Eduardo Henrique Tozetto

Automação do fluxo de projeto de circuitos integrados através do desenvolvimento de uma interface gráfica paramétrica implementada em TCL/TK

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Eletrônica, Microeletrônica e Optoeletrônica.

Orientador: Prof. Dr. José Antonio Siqueira Dias

Campinas, SP
2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Tozetto, Eduardo Henrique
T669a Automação do fluxo de projeto de circuitos integrados através do desenvolvimento de uma interface gráfica paramétrica implementada utilizando-se pacotes TCL/TK
Faculdade de Engenharia Elétrica e de Computação
Eduardo Henrique Tozetto. – Campinas, SP: [s.n.], 2007.

Orientador: José Antonio Siqueira Dias.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Circuitos integrados. 2. Computação gráfica.
I. Dias, José Antonio Siqueira. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Título em Inglês:	Integrated circuit design flow automation using a parametric graphical interface implemented using TCL/TK packages.
Área de concentração:	Eletrônica, Microeletrônica e Optoeletrônica.
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Elnatan Chagas Ferreira, Oséas Valente de Avilez Filho, Marco Antonio Robert Alves.
Data da defesa:	31/07/2007
Programa de Pós-Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Eduardo Henrique Tozetto

Data da Defesa: 31 de julho de 2007

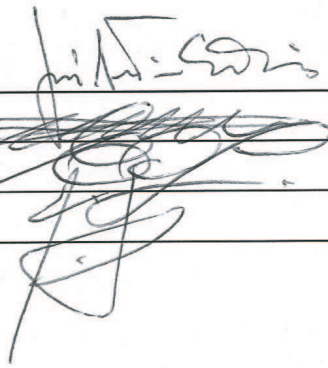
Título da Tese: "Automação do Fluxo de Projeto de Circuitos Integrados Através do Desenvolvimento de uma Interface Gráfica Paramétrica Implementada em TCL/TK"

Prof. Dr. José Antonio Siqueira Dias (Presidente): _____

Prof. Dr. Nivaldo Vicençotto Serran: _____

Prof. Dr. Elnatan Chagas Ferreira: _____

Prof. Dr. Oséas Valente de Avilez Filho: _____



Resumo

O contexto econômico competitivo em que as empresas que desenvolvem ferramentas para projeto de CIs estão inseridas dificulta o estabelecimento de padrões e plataformas de desenvolvimento comuns. Em geral, a necessidade de inúmeras ferramentas resulta em um ambiente de projeto fragmentado. Este trabalho apresenta uma ferramenta desenvolvida através da implementação de interfaces gráficas paramétricas em TCL/TK, que integra funções gerais, permitindo a rápida codificação de procedimentos e seu acesso através de elementos gráficos. A ferramenta desenvolvida serve para facilitar e otimizar as tarefas envolvidas no aprimoramento das técnicas de projeto de Circuitos Integrados através da elaboração de métodos e scripts visando à automação de etapas do fluxo de projeto.

Palavras-chave: Interface gráfica paramétrica, TCL, TK, Projeto de circuitos integrados.

Abstract

The competitive environment in which the companies who develop software tools for the design of integrated circuits creates many barriers to the establishment of standards and common platforms. Usually the need for several software tools leads to a design environment which is fragmented and difficult to manage. This work presents the development of software tool, based on graphical parametric user interfaces in TCL/TK, which integrates many general functions and allows for a quick codification of procedures and its access through the graphics elements. The developed tool optimizes and facilitates the tasks employed in the improvement of the techniques used in integrated circuits design through the elaboration of methods and scripts dedicated to the automation of the design flow steps.

Keywords: Parametric, Graphical user interface, TCL, TK, Integrated circuit design.

Agradecimentos

Agradeço ao meu orientador pelo voto de confiança e ajuda no alinhamento da pesquisa, à família pelas raízes, à Freescale Semicondutores do Brasil por incentivar as atividades acadêmicas através da flexibilidade de horário. Em especial à minha esposa, sempre ao lado neste caminhar.

Dedico esta obra a todos aqueles que na mediação de sua essência, encontrem uma aplicação pessoal e justifiquem cada vez mais sua existência.

Sumário

Lista de Figuras **ix**

Glossário **x**

1 Introdução **1**
1.1 Tema . . . 1
1.2 Motivações . . . 2
1.3 Objetivos . . . 2
1.4 Metodologia . . . 3
1.5 Revisão bibliográfica . . . 3
1.6 Estrutura da dissertação . . . 3

2 Projeto de circuitos integrados **5**
2.1 Ambiente computacional . . . 5
2.1.1 Ferramentas gerais . . . 6
2.1.2 Electronic design automation . . . 6
2.1.3 Ferramentas internas . . . 8
2.2 Fluxo de projeto . . . 9
2.2.1 Metodologia . . . 9
2.2.2 Parametrização . . . 13

3 Arquitetura **14**
3.1 Elementar . . . 14
3.2 Interface gráfica paramétrica . . . 16
3.2.1 Requisitos . . . 17
3.2.2 Diagrama de blocos . . . 18
3.2.3 Blocos funcionais . . . 18
3.2.4 Benefícios . . . 22

4 Implementação **24**
4.1 Metodologia . . . 24
4.2 Tool Command Language . . . 25
4.2.1 Thread . . . 26
4.2.2 Expect . . . 26

4.2.3 TCLLIB 27
4.2.4 Sistema de abas 28
4.2.5 TclDot 28
4.3 Técnicas de programação 29
4.3.1 Meta programação 29
4.3.2 Eventos 29
4.3.3 Reflexão 30
4.3.4 Autonomic computing 30
4.3.5 Cliente/Servidor 30
4.3.6 Comandos canônicos 31
4.4 Módulos 32
4.4.1 Bootstrap 32
4.4.2 Construtores de interface 33
4.4.3 Arquivos de configuração 36
4.4.4 Gerador de grafos 38
4.4.5 Controlador X-window 38
4.4.6 Parser verilog 38
4.5 Aplicações 39
4.5.1 Meta aplicação 39
4.5.2 Simulação mixed-signal 39
4.5.3 Automação de documentação 41

5 Considerações finais **44**
5.1 Resultados 44
5.2 Conclusões 45
5.3 Trabalhos futuros 45

Referências bibliográficas **47**

Lista de Figuras

2.1 Fluxo de projeto de um “System On Chip” genérico. 10
2.2 Métodos “bottom-up” e “top-down” aplicados ao desenvolvimento de CIs 11
2.3 Convergência entre os métodos “bottom-up”e “top-down”. 12

3.1 Arquitetura X Window. 15
3.2 Gerenciador de janelas Enlightenment. 16
3.3 Diagrama de blocos da interface paramétrica. 19

4.1 Procedimento para criação de threads. 26
4.2 Código utilizado para comunicação com terminal remoto. 27
4.3 Código utilizado para automação do comando sudo. 27
4.4 Interface gráfica resultante da parametrização do “RNotebook”. 28
4.5 Código utilizado para parametrização do “RNotebook”. 29
4.6 Código que implementa funcionalidades de servidor. 31
4.7 Código que implementa funcionalidades de cliente. 32
4.8 Código que implementa o monitor da interface gráfica paramétrica. 32
4.9 Detector de processos genéricos. 33
4.10 Editor de interface gráfica interativo VisualTcl. 34
4.11 Parâmetros para criação do vetor de botões. 35
4.12 Procedimentos para criação de vetores de botões. 36
4.13 Sistema resultante da implementação de rotinas para parametrização de interfaces. . . 37
4.14 Procedimentos para automação do interpretador eesh 39
4.15 Arquitetura do software xnee. 40
4.16 Simulação de topo convencional. 41
4.17 Simulação de topo utilizando a interface gráfica paramétrica. 42
4.18 Automação de procedimentos para documentação. 43

Glossário

API - Advanced Program Interface

CI - Circuito Integrado

DFII - Design Framework II

EDA - Electronic Design Automation

GNU - GNU is Not Unix

PDK - Process Design Kit

SI2 - Silicon Initiative 2

SOC - System On Chip

TCL - Tool Command Language

TK - Tool Kit

Capítulo 1

Introdução

Neste capítulo será apresentada a contextualização acadêmica. Trata-se de uma sucinta discussão do tema abordado para introdução do panorama geral e posterior levantamento das motivações e objetivos. A metodologia, de modo complementar, fundamentará as decisões tomadas na orientação da pesquisa e os critérios utilizados na implementação da arquitetura proposta. Fazendo a conexão com o que será tratado, uma breve revisão bibliográfica analisará a categoria dos trabalhos citados. A análise da estrutura da dissertação e a identificação de roteiros de leitura seqüencialmente relevantes conclui esta introdução.

1.1 Tema

Aparelhos eletrônicos encontrados no dia-a-dia do século XXI, devido à sua popularidade, representam um bom ponto de partida para se compreender a importância dos Circuitos Integrados (CIs) em seu contexto mais amplo. Artefatos industriais como computadores, telefones móveis, televisores, aparelhos de DVD, GPS, câmeras digitais, dentre outros equipamentos são o resultado de um processo produtivo extremamente complexo. Os CIs, ao implementarem funcionalidades elétricas e eletrônicas para o gerenciamento de potência, sensorização, processamento, controle e comunicação, são componentes fundamentais na concepção destes sistemas. Também conhecido por "chip", é um dispositivo que agrega muitos transistores, resistores, capacitores e nos circuitos de RF, indutores, interligados em escala micrométrica [1].

A indústria de semicondutores e a comunidade acadêmica, através da alocação de recursos tecnológicos e gerenciamento intelectual, são a força motriz no desenvolvimento da microeletrônica. Arquiteturas organizacionais são definidas visando a obtenção constante de inovações, tornando viável a produção de sistemas cada vez mais sofisticados, com melhor desempenho e custos relativamente mais baixos. Neste contexto, as etapas que resumem, de forma muito simplificada, o processo de desenvolvimento e fabricação de um chip, têm início com a identificação de oportunidades para atender às demandas do mercado consumidor, passam pela especificação, estudo de viabilidade, planejamento, projeto, fabricação e terminam com os testes e a qualificação.

Os centros de projeto são responsáveis pela implementação de sistemas integrados, ou em inglês "System-On-Chip" (SOC) [2] através da utilização de recursos computacionais avançados, indispensáveis em toda a cadeia produtiva. O gerenciamento destes recursos, visando um fluxo de projeto

eficiente e acessível, é preocupação constante em centros de excelência. Este trabalho está, portanto, diretamente relacionado com o desenvolvimento de conceitos visando a elaboração de uma interface gráfica paramétrica e sua aplicação na automação das etapas do fluxo de projeto de CIs.

1.2 Motivações

Dentre as principais motivações podem se destacar o grande número de ferramentas utilizadas e as questões relacionadas com sua interoperabilidade. A interação entre o projetista e o ambiente computacional e finalmente a falta de trabalhos que integrem os recursos computacionais através de abordagens e métodos utilizando interfaces gráficas paramétricas. Tais motivações mostram-se ainda mais relevantes na medida em que o fluxo de projeto de circuitos integrados apresenta pontos com grande potencial de gerenciamento e otimização através do sistema proposto.

O contexto econômico competitivo em que as empresas [3, 4, 5] que desenvolvem ferramentas para projeto de CIs estão inseridas dificulta o estabelecimento de padrões e plataformas de desenvolvimento comuns. Em geral, a necessidade de inúmeras ferramentas, concebidas por diferentes empresas, resulta em um ambiente de projeto fragmentado.

A múltipla representação de um mesmo elemento do chip em desenvolvimento e a distribuição funcional das etapas do fluxo de projeto em diferentes ferramentas é comum, resultando em inúmeras interações com o ambiente de projeto. Este fato demanda a entrada recorrente de informações e dados como condição para a continuidade das atividades de projeto, dificultando assim a parametrização do seu fluxo e conseqüentemente sua automação.

Devido à necessidade de gerenciamento das ambigüidades, o ambiente de projeto se torna ineficiente e com maior probabilidade de erros. Diante desta situação, é bastante útil o desenvolvimento de código personalizado, que contorne as dificuldades anteriormente apontadas e promova a automação das etapas do fluxo de projeto. Deste modo, obtêm-se um ambiente computacional para a concepção de sistemas de microeletrônica com maior eficiência e qualidade. Como conseqüência das exigências do mercado, projetos executados mais rapidamente, com menor probabilidade de erros e implementações mais otimizadas são elementos imprescindíveis para a introdução de produtos de modo mais competitivo no mercado.

Outra motivação importante concentra-se na inexistência de métodos que abordem o ambiente computacional envolvido no projeto de CIs de modo mais abrangente. Apesar da grande disponibilidade de recursos, informações e ferramentas, pouco foi desenvolvido visando integrar a infraestrutura computacional através de uma interface mestra genérica.

1.3 Objetivos

As metas estabelecidas neste item estão diretamente alinhadas com as motivações levantadas anteriormente. Podem ser mencionados alguns traços gerais, como especificar e implementar uma interface gráfica paramétrica em TCL/TK para integração de funções genéricas, permitindo a rápida codificação de procedimentos e seu acesso através de elementos gráficos.

Como objetivo fundamental pode ser citada a aplicação do sistema proposto no aprimoramento das técnicas de projeto de circuitos integrados. Implementação de procedimentos em TCL para cria-

ção da infra-estrutura computacional e execução pontual das ferramentas computacionais utilizadas. Além disso, a integração destes procedimentos através da interface gráfica paramétrica, deve permitir a automação de etapas completas do fluxo de projeto, tornando-as acessíveis a outros projetistas.

A utilização da própria ferramenta para facilitar e otimizar as tarefas envolvidas na sua própria concepção merece destaque dentre os objetivos. Para isto, recursos avançados de engenharia de software serão empregados.

1.4 Metodologia

Os recursos utilizados neste trabalho estão concentrados na linguagem de programação TCL e suas extensões. Programas genéricos, disponíveis nas principais distribuições GNU/Linux [6] e ferramentas proprietárias, específicas para o projeto de CIs [4], foram também extensivamente utilizadas.

Três atividades principais nortearam o desenvolvimento deste trabalho. A primeira, responsável pela elaboração de conceitos e desenvolvimento da interface gráfica paramétrica, independentemente de sua aplicação; a segunda, ligada ao desenvolvimento de "scripts" isolados relativos às tarefas computacionais, principalmente aquelas envolvidas nas etapas do fluxo de projeto de circuitos integrados. Nesta fase são determinados os comandos elementares, normalmente "encobertos" pelas interfaces gráficas específicas de cada ferramenta; a terceira, integra os elementos provenientes das duas primeiras e consiste na determinação de parâmetros para a interface gráfica proposta, bem como procedimentos adicionais visando concatenar a execução automática dos procedimentos desenvolvidos no segundo passo.

1.5 Revisão bibliográfica

Observando as tarefas desenvolvidas na metodologia, as referências bibliográficas se dividem basicamente em dois grandes conjuntos: o primeiro relativo à programação utilizando-se TCL/TK [7], e o segundo envolvendo os manuais das ferramentas computacionais utilizadas na concepção de circuitos integrados. Estão incluídos neste conjunto as referências das linguagens de programação para descrição de hardware [8] e verificação [9].

Embora tenha sido utilizado no projeto de topologias avançadas como controladores e sistemas para gerenciamento de potência, este trabalho possui como foco fundamental os processos e as ferramentas envolvidas no projeto de sistemas de microeletrônica. Neste sentido, as referências bibliográficas tendem a se concentrar nos métodos e nas ferramentas de projeto. Apesar de topologias ou técnicas utilizadas em circuitos não serem diretamente tratadas, toda e qualquer atividade neste sentido, devido a sua sofisticação, se valerá dos conceitos aqui descritos para otimização do processo de desenvolvimento.

