inteiros
- REC_TXT, campo textual
- REC_OBJ, objeto
- REC_LIN, representação
- REC_VER, versão
- ENV, estrutura (u_mens) para a mensagem a ser enviada
- ENV_VAL, vetor de números inteiros
- ENV_TXT, campo textual
- ENV_OBJ, objeto
- ENV_LIN, representação
- ENV_VER, versão

ANEXO A-2 ARQUIVOS DESCRITORES DE PROJETOS

Conteúdo do arquivo descritor do projeto GERAL (*~/proj/GERAL.prj*)

```
#PROJETO  GERAL
#USUARIOS  TODOS

#MENSAGEM
Este projeto e' de uso geral, todos os usuarios tem acesso.
O local de armazenagem dos dados e' compartilhado e deve conter
apenas informacoes temporarias, como pequenos testes.
#DIRETORIO  /home/rubi/giba/proj/GERAL.dir
#OBJETOS /home/rubi/giba/proj/GERAL.obj
```

Conteúdo do arquivo de ferramentas do projeto GERAL (*~/proj/GERAL.fer*)

```
#Editores:
Mascara  sela
Simbolico  ei
Esquematico  esqueleto

#Simuladores:
Spice  spice
Aramos  aramos
SHC  hdc

#Controle:
Projeto  contproj
Versoes contver
```

Conteúdo do arquivo de objetos do projeto GERAL (*~/proj/GERAL.obj*)

```
#OBJETOS
reg hdc atual 1
reg.c hdc atual 2
reg.h hdc atual  3
reg spice atual 4
reg spice v_inicio 5
contador.c hdc atual 6
contador.h hdc atual 7
contador hdc atual 8
carry hdc atual 9
carry.c hdc atual 10
carry.h hdc atual 11
fft hdc atual 12
fft.c hdc atual 13
fft.h hdc atual 14
pla hdc atual 15
pla.c hdc atual 16
pla.h hdc atual 17
```

ANEXO A-3 PARSER DA LINGUAGEM LDS (*LEX* E *YACC*)

Analisador Léxico da linguagem LDS

Conteúdo do arquivo ~/*src/parser/lds.lx*

```
%{
/*****************************************************************************
      Analisador Lexico para a linguagem LDS h
            Versao: 15/01/90
        Por: Gilberto Marchioro
*****************************************************************************/
/*
 * Funcao : YYLEX
 *
 * Realiza a analise lexica, retornando o tipo de token. Para tokens
 * variaveis, a union yylval e alimentada.
 *
 * O arquivo de entrada yyin deve estar previamente aberto
 * Os erros na analise irao para o arquivo yyerr que deve estar aberto
 *
 * Os tokens sao os seguintes:
 *   - variaveis:          NUM_INTEIRO NUM_FLOAT PALAVRA STRING
 *   - palavras reservadas: INICIO FIM TECNOL PROF ELEM PARAM
 *                  TRANS FIO CONT PINO INSTANCIA POCO
 *                  OR REP SEP DESCR CONEC NOME SERIE
 *                  GRUPO FIM_GRUPO WL MONTA
 *                  NL NO SL LN LS ON OS
 * Os valores referentes as variaveis retornam em uma union :
 * %union
 * { int   intg;
 *   float flt;
 *   char  *str;
 *   char **lstr;
 *   void *pnt;
 * } ;
 * OBS: o campo yylval.str deve ser liberado apos sua utilizacao
 */
%}
```

```
%%
[ \t\n]              ;
[\,\=\(\)\*\+\#\:\<\>]        return (yytext[0]);
\;.*                          {
   return (';');
   /* linha comentario ignorada */
}
"/*"[^"*/"]*"*/"        /* ignora comentarios */;
[0-9]+                {
   sscanf(yytext, "%d", &(yylval.intg));
   return (NUM_INTEIRO);
}
[\-]?[0-9]+[\.[0-9]+]? {
   sscanf(yytext, "%f", &(yylval.flt));
   return (NUM_FLOAT);
}
\"[^\"]*\"          {
   yytext[--yyleng] = '\0';
   yylval.str = (char *) strdup(yytext + 1);
   return (STRING);
}
"INICIO"        { return(INICIO);    }
"FIM"           { return(FIM);       }
"TECNOL"        { return(TECNOL);    }
"PROF"              { return(PROF);      }
"ELEM"              { return(ELEM);      }
"PARAM"             { return(PARAM);     }
"TRANS"             { return(TRANS);     }
"WL"                { return(WL);        }
"FIO"               { return(FIO);       }
"CONT"              { return(CONT);      }
"PINO"              { return(PINO);      }
"POCO"              { return(POCO);      }
"GRUPO"             { return(GRUPO);     }
"FIM_GRUPO"           { return(FIM_GRUPO); }
"INSTANCIA"           { return(INSTANCIA); }
"MONTA"             { return(MONTA);     }
"OR"                { return(OR);        }
"NL"                { return(NL);        }
"NO"                { return(NO);        }
"SL"                { return(SL);        }
"LN"                { return(LN);        }
```

```
"LS"                { return(LS);       }
"ON"                { return(ON);       }
"OS"                { return(OS);       }
"REP"               { return(REP);      }
"SEP"               { return(SEP);      }
"DESCR"             { return(DESCR);    }
"CONEC"             { return(CONEC);    }
"NOME"              { return(NOME);     }
"SERIE"             { return(SERIE);    }
[a-zA-Z][a-zA-Z0-9\_]*   {
   if (yyleng > 1) {
      yylval.str = (char *) strdup(yytext);
      return (PALAVRA);
   } else
      return (yytext[0]);
}
.          printf("Caractere invalido \'%c\'=%d\n",yytext[0],(int) yytext[0]);
%%
```