1.6 Estrutura da dissertação

De modo linear, o segundo capítulo concentra as informações e conceitos relativos ao projeto de circuitos integrados em geral, envolvendo seu fluxo e as ferramentas utilizadas. São levantadas as dificuldades inerentes ao fluxo de projeto de CIs, bem como aquelas resultantes do contexto tecnológico

atual. São também analisados como estes problemas foram abordados por diferentes pesquisadores em domínios específicos. O terceiro capítulo diz respeito à arquitetura do sistema proposto com base nas necessidades identificadas no capítulo 2. Neste capítulo é feita uma descrição comportamental da arquitetura através da especificação de seus módulos funcionais. A dinâmica do sistema é também abordada com ênfase em suas características de autonomia. No quarto capítulo está detalhada a implementação da interface gráfica paramétrica. Os aspectos relacionados com a linguagem TCL são evidenciados. Aplicações como a automação do ambiente de verificação Mixed-Signal [10], verificação física e automação de documentação são então sucintamente descritas. A dissertação é concluída no quinto capítulo, a partir dos resultados provenientes das aplicações do capítulo 4. Neste capítulo devem ficar claras as contribuições que este trabalho oferece àqueles que utilizam intensivamente recursos computacionais, principalmente no projeto de CIs. Finalmente são delineados os próximos passos com possíveis direcionamentos para a pesquisa.

O segundo capítulo fundamenta a pesquisa ao identificar as dificuldades encontradas no ambiente de projeto, portanto leitores com interesse apenas na utilização da interface gráfica paramétrica, podem se ater ao capítulo 2 e, especificamente as seções do capítulo 4, no que concerne às aplicações. Para o aproveitamento máximo do sistema desenvolvido, torna-se bastante útil uma análise dos resultados, servindo de motivação para a extensão do sistema. Interessados em compreender o funcionamento do sistema, visando a otimização de sua utilização, podem incluir o capítulo 3 no roteiro de leitura. Os capítulos 3 e 4, isoladamente, compõem o corpo de conhecimento para a concepção da interface paramétrica desenvolvida em (TCL/TK) [7]. Estes capítulos são fundamentais para a criação de novas interfaces em outros contextos computacionais.

Capítulo 2

Projeto de circuitos integrados

Uma forma bastante interessante de expor o processo de produção de CIs é fazer o caminho inverso da cadeia produtiva. O mercado consumidor exerce pressão através da demanda constante por inovações, resultando na especificação de equipamentos cada vez mais sofisticados, menores e com melhor desempenho. Isto representa grande desafio para todos os envolvidos na cadeia produtiva.

Fundições são responsáveis pela efetiva fabricação dos micro-chips utilizados na indústria em geral. A matéria prima utilizada para a fabricação de um chip é o silício com altíssimo grau de pureza em forma de discos. Estruturas tridimensionais correspondentes aos dispositivos eletrônicos são criadas através de processos que envolvem etapas de implantação, corrosão, metalização, deposição e remoção de filmes, fotolitografia e dopagem [11].

Além do desenvolvimento de processos tecnológicos, levando-se em conta a fabricação de dispositivos integrados de alta complexidade, é também atribuição da fundição ou algum grupo com estreito relacionamento, o mapeamento dos parâmetros de processo em um conjunto de arquivos contendo bibliotecas de componentes, regras de projeto e modelos [12, 13, 14]. O pacote com o conjunto de arquivos é normalmente denominados PDK, em inglês "Process Design Kit".

O desenvolvimento de circuitos integrados exige um complexo aparato tecnológico e organizacional. Centros de projeto, ao atuarem como agentes de desenvolvimento e criação, desempenham um papel crucial na concepção de novos produtos. Através da utilização de recursos computacionais e de infra-estrutura, transformam especificações em implementações extraindo a máxima performance de determinado PDK [15]. Dentro de toda a dinâmica existente nesta estrutura, o projeto em si pode ser entendido como ponto crítico, representando grande parte dos esforços das empresas fabricantes de semicondutores. Qualquer artefato de engenharia complexo, para ser desenvolvido de forma otimizada, deve ser concebido dentro do paradigma de projetos [16].

2.1 Ambiente computacional

Consumidores finais, indiretamente através de especificações, fundições e engenheiros de projeto encontram seu ponto de convergência em complexos ambientes computacionais. O ambiente de projeto atual conta com um vasto conjunto de programas levando a uma multiplicidade de opções. O conjunto de ferramentas usadas efetivamente em determinado fluxo de projeto depende de muitos fatores. Estão envolvidas em sua determinação variáveis como os conhecimentos que o grupo de

projeto possui, possibilidades de reuso, complexidade do projeto, necessidade de funcionalidades específicas, critérios de performance, interoperabilidade e, no caso das ferramentas proprietárias, a disponibilidade de licenças. Desta forma, o ambiente computacional emerge dinamicamente de uma combinação de ferramentas gerais, proprietárias e internas.

2.1.1 Ferramentas gerais

Este tipo de software compreende normalmente o sistema operacional GNU/Linux [6] e sua infinidade de aplicativos como leitores de correio eletrônico, editores de texto, imagens, diagramas e navegadores de internet. São usualmente de código livre embora, devido à extensiva utilização da plataforma Windows por clientes e engenheiros de aplicação, ainda é comum a utilização de ferramentas proprietárias para funções básicas. Conseqüentemente torna-se interessante a utilização de formatos eletrônicos compatíveis, acrescentando um nível a mais de complexidade com a ampla disponibilidade de programas nas duas plataformas.

Os aplicativos utilizados para desenvolvimento e programação [6] como linguagens, compiladores e interpretadores podem ser categorizados como gerais, por serem intensivamente utilizados em outros domínios. Sua utilização no desenvolvimento de um chip se estende desde a concepção do sistema até a elaboração de ambientes complexos para testes e verificação, onde é comum a integração com ferramentas específicas usadas no projeto de CIs.

2.1.2 Electronic design automation

As ferramentas computacionais de projeto de circuitos integrados, normalmente referenciadas pela sigla em inglês EDA, “Electronic Design Automation” [13, 17], são aplicativos especializados na automação de etapas do fluxo de projeto. Em [18] é realizada uma retrospectiva sobre a área apontando novos horizontes para sua evolução. Diversas abordagens, que possuem em comum o desenvolvimento de determinada infra-estrutura conceitual e a elaboração de algoritmos para a automação de tarefas específicas do fluxo de projeto, estão apresentadas em [19].

As referências acima oferecerem uma idéia geral das ferramentas de projeto, de modo que é interessante recorrer às empresas que desenvolvem ferramentas comerciais para se ter uma noção mais clara do ambiente de projeto industrial e acadêmico atual. Uma rápida análise das ferramentas em [3, 4, 5] mostra como pode ser complexo o ambiente de projeto dependendo do grau de sofisticação do circuito ou sistema integrado a ser implementado.

Interoperabilidade

As ferramentas de EDA utilizadas na indústria são em geral proprietárias, fato que dificulta sua interoperabilidade uma vez que o acesso às ferramentas, por critérios de propriedade intelectual, fica limitado a suas interfaces denominadas APIs, ou em inglês “Advanced Program Interface”. A competição existente entre as empresas de desenvolvimento de software prejudica o surgimento de consórcios e conseqüentemente a padronização de modelos, linguagens e estruturas de dados, resultando em uma verdadeira “torre de babel” em se tratando de programas para o auxílio no projeto de circuitos integrados [18]. Tais ferramentas, ao serem orientadas somente para a execução de determinada tarefa do fluxo de projeto, precisam de extensiva personalização para sua conformidade com

um ambiente que inclua a troca de dados entre programas de diferentes desenvolvedores. Devido ao seu desenvolvimento sob a influência de pressões de concorrência, ferramentas de EDA de diferentes "vendors" tendem a apresentar funcionalidades semelhantes, mas com abordagens completamente diferentes do ponto de vista de operação, base de dados e tecnologias. Outra característica importante é sua generalidade em detrimento da integração dos processos específicos de um determinado grupo de projetos, devido à necessidade de padronizar soluções para minimizar custos de desenvolvimento.

Neste contexto surgem iniciativas como o SI2, sigla em inglês para Silicon Initiative [20], que tem como principal desenvolvimento a base de dados comum denominada "Open Access". Ferramentas comerciais como o Silicon Navigator [21] apresentam elementos de arquitetura interessantes de serem observados, pois são a resposta comercial aos problemas mencionados. Apesar dos avanços, ainda não foram estabelecidos padrões aceitos pela indústria como um todo, em grande parte pelo custo de migração a partir de ambientes antigos.

Ferramentas pontuais

A máxima liberdade para projetistas ocorre quando a maior parte possível das ferramentas de projeto são utilizadas como pontuais, ou em inglês, "Point Tools". São ferramentas que têm uma funcionalidade específica e atuam de modo pontual no fluxo de projeto, com entradas e saídas bem definidas, como o roteamento de um bloco ou a simulação de um "netlist". A utilização da ferramenta na linha de comando torna isto viável. Neste caso ficam bem definidos os parâmetros de entrada que podem ser relacionados e reutilizados de forma sistemática.

A utilização independente das ferramentas de EDA/CAD de modo pontual oferece ao projetista maior controle dos processos que envolvem o projeto de circuitos integrados. Por outro lado, devido à complexidade do ambiente e a quantidade das diferentes ferramentas, torna-se necessária à automação destes processos. A utilização de ferramentas pontuais de modo manual é muitas vezes inviável, devido à quantidade de parâmetros, operações necessárias e sua constante repetição. Além de consumir muito tempo, não se poderia ter garantia de que todos os passos de um procedimento foram adequadamente executados.

Como resposta a esta situação, ferramentas de um determinado desenvolvedor são integradas em ambientes de desenvolvimento denominados "frameworks", como por exemplo o "Design Framewok II" ou DFII [4]. A integração ao ser feita por um ou outro desenvolvedor conta com propriedades específicas do conjunto de ferramentas integradas nativas [22], além disso, qualquer implementação fica restrita aquele ambiente de programação e seus recursos. Ferramentas de terceiros podem ser integradas com o custo do desenvolvimento de código extra.

Mesmo a possibilidade de convergência para um único desenvolvedor certamente enfrentaria problemas, pois atualmente, não existem ainda empresas que dominem e possuam ferramentas para o fluxo de projeto como um todo. Além disso, o "vendor" escolhido seria necessariamente um dos maiores neste ramo de atividade. Neste caso o problema se repete internamente, pois normalmente tal posição no mercado fez-se à base de aquisições de companhias menores. Mesmo em menor grau, "frameworks" proprietários tendem a sofrer das mesmas idiossincrasias existentes no ambiente de projeto onde são utilizadas apenas ferramentas pontuais.

Interfaces gráficas

As interfaces gráficas das ferramentas de projeto comerciais podem ser funcionalmente categorizadas tendo-se em mente a integração de ferramentas pontuais, àquelas com recursos gráficos para a visualização de diversos elementos do projeto, esquemáticos, formas de onda, "floorplans", "layouts", módulos "verilog", dentre outros ou ainda algorítmicas, como simuladores e roteadores automáticos.

Em todos os casos estas interfaces são estáticas do ponto de vista do desenvolvedor da ferramenta, apenas disponibilizando os recursos para que sejam feitas personalizações, que ficam a cargo do usuário. Para este tipo de expansão a grande maioria das ferramentas de projeto utiliza TCL como linguagem de "script". O DFII é uma das exceções. Orientado primordialmente para o projeto de circuitos analógicos, utiliza "Skill", um dialeto da linguagem "Lisp" como linguagem de script.

O desenvolvimento de qualquer código dentro de uma ferramenta proprietária, mesmo com uma linguagem de código aberto, como é o caso do TCL, torna necessário o uso de uma complexa infra-estrutura básica, o que seria inviável para os propósitos deste trabalho, pois dificultaria a reutilização do núcleo do sistema em diferentes contextos.

2.1.3 Ferramentas internas

Devido aos aspectos anteriormente apontados, surge demanda para a implementação de sistemas computacionais extras personalizados com funções voltadas para o desenvolvimento de plataformas de integração através de "frameworks" [23], implementação de gerenciadores da infra-estrutura computacional [24] e "kits" de ferramentas [25] ou [26]. A principal diferença entre "frameworks" e as outras abordagens tem relação direta com suas respectivas amplitudes. Enquanto "frameworks" tendem a ser projetos amplos de médio e longo prazo, normalmente com o trabalho conjunto de um grupo de pesquisadores, kits de ferramentas e gerenciadores de infra-estrutura são usualmente desenvolvidos por uma pessoa e se caracterizam por uma evolução rápida, no entanto, de abrangência limitada.

A interface gráfica paramétrica é um conceito mais geral, podendo ter seu desenvolvimento direcionado para formar um "framework", "kit" de ferramentas ou mesmo uma IDE, sigla em inglês para "Integrated Design Environment" [27], dependendo do esforço empregado. Em um ambiente de projeto, onde os recursos computacionais são utilizados de modo intensivo, a geração de pequenos programas, nas mais variadas linguagens, é bastante expressiva. A interface paramétrica também pode atuar na sistematização desta produção e facilitar o acesso de modo uniforme a estes recursos.

O encapsulamento e respeito a uma rígida especificação mecânica e eletrônica dentro de prazos e custos razoáveis não são suficientes para garantir a competitividade necessária no mercado atual. Medidas metodológicas e o desenvolvimento de sistemas dedicados são requisitos mínimos para agilizar o processo de desenvolvimento. O desenvolvimento de código especializado é, portanto, fundamental para a concepção de uma plataforma de desenvolvimento eficiente com recursos computacionais acessíveis e que otimizem o fluxo de projeto.

Ferramentas internas podem ainda auxiliar em tarefas como o gerenciamento do complexo fluxo de projeto, automação de tarefas, pesquisas à base de dados, visualização gráfica, geração de código, parametrização de dados de projeto, criação, manutenção, monitoramento e verificação da infra-estrutura computacional e arquitetura de projetos. Como fica evidente, o gerenciamento de diversos aplicativos em uma rede de computadores torna-se uma tarefa bastante complexa. Além do processo

de instalação, deve-se implementar uma política de gerenciamento das versões utilizadas de modo que cada projeto possua uma configuração específica. Pode ser também incluído nesta categoria o código utilizado para a personalização e configuração de ferramentas pré-existentes.

2.2 Fluxo de projeto

Nesta seção será apresentado um fluxo de projeto genérico, independentemente das ferramentas utilizadas e levando-se em conta como substrato as metodologias descritas anteriormente. Abordagens mais formais e sistemáticas podem ser encontradas em [28], onde o processo de implementação é analisado utilizando grafos hierárquicos concorrentes.

A orientação das atividades de projeto, em cenários comerciais cada vez mais dinâmicos e na existência de grande oscilação entre estudos de viabilidade, desenvolvimento de blocos para reutilização e projetos propriamente ditos, demanda do grupo de projeto agilidade na criação e replicação de estruturas básicas denominadas plataformas. De modo mais pragmático em [12] são empregados conceitos de reuso visando o desenvolvimento de plataformas de projeto. Para este trabalho, será suficiente uma visibilidade geral do fluxo de projeto e a especificação da etapa de simulação "mixed-signal" utilizada de modo recorrente na implementação, verificação e modelagem de sistemas.

Partindo-se dos elementos fundamentais disponíveis na biblioteca de um determinado processo, o circuito integrado é estruturado de forma hierárquica com módulos e sub módulos [9]. Esta estrutura hierárquica permite uma melhor distribuição funcional, facilitando a determinação de arquiteturas, além de permitir a alocação de recursos humanos de forma mais intuitiva. Os módulos podem ser divididos em duas grandes categorias, uma essencialmente analógica, mas com módulos digitais e outra com característica oposta. Do ponto de vista computacional, isto representa um grande desafio para os desenvolvedores de ferramentas de EDA/CAD [29].

A figura 2.1 representa de modo simplificado o fluxo de projeto de um SOC. Neste diagrama é possível observar o particionamento entre hardware e software durante o desenvolvimento. Além da utilização de ferramentas específicas em cada domínio, o ponto de contato ainda precisa de sistemas para manter a consistência do ambiente de projeto ao longo de sua evolução, exemplificando assim a necessidade de grande quantidade de ferramentas.

Outra abordagem interessante para se analisar o fluxo de projeto está na identificação de ferramentas específicas. Neste sentido, podem ser mencionados programas para o gerenciamento do fluxo de projeto, como o "FlowExpert", da "Mentor Graphics" [5], que tem por objetivo descrever, disparar e acompanhar as atividades de projeto e a família de produtos "FlowTracer" da "Runtime Design Automation" [30], com a finalidade de gerenciamento de licenças, redes de computadores e fluxos de projeto.

2.2.1 Metodologia

A metodologia de projeto pode ser sumarizada por um intrincado processo recursivo onde os métodos de baixo para cima e de cima para baixo, ou respectivamente em inglês, "bottom-up" e "top-down" [13], ocorrem simultaneamente. Enquanto o método "top-down" possui como processo fundamental a análise e o particionamento a partir de macro objetos do sistema, o método "bottom-up" inicia o desenvolvimento a partir de módulos elementares para, então, através de sucessivas

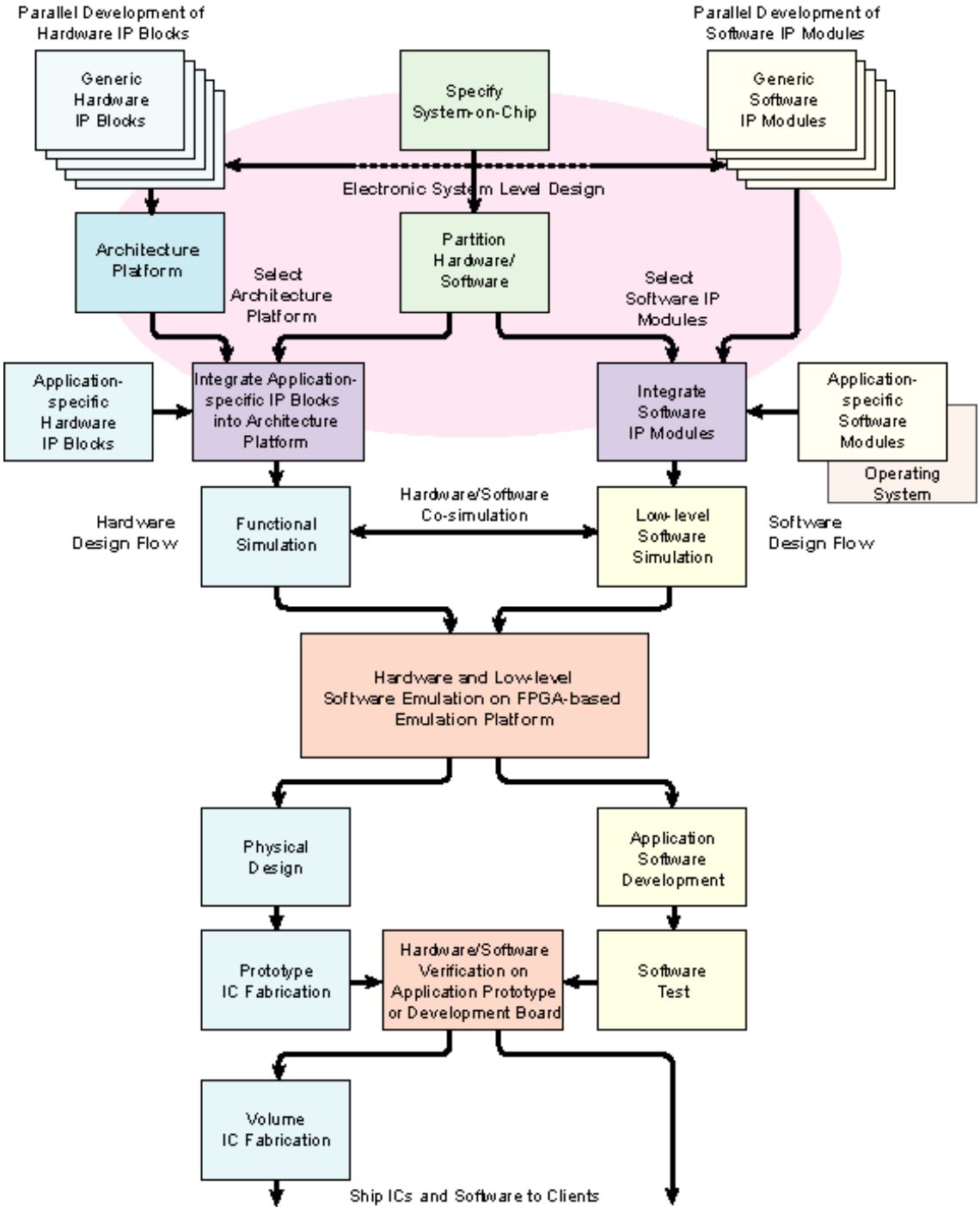


Fig. 2.1: Fluxo de projeto de um “System On Chip” genérico.

integrações hierárquicas, criar blocos com níveis de abstração mais elevados. Enquanto a metodologia “top-down” confere velocidade, a metodologia “bottom-up” caracteriza-se pela acurácia. A figura 2.2 aplica estes conceitos no projeto de circuitos integrados, ilustrando, a partir do nível de abstração mais alto possível, o processo de desenvolvimento de um chip por um centro de projeto.

Partindo de elementos como a especificação e o kit de tecnologia, são empregados avançados recursos computacionais e ferramentas de EDA/CAD para a concepção de um chip. Como produto deste trabalho são realizadas diversas aplicações e os resultados obtidos servem como parâmetro de realimentação para a implementação de sistemas de qualidade. Este diagrama pode ser visto como canônico, uma vez que pode ser aplicado de modo recursivo em todo fluxo de projeto.

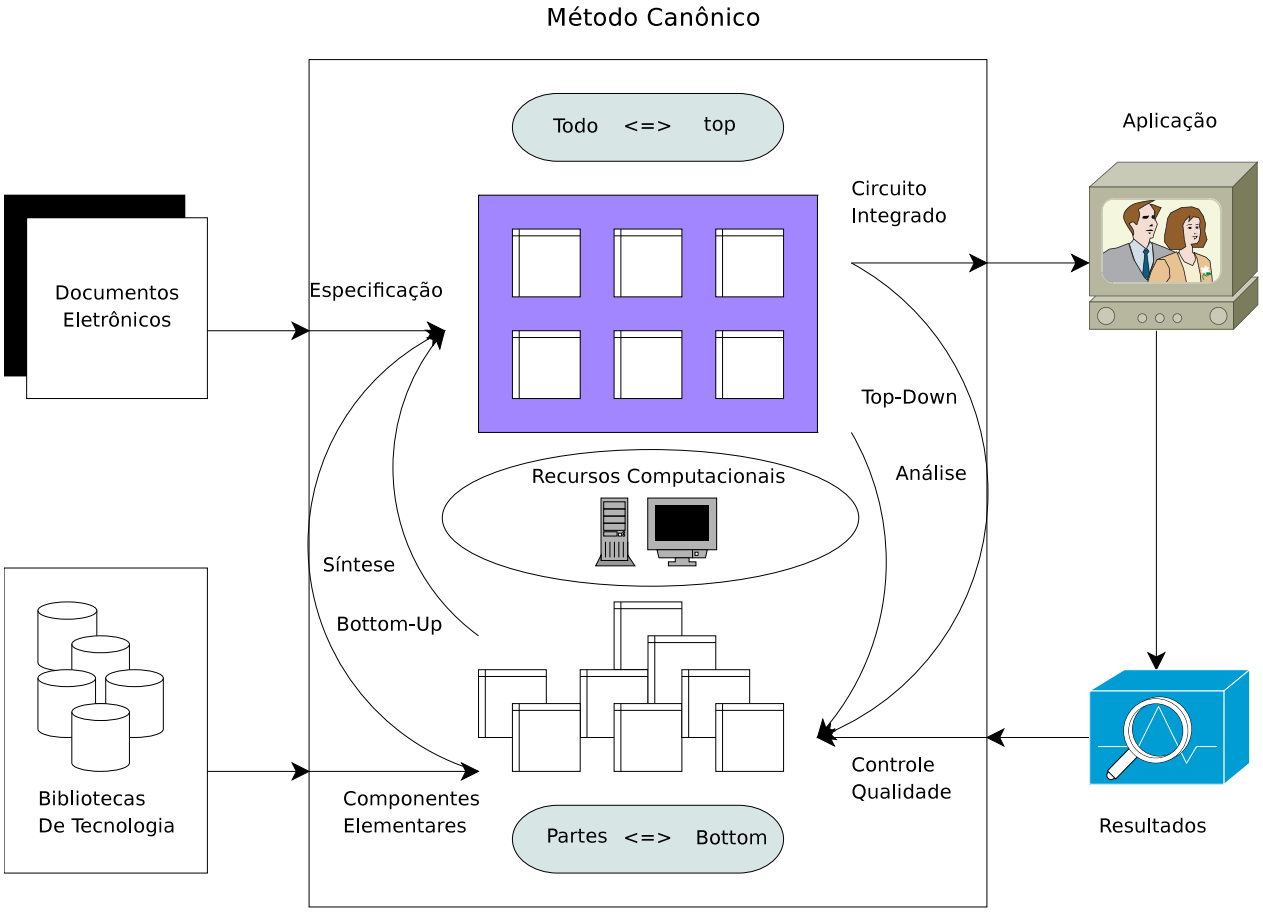


Fig. 2.2: Métodos “bottom-up” e “top-down” aplicados ao desenvolvimento de CIs

No ambiente de projeto real, estas duas abordagens ocorrem em paralelo, com predominância de uma abordagem em detrimento da outra. Fatores como a complexidade do projeto, a etapa do fluxo, existência de ferramentas para simulação sistêmica e gerenciamento da base de dados, competência técnica dos projetistas, métodos gerenciais e até mesmo a arquitetura organizacional influenciam diretamente neste equilíbrio ao longo do ciclo de execução de projetos.

Independentemente da categoria do circuito a ser implementado (digital, analógico ou “mixed-signal”) podem ser empregadas técnicas predominantemente “bottom-up”ou predominantemente “top-down”. Em geral o projeto de circuitos lógicos, utilizando-se linguagens de descrição de hardware, é inerentemente top-down, enquanto que o projeto analógico é “bottom-up”. De modo complementar, a síntese lógica assume como ponto de partida uma biblioteca de tecnologia com células digitais pré-definidas. Na figura 2.3 está ilustrada a convergência entre os dois métodos, levando-se em conta atividades de projeto de alto nível.

Pode-se dar ênfase ao método “top-down” quando o “floorplan” (otimização de blocos levando-se em conta a dimensão, interface com outros blocos e razão de aspecto) é detalhado e são criadas abstrações somente com o contorno dos blocos inferiores. A área, geometria e razão de aspecto, posicionamento de pinos e canais de roteamento, dentre outros parâmetros, são estimados logo no início do fluxo. A ênfase no método “bottom-up” é dada quando o “layout” de sub-blocos é feito sem um detalhamento do “floorplan”, resultando na necessidade de retrabalhar os blocos ao serem integrados, visando a otimização de área, pinos e outros elementos que surgem quando se eleva o nível de abstração. De modo geral, o desenvolvimento de sub-blocos para sua posterior integração, apesar de um processo essencialmente “bottom-up”, ainda possui elementos “top-down”, uma vez que existe uma diretriz sistêmica para o direcionamento da implementação visando a conformidade com determinada especificação.

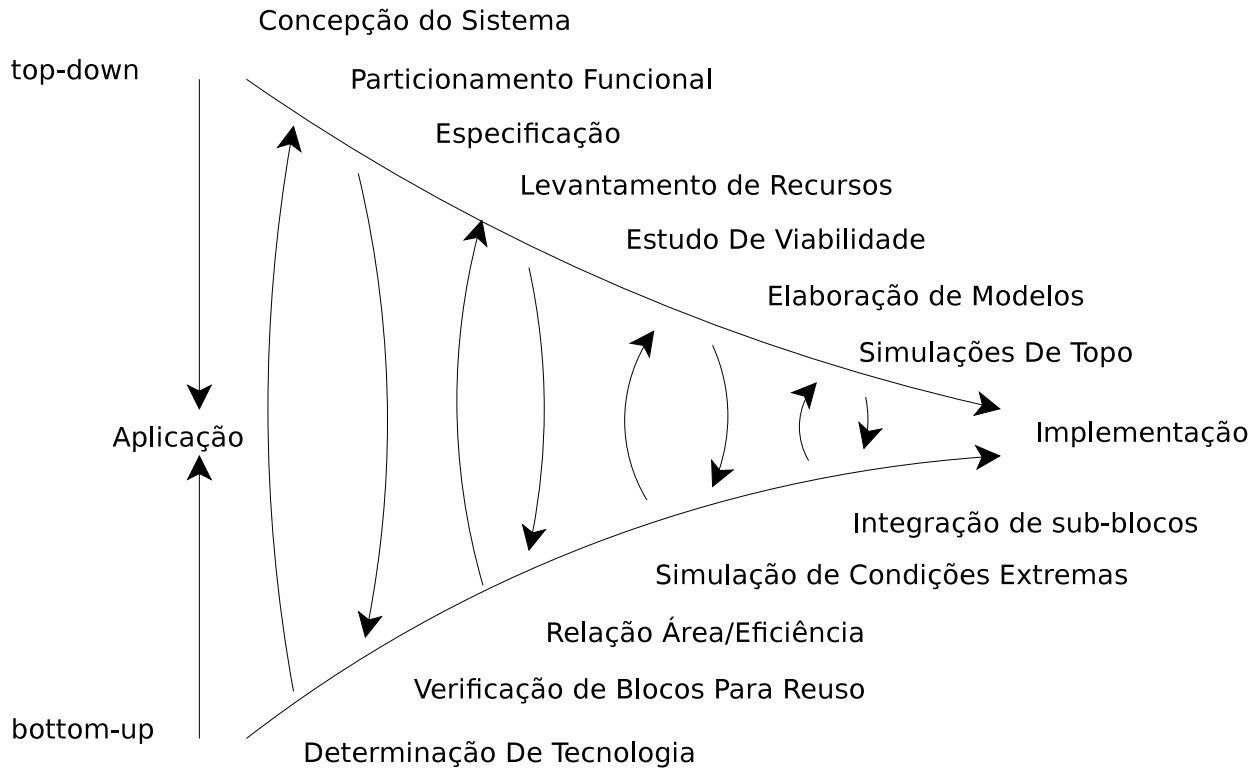


Fig. 2.3: Convergência entre os métodos “bottom-up”e “top-down”.

As ferramentas de projeto tem sido desenvolvidas sob o paradigma de linguagens de sistema que influenciam diretamente nos métodos envolvidos na concepção de sistemas eletrônicos. Passos discretos são necessários para a exploração de alternativas de projeto e implementação de sistemas. Em [31], existe a proposta bastante inovadora do simulador SelfHDL e um conjunto metodológico bastante completo que ilustra claramente este problema e alternativas para solucioná-lo. De certa forma, o ambiente proposto, principalmente do ponto de vista dos recursos de automação, encontra correspondência com esta referência bibliográfica.

2.2.2 Parametrização

A parametrização do fluxo de projeto é uma poderosa técnica, pois permite a rápida replicação de determinadas estruturas ao separar do processo as variáveis importantes, ficando desta forma disponíveis como elementos de entrada. Isto permite, por exemplo, a validação do fluxo de projeto com diferentes tecnologias, partições, módulos, em suma diferentes cenários dependendo do seu mapeamento paramétrico.

Em um fluxo de projeto não parametrizado, utilizando-se ferramentas convencionais, observa-se a necessidade recorrente de entrada de informação, ou seja, repetidas interações com a ferramenta. Como resultado o conjunto de parâmetros fica distribuído no fluxo de projeto.

Capítulo 3

Arquitetura

A metodologia utilizada e a arquitetura proposta resultante devem ser vistas como primordiais, pois devido a grande variação do ambiente de projeto, qualquer implementação da interface gráfica paramétrica terá validade somente no ambiente em que foi originalmente desenvolvida. Como visto anteriormente, existe uma multiplicidade de combinações de ferramentas e métodos, o que dificulta a especificação de sistemas genéricos. Os conceitos abordados neste capítulo, devido ao seu nível de abstração, pode, por outro lado, ser aplicados em outros domínios através de implementações específicas, mas utilizando a mesma arquitetura básica.