Analisador Sintático da linguagem LDS

Conteúdo do arquivo *~/src/parser/lds.yy*

```
/*************************************************************************
      Analisador Sintatico para a linguagem LDSh
            Versao: 20/10/90
        Por: Gilberto Marchioro
*************************************************************************/
/*
 * Funcao : YYPARSE
 *
 * Realiza a analise sintatica alimentando a estrutura do editor
 */
%start programa
%union
{ int    intg;
  float  flt;
  char  *str;
  char **lstr;
  void  *pnt;
}
```

```
/* TERMINAIS */
%token <intg>  NUM_INTEIRO
%token <flt>   NUM_FLOAT
%token <str>   PALAVRA STRING
%token <notype> INICIO FIM TECNOL PROF ELEM PARAM TRANS FIO
CONT PINO INSTANCIA POCO
%token <notype> TOR REP SEP DESCR CONEC NOME SERIE GRUPO FIM_GRUPO
WL MONTA
%token <notype> NL NO SL LN LS ON OS OR

/* NAO TERMINAIS */
%type <intg>    primitiva /* grupo instancia */ c_tipo
%type <intg>    num orient tipo_s
%type <intg>    tipo_p /* sentido */
%type <flt>   real
%type <str>    /* string */  palavra

%{
/* CODIGO C */
#include <ctype.h>
#include <stdio.h>
#include "silex.h"
#include "../include/def_lds.h"

t_u_LDS_GLOBAL   com;
t_D_TEXTOS lista[NUM_MAX_TEXTOS];

t_ferram *lds;
%}
%%

programa   : comandos
/*       | hierarquia */;

comandos   :   declaracao ';' l_comandos  FIM ';'      { envia_dado(LDS_FIM);
}
       | /* vazio */
       ;

/* hierarquia      : monta texto ';' hierarquia
            | ';'
            | ;*/
```

```
l_comandos     : primitiva    { envia_dado($1);     }
               texto ';' l_comandos
/*               | grupo     { envia_dado($1);     }
               texto ';' l_comandos */
/*         | instancia    { envia_dado($1);     }
               texto ';' l_comandos */
               | ';'
      | ;

primitiva    : transistor          { $$=LDS_TRANS; }
      | fio                  { $$=LDS_FIO;    }
      | contato              { $$=LDS_CONT;    }
      | pino                 { $$=LDS_PINO;    }
      | poco                 { $$=LDS_POCO;    } ;

/*   DECLARACAO    */

declaracao     : INICIO palavra d_dimen d_tecn d_prof d_taman d_param {
          EMPILHA_TEXTO($2,&(lds→env_dado.un.celula));
          envia_dado(LDS_INICIO); } ;

d_dimen        : '(' num ',' num ')'     {
           lds→env_dado.un.inicio.dim_x= $2;
          lds→env_dado.un.inicio.dim_y= $4; }
               | ;

d_tecn      : TECNOL '=' palavra   { EMPILHA_TEXTO($3,&(lds→env_dado.un.inicio.tecnol))
}
               | ;

d_prof         : PROF '=' num         { lds→env_dado.un.inicio.prof= $3; }
               | ;

d_taman        : ELEM '=' num         { lds→env_dado.un.inicio.tam= $3; }
               | ;

d_param      : PARAM '=' '<'   { lds→env_dado.un.inicio.n_id_1= lds→env_dado.num_str+1;
}
         d_lista ':'     { lds→env_dado.un.inicio.n_id_2= lds→env_dado.num_str+1;
}
          d_lista '>'
               | ;
```

```
d_lista      : palavra        { int i; EMPILHA_TEXTO($1,&i); }
        ',' d_lista
          | palavra          { int i; EMPILHA_TEXTO($1,&i); } ;

/*   TRANSISTOR    */

transistor     : TRANS tipo_s t_posic orient t_fator  {
        lds→env_dado.un.trans.cam= $2;
        lds→env_dado.un.trans.orient = $4; } ;
t_posic        : '(' num ',' num ')'     {
        lds→env_dado.un.trans.x= $2;
         lds→env_dado.un.trans.y= $4; } ;
t_fator        : 'W' '*' real      { lds→env_dado.un.trans.w    = $3; }
          | 'L' '*' real       { lds→env_dado.un.trans.l    = $3; }
          | WL '=' real        {
        if ($3<1.0)  lds→env_dado.un.trans.l= 1.0/$3;
        else     lds→env_dado.un.trans.w= $3; }
          | ;

/*   FIO    */

fio            : FIO tipo_p f_posic f_fator {
        lds→env_dado.un.fio.cam= $2; } ;
f_posic        :  '(' num ',' num ')' '(' num ',' num ')' {
        lds→env_dado.un.fio.x1= $2;
        lds→env_dado.un.fio.y1= $4;
        lds→env_dado.un.fio.x2= $7;
        lds→env_dado.un.fio.y2= $9; } ;
f_fator        : 'W' '*' real         { lds→env_dado.un.fio.w= $3; }
      | ;


/*   CONTATO    */

contato        : CONT c_tipo c_posic { lds→env_dado.un.contato.cam = $2; }
c_tipo         : 'P'          { $$ = CP; }
      | 'D'          { $$ = CD; }
      | 'B'          { $$ = CB; }
      | 'S'          { $$ = CS; }
      | 'V'          { $$ = CV; } ;
c_posic        : '(' num ',' num ')'     {
```

```
        lds→env_dado.un.contato.x    = $2;
        lds→env_dado.un.contato.y    = $4; } ;

/*   PINO    */

pino        : PINO palavra tipo_p p_posic {
        EMPILHA_TEXTO($2,&(lds→env_dado.un.pino.nome));
        lds→env_dado.un.pino.cam= $3; } ;
p_posic     : '(' num ',' num ')'    {
        lds→env_dado.un.pino.x= $2;
        lds→env_dado.un.pino.y= $4; } ;

/*   POCO    */

poco        : POCO tipo_s w_posic    { lds→env_dado.un.poco.cam    = $2; } ;
w_posic     : '(' num ',' num ',' num ',' num ')' {
        lds→env_dado.un.poco.x1= $2;
        lds→env_dado.un.poco.y1    = $4;
                lds→env_dado.un.poco.x2    = $6;
        lds→env_dado.un.poco.y2    = $8; } ;

/*   GRUPO    */

/*grupo        : GRUPO PALAVRA g_atrib    {
        EMPILHA_TEXTO($2,&(lds->env_dado.un.grupo.nome));
        envia_dado(LDS_GRUPO); }
            lista_primitivas
          FIM_GRUPO        {
        $$= LDS_FIM_GRUPO; } ;

g_atrib     : ',' num        { lds->env_dado.un.grupo.num    = $2; }
          | ;
*/
/*   INSTANCIA    */

/*instancia     : INSTANCIA palavra i_posic i_orient i_repet {
        EMPILHA_TEXTO($2,&(lds->env_dado.un.instancia.nome));
        $$= LDS_FIM_INSTANCIA } ;
i_posic     : '(' num ',' num ')'    {
        lds->env_dado.un.instancia.x    = $2;
        lds->env_dado.un.instancia.y    = $4; } ;
i_orient    : OR '=' orient    { lds->env_dado.un.instancia.orient= $3; }
```

```
        | ;
i_repet         : REP '=' num ',' '(' orient ')' {
            lds->env_dado.un.instancia.repeticao= $3;
            lds->env_dado.un.instancia.sentido= $6; };
        | ;
*/
/*    MONTA     */

/*monta           : MONTA orient palavra '=' lista_celulas {
            lds->env_dado.un.monta.sentido= $2;
            EMPILHA_TEXTO($3,&(lds->env_dado.un.monta.nome));
            envia_dado(LDS_FIM_GRUPO); } ;

lista_celulas   : celula m_opcao    { envia_dado(LDS_FIM_GRUPO); }
                  lista_celulas
                  | celula          { envia_dado(LDS_FIM_GRUPO); } ;
m_opcao           : '+'             { lds->env_dado.un.monta_1.op_montagem= OPM+; }
                  | '#'             { lds->env_dado.un.monta_1.op_montagem= OPM#; }
                  | '+' num '+'       {
            lds->env_dado.un.monta_1.op_montagem= OPMS;
            lds->env_dado.un.monta_1.separ=$2; } ;
celula          : palavra m_orient m_repet m_descr m_conex mascara ;
m_orient        : OR '=' orient     { lds->env_dado.un.monta_1.orien= $3; }
        | ;
m_repet         : REP '=' num           { lds->env_dado.un.monta_1.repet= $3; }
        | REP '=' num ',' orient {
            lds->env_dado.un.monta_1.repet= $3;
            lds->env_dado.un.monta_1.orien= $5; } ;
m_descr       : DESCR '=' palavra   { EMPILHA_TEXTO($3,&(lds->env_dado.un.monta_1.descr)
}
        | ,
m_conex         : CONEC '=' m_opcoes
        | ;
m_opcoes        : NOME          { lds->env_dado.un.monta_1.tipo_conec= OPCN; }
        | SERIE         { lds->env_dado.un.monta_1.tipo_conec= OPCS; }
        | '('           {
            lds->env_dado.un.monta_1.conex= OPCL;
            lds->env_dado.un.monta_1.lista_1= lds->env_dado.num_str; }
          lista_pinos ':'   { lds->env_dado.un.monta_1.lista_2= lds->env_dado.num_str;
}
            lista_pinos ')' ;
lista_pinos     : palavra         { int i; EMPILHA_TEXTO($1,&i); }
            ',' lista_pinos
```

```
            | palavra        { int i; EMPILHA_TEXTO($1,&i); } ;
*/

/*   TEXTO    */

texto       : STRING t_atrib t_relat {
            EMPILHA_TEXTO($1,&(lds→env_dado.un.texto.texto));
            envia_dado(LDS_TEXTO); }
      | ;
t_atrib     : ',' real        { lds→env_dado.un.texto.atrib= $2; }
      | ;
t_relat     : '(' num ',' num ')'    { lds→env_dado.un.texto.x= $2;
              lds→env_dado.un.texto.y= $4; }
      | ;

/*   GERAIS    */
orient      : 'H'          { $$= OH; }
            | 'V'          { $$= OV; } ;

tipo_p      : 'P'          { $$= CP; }
            | 'D'          { $$= CD; }
            | 'B'          { $$= CB; }
            | 'M'          { $$= C1; }
            | 'm'          { $$= C2; } ;

/* sentido      : NL            { $$= SNL; }
            | NO            { $$= SNO; }
            | SL            { $$= SSL; }
            | LN            { $$= SLN; }
            | LS            { $$= SLS; }
            | ON            { $$= SON; }
      | OS          { $$= SOS; } ; */

tipo_s      : 'P'          { $$= Pn; }
      | 'N'         { $$= Np; } ;

/*   TIPOS DEFINIDOS    */

num         : NUM_INTEIRO       { $$ = yylval.intg; } ;
real        : NUM_FLOAT         { $$  = yylval.flt; }
      | NUM_INTEIRO       { $$  = (float ) yylval.intg; } ;
palavra     : PALAVRA           { $$  = yylval.str; } ;
```

```
/* string        : STRING              { $$ = yylval.str; } ; */

%%

#include "lex.yy.c"

int main(argc,argv)
int argc;
char *argv[];
{
   if (INICIA_FERRAM(&lds,argc,argv))
      _exit(1);

   lds→env_dado.uni=&com;
   lds→env_dado.texto=lista;

   lds→num_str=0;
   memset(&(lds→env_dado.uni),0,sizeof(t_u_LDS_GLOBAL));

   switch(RECEBE_M(lds))
   {
      case ATIVA_PIPE_DADO    :
         lds→DADO=lds→rec_men.un.valor[0];
         break;
      case PROCESSA    :
         if (!lds→DADO)
            ENVIA_M(lds,OP_BUG);
         else
         {
            yyin=lds→DADO;
            envia_dado(CABECALHO);
            envia_dado(LDS);
              yyparse();
            envia_dado(CABECALHO_FINAL);

            if (num_err) ENVIA_M(lds,BUG_PROCESSO);
            else ENVIA_M(lds,OP_OK);
         }
         break;
      case FINALIZA    : _exit(0);
   }
}
```

```
envia_dado(com)
int com;
{
    int i;
   lds→env_dado.comando=com;
   ENVIA_COMANDO(lds,sizeof(t_u_LDS_GLOBAL));
   for (;lds→env_dado.num_str;--lds→env_dado.num_str)
      free(lds→env_dado.texto[lds→env_dado.num_str]);
   memset(&(lds→env_dado.uni),0,sizeof(t_u_LDS_GLOBAL));
}

yyerror(s)  { printf("%s\n",s); }

yywrap() {  return(1);  }
```