Para que se possa efetivamente situar a interface gráfica paramétrica no contexto computacional atual, inicialmente será feito o levantamento de sistemas básicos. Isto permitirá a determinação dos requisitos necessários à implementação com base em elementos pré-existentes. Como será visto, esta abordagem permite o máximo reuso possibilitando a criação de sistemas com funcionalidades complexas e máxima eficiência com custo relativamente baixo de programação.

A união destes elementos com as necessidades apresentadas no capítulo anterior estabelecem as condições de contorno que permitem determinação dos requisitos. Visando uma compreensão mais detalhada, a descrição da arquitetura será feita através de seus aspectos estruturais e funcionais na forma de figuras, diagramas e descrições textuais. Ao final do capítulo, será realizado o levantamento dos principais benefícios da arquitetura proposta.

3.1 Elementar

Como descrito nos capítulos anteriores, este trabalho foi desenvolvido visando sua reutilização por especialistas que queiram construir interfaces paramétricas como ferramenta para a automação de suas atividades envolvendo recursos computacionais. Para que se possa efetivamente situar a interface paramétrica no contexto computacional atual, os aspectos fundamentais do ambiente serão inicialmente apresentados. Os elementos básicos do ambiente computacional, apesar de gerais, devem ser analisados, pois devido à natureza dos aplicativos utilizados [32], existem múltiplas escolhas que podem dificultar ou mesmo inviabilizar certas funcionalidades do sistema proposto. O repertório de programas e bibliotecas disponíveis e a configuração de parâmetros são os principais elementos encontrados neste contexto.

A partir do sistema operacional, níveis de abstração conferidos por camadas de software, per-

mitem a elaboração de sistemas que podem ser reutilizados em diferentes plataformas, mantendo assim a generalidade da pesquisa. Neste trabalho o "hardware" será tratado como uma abstração, uma vez capaz de processar a infra-estrutura computacional básica, sua especificação exata é irrelevante. Entende-se por ambiente computacional básico uma estação de trabalho GNU/Linux, o que inclui acesso a servidores com ferramentas gerais, incluindo o núcleo da linguagem TCL, bibliotecas e pacotes. A arquitetura apresentada na figura 3.1 é o substrato para a utilização dos "softwares" disponíveis nas diferentes modalidades, "free-software" e "open-source", principalmente voltadas para aplicações gerais.

Apesar da utilização de servidores Unix ainda ser comum, o sistema proposto tomou como prioridade o sistema operacional Linux [6], devido às suas características únicas de acessibilidade e performance. Outra motivação foi a recente popularização de estações e servidores rodando GNU/Linux para o projeto de CIs, uma vez que desenvolvedores de ferramentas proprietárias passaram a disponibilizar versões para esta plataforma.

Na figura 3.1 estão representados os diferentes sistemas usualmente encontrados na composição básica de uma estação de trabalho. Iniciando pelos componentes do sistema operacional, o kernel, o ambiente de janelas X-Window [33] e o gerenciador de janelas Enlightenment [34]. Ferramentas gerais de comunicação, documentação, gerenciamento pessoal e produtividade, juntamente com as específicas de EDA/CAD, são executadas nesta arquitetura computacional. A interface gráfica paramétrica se coloca como elemento integrador deste sistema ao permear toda a estrutura com acesso a todos os elementos do sistema, de baixo e alto nível. O sistema como aparece para o usuário final está exemplificado na figura 3.2.

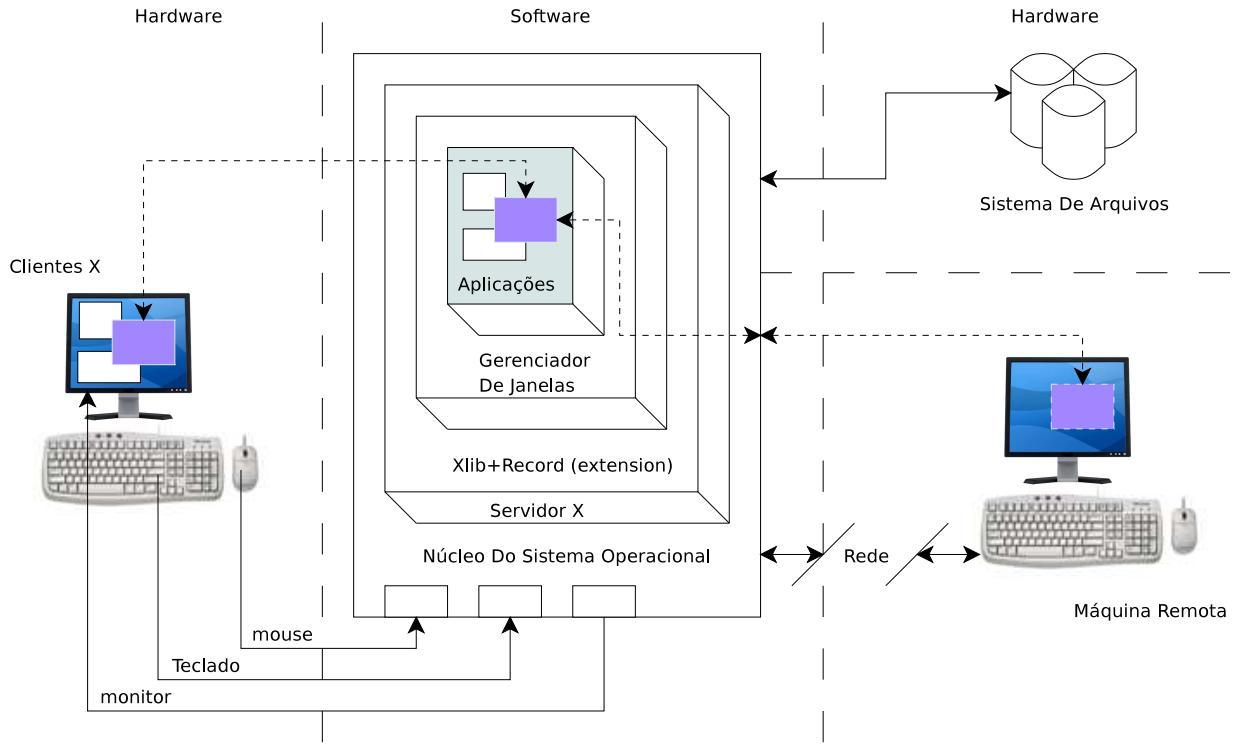


Fig. 3.1: Arquitetura X Window.

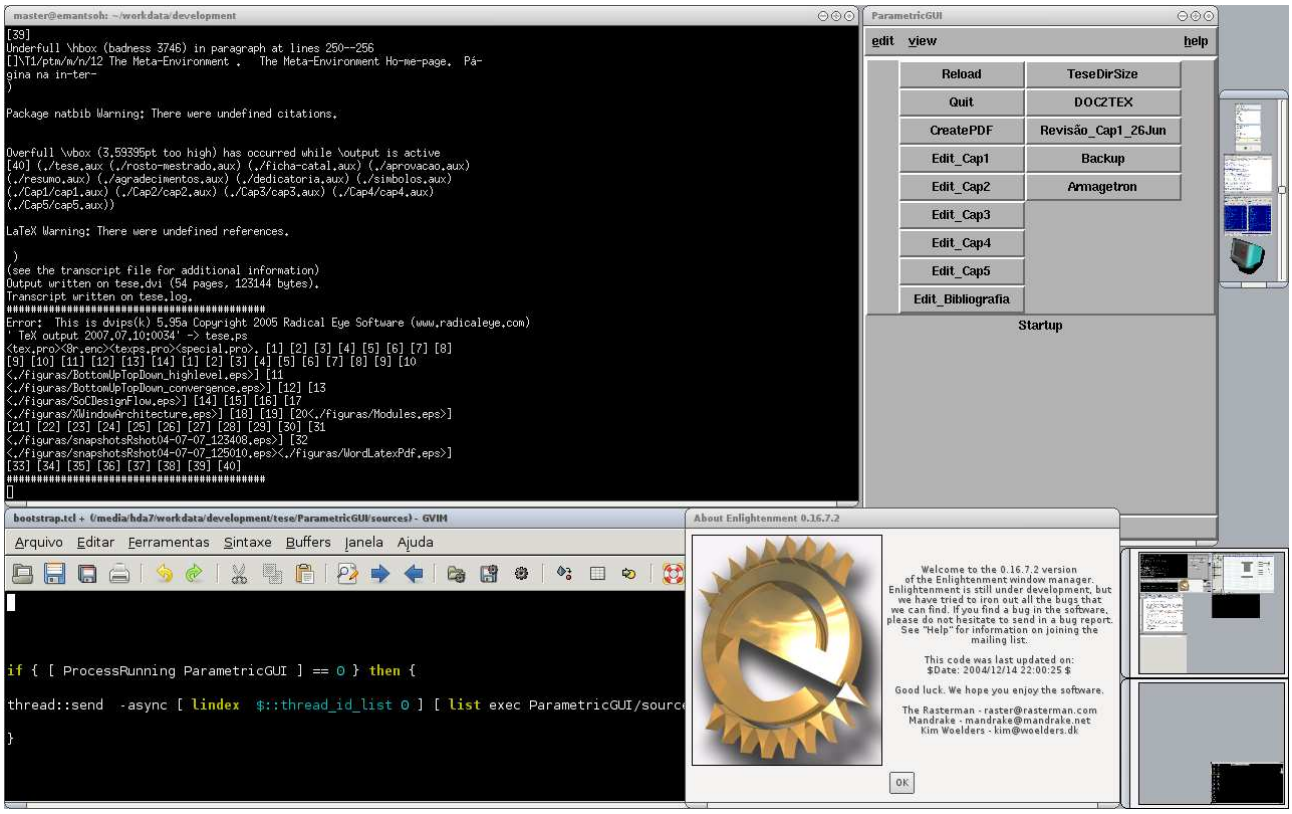


Fig. 3.2: Gerenciador de janelas Enlightenment.

3.2 Interface gráfica paramétrica

Diante da estrutura organizacional e tecnológica exposta no capítulo 2, serão elaborados pontos considerados fundamentais para a determinação dos requisitos e conseqüentemente desenvolvimento da interface gráfica paramétrica. A arquitetura toma por base um modelo estrutural de alto nível da infra-estrutura computacional básica da figura 3.1.

A arquitetura básica da interface paramétrica é composta pelo módulo de bootstrap, o núcleo do sistema, arquivos de configuração e o construtor de interfaces. Estes módulos operam em conjunto para a criação da interface gráfica paramétrica. Tendo-se em mente a implementação das aplicações, alguns módulos específicos como o verilog "parser", o controlador do gerenciador de janelas "X-Window" e o gerador de grafos, foram também desenvolvidos, mas mantidos como extensões e não fazendo parte do sistema básico.

A implementação de monitores para o gerenciamento e controle de padrões interativos permite fazer com que a interface gráfica possa sofrer mutações dependendo dos eventos gerados pelo usuário ou pelo próprio sistema. Em uma situação hipotética, um engenheiro eletricista interessado em ações poderia ter sua estação transformada em um sistema de negociação (Trading System) de modo automático, com o fechamento de aplicativos, abertura de outros, estruturação das janelas na área de trabalho, execução de back-up, dentre outras tarefas pertinentes a um procedimento de chaveamento de contextos computacionais como este.

3.2.1 Requisitos

A descrição dos requisitos permitirá a elaboração de critérios mais precisos tendo-se em vista a implementação da arquitetura e servirá de "framework" conceitual para a concepção da interface gráfica paramétrica. Estes critérios têm como alvo o sistema proposto, independentemente dos elementos de integração, que podem ser quaisquer. Neste sentido, deve ficar clara a distinção entre os blocos internos e sistemas externos.

Implementação

A interface gráfica paramétrica deve possuir uma implementação aberta do ponto de vista dos recursos utilizados para sua construção. Apesar de integrar ferramentas proprietárias do fluxo de projeto, a ferramenta desenvolvida deve utilizar elementos que possuam licenças GNU e que permitam seu uso sem restrições legais.

O sistema deve ser o mais auto contido possível a partir de dependências primárias. Na medida do possível, manter o sistema com o menor número de elementos externos à linguagem TCL. Nas ocasiões em que o custo de desenvolvimento não se justificar, optar por elementos de fácil disponibilidade e integração. Isto irá minimizar o número de domínios mantendo reduzido o número de dependências necessárias.

Ser extensível permitindo que novos módulos sejam adicionados com facilidade. Tornar o sistema o mais simples possível do ponto de vista dos requisitos de pacotes, tornando possível a adição e remoção de extensões mantendo funcionalidades básicas. Especificar um conjunto mínimo de pacotes. Deve também ser possível a extensão de sua estrutura de dados e de configuração sem a necessidade de re-programar o sistema como um todo.

Ser implementável e gerenciável por uma pessoa. O sistema deve ser pequeno o suficiente para ser desenvolvido em pouco tempo e com recursos limitados e apresentar coerência em sua estrutura para que possa ser compreendido por completo.

Permitir fácil acesso ao código responsável pela execução de um procedimento. Desta forma pode-se experimentar novas abordagens e parâmetros alternativos, que podem ser incorporados através de novos elementos de interface de forma automática.

Possibilidade de controle dos processos autônomos permitindo sua interrupção, interação e continuidade durante a execução da série de comandos. Facilita a detecção de erros na implementação, confere visibilidade aos comandos canônicos utilizados na implementação de determinada funcionalidade.

Funcionais

Possuir funcionalidades gráficas paramétricas. Permitir a parametrização dos elementos gráficos. A interface gráfica, ao ter seus parâmetros modificados, deve ser capaz de atualizar suas unidades gráficas sem interrupção de sua execução.

Trabalhar com grafos e árvores. O sistema deve apresentar recursos para a interação com estruturas matemáticas como grafos, árvores e matrizes, tanto em modo texto como em modo gráfico.

Contribuir com ferramentas que auxiliem no gerenciamento da multiplicidade de opções existentes no ambiente de projeto atual. Possibilitar o acompanhamento do projeto em tempo real, mantendo para isto atualizada grande quantidade de resultados disponíveis através da interface paramétrica.

Manter o projeto consistente com o tratamento de eventos por meio de procedimentos, que coordenados garantam a consistência de inúmeras dependências existentes no complexo ambiente de projetos de CIs.

Permitir o controle de ferramentas remotamente em modo texto e gráfico. Incluir na arquitetura módulos que tenham a habilidade de automatizar processos interativos, incluindo o ambiente gráfico.

Possuir capacidades autônomas para gerar a infra-estrutura de design necessária, criando os arquivos de configuração necessários para a execução de determinada ferramenta do fluxo de projeto.

Colaboração

A infra-estrutura computacional desenvolvida deve favorecer a colaboração. Devido à complexidade do fluxo de projeto, é fundamental que haja colaboração no desenvolvimento de elementos canônicos para a interface gráfica paramétrica. Neste sentido o sistema deve permitir a troca de "scripts" de modo estruturado através de uma plataforma comum.

3.2.2 Diagrama de blocos

Na figura 3.3 os módulos da interface gráfica paramétrica estão evidenciados. Neste diagrama está representada a interface gráfica paramétrica criada para automação da documentação. Quando o sistema é iniciado com a estrutura paramétrica mínima, observa-se a criação de uma janela padrão vazia, ou seja, desprovida de qualquer elemento gráfico que não seja simplesmente a janela. Esta é a saída básica do interpretador "wish", que é essencialmente o núcleo da linguagem TCL com a extensão TK. Nesta situação apenas parâmetros básicos, como o nome do arquivo contento o módulo principal são definidos.

Quando são criados arquivos de configuração com os parâmetros elementares, a interface paramétrica ao ser iniciada apresenta em sua janela principal três elementos distintos, que estão dispostos em diferentes painéis, ou em inglês, como utilizado pelo pacote TK, "frames". Esta divisão é arbitrária e pode ser modificada, bastando alterar o arquivo de configuração correspondente. Eventualmente torna-se necessário a alteração dos procedimentos, tendo em vista a adição ou exclusão de novos parâmetros.