ANEXO A-4 DEFINIÇÃO DA ESTRUTURA E FORMATO DE COMUNICAÇÃO DA LINGUAGEM LDS

Conteúdo do arquivo *~/src/include/def_lds.h*

```
/*   DEFINICAO DA ESTRUTURA E COMUNICACAO DA LINGUAGEM LDS
*/

enum {   OH, OV   };
enum {   CP, CD, CB, C1, C2, CS, CV   };
enum {   SNL, SNO, SSL, SLN, SLS, SON, SOS   };
enum {   Pn, Np   };
enum {   OPM, OPV, OPS   };
enum {   OPCN, OPCS, OPCL   };
enum {     LDS_INICIO = 1, LDS_FIM, LDS_TRANS, LDS_FIO, LDS_CONT,
LDS_PINO,
  LDS_POCO, LDS_GRUPO, LDS_FIM_GRUPO, LDS_INSTANCIA, LDS_MONTA,
   LDS_TEXTO   };

typedef struct {        /* INICIO */
   STR            celula, tecnol
   int            dim_x, dim_y;
   int            prof, tam;
   STR            n_id_1, n_id_2;
}            t_LDS_INICIO;

/* FIM */
typedef struct {        /* TRANS */
   char           cam;
   int            x, y;
   char           orient;
   float          w, l;
}            t_LDS_TRANS;

typedef struct {        /* FIO */
   char           cam;
   int            x1, y1, x2, y2;
```

```
    float           w;
}               t_LDS_FIO;

typedef struct {        /* CONTATO */
    char            cam;
    int             x, y;
}               t_LDS_CONTATO;

typedef struct {        /* PINO */
    STR             nome;
    char            cam;
    int             x, y;
}               t_LDS_PINO;

typedef struct {        /* POCO */
    char            cam;
    int             x1, y1, x2, y2;
}               t_LDS_POCO;

typedef struct {        /* GRUPO */
    STR             nome;
    int             num;
}               t_LDS_GRUPO;

/* FIM_GRUPO */
typedef struct {        /* INSTANCIA */
    STR             nome;
    int             x, y;
    char            orient;
    int             repeticao;
    char            sentido;
}               t_LDS_INSTANCIA;

typedef struct {        /* MONTA */
    STR             nome;
    char            sentido;
}               t_LDS_MONTA;

typedef struct {        /* MONTA_1 */
    STR             nome, descricao;
    char            orient;
    int             repet;
```

```
    char            sentido;
    char            tipo_conec;
    int             lista_1, lista_2;
    char            op_montagem;
    int             sep_montagem;
}               t_LDS_MONTA_1;

typedef struct {        /* TEXTO */
    STR             texto;
    float           atrib;
    int             x, y;
}               t_LDS_TEXTO;

typedef union {             /* UNIAO */
    t_LDS_INICIO    inicio;
    t_LDS_TRANS     trans;
    t_LDS_MONTA     monta;
    t_LDS_MONTA_1   monta_1;
    t_LDS_INSTANCIA instancia;
    t_LDS_GRUPO     grupo;
    t_LDS_POCO      poco;
    t_LDS_PINO      pino;
    t_LDS_CONTATO   contato;
    t_LDS_FIO       fio;
    t_LDS_TEXTO     texto;
}               t_u_LDS_GLOBAL;
```

ANEXO A-5 DEFINIÇÃO DO PROTOCOLO DE TROCA DE DADOS

Conteúdo do arquivo *~/src/include/pipe_dados.h*

```
/*    DEFINE O PROTOCOLO DE DADOS     */

/*
 * Forma de transmissao de arquivos:
 *    CABECALHO = 0
 *    LINGUAGEM = (CIF|LDS|SPICE)
 *    LISTA_COMANDOS
 *    CABECALHO_FINAL = ~0
 *
 * LISTA_COMANDOS
 *    ESTRUTURA DO COMANDO
 *    [ STRINGS
 *      PARAMETROS_ESPECIAIS ]
 *
 * STRINGS
 *    TAMANHO
 *    STRING
 */

/* LINGUAGENS */

enum {    CIF = 1, LDS, SPICE    };

#define CABECALHO    0
#define CABECALHO_FINAL    ~0
#define NUM_MAX_TEXTOS 20

typedef struct {        /* LISTA_TEXTOS */
   int            tam;
   char           *str;
}            t_D_TEXTOS;

typedef int     STR;

typedef struct {        /* GLOBAL */
```

```
    int             comando;
    int             num_str;
    void            *un;
    t_D_TEXTOS      **texto;
}           t_DADO;
```

ANEXO A-6 DEFINIÇÃO DO PROTOCOLO DE TROCA DE MENSAGENS

Conteúdo do arquivo ~/*src/include/pipe_mens.h*

```
/*   Definicao da comunicacao de dados via PIPE     */

#define RESERVADO 1000
/* MENSAGENS GLOBAIS */

enum {
   ENVIA_TELA = 0,
   ARQUIVO_NAO_ENCONTRADO,
   ATIVA_PIPE_DADO,
   BIBLIOTECA,
   BUG_GERAL,
   BUG_PROCESSO,
   CARGA_PROJETO,
   CONDICOES_CARGA,
   CONDICOES_SALVA,
   DESCRITOR_INCONSISTENTE,

   DESTROY_NEGADO,
   DIRETORIO,
   DIRETORIO_NAO_ENCONTRADO,
   EXIT,
   FIM,
   FIM_LISTA,
   FIM_OK,
   FIM_RECEPCAO,
   FIM_SALVAMENTO,
   FIM_TRANSMISSAO,

   FINALIZA,
   INICIO_CARGA,
   INICIO_ENVIO,
   INICIO_FLUXO,
   INICIO_RECEPCAO,
   INICIO_TRANSMISSAO,
   INSERE_OBJETO,
```

LIBERA_PIPE_DADO,
LINGUAGEM,
LISTA_ARQUIVOS,
LISTA_FERRAMENTAS,

LISTA_LINGUAGENS,
LISTA_OBJETOS,
LISTA_PROJETOS,
LISTA_VERSOES,
MENSAGEM_PERDIDA,
MENSAGEM_PRONTA,
MENSAGEM_RECEBIDA,
MONTA_PATH,
MORTE_FILHO,
MOVE_TO_ID,

MOVE_TO_TMP,
OBJETO,
SETA_OBJETO,
OP_BUG,
OP_OK,
PARSER_NAO_ENCONTRADO,
PIPE_DADO,
PODE_FINALIZAR,
PROCESSA,
PROJETO,

QUAL_DESCRITOR,
RECEBE_PIPE_DADO,
SEM_PERMISSAO,
SOLICITA_ATIVACAO,
SOLICITA_CARGA,
SOLICITA_CARGA_FLUXO,
SOLICITA_FIM,
SOLICITA_FLUXO_CARGA,
SOLICITA_FLUXO_SALVA,
SOLICITA_SALVA,

TESTA_CANAL,
RETIRA_OBJETO,
RETRANSMISSAO,
VERIFICA_OBJETO,

```
    VERIFICA_PROJETO,

    /* MENSAGENS DO HDC */
    ENVIA_VAR,
    LISTA_NOMES_CICLO,
    LISTA_NOMES_VAR,
    NUM_ONDAS,
    PASSO,
    RECEBE_VAR,
    RODA
};


typedef struct {        /* ARQUIVO */
    char            objeto[30], linguagem[30], versao[30];
}               t_PIPE_ARQUIVO;

typedef union {
    char            texto[80];
    int             valor[10];
    t_PIPE_ARQUIVO  arquivo;
}               u_mens;

typedef struct s_MENS {         /* GLOBAL */
    int             fonte, destino;
    int             mens, sincr;
    u_mens          un;
    struct s_MENS  *prox;

}               t_MENS;
```

ANEXO A-7 DESCRIÇÃO DO HARDWARE EM HDC

Conteúdo do arquivo *contador.c*

```
/*****************************************************************************/
/* CONTADOR.C                                                                */
/*****************************************************************************/

#include "contador.h"
#include "circuito.h"

/*****************************************************************************/
/* INICIALIZACAO DOS PARAMETROS GLOBAIS                                      */
/*****************************************************************************/

void
Inicializacao_parametros()
{
   n_fases = 1;
   true = ~0;
   false = 0;
   def_resolucao = 5;   /* DEFINE A LARGURA DO PULSO DE RELOGIO    */
   def_altura = 20;   /* DEFINE A ALTURA DO PULSO DE RELOGIO     */
    def_convergencia = 30;    /* DEFINE O # DE INTERACOES P. CONVER-
GENCIA */
  def_estabilizacao = 100;/* DEFINE O # DE INTERACOES P. ESTABILIZACAO
*/
   tam_circ = sizeof(circuito);   /* DEFINE O TAMANHO DA ESTRUTURA
                * CIRCUITO   */

   sn = (circuito *) calloc(1, sizeof(circuito));
   sa = (circuito *) calloc(1, sizeof(circuito));
   fescrita = 0;        /* PARA ESCREVER EM ARQUIVO ->1, SE NAO ->0 */

}

/*****************************************************************************/
/* NODOS QUE SAO ALTERADOS DURANTE A EXECUCAO DA SIMULA-
CAO COM A TECLA <F4> */
```

```
/* */
/* Nodo(char *, int *, int, int);                    */
/* */
/* Nodo("Idetificacao", &resultado, variavel_a_ser_alterada, radical);  */
/* */
/* radical: 16 para representacao hexa ou 10 para representacao em decimal */
/* */
/* obsevacao: o conteudo apontado por "sa" deve ser igual ao de "sn"    */
/*************************************************************************/

void
Altera_nodos()
{
    /*
     * Nodo("B1  ", &resultado, sa->flip_flop_T[1].Q, 16);
     * sn->flip_flop_T[1].Q = sa->flip_flop_T[1].Q = resultado;
     */
}

/*************************************************************************/
/* INICIALIZACAO DAS VARIAVEIS                       */
/* */
/* obsevacao: o conteudo apontado por "sa" deve ser igual ao de "sn"    */
/*************************************************************************/

void
Inicial()
{
    int             i;
    for (i = 0; i < 4; i++) {
        sa→flip_flop_T[i].T = sn→flip_flop_T[i].T = true;
        sa→flip_flop_T[i].x1 = sn→flip_flop_T[i].x1 = false;
        sa→flip_flop_T[i].x2 = sn→flip_flop_T[i].x2 = true;
        sa→flip_flop_T[i].x3 = sn→flip_flop_T[i].x3 = true;
        sa→flip_flop_T[i].x4 = sn→flip_flop_T[i].x4 = false;
        sa→flip_flop_T[i].x5 = sn→flip_flop_T[i].x5 = true;
        sa→flip_flop_T[i].x6 = sn→flip_flop_T[i].x6 = true;
        sa→flip_flop_T[i].nQ = sn→flip_flop_T[i].nQ = true;
        sa→flip_flop_T[i].Q = sn→flip_flop_T[i].Q = false;
    }
}
```

```
/*****************************************************************************/
/* DESCRICAO DO CIRCUITO                                                     */
/*****************************************************************************/

void        Funcao_ff_T();

void
Descricao_Circuito()
{
   int          i;
   sn→flip_flop_T[1].T = sa→flip_flop_T[0].Q;
   sn→flip_flop_T[2].T = (sa→flip_flop_T[1].Q & sa→flip_flop_T[0].Q);
   sn→flip_flop_T[3].T = (sa→flip_flop_T[2].Q & sa→flip_flop_T[1].Q
            & sa→flip_flop_T[0].Q);
   for (i = 0; i < 4; ++i) {
      Funcao_ff_T(&sn→flip_flop_T[i], &sa→flip_flop_T[i], f[1]);
   }
}

void
Funcao_ff_T(ffsn, ffsa, f1)
   ff_T         *ffsn, *ffsa;
   char          f1;
{
   ffsn→x1 = ~(ffsa→T & f1 & ffsa→nQ);
   ffsn→x2 = ~(ffsa→T & f1 & ffsa→Q);
   ffsn→x3 = ~(ffsa→x1 & ffsa→x4);
   ffsn→x4 = ~(ffsa→x2 & ffsa→x3);
   ffsn→x5 = ~(ffsa→x3 & ~f1);
   ffsn→x6 = ~(ffsa→x4 & ~f1);
   ffsn→Q = ~(ffsa→x5 & ffsa→nQ);
   ffsn→nQ = ~(ffsa→x6 & ffsa→Q);
}

/*****************************************************************************/
/* EXIBICAO GRAFICA DAS VARIAVEIS                                            */
/* */
/* JANELA(int, double, char *);                                              */
/* VALOR(int, double, char *, int);                                          */
/* */
/* JANELA(numero_da_janela, valor_a_ser_exibido, "Idetificacao");            */
/* VALOR(numero_do_valor, valor_a_ser_exibido, "Idetificacao", radical);     */
```

```
/* */
/* radical: 16 - hexadecimal, 10 - decimal, 2- binario          */
/*****************************************************************************/

void
Exibicao()
{
   JANELA(0, f[1], "f[1]");
   JANELA(1, sa→flip_flop_T[0].Q, "B0   ");
   JANELA(2, (double) sa→flip_flop_T[1].Q, "B1   ");
   JANELA(3, (double) sa→flip_flop_T[2].Q, "B2   ");
   JANELA(4, (double) sa→flip_flop_T[3].Q, "B3   ");

   VALOR(0, (double) (bool(sa→flip_flop_T[0].Q) + (bool(sa→flip_flop_T[1].Q) ≪
1) + (bool(sa→flip_flop_T[2].Q) ≪ 2) + (bool(sa→flip_flop_T[3].Q) ≪ 3)), "Cont",
16);
   VALOR(1, (double) sa→flip_flop_T[0].Q, "Q", 10);
   VALOR(2, (double) sa→flip_flop_T[0].nQ, "nQ", 10);

}
```

Conteúdo do arquivo *contador.h*

```
/*****************************************************************************/
/*                    CONTADOR.H                                             */
/*****************************************************************************/

typedef struct
{
  char T,nQ,Q;
  char x1,x2,x3,x4,x5,x6;
} ff_T;

typedef struct
{
  ff_T flip_flop_T[4];
} circuito;
```