3.2.3 Blocos funcionais

Para a compreensão da arquitetura, deve-se partir dos elementos mais gerais, que dizem respeito à interação entre homem-computador. A interação com o ambiente computacional conta com dois paradigmas cuja fundamentação se encontra nos dois modo de operação de sistemas, o textual, mais natural para programadores e o modo gráfico mais intuitivo para projetistas analógicos.

Sistemas não otimizados possuem como característica utilizar seus recursos na direção de facilitar o uso para iniciantes, utilizando demasiadamente recursos gráficos. Sistemas avançados utilizam elementos gráficos e em modo texto em um ambiente híbrido. Isto pode ser facilmente identificado ao se comparar sistemas operacionais como o Windows e o GNU/Linux. Enquanto o Windows gasta recursos para se tornar mais acessível e atraente aos usuários, retirando os elementos de modo texto, o GNU/Linux através de seu desenvolvimento descentralizado, avança nas duas frentes, tornando-se

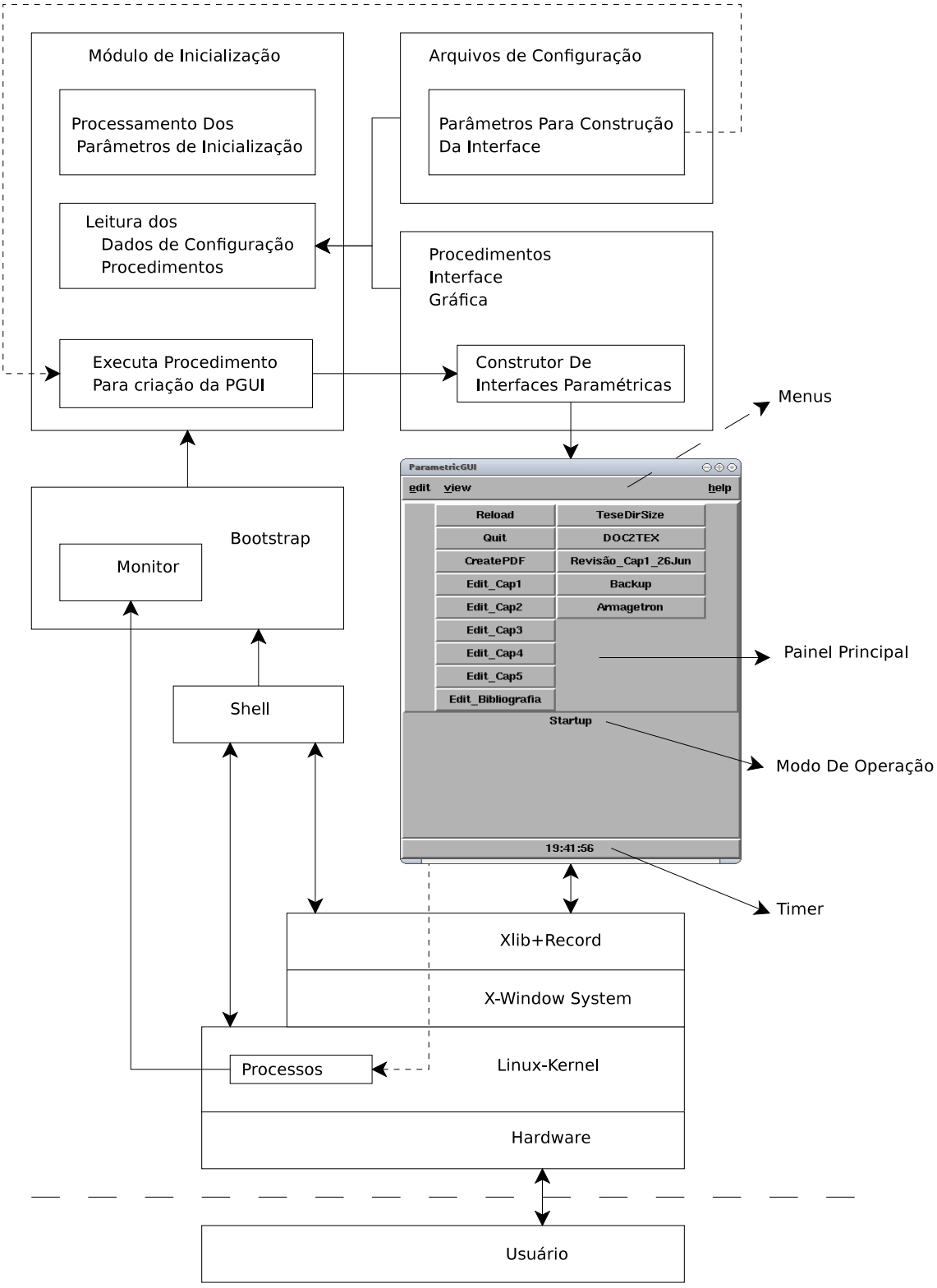


Fig. 3.3: Diagrama de blocos da interface paramétrica.

cada vez mais complexo, sem contanto deixar de lado as características elementares do modo texto, que representam poderosas ferramentas de programação para usuários avançados.

A interface gráfica paramétrica transcende a interação com o usuário. Ao admitir que a utilização do sistema está direcionada para seu desenvolvimento, cria-se um plano onde não há distinção entre desenvolvedor e usuário. Ao uní-los em uma única entidade, obtêm-se uma poderosa ferramenta de programação. Esta compreensão possibilita dentre outras coisas a automação do próprio desenvolvimento da interface, possibilitando uma experiência muito semelhante às características encontradas em ambientes de programação que implementam os recursos da linguagem Smalltalk [35].

Bootstrap

O módulo de bootstrap é um pequeno programa responsável por invocar programas de inicialização e monitorá-los a fim de garantir sua permanente execução. O mesmo é responsável pela manutenção vital do núcleo do sistema. Sem ele, quando algum procedimento do sistema é alterado e eventualmente ocorrer um erro, o funcionamento da interface gráfica paramétrica será encerrado. Com o sistema de bootstrap é possível adotar diversas estratégias para que o sistema continue em funcionamento mesmo quando haja um erro. Mecanismos inteligentes podem ser implementados para monitorar os processos da máquina servindo para manter em execução grande número de processos.

Na ocorrência de um erro, pode-se proceder de diversas maneiras. A estratégia mais simples é manter um temporizador que verifica se o núcleo do sistema está ativo, em caso contrário, o mesmo é reinicializado. Como fica evidente, quando houver uma falha, o sistema ficará fora de funcionamento até que o temporizador, no pior caso, conclua um ciclo. Outra abordagem, um pouco mais elaborada, faz uma tentativa de carregar o sistema antes de parar sua execução atual e havendo algum tipo de erro no código, retorna uma mensagem e mantém em execução o sistema original. Desta forma, ao se tentar carregar o sistema com determinada modificação em seus parâmetros ou procedimentos, pode-se abortar a operação em caso de falha antes de finalizar a instância original. O problema neste caso está nas situações que não estão relacionadas com a mudança do código fonte, por exemplo, espaço insuficiente em disco, que pode ser uma condição temporária.

Para ser efetivo, o sistema de bootstrap deve implementar ambas as abordagens, principalmente quando o sistema em desenvolvimento estiver ao mesmo tempo sendo utilizado no gerenciamento de tarefas cuja interrupção deve ser evitada. Eventualmente pode-se disparar diferentes sistemas para estas atividades. Do ponto de vista de integração do ambiente computacional isto representa maior fragmentação, podendo ser contornada por um gerenciador de seções, que dentre outras coisas permitiria o gerenciamento dos diferentes estágios de evolução da interface gráfica paramétrica.

Arquivos de configuração

Parâmetros tratados inicialmente na arquitetura como elementos gerais, agora são especificados dentro do universo da linguagem TCL como sendo basicamente estruturas de dados, variáveis, vetores ou mesmo estruturas mais complexas como matrizes e grafos armazenados em disco sob a forma de arquivos de configuração.

Independentemente de como for feita a implementação, os parâmetros devem ser passados para os procedimentos dentro do módulo construtor de interface. Como este construtor está no domínio da linguagem TCL, de um modo ou de outro, os parâmetros devem ser passados neste domínio.

Do ponto de vista do fluxo de projeto, a determinação de uma série de informações relativas à execução das diferentes ferramentas de EDA/CAD de forma manual, permitiu a elaboração de um arquivo de configuração com matrizes concentrando os parâmetros de projeto em único lugar, minimizando as redundâncias existentes em comparação com a entrada de informações no fluxo interativo convencional.

Parametrização dinâmica diz respeito ao método no qual existe pré-processamento dos parâmetros antes de sua utilização. Neste caso, novos níveis de complexidade são adicionados, do ponto de vista da codificação, o que garante grande poder e concisão na descrição dos parâmetros. A utilização de código permite dentre outras coisas: extração de parâmetros em tempo real acessando recursos do sistema operacional, expansão de parâmetros através da utilização de iterações e a utilização de variáveis, permitindo a criação de meta parâmetros. A parametrização dinâmica pode ainda ser utilizada para compactar a entrada de dados, por exemplo, quando os elementos estão de certa forma indexados.

Núcleo do sistema

O núcleo do subsistema principal é responsável pela inicialização, que compreende o tratamento dos argumentos permitindo iniciar o sistema em uma determinada condição ou estado. Diferentes contextos de interface podem ser construídos para acesso imediato após a execução do programa. Múltiplas instâncias permitem acesso à ampla variedade de controles, tornando possível o gerenciamento de diversas atividades de projeto simultaneamente.

As extensões são também carregadas juntamente com os arquivos de configuração escritos em TCL e os procedimentos. Existe comunicação direta entre este módulo e o módulo de bootstrap, que monitora seu processo mantendo sua execução constante. Quando, através da interface gráfica, é dado um comando para que o sistema seja encerrado, o módulo de bootstrap deve ser informado para não recarregar o mesmo novamente.

Construtor de interface

Este módulo é responsável pela efetiva construção da Interface Gráfica como aparece ao usuário. De modo genérico está restrito e é governado pela implementação do sistema X-Window, como descrito anteriormente. Pode-se distinguir neste bloco do sistema os conceitos de elementos gráficos canônicos, que em última instância, são os constructos básicos disponíveis na linguagem, considerada sem outras extensões, tendo-se em mente a manutenção do requisito de manter o sistema autocontido ou hermético. Uma vez definidos os elementos gráficos canônicos, outras unidades são agregadas através de sua múltipla utilização. Extensões são adicionadas para formar elementos gráficos mais complexos.

Parser

Diante da necessidade de se obter informações de diferentes domínios, este módulo é bastante útil para transformar diversas fontes de informação em parâmetros. Procedimentos para analisar código verilog podem ser usados como entrada para a criação de botões que permitem a execução de programas, como por exemplo, simuladores. A criação de novas hierarquias e novos módulos verilog

resultaria na modificação da interface gráfica de modo a refletir esta atualização, sem a necessidade de modificação do seu código.

Gerador de grafos

O gerenciamento de grandes bases de dados, como a informação de sistemas de microeletrônica, passa necessariamente por algum sistema que permita sua visualização. Os ambientes de projeto atuais empregam o paradigma de árvore, semelhante ao da estrutura de diretórios para descrever a dependência hierárquica entre módulos. Uma visualização na forma de grafos permite a verificação da quantidade de dependências, reutilização de sub-blocos e identificação de sistemas independentes através de grafos desconexos.

A transformação de diversas estruturas de dados gerada por outras ferramentas em diagramas é o objetivo oposto de ferramentas como o DIA [36], que são extremamente úteis para a criação manual de diagramas. Procedimentos implementados neste módulo são responsáveis pela transformação de forma rápida e automática de estruturas de dados em grafos.

Controlador X-window

Para o controle total da estação de trabalho foi implementado um módulo responsável pelo controle do X-Window através do gerenciador de janelas. Dentre outras funcionalidades este módulo possui a capacidade de extrair informação relativa ao ambiente X e tomar ações. Exemplos são: o redimensionamento de janelas ou a gravação e reprodução de eventos, que permitem a automação gráfica de atividades.

3.2.4 Benefícios

Dentre os principais benefícios da arquitetura proposta, podemos destacar:

- Rápido desenvolvimento, além de simples, técnicas de meta-programação auxiliam no desenvolvimento do próprio sistema;
- Possibilita compartilhar código de modo sistemático e organizado, auxiliando na colaboração entre projetistas;
- O projetista possui a chance de desenvolver fluxos de projeto alternativos e mantê-los de modo organizado e acessíveis através da interface;
- Possibilita o uso das ferramentas do ambiente de projeto como “point tools”, tornando possível sua integração em um fluxo de projeto complexo de modo consistente;
- Detalhes do fluxo de projeto podem ser encapsulados, tornando o mesmo transparente para outros projetistas que queiram apenas utilizar a ferramenta, sem se preocupar com os detalhes de implementação e extensão;
- Torna clara a divisão entre o que o projetista deve saber e o que é parte da ferramenta;

- Simplifica e padroniza métodos e processos através de sua automação e encapsulamento na interface gráfica paramétrica;
- O sistema pode ser tão fácil de usar quanto se queira. Os mecanismos de contração e expansão de comandos possibilitam o agrupamento de diversas etapas em uma única, encapsulada em um simples botão, ou o contrário, tornando claro para o usuário a seqüência de etapas executadas pela interface;
- Permite sua rápida alteração e orientação para diferentes contextos através da implementação de elementos de configuração dinâmicos;

Capítulo 4

Implementação

4.1 Metodologia

Ao invés de implementar o sistema proposto utilizando somente o repertório básico da linguagem de programação, procedeu-se inicialmente intensa pesquisa para encontrar módulos prontos que pudessem ser facilmente integrados. Esta abordagem possibilitou o aumento da complexidade do sistema com a atribuição de funcionalidades sofisticadas sem demandar demasiado recurso, agilizando o processo de desenvolvimento.

O método utilizado na concepção da arquitetura e sua implementação estão detalhados através da seguinte seqüência hierárquica de etapas. É possível identificar os três passos fundamentais descritos no primeiro capítulo. Elaboração da interface básica, independente de aplicação, desenvolvimento de "scripts" isolados e, finalmente, a integração de ambos na concepção de uma aplicação. Os recursos computacionais apresentados na seção 3.1 serão utilizados como substrato para a execução das seguintes etapas.

- Implementação da arquitetura computacional.
  - Concepção da arquitetura.
  - Descrição dos módulos funcionais.
  - Elaboração de diagramas descritivos e de eventos.
  - Determinação e elaboração de requisitos.
- Implementação de blocos fundamentais básicos.
  - Elaboração da infra-estrutura básica.
  - Criação de meta elementos de interface.
  - Utilização e testes de meta elementos para desenvolver o próprio sistema.
- Determinação de tarefa computacional.
  - Utilização de ferramentas do modo como são disponibilizadas originalmente.
  - Execução manual.

    * Execução da tarefa pelos métodos conhecidos.
    * Determinação dos comandos elementares.

- Implementação de scripts que executem os comandos elementares.
  - Substituição de algumas etapas pela execução dos scripts.
  - Execução semi-automática do fluxo de projeto.
  - Criação de macro procedimentos e objetos.
- Verificação das ferramentas, pacotes e extensões disponíveis.
  - Determinação dos parâmetros de entrada.
  - Extensão do arquivo de configuração
  - Extensão do conjunto de funções e objetos.
  - Escolha das ferramentas gerais.
  - Composição dos elementos de entrada e parâmetros.
  - Execução das ferramentas com os parâmetros.
- Implementação de procedimentos para execução das ferramentas.
  - Concepção dos monitores para identificar mudanças de estado do sistema.
  - Determinação dos elementos de saída.
  - Tratamento dos elementos de saída.
  - Mensagens normais.
  - Mensagens de erro.
  - Implementação de rotinas para geração de relatórios.
  - Execução de procedimentos específicos.

4.2 Tool Command Language

A primeira consideração que deve ser feita, com relação à implementação de um software, diz respeito às linguagens de programação que serão utilizadas. Vários critérios podem ser utilizados para esta determinação, dentre eles podem ser destacados: as diferenças entre linguagens de sistema e linguagens de script, apresentada de modo bastante sistemático em [37], os conhecimentos do programador, a existência de código para reuso e sua viabilidade na satisfação dos requisitos da aplicação.