ANEXO A-8 DEFINIÇÃO DO DESCRITOR DAS FERRAMENTAS

Conteúdo do arquivo *~/include/silex.h*

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/stat.h>
#include <sys/time.h>

#include "pipe_mens.h"
#include "pipe_dados.h"

typedef struct s_ferram {
    char            nome[50];
    char            nomint[50];
    int             id;
    int             pid;
    fd_set          Mfds, Dfds;
    int             MSG, DADO;
    struct sockaddr_un MSock, DSock;
    t_MENS          rec_men, env_men;
    t_MENS          *ini_rec_men;
    t_DADO          rec_dado, env_dado;
    struct timeval  tempo_espera;
    int             pai, filho;
    void            *con;
    struct s_ferram *prox;
}               t_ferram;

t_ferram        *D;

#ifdef MAIN
static Notify_value notifica_pipe();
static Notify_value notifica_fim();
#endif

#include "pipe.c"
```

ANEXO A-9 EXEMPLO DE CODIFICAÇÃO DE UMA INTERFACE GRÁFICA

Conteúdo do arquivo ~/*include/interface.c*

```
/*
 * DECLARACAO DA INTERFACE GRAFICA cap.c - Rotinas de notificacao e
chamada
 */

#include <stdio.h>        /* Bibliotecas padrao */
#include <sys/param.h>
#include <sys/types.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/textsw.h>
#include <xview/xv_xrect.h>
#include <gdd.h>
#include "cap_ui.h"

Attr_attribute  INSTANCE;

void
main(argc, argv)
    int             argc;
    char            **argv;
{
    cap_Janela_base_objects *cap_Janela_base;    /* Descritor da
                         * interface */


   xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);   /* Iniciacao do ambiente
*/
    INSTANCE = xv_unique_key();

    /* Rotina de criacao da interface */
    cap_Janela_base = cap_Janela_base_objects_initialize(NULL, NULL);

    xv_main_loop(cap_Janela_base→Janela_base);    /* Repassa controle ao
```

```
                    * Xview */
    exit(0);
}

/* Rotina de Criacao do elemento texto1 */
Xv_opaque
cap_Janela_base_texto1_create(ip, owner)
    caddr_t         ip;
    Xv_opaque       owner;
{
    extern Panel_setting Notifica_texto1();
    Xv_opaque       obj;

    obj = xv_create(owner, PANEL_TEXT,
            XV_KEY_DATA, INSTANCE, ip,
            XV_X, 12, XV_Y, 12,
            XV_WIDTH, 113, XV_HEIGHT, 15,
            PANEL_LABEL_STRING, "Texto:",
            PANEL_VALUE_X, 61, PANEL_VALUE_Y, 12,
            PANEL_LAYOUT, PANEL_HORIZONTAL,
            PANEL_VALUE_DISPLAY_LENGTH, 8,
            PANEL_VALUE_STORED_LENGTH, 80,
            PANEL_READ_ONLY, FALSE,
            PANEL_NOTIFY_PROC, Notifica_texto1,
            NULL);
    return obj;
}

/* Rotina de notificacao de interacao com o elemento texto1 */
Panel_setting
Notifica_texto1(item, event)
    Panel_item      item;
    Event           *event;
{
  cap_Janela_base_objects *ip = (cap_Janela_base_objects *) xv_get(item, XV_KEY_DATA,
INSTANCE);
    char            *value = (char *) xv_get(item, PANEL_VALUE);

    return panel_text_notify(item, event);
}

/* Rotina de criacao do elemento botao */
Xv_opaque
```

```
cap_Janela_base_botao1_create(ip, owner)
    caddr_t         ip;
    Xv_opaque       owner;
{
    extern void     Notifica_botao1();
    Xv_opaque       obj;

    obj = xv_create(owner, PANEL_BUTTON,
            XV_KEY_DATA, INSTANCE, ip,
            XV_X, 152, XV_Y, 12,
            XV_WIDTH, 51, XV_HEIGHT, 19,
            PANEL_LABEL_STRING, "Botao",
            PANEL_NOTIFY_PROC, Notifica_botao1,
            NULL);
    return obj;
}

/* Rotina de notificacao do acionamento do elemento botao1 */
void
Notifica_botao1(item, event)
    Panel_item      item;
    Event           *event;
{
  cap_Janela_base_objects *ip = (cap_Janela_base_objects *) xv_get(item, XV_KEY_DATA,
INSTANCE);
}

/* ... Continuacao da definicao e rotinas notificadoras ... */
```

BIBLIOGRAFIA

[BHA 90] BHAT, P.; TAKU, F. A Seven-layer Model of Framework functionality. **Electronic Engineering**, London, p.67-73, Sept. 1990.

[BOW 90] BOWER, Wayne; et al. A Framework for Industrial Layout Generators. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 27. JUNE, 1990. **Proceedings...** Florida: ACM/IEEE, June 1990. p. 419-424.

[BRO 87] BROUWERS, Jean; GRAY, Moshe. Integrating the Eletronic Design Process. In: VLSI SYSTEMS DESIGN, 1987, Santa Clara, CA. **Proceedings...** Santa Clara: [s.n.] June 1987.

[CAD 88] CADENCE Design Systems. **Design Framework**. San José: Cadence, 1988. v.1.

[CAR 87] CARRO, Luigi; SUZIM, Altamiro A. Metodologia em Concepção de Circuitos Integrados. In: CONGRESSO BRASILEIRO DE MICROELETRÔNICA, 2., 1987, São Paulo. **Anais...** São Paulo: SBMICRO, 1987. p.243-247.

[CAR 88] CARRO, Luigi; et al. GROM: Gerador de ROMs Parametrizável. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 3., 1988, São Paulo. **Anais...** São Paulo: SBMICRO, 1990. p.471-481.

[CAR 88a] CARRO, Luigi. Comparação de Ambientes de CAD para VLSI. In: SEMINÁRIO INTERNO DO GRUPO DE MICROELETRÔNICA, 4., 1988, Porto Alegre. **Anais...** Porto Alegre: SIM, 1988. p. 37-39.

[CAR 89] CARRO, Luigi. **Gerador Parametrizável de Partes Operativas CMOS**. Porto Alegre: CPGCC da UFRGS, 1989. 221 p. Dissertação de Mestrado.

[CAR 91] CARRO, Luigi; SUZIM, A. A.; JUNQUEIRA, A.; MARCHIORO, G.F. Perspectivas para a microeletrônica: a influência do CAD. In: CONGRESSO NACIONAL EM INFORMÁTICA, 24., **Anais...** São Paulo: SUCESU, 1991. P.227-231.

[CAS 91] CASACURTA, Alexandre. SELA - Sistema de Edição de Layout. In: SEMINÁRIO INTERNO DE MICROELETRÔNICA, 7., 1991, Capão Novo. **Anais...** Capão Novo: SIM, 1991.

[CLA 88] CLAESEN, L.; SEVERYNS R.; et al. Open Framework of Interactive and Communicating CAD Tools. In: TOOL INTEGRATION AND DESIGN ENVIRONMENTS., 1988, North-Holland. **Proceedings...** North-Holland: IFIP, 1988. p.133-155.

[COR 91] CÔRTES, M. Lúcio; et al. Planejamento de EDA: a experiência do CPqD. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 6., 1991, Minas Gerais. **Anais...** Belo Horizonte: SBMICRO, 1991, p.167-175.

[DAN 89] DANIEL, J. An Object Oriented Approach to CAD Tool Control within a Design Framework. In: IEEE DESIGN AUTOMATION CONFERENCE, 26., 1989, New York. **Proceedings...** New York: IEEE, 1989. p.197-202.

[DEM 87] DeMAN, H. Evolution of CAD Tools Towards Third Generattion Custom VLSI Design. **Revue de Physique Appliquee**, v.22, p.31-45, Jan. 1987.

[DEW 90] DEWHURST, Stephen C.; STARK, Kathy T. **Programando em C++**. [s.l.]: Campus, 1990.

[FER 89] FERREIRA, A.; et al. Integração de Ferramentas no contexto de uma Estação de Trabalho. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 4., abril 1989, Porto Alegre. **Anais...** Porto Alegre: SBC, 12-14 Abril, 1989. p. 83-94.

[FID 90] FIDUK, Kenneth W. Design Methodology Management - A CAD Framework Initiative Perspective -. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 27., June, 1990. **Proceedings...** Florida: ACM/IEEE, June 1990. p. 278-283.

[FOO 90] FOO, Simon Y.; TAKEFUJI Y. Databases and Cell-selection Algorithms for VLSI cell Libraries. In: IEEE COMPUTER. **Proceedings...** [s.l.: s.n.], Feb. 1990.

[FRA 91] FRANCESCHINI, Gustavo. A VLSI Eletrical Simulator. In: SEMINÁRIO INTERNO DE MICROELETRÔNICA, 7., 1991, Rio Grande do Sul. **Anais...** Capão Novo: SIM, 1991.

[FRE 91] FREITAS, Flavio S. G. **Arquitetura para o Suporte de Aplicações Distribuídas em Ambiente UNIX**. Porto Alegre: CPGCC da UFRGS, 1991. 72 p. Trabalho Individual.

[FUL 88] FULTON, Jim. **X Window System - Version 11**. Massachusetts Institute of Technology, 1988. 35 p.

[GAJ 88] GAJSKI, D. Introduction to Silicon Compilation. In: SILICON COMPILATION. New York: Addison Wesley, 1988. p. 1-48.

[GLA 85] GLASSER, L.; DOBBERPUHL D. **The Design and Analysis of VLSI Circuits**. Cambridge: Addison-Wesley, 1985. 473 p.

[GOM 88] GOMES, R. F. DARC: um Verificador de Regras de Projeto de Circuitos Integrados Utilizando Programação em Lógica. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 3., 13-15 abril, 1988, Gramado. **Anais...** Porto Alegre: SBC, 1988. p. 85-94.

[GRI 92] GRIMBERT, F.; MORAES, F.; REIS, R. A. L. **Usando o Magic.** Porto Alegre: CPGCC da UFRGS, jan 1992. 60 p. Relatório Técnico.

[JUN 92] JUNQUEIRA, A. A. **RISCO: Microprocessador RISC de 32 bits.** Porto Alegre: CPGCC da UFRGS, 1992. Dissertação de Mestrado.

[KER 86] KERNIGHAN, Brian; RITCHIE, Dennis. **Linguagem de Programação C.** Rio de Janeiro: Campus, 1986. 208 p.

[KOK 91] KOK, Hans. **Sistema Navigator.** Comunicação Informal, Jan./Jun. 1991.

[LAM 86] LAMPORT, Leslie. **LATEX, a Document Preparation System.** Digital Equipment Corporation, Addison-wesley, maio 1986.

[MAL 89] MALINIAK, David. Modular Tools, Framework Refine IC-Design Systems. In: ELETRONIC DESIGN-INTERNATIONAL, **Proceedings...** Maio 1989. p. 47-51.

[MAR 89] MARCHIORO, Gilberto F.; SUZIM, Altamiro A. **CHARRUA - Editor Simbólico para Circuitos Integrados**, Porto Alegre: CPGCC da UFRGS, 1989. 95 p. Trabalho Individual.

[MAR 90] MARCHIORO, Gilberto F.; SUZIM, Altamiro A. SILEX sobre X WINDOW, IN: SIMPÓSIO INTERNO DE MICROELETRÔNICA, 4., 1990, Rio Grande do Sul. **Anais...** Tramandaí: SIM, 1990. p. 15-18.

[MAR 90a] MARCHIORO, Gilberto F.; SUZIM, Altamiro A. Editor Simbólico Hierárquico para Circuitos. IN: CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA, 10., 1990, Assunção. **Anais...** Paraguai: CLEI, 1990. p. 580-588.

[MAR 91] MARCON, Cesar Missio; et al. **Manual do Sistema HDC**. Porto Alegre, CPGCC da UFRGS, 1991. Relatório Técnico.

[MEA 80] MEAD, CARVER; CONWAY L. **Introduction to VLSI Systems**. [s.l.]: Addison-wesley, 1980.

[MEN 90] Mentor Graphics Corporation. **GENIE User Guide**. Oregon: [s.n.], 1990. 242 p.