A escolha da Linguagem TCL [38], dentro do contexto apresentado no capítulo 2, é bastante natural. A análise dos critérios mencionados mostra que as necessidades da arquitetura são mais facilmente atendidas quando se adota uma linguagem de script. Primeiramente devido ao caráter de integração do sistema e de modo complementar, o fato de ser uma linguagem interpretada, sem

a necessidade da etapa de compilação. Além disso, existem diversos trabalhos que tratam de sua integração com outras linguagens [39].

Com o objetivo de tornar este sistema viável de ser reconstruído por qualquer projetista com a intenção de automatizar sua interação com os recursos computacionais, foram empregadas bibliotecas cujo desenvolvimento está consolidado. Embora os sistemas descritos no capítulo 2 sejam bastante atraentes para uso no ambiente industrial, a indisponibilidade destes aplicativos contribui para que sua relevância seja maior no aspecto conceitual do que prático.

Os elementos externos que podem ser integrados ao TCL são virtualmente ilimitados. Um dos critérios para a utilização de um determinado pacote é sua disponibilidade e atividade. Pacotes fáceis de se encontrar e que tenham ampla atividade em seu desenvolvimento são recomendados. Desta forma, pode-se manter o sistema desenvolvido em constante atualização. O conjunto de pacotes descritos a seguir atende de maneira adequada aos critérios de pertinência, disponibilidade e tamanho.

4.2.1 Thread

Este pacote foi utilizado para implementar recursos no sistema de bootstrap que permitissem o monitoramento de diversas interfaces gráficas paramétricas ao mesmo tempo. A habilidade de distribuir diferentes aplicações em diferentes processos foi utilizada para a comparação de resultados com diferentes arquivos de configuração e para a utilização simultânea de diferentes interfaces gráficas.

O procedimento na figura 4.1 exemplifica como implementar uma rotina para automação do processo de criação de processos. Funções de mais alto nível são responsáveis pelo tratamento de parâmetros e criação sistemática de processos, permitindo a execução e acompanhamento de diversos programas simultaneamente.

```
proc CreateThread { } {

   set   ::current_thread_id  [ thread::create  {  thread::wait  } ]
   set   current_time         [ clock format [clock sec] -format %H:%M:%S  ]
   #puts $current_time

   set script { puts "Created Thread: [thread::id] "  }

   thread::send -async $::current_thread_id  $script result
   thread::release

return $::current_thread_id

}
```

Fig. 4.1: Procedimento para criação de threads.

4.2.2 Expect

Desenvolvido para automação de processos interativos, este pacote foi essencial, sendo usado com o objetivo de acionar remotamente terminais, e com isso permitir a intervenção manual em processos automatizados. A utilização de terminais remotos possibilita que o projetista tenha controle total sobre o que o "script" está fazendo, podendo inclusive parar uma execução, tomando o controle da mesma. Modificações positivas podem ser encapsuladas em novos comandos criando-se simultaneamente cadeias de utilização e desenvolvimento.

Outra vantagem de se utilizar terminais remotos foi permitir a verificação e o acompanhamento dos comandos executados, permitindo que usuários menos experientes aprendam a medida em que estiverem utilizando a interface gráfica paramétrica. Esta abordagem possibilita o acesso imediato aos comandos mais elementares, na medida em que os procedimentos estejam devidamente expandidos até seus comandos canônicos.

Os procedimentos de alto nível utilizados para criação e comunicação com terminais remotos está ilustrado na figura 4.2. Nesta figura pode-se identificador o mecanismo para envio de um comando, que é passado como parâmetro, para determinada "Shell".

```
proc xtermSend { A i_exp_send } {

  exp_send -raw -i  $i_exp_send -- $A

}

proc sendTo {to} {
    exp_send -raw -i $to -- $::expect_out(buffer)
}
```

Fig. 4.2: Código utilizado para comunicação com terminal remoto.

Na figura 4.3 esta exemplificado um uso fundamental para administração de sistemas. Muitas tarefas administrativas, como a manutenção da área de projetos, necessitam de constante atenção. Atividades gerais como o gerenciamento do grupo de usuários, senhas, servidores de e-mail, quotas, utilização de licenças, podem ser automatizadas de forma incremental através da implementação contínua de rotinas efetivadas e testadas para cada função. Outra utilização importante deste pacote foi na automação da mudança de grupos e usuários, pois devido a abertura de um novo interpretador, torna-se necessária a execução de "scripts" reentrantes sofisticados.

```
proc UserCommand { user alias command } {

set timeout 100
set prompt "(:|%|\\\$)\[^!-~]*$"
log_user 1

#puts $user
#puts $alias
#puts $command

spawn sudo su - $user
expect   -i $spawn_id -re $prompt {}
exp_send -i $spawn_id "$alias \n"
expect   -i $spawn_id -re $prompt {}
exp_send -i $spawn_id "$command \n"
expect   -i $spawn_id -re $prompt {}
}
```

Fig. 4.3: Código utilizado para automação do comando sudo.

4.2.3 TCLLIB

Este é um pacote básico. Sua utilização garante acesso a comandos relativos ao sistema de arquivos que não estão disponíveis na distribuição básica da linguagem. Estruturas matemáticas mais

complexas, como grafos e matrizes, também estão disponíveis e se mostram essenciais para a criação de estruturas paramétricas mais complexas.

4.2.4 Sistema de abas

O sistema de abas da figura 4.4, conhecido por "tabs" ou "notebook" em inglês, tem como função a troca de conteúdo da área principal de diversos aplicativos. Este tipo de sistema gráfico paramétrico não está disponível, pelo menos até a versão corrente à edição desta dissertação, dentro dos elementos gráficos básicos do núcleo do pacote TK.

A necessidade de criação do sistema de abas na interface gráfica tornou necessário considerar as seguintes possibilidades. Desenvolver os procedimentos a partir das primitivas da linguagem TK, estendendo os elementos gráficos disponíveis, ou então, utilizar bibliotecas gráficas sofisticada, que podem ser encontradas em [40].

Atendo-se a premissa de tornar o sistema desenvolvido o mais auto contido possível, optou-se por uma terceira alternativa. Alinhada a metodologia proposta, foi feita uma pesquisa a fim de se identificar desenvolvimento pré-existentes para reutilização. Neste sentido, após pesquisa, foi possível encontrar na internet procedimentos prontos para realizar esta função.

Fig. 4.4: Interface gráfica resultante da parametrização do "RNotebook".

Várias alternativas foram testadas levando-se em conta a simplicidade do código e sua compreensão como um todo. O conjunto de procedimentos desenvolvidos por Daniel Roche, disponibilizados na Internet sob a denominação de "Resizeable Notebook", foi o que obteve a melhor relação entre custo e benefício. O mesmo apresentou bons resultados com relativa facilidade para integração do código na interface paramétrica. Na figura 4.5 está exemplificado um procedimento para criação da interface paramétrica da figura 4.4.

4.2.5 TclDot

Para a construção do sistema gerador de grafos foi utilizado o pacote Tcldot [41]. Este pacote gera automaticamente o "layout" de grafos a partir da definição de nós e arcos.

Inicialmente foi feito um estudo do aplicativo através dos scripts de demonstração. Foram implementados pequenos procedimentos para transformar estruturas de diretórios em grafos automaticamente. Com esta abordagem pode-se visualizar a estrutura de diretórios de uma perspectiva diferente, uma vez que links simbólicos, por exemplo, podem ser adequadamente representados.

A integração deste módulo com o "parser" verilog permitiu a geração de grafos de dependências. A implementação de rotinas para monitoramento do sistema de arquivos tornou possível a visualização dinâmica da base de dados de diversos projetos.

```
proc CreateTabs { tab_list  frame_path } {

puts "Creating tabs: $tab_list"

set main_frame       [ lindex [ split $frame_path "." ] 1 ]
set rnotebook_frame  $frame_path

frame $main_frame -borderwidth 2 -relief raised
pack  $main_frame -side top -fill both -expand 1

puts "Creating Tab: $rnotebook_frame"

Rnotebook:create $rnotebook_frame -tabs $tab_list -borderwidth 2
pack  $rnotebook_frame -fill both -expand 1 -padx 1 -pady 1 -side left

}
```

Fig. 4.5: Código utilizado para parametrização do “RNotebook”.

4.3 Técnicas de programação

4.3.1 Meta programação

Meta-programação é utilizada quando são desenvolvidos programas cujo objetivo é gerar código. Trata-se de uma técnica de programação amplamente conhecida entre os que utilizam SmallTalk [35] ou aqueles que desenvolvem sistema como compiladores.

A interface gráfica paramétrica pode ser vista como exemplo de meta-programação, pois o núcleo do sistema não possui função algorítmica específica. Os procedimentos têm como objetivo a implementação de funcionalidades que auxiliam no desenvolvimento do próprio sistema, como menus e botões. O principal propósito da utilização desta técnica é aprimorar a interação do usuário com o sistema computacional visando o desenvolvimento do próprio ambiente computacional, independentemente de uma aplicação específica.

De modo prático, este tipo de abordagem foi utilizado extensivamente para a criação de elementos gráficos como listas e botões, visando o acesso a procedimentos que permitam a criação de novas interfaces. Foi também empregada para o desenvolvimento de parametrização dinâmica e geração de arquivos de configuração codificados em TCL. A automação da geração de estruturas de dados para utilização dinâmica pela interface gráfica na construção de elementos gráficos ou execução de procedimentos, pode também ser vista como técnica de meta programação.

A técnica utilizada neste trabalho contou com a implementação de procedimentos através da introdução direta no código fonte de diretrizes para a criação de estruturas de código. Foram utilizados também arquivos prontos, realizando-se substituições de “strings” para gerar código final executável pela máquina virtual TCL. Estes arquivos são denominados “templates” ou modelos.

4.3.2 Eventos

Eventos são responsáveis por mudanças no ambiente de projeto. Qualquer mudança na especificação do projeto, por exemplo, deve resultar em uma série de eventos visando o realinhamento a esta nova realidade. Em princípio, qualquer mudança no estado do sistema pode ser considerada um evento. A implementação de monitores em software torna isto viável. O módulo de bootstrap, por exemplo, utiliza como evento para a tomada de uma ação a ausência do processo referente ao núcleo

da interface gráfica paramétrica.

A programação orientada a eventos [7], do ponto de vista de monitoramento de arquivos, não é suportada pelos comandos do corpo básico da linguagem. O comando "fileevent", gera um evento quando um determinado arquivo está pronto para ser escrito ou lido, mas não quando foi modificado no sistema de arquivos, por exemplo. Para monitoramento de arquivos, parte-se inicialmente de um temporizador e através de comandos específicos para interação com o sistema de arquivos, compara-se atributos antigos com novos.

4.3.3 Reflexão

Reflexão [7] é uma técnica de programação utilizada quando informações sobre o próprio programa, como variáveis por exemplo, são utilizadas no processamento. Tendo-se em vista o auto-desenvolvimento do sistema, procedimentos para gerenciamento do próprio código fonte foram implementados. Botões para compactação e envio de arquivos via correio eletrônico, possibilitaram a implementação de sistemas autônomos de "backup". A utilização de diversas interfaces gráficas gerenciadas através de sistemas de bootstrap, torna necessário que o sistema realize verificações através de estruturas de dados para auto-representação.

4.3.4 Autonomic computing

A Computação Autônoma tem sido o termo utilizado pela indústria para designar produtos que tenham capacidade de autogerenciamento. Dentro desta linha, um ramo bastante pragmático de desenvolvimeto vem sendo explorado sob diferentes denominações. "Autonomic Computing" [42] foi inicialmente proposto pela IBM, "Adaptive Enterprise" tem sido utilizado pela HP para diferenciar seus produtos e finalmente "Dynamic Systems" como alternativa pela Microsoft. A área de tecnologia da informação tem sido o principal foco de aplicação em que estas empresas vêm direcionando recursos.

Não foram encontradas, durante a pesquisa bibliográfica, publicações relacionadas com o emprego de abordagens deste tipo para automação do fluxo de projeto de CIs. Do ponto de visa de implementação, sistemas autônomos têm sido amplamente estudados dentro do contexto dos sistemas inteligentes, mais especificamente no que se refere à teoria de agentes computacionais. A automação de etapas específicas do fluxo de projeto, por outro lado, têm empregado tais técnicas. Das iniciativas profissionais e acadêmicas pertencentes ao universo da computação autônoma, não foram identificadas, iniciativas de utilização de suas técnicas para integração do fluxo de projeto como um todo.

4.3.5 Cliente/Servidor

No ambiente de projeto a existência de imensas bases de dados desenvolvidas ao longo dos anos é comum. Um comando simples para listar a versão dos objetos na base de dados pode levar muito tempo além de consumir recursos desnecessários, quando executado por diversas pessoas. A mesma situação ocorre quando se deseja compartilhar resultados de simulação. Pode-se adotar a estratégia de compartilhar apenas os dados produzidos pelo simulador, ao invés de se rodar a simulação novamente, consumindo recursos e tempo.

Em TCL é muito simples implementar sistemas com características de comunicação cliente/servidor. Neste sentido a implementação deste tipo de arquitetura permite a troca de dados de modo eficiente através da centralização de funcionalidades em uma interface gráfica paramétrica e compartilhando os resultados com outras. Para isto foram implementados módulos simples que funcionam, dependendo do contexto, como cliente ou servidor.

Como exemplo podemos citar uma situação recorrente onde se necessita obter da base de dados do projeto a informação referente à lista de arquivos "travados". Através de métodos convencionais, todos que precisem da informação irão executar determinada ferramenta, tornando a operação redundante e desperdiçando recursos. Nesta situação, a lista poderia ser obtida por um indivíduo e disponibilizada por outros.

Nas figuras 4.6 e 4.7, estão os mecanismos básicos para a implementação de propriedades de comunicação remota entre interfaces paramétricas. Fica evidente o modelo de programação cliente/servidor, estruturado em requisições e respostas.

```
proc Server {port} {
   set s [socket -server AcceptConnection $port]
   vwait forever
}

proc AcceptConnection {sock addr port} {
   puts "Accept $sock from $addr port $port"
   set client_info(addr,$sock) [list $addr $port]
   fconfigure $sock -buffering line
   fileevent $sock readable [list ProcessRequest $sock]
}

proc ProcessRequest {sock} {

   if {[eof $sock] || [catch {gets $sock client_request}]} {

   close $sock
   unset client_info(addr,$sock)

   } else {

   # puts $client_request
   ExecClientRequest

   }
}
```

Fig. 4.6: Código que implementa funcionalidades de servidor.

4.3.6 Comandos canônicos

Comandos Canônicos podem ser vistos como os comandos existentes em uma Shell Unix/Linux. Qualquer programa, neste universo, pode ser resumido a uma entrada com parâmetros, uma execução ou processamento, uma eventual mudança de estado do sistema e a produção de algum tipo de saída, que pode ser na forma de um relatório ou de uma mudança do sistema.

Comandos canônicos apresentam a propriedade de não serem divisíveis em outros comandos. Seu processo pode ser reduzido a três diferentes etapas distintas, determinação de parâmetros de

```
proc SendRequest { host port request } {
   set socket_id [socket $host $port]
   fconfigure $socket_id -buffering line
   puts $socket_id "request"
   gets $socket_id line
   set $::answer $socket_id
}
```

Fig. 4.7: Código que implementa funcionalidades de cliente.

entrada, execução ou processamento e parâmetros ou resultados de saída, que podem ser também a modificação do estado do sistema, como por exemplo, no processo de criação de um diretório do sistema de arquivos.

Um processo complexo pode ser encarado como um passo canônico na medida em que procedimentos são desenvolvidos reduzindo o mesmo aos seus elementos de entrada, seu comando de disparo ou execução e os elementos de saída. Todo processo, por mais complexo que seja pode ser sintetizado a estes três passos, entrada de parâmetros, processamento, análise de resultados. Um exemplo simples seria a simulação de um circuito. Esta simulação pode ser traduzida através dos seguintes passos canônicos: entrada de circuito, simulação e resultados. A entrada de parâmetro pode ser a topologia de um circuito eletrônico através de seu "netlist". O processamento compreende a etapa de simulação e os resultados a exibição dos sinais elétricos do circuito através de formas de ondas.

4.4 Módulos

A seguir estão detalhados os critérios de implementação dos principais módulos. Serão apresentados fragmentos do código visando ilustrar a implementação, tornando acessíveis os diferentes elementos de programação empregados.

4.4.1 Bootstrap

Na figura 4.8 temos um trecho do código que demonstra a implementação em alto nível de um monitor para a interface gráfica paramétrica implementado no módulo de "bootstrap". Nesta figura, é observada a utilização de procedimentos de baixo nível como o "MonitorProcess", responsável pelo efetivo monitoramento dos processos da máquina.

```
if { [ MonitorProcess ParametricGUI ] == 0 } then {
thread::send  -async [ lindex  $::thread_id_list 0 ] [ list exec ParametricGUI.tcl & ]
}
```

Fig. 4.8: Código que implementa o monitor da interface gráfica paramétrica.

Um procedimento semelhante ao MonitorProcess está na figura 4.9. Trata-se de um detector de processos genéricos, que utiliza como comando canônico "ps", que gera um relatório instantâneo dos

processos.

```
proc DetectGenericProcess { Executable_name } {

set result [ exec ps -af  ]
set exists 0

foreach line $result {

   #puts $line
   if { [ file tail $line ] == "$Executable_name" } then { set exists  1 }

}

#puts $result
#puts $exists

return $exists

}
```

Fig. 4.9: Detector de processos genéricos.

4.4.2 Construtores de interface

Interativos

Inicialmente serão feitas algumas considerações acerca dos mecanismos utilizados para a construção de interfaces. Assim, justificam-se os critérios adotados na elaboração do trabalho. Existem basicamente duas formas básicas de se escrever programas que tenham como resultado de sua execução a criação de uma determinada interface gráfica para acesso a funções específicas.

O primeiro modo, mais elementar, consiste na entrada dos comandos que descrevem o comportamento da interface de modo seqüencial. Como será visto, é mais adequado para se construir a interface gráfica paramétrica. O outro depende da utilização de outra ferramenta, normalmente denominada, construtor de interface gráfica com o usuário ou em inglês "gui builder". Este tipo de aplicativo tem como principal benefício o desenvolvimento rápido de Interfaces Gráficas complexas, mas que não tendem a mudar. O processo interativo, justamente aquilo que queremos automatizar no ambiente de projeto de CIs, compõe seu método de utilização básico. O custo desta facilidade é um código que além de não ser otimizado, é difícil de ser editado sem o aplicativo. Na figura 4.10 pode-se visualizar as diferentes janelas utilizadas por este aplicativo na construção de interfaces.

Outro aspecto interessante encontrado em aplicativos construtores de interface é que os mesmos não abordam a parametrização da interface gráfica que está sendo desenvolvida de modo direto. As mesmas dificuldades encontradas no uso das ferramentas de projeto de CIs, devido a distribuição funcional no ambiente computacional, ocorrem aqui também. Seria possível desenvolver meta elementos gráficos, resultando na implementação de procedimentos dedicados, não previstos no ambiente de programação disponibilizado. Uma diferença importante entre estas duas abordagens e a interface gráfica paramétrica está no método utilizado para implementação. Enquanto construtores visam facilitar a implementação de interfaces através da interação com o usuário, a interface paramé-

trica tem sua construção codificada em parâmetros, facilitando principalmente sua extensão de modo autônomo.

Apesar de existirem vários construtores de interface como o VisualTCL [40], sua utilização para os propósitos desta pesquisa não seria adequada. Tendo-se em mente a metodologia proposta, as funcionalidades descritas inviabilizam a utilização de construtores de interface interativos neste trabalho. Os objetivos assinalados pelos sistemas, do ponto de vista da interação com o desenvolvedor, são opostos. Os critérios primordiais para a interface gráfica paramétrica são a simplicidade da codificação, parametrização e automação dos processos, inclusive do desenvolvimento da própria interface. Além disso, a automação do fluxo de projeto deve resultar em interfaces com elementos gráficos extremamente simples, para acesso imediato aos procedimentos, reduzindo a complexidade do processo de codificação.

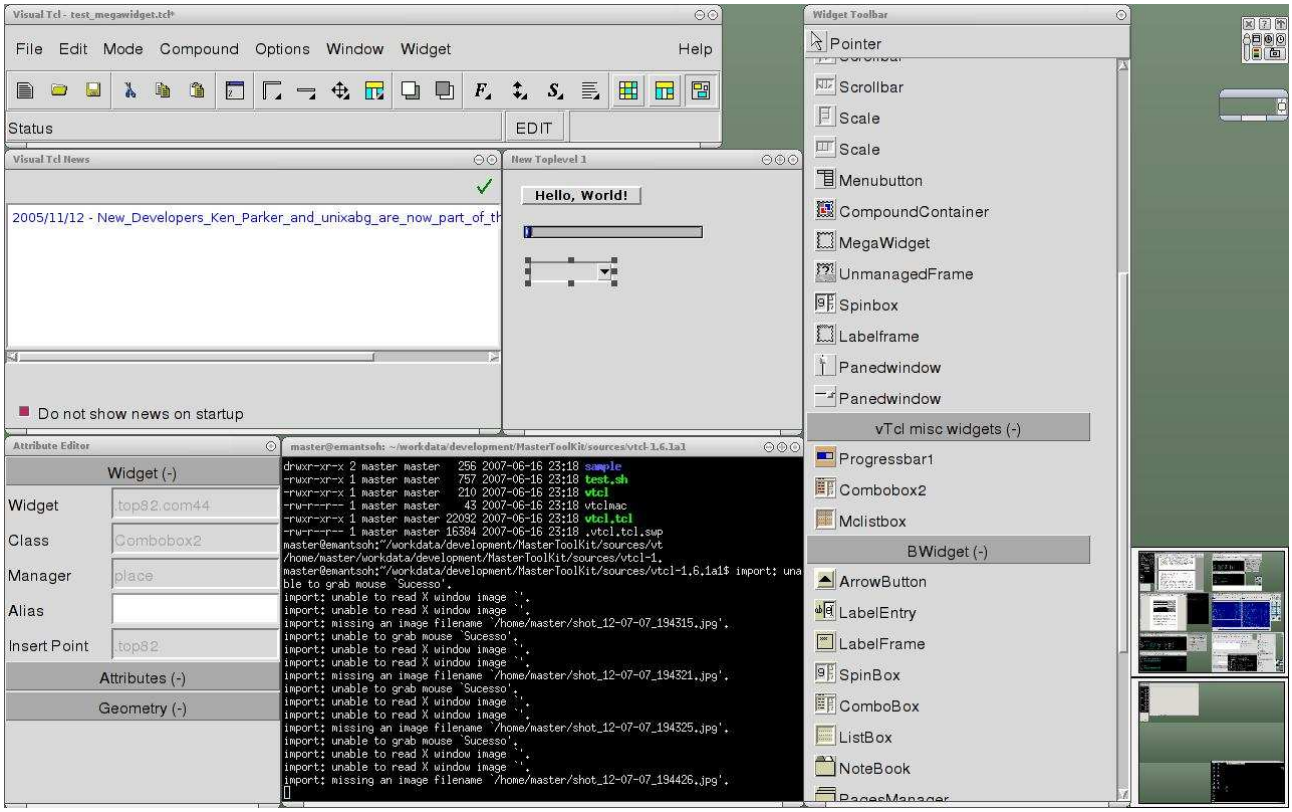


Fig. 4.10: Editor de interface gráfica interativo VisualTcl.

Paramétricos

A criação de uma interface gráfica de modo linear é realizada através de um conjunto serial de declarações, cujos parâmetros como funções, textos, cores, posicionamento, dentre outros elementos, estão codificados de forma fixa no programa. Fica evidente que para a extensão da interface seria necessário editar diretamente o código fonte, mesmo que poucas diferenças fossem solicitadas. O desenvolvimento de uma interface gráfica paramétrica requer a distinção entre o código utilizado para efetivamente gerar a interface e seus parâmetros. O desenvolvimento de rotinas especializadas para a

criação da interface gráfica que tome como parâmetros estruturas de dados desenvolvidas na linguagem TCL, como vetores, matrizes e grafos, é o conceito primordial utilizado de modo recorrente para a implementação de interfaces gráficas para variadas aplicações.

Na figura 4.11 temos exemplificada uma estrutura bastante simples. Trata-se de um vetor com os parâmetros para a construção de uma interface gráfica simples com botões. Este vetor é definido quando o sistema está iniciando e fica fisicamente localizado no arquivo de configuração.

```
set ::command_01 "EditFile ooffice Cap1/cap1.doc"
set ::command_02 "EditFile ooffice Cap2/cap2.doc"
set ::command_03 "EditFile ooffice Cap3/cap3.doc"
set ::command_04 "EditFile ooffice Cap4/cap4.doc"
set ::command_05 "EditFile ooffice Cap5/cap5.doc"
set ::command_06 "EditFile ooffice tese.bib.doc"

array unset ::button_array
array set ::button_array {

Reload               { $::top_frame.a1 reload         bottom   both   1 1  grid }
Quit                 { $::top_frame.a2 exit           bottom   both   2 1  grid }
CreatePDF            { $::top_frame.a3 CreatePDF      bottom   both   3 1  grid }
Edit_Cap1            { $::top_frame.01 $::command_01  bottom   both   4 1  grid }
Edit_Cap2            { $::top_frame.02 $::command_02  bottom   both   5 1  grid }
Edit_Cap3            { $::top_frame.03 $::command_03  bottom   both   6 1  grid }
Edit_Cap4            { $::top_frame.04 $::command_04  bottom   both   7 1  grid }
Edit_Cap5            { $::top_frame.05 $::command_05  bottom   both   8 1  grid }
Edit_Bibliografia    { $::top_frame.06 $::command_06  bottom   both   9 1  grid }
TeseDirSize          { $::top_frame.08 TeseDirSize    bottom   both   1 2  grid }
DOC2TEX              { $::top_frame.a8 DOC2TEX        bottom   both   2 2  grid }
RevisÃ£o_Cap1_26Jun  { $::top_frame.a9 diff_rev_A     bottom   both   3 2  grid }
Backup               { $::top_frame.09 CreateTeseTgz  bottom   both   4 2  grid }
Armagetron           { $::top_frame.07 Armagetron     bottom   both   5 2  grid }

}
```

Fig. 4.11: Parâmetros para criação do vetor de botões.

Um possível procedimento que toma como entrada o vetor definido na figura 4.11, está ilustrado na figura 4.12. Neste procedimento podemos identificar o tratamento para todos os parâmetros definidos no vetor. É interessante notar que a definição dos parâmetros segue de modo estreito os recursos disponibilizados pela linguagem. Na medida em que níveis de complexidade são adicionados, novos parâmetros, arbitrários, podem ser necessários, tornando difícil a construção automática de procedimentos como este.

O resultado da execução do procedimento pode ser visualizado na figura 4.13. Nesta figura pode ser observado o quão fácil fica a extensão da interface, uma vez que o procedimento para sua construção tenha sido definido. Para isto deve ser adicionada uma linha a mais no vetor de botões e criar o procedimento que será executado quando o botão for pressionado.

Com a arquitetura proposta, ilustrada na figura 3.3, encontramos um ponto que permite até mesmo a automação completa da construção da interface. Para isto basta desenvolver procedimentos especiais cuja saída são os parâmetros utilizados para a construção da interface. Além disso, este tipo de software não restringe a utilização de técnicas mais sofisticadas contando atualmente com funcionalidades que permitam usar diversas extensões, dentre elas aquelas que suportam programação orientada a objetos.

```
proc CreateButtonArray {  } {

Banner "Creating Button List "

array statistics  ::button_array

foreach button_name [ array name ::button_array ] {

   set button_frame        [ lindex $::button_array($button_name) 0 ]
   set button_command      [ lindex $::button_array($button_name) 1 ]
   set button_orientation  [ lindex $::button_array($button_name) 2 ]
   set button_fill         [ lindex $::button_array($button_name) 3 ]
   set button_x            [ lindex $::button_array($button_name) 4 ]
   set button_y            [ lindex $::button_array($button_name) 5 ]
   set button_wm           [ lindex $::button_array($button_name) 6 ]

eval  button   $button_frame -text $button_name       -command $button_command

   if { $button_wm == "grid" } then {

   eval     grid $button_frame -row $button_x -column $button_y -sticky ew

   } elseif { $button_wm == "pack" } then {

   eval     pack     $main_frame -side $button_orientation  -expand  1  -fill $button_fill

   }

}

}
```

Fig. 4.12: Procedimentos para criação de vetores de botões.

4.4.3 Arquivos de configuração

Arquivos de configuração são utilizados para armazenar os parâmetros relativos ao sistema, procedimentos ou extensões. O conjunto de parâmetros encapsulados em uma entidade única, pode ser gerado de modo manual ou automático. Alinhado a metodologia definida, inicialmente os arquivos de configuração foram escritos em TCL, ficando assim restritos, ao menos em primeira instância, ao domínio da linguagem TCL.

Arquivos TCL utilizados como parâmetros podem ser vulneráveis na medida em que podem conter código malicioso, mas neste caso seria necessária a distinção dos agentes responsáveis pelo código em si e pela edição dos arquivos de configuração. No sistema proposto, em princípio, a mesma pessoa que edita o arquivo de configuração deve estar apto a transformar os procedimentos, de modo que esta preocupação não é importante de imediato.

Elementos de configuração transcendem a simples entrada de parâmetros. Eles dizem respeito a uma espécie de elaboração paramétrica dinâmica, uma vez que é possível incluir código para a criação de estruturas de dados em tempo de execução.

Com a adição de procedimentos que realizam a tradução de arquivos em outras linguagens, podem ser adotados formatos mais amigáveis para a definição de arquivos de configuração como tabelas. Arquivos de interface podem ser utilizados para criar uma camada de abstração e permitir que os parâmetros sejam escritos com outra sintaxe. Neste caso são necessários procedimentos para conversão do arquivo paramétrico de interface nas estruturas TCL correspondentes.

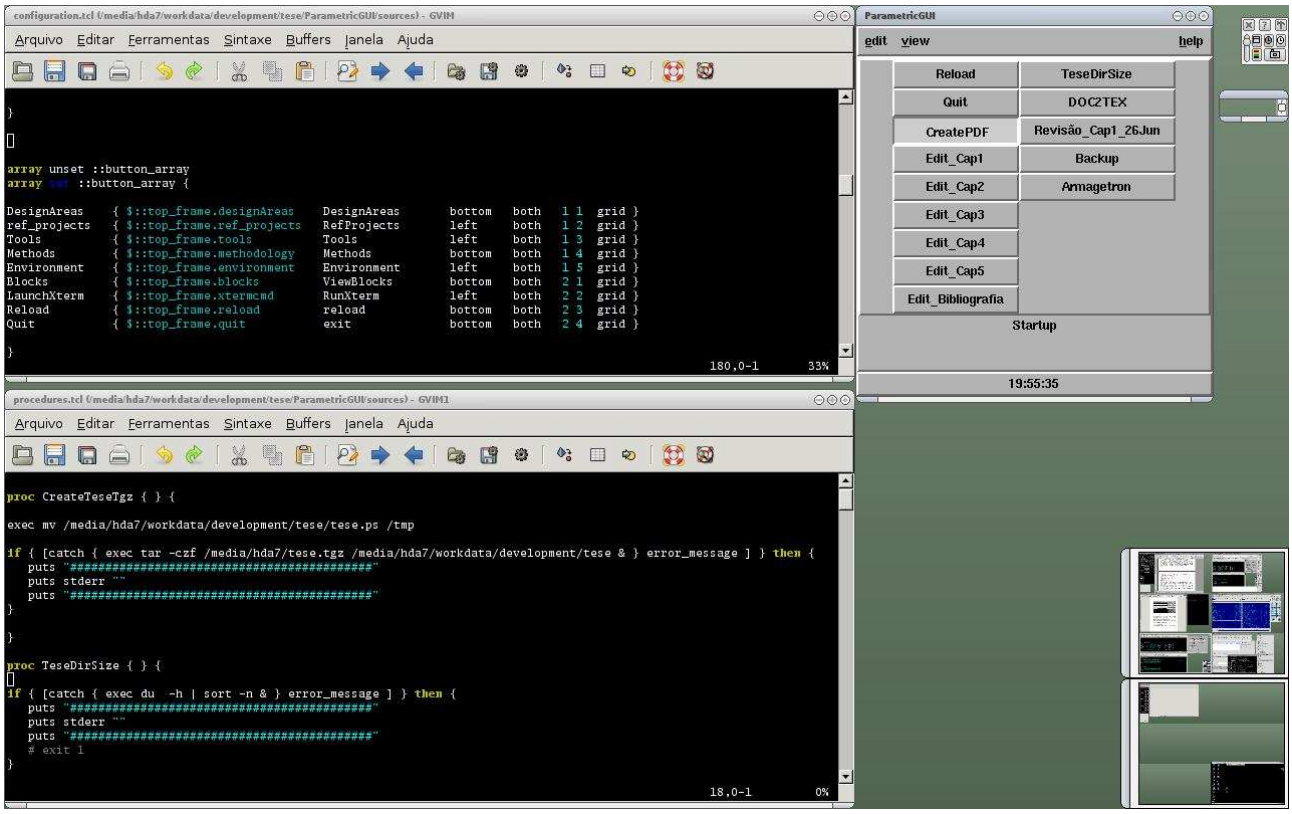


Fig. 4.13: Sistema resultante da implementação de rotinas para parametrização de interfaces.