[MEN 92] Mentor Graphics Corporation. **FALCON Framework for Concurrent Design**. Oregon: [s.n.], 1992.

[MEN 92] Mentor Graphics Corporation. **GDT Designer for New Users**. Oregon: [s.n.], 1992.

[MIC 90] MICHELI, Giovanni; et al. The Olympus Synthesis System. In: IEEE DESIGN & TEST OF COMPUTERS. Proceedings... Oct. 1990. p. 37-53.

[MOR 87] MORSCHEL, I. J.; BARONE D. A. C. GERME - Um Gerenciador de Projetos como Suporte às Estações de Trabalho. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 3., abril 1987, Porto Alegre. **Anais...** Porto Alegre: SBC, 1987. p. 119-128.

[MOR 90] MORAES, F. G.; PEREIRA, C. E.; MARCHIORO, G. F. ESQUELETO - um Editor de Esquemas Elétricos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 5., Julho 1990, Campinas. **Anais...** Campinas: SBMICRO, 1990.

[MOR 90a] MORAES, F. G.; REIS, R. EXTRALO - Extrator Lógico. In: CONGRESSO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 5., out. 1990, Ouro Preto. **Anais...** Ouro Preto: SBMICRO, out. 1990. p. 167-176.

[MOR 90b] MORAES, F. G. TRAGO - Síntese Automática de Leiaute para Circuitos em Lógica Aleatória. Porto Alegre: CPGCC da UFRGS, 1990. 199 p. Dissertação de Mestrado.

[MOR 91] MORAES, Fernando G.; REIS Ricardo A. L. **TENTOS - Gerenciador de Software para Microeletrônica**. Porto Alegre: CPGCC da UFRGS, 1991. 36 p. Relatório Técnico.

[NIE 86] NIESSEN, C. Abstraction Requirements in Hierarchical Design Methods. **Design Methodologies**. Amsterdan: North-Holland, 1986. p. 151-182.

[OLI 91] OLIVEIRA, Alexandro. CHARRUA - Symbolic Editor. In: SEMINÁRIO INTERNO DE MICROELETRÔNICA, 7., 1991, Rio Grande do Sul. **Anais...** Capão Novo: SIM, 1991.

[PER 89] PEREIRA, C. E.; WONG, Z. W.; BARONE D. A. C. Estudo de Caso: Vantagens e Desvantagens de um Projeto com Ferramentas de PAC não integradas. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 4., Abril 1989, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1989. p. 95-105.

[PER 90] PEREIRA, Carlos Eduardo. **AMARGO - Ambiente Hierárquico de Montagem de Circuitos Integrados**. Porto Alegre: CPGCC da UFRGS, 1990. 282 p. Dissertação de Mestrado.

[PRE 88] PREAS, B. T.; LORENZETTI M. J. **Physical Design Automation of VLSI Systems**. Menlo Park: Benjamin/Cummings, 1988. 510 p.

[RAM 87] RAMMIG, F. **Tool Integration and Design Environments.** Paderborn: North-holland, 1987. 323 p.

[ROS 91] ROSA, Fernando R. **Programação Distribuída Baseada em Objetos.** Porto Alegre: CPGCC da UFRGS, 1991. 65 p. Trabalho Individual.

[SAR 91] SARMENTO, Helena. Ferramentas de CAD e sua Integração. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 6., 1991, Minas Gerais. **Anais...** Belo Horizonte: SBMICRO, 1991.

[SDA 88] SDA Systems. **Design Framework I**, v. 1., 1988.

[SIL 89] Silicon Compiler Systems Corporation. **Genesil Designer**, v. 1., 1989.

[SOT 91] SOTO, Fernando. GGMOD: a generator of module generators. In: SEMINÁRIO INTERNO DE MICROELETRÔNICA, 7., 1991, Rio Grande do Sul. **Anais...** Capão Novo: SIM, 1991. p. 365-374.

[STE 89] STEMMER, M.; REIS, R. EXTRIBO: um Extrator Hierárquico de Circuitos. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 4., abril, 1989. Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1989. p. 1-9.

[SUN 88] SUN - Microsystem. **Security Features Guide**, [s.l.]: SUN Microsystem, May 1988.

[SUN 90] SUN - Microsystem. **Network Programming Guide**, [s.l.]: SUN Microsystem, May 1990. 356 p.

[SUN 90a] SUN - Microsystem. **System Services Overview**, [s.l.]: SUN Microsystem, May 1990. 532 p.

[SUZ 86] SUZIM, Altamiro Amadeu. Sistema de CAD para Microeletrônica, In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 1., 1986, São Paulo. **Anais...** Campinas: SBMICRO, 1986. p. 341-350.

[SUZ 88] SUZIM, A. A. A CAD Frame for VLSI Design. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 3., Abril 1988, Gramado. **Anais...** Gramado: SBC, Abr. 1988. p. 129-145.

[SUZ 89] SUZIM, Altamiro Amadeu. **Sistemas Digitais: Uma Visão Integrada do Processo de Síntese**. Curitiba: UFPr., 1989.

[SWI 88] SWICK, Rlaph R. **X Toolkit Athena Widget - C Language Interface**, Digital Equipament Corporation, 1988.

[VAN 89] VANHORN, Earl; REZAC, Roy R. Experience with the D-Bus Architecture for a Design Automation Framework. In: ACM/IEEE Design Automation Conference, 26., Ago, 1989. **Proceedings...** [s.l.]: IEEE, 1989. p. 209-214.

[VAS 91] VASCONCELLOS, Vinicius; BASTOS, Eduardo. Implementation of Symbols Properties for Esqueleto Schematic Editor. In: SEMINÁRIO INTERNO DE MICROELETRÔNICA, 7., 1991, Rio Grande do Sul. **Anais...** Capão Novo: SIM, 1991.

[VLA 81] VLADIMIRES, C. U.; et al. **SPICE Version 26, User's Guide**. Berkeley: University of California, 1981.

[YAG 91] YAGER, Tom. X Terminals for Workstation Power at PC Prices. **Byte Magazine**, [s.l.], p. 238-244, May 1991.

[WAG 88] WAGNER, Flavio R. Ambientes Integrados de Projeto de Circuitos VLSI. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 1988, Rio Grande do Sul. **Anais...** Gramado: SBC, 1988. p. 147-60.

[WAT 87] WATKINS, M. L. Software Architecture and the UNIX Operating System: An Introduction to Interprocess Communication. **Hewllet-Packard Journal**, Palo Alto, v. 38, n. 6, p. 26-36, June 1987.

[WES 85] WESTE, NEIL H.; ESHRAGHIAN, K. **Principles of CMOS VLSI Design: a System Perspective**. [s.l.]: Addison-wesley, 1985.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"SILEX - Sistema para a Integração de Ferramentas de Projeto de Circuitos Integrados"

Dissertação apresentada aos Srs.

Prof. Dr. Altamiro Amadeu Suzim

Prof. Dr. José Monteiro da Mata (DCC/UFMG)

Prof. Dr. Ricardo Augusto da Luz Reis

Profa. Dra. Taisy Silva Weber

Visto e permitida a impressão

Porto Alegre, 06/04/92

Prof. Dr. Altamiro Amadeu Suzim
Orientador

Prof. Dr. Ricardo Augusto da L. Reis
Coordenador do Curso de Pós-Graduação
em Ciência da Computação