Temporizador

Os temporizadores foram implementados utilizando-se uma função recursiva. O período do temporizador é determinado pela instrução “after”. Ao ser iniciado, o temporizador agenda sua própria execução de modo recursivo através do comando “after”. É também passado como parâmetro um “scrip” em Tcl, que é executado no intervalo de tempo determinado. O “script” em TCL pode ser uma função complexa, como por exemplo verificar se determinada base de dados está consistente do ponto de vista da correspondência entre “layout” e esquemático.

Em caso de inconsistência, quando for detectado que um esquemático possui data mais recente que um “layout”, a ferramenta pode executar programas de verificação física para identificar se isto representa uma modificação que requer alterações no “layout” para torná-lo novamente consistente. Níveis de modificação podem ser definidos, permitindo que o software utilize, por exemplo, ferramentas de roteamento automático para modificar uma topologia cuja implementação inicial, por falhas sistêmicas de implementação, processo ou metodologia, não atenda a especificação. O sistema pode ainda enviar uma mensagem de correio eletrônico ou gerar um relatório informando os principais problemas encontrados e as decisões tomadas.

4.4.4 Gerador de grafos

Ferramentas como o DIA [36] são extremamente úteis para a criação manual de diagramas. Quando se quer transformar, de forma rápida e automática, estruturas de dados em diagramas, esta ferramenta deixa de ser atrativa, uma vez que é necessária extensiva interação com o programa para posicionamento e ligação de objetos no diagrama. Com o intuito de gerar grafos e diagramas cuja aparência se assemelhe aos diagramas feitos manualmente, existem ferramentas como o Graphviz.

O Graphviz está disponível como uma extensão da linguagem TCL denominada TclDot. Embora disponha de uma API bastante flexível, sua integração à interface gráfica paramétrica passou por cuidadoso estudo de sua sintaxe e elaboração de procedimentos para sua utilização de forma parametrizada.

4.4.5 Controlador X-window

Qualquer sistema computacional pode ser implementado a partir de elementos que podem oferecer ou não funcionalidades, cuja ausência implica na necessidade de implementação. Aliado aos critérios estabelecidos anteriormente, para a implementação do controlador de janelas foi necessário utilizar o gerenciador de janelas Enlightenment.

O gerenciador de janelas Enlightenment possui recursos fáceis de serem utilizados para controlar remotamente os recursos do servidor de janelas. Assim torna-se possível a automação de recursos gráficos como o posicionamento, dimensão dentre outros atributos das janelas.

Após testar de diversos gerenciadores de janelas, foi possível concluir que o Enlightenment, além de possuir todas as funcionalidades dos outros gerenciadores e ser leve, ainda conta com o aplicativo "eesh", um terminal para o controle interativo das janelas que funciona como um IPC (Inter Process Communication Protocol). Outros gerenciadores podem ser usados, mas neste caso, seria necessária a instalação de programas adicionais, dedicados para esta tarefa. Na figura 4.14 temos um exemplo de codificação para obtenção da condição das janelas a partir do aplicativo eesh.

O Xnee [43] foi outro programa fundamental para completar a automação dos recursos do "X-Window". Este software permite que se grave eventos gerados a partir dos elementos de entrada, como mouse e teclado, e permite sua posterior reprodução. A figura 4.15 mostra a arquitetura implementada pelo Xnee.

4.4.6 Parser verilog

Tendo-se como principio a avaliação de código, este módulo foi implementado utilizando-se bibliotecas prontas, mais especificamente aquelas encontradas na página da internet Veripool [44]. A integração de código desenvolvido em ambientes mais refinados como o Meta Environment [45] permitiriam maior avanço nesta parte do sistema. A utilização de bibliotecas comuns como o Graphviz [41] torna esta atividade mais fácil, além de servir de exemplo para o módulo responsável pela geração de grafos.

```
set window_list [ exec eesh -ewait window_list extended ]
set window_list_splited [ split $window_list "\n" ]

#puts $window_list
#puts $window_list_splited

set verbose off

foreach window $window_list_splited {

        set window_id                   [ lindex [ split $window ":" ] 0 ]
        set window_name                 [ lindex [ split $window ":" ] 1 ]
        set window_cord_aspect          [ lindex [ split $window ":" ] 5 ]
        set window_cord_x               [ lindex [ split $window_cord_aspect " " ] 1 ]
        set window_cord_y               [ lindex [ split $window_cord_aspect " " ] 2 ]
        set window_atributes [ split $window " : "                      ]
        set window_id                   [ lindex $window_atributes 0    ]
if { $verbose == "on" } {
        puts $window
        puts $window_atributes
        puts $window_id
        puts $window_name
        puts $window_cord_x
        puts $window_cord_y
        puts $window_cord_aspect
        puts " $window_name $window_cord_x $window_cord_y "

        puts $logfileid $window
        puts $logfileid $window_atributes
        puts $logfileid $window_id
        puts $logfileid $window_name
        puts $logfileid $window_cord_x
        puts $logfileid $window_cord_y
        puts $logfileid " $window_name $window_cord_x $window_cord_y "
}

}
```

Fig. 4.14: Procedimentos para automação do interpretador eesh .

4.5 Aplicações

O campo de aplicação do sistema proposto é virtualmente ilimitado. Tendo em mente a ilustração dos conceitos propostos, foram desenvolvidas diversas aplicações relativas ao desenvolvimento do próprio sistema, manutenção da infra-estrutura computacional e automação do fluxo de projeto relativo a simulação “mixed-signal”.

4.5.1 Meta aplicação

A aplicação inicial do sistema pode ser categorizada como uma meta-aplicação, uma vez que a mesma se concentra na concepção de funcionalidades com o objetivo de facilitar o próprio desenvolvimento do sistema. Foram criados menus especiais para permitir, por exemplo, a implementação de menus para rápido acesso ao código fonte do sistema, rotinas para “backup” e verificação do espaço em disco utilizado.

4.5.2 Simulação mixed-signal

As figuras 4.16 e 4.17 ilustram a utilização do simulador “AMSDesigner” [4]. A primeira possui exemplificado o fluxo de projeto integrado ao “framework” proprietário “DFII”. A segunda, a auto-

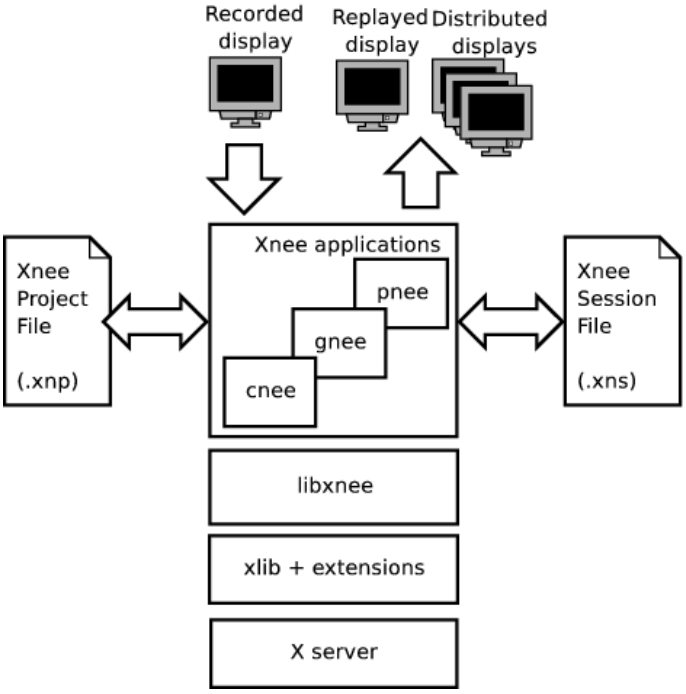


Fig. 4.15: Arquitetura do software xnee.

mação do fluxo com a interface gráfica paramétrica, evidenciando o nível de automação que pode ser alcançado ao se implementar e aplicar a arquitetura proposta.

No primeiro caso fica evidente a necessidade de se percorrer as diversas interfaces gráficas, mesmo no caso em que a simulação tenha seus parâmetros definidos anteriormente. A entrada de informação referente às diversas variáveis, usadas para definir os critérios de simulação, está distribuída pelas diversas interfaces gráficas.

Na figura 4.17 estão dispostas a interface gráfica paramétrica e o aplicativo “AMSDesigner” [4]. Os botões foram implementados de modo que, a partir dos parâmetros relativos à versão do topo do “chip”, o nome do módulo de mais alto nível e os elementos de interconexão, sejam executados diversas etapas para preparação da infra-estrutura e base de dados visando uma simulação completa.

Pressionando-se os botões na seqüência, são realizadas as seguintes operações: limpeza do diretório temporário utilizado para simulação; “download” da base dados correspondente à versão informada; geração de “netlist” dos blocos analógicos; geração do “netlist” da tecnologia; criação de arquivos do tipo “include”, com todos os blocos da biblioteca de tecnologia e do projeto; compilação; elaboração e finalmente a simulação, que pode ser em segundo plano ou simulação interativa.

Grande parte da dificuldade de automação do fluxo de projeto está na determinação de inúmeros parâmetros distribuídos ao longo da uma extensiva cadeia de interações entre o projetista e o ambiente computacional. Uma vez determinados e encapsulados na interface gráfica paramétrica, ficam acessíveis a outros projetistas. Além de permitir a qualquer um o acesso a um ambiente de simulação complexo, foi possível economizar com a utilização de licenças, uma vez que o número de ferramentas utilizadas foi menor.

Durante o desenvolvimento desta aplicação foram criadas diversas interfaces. Nos casos extremos, foi possível concentrar conjuntos de etapas do fluxo em apenas um botão. Este procedimento

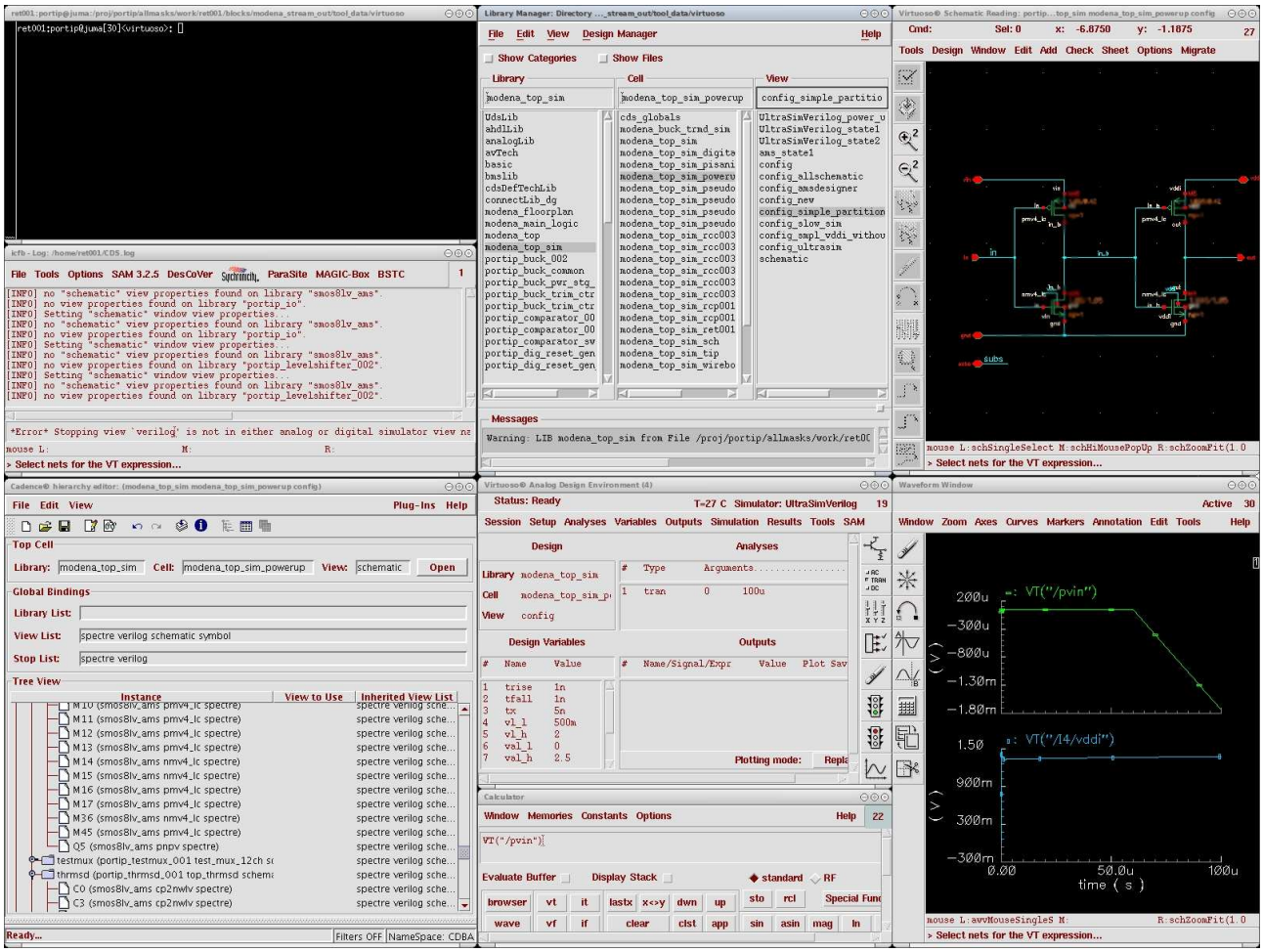


Fig. 4.16: Simulação de topo convencional.

tornou possível a implementação de metodologias mais complexas. A simulação de sub-blocos do sistema em desenvolvimento, mantendo-se consistente o controle de versão dos arquivos, ferramentas e tecnologias utilizadas ao serem automatizados oferecem a possibilidade de criação de fluxos de projeto mais elaborados. O gerenciamento automático e a parametrização de variáveis de simulação permite a criação sistemática de cenários de simulação a partir de um modelo.

4.5.3 Automação de documentação

Esta dissertação foi elaborada utilizando-se o sistema GNU/Linux, distribuição “Ubuntu 7.04”. O pdf foi gerado através do software “Latex”. Os arquivos com extensão tex e bib foram elaborados essencialmente utilizando-se o gvim para edição direta e o Microsoft “Word” e o “Openoffice” para geração de arquivos doc, posteriormente convertidos automaticamente através da interface gráfica paramétrica. A sincronização dos arquivos doc e dos arquivos tex foi feita inicialmente de modo manual e posteriormente de modo automático com o próprio sistema descrito no trabalho. Quando o arquivo de extensão doc muda, o sistema que monitora o arquivo chama uma rotina que executa uma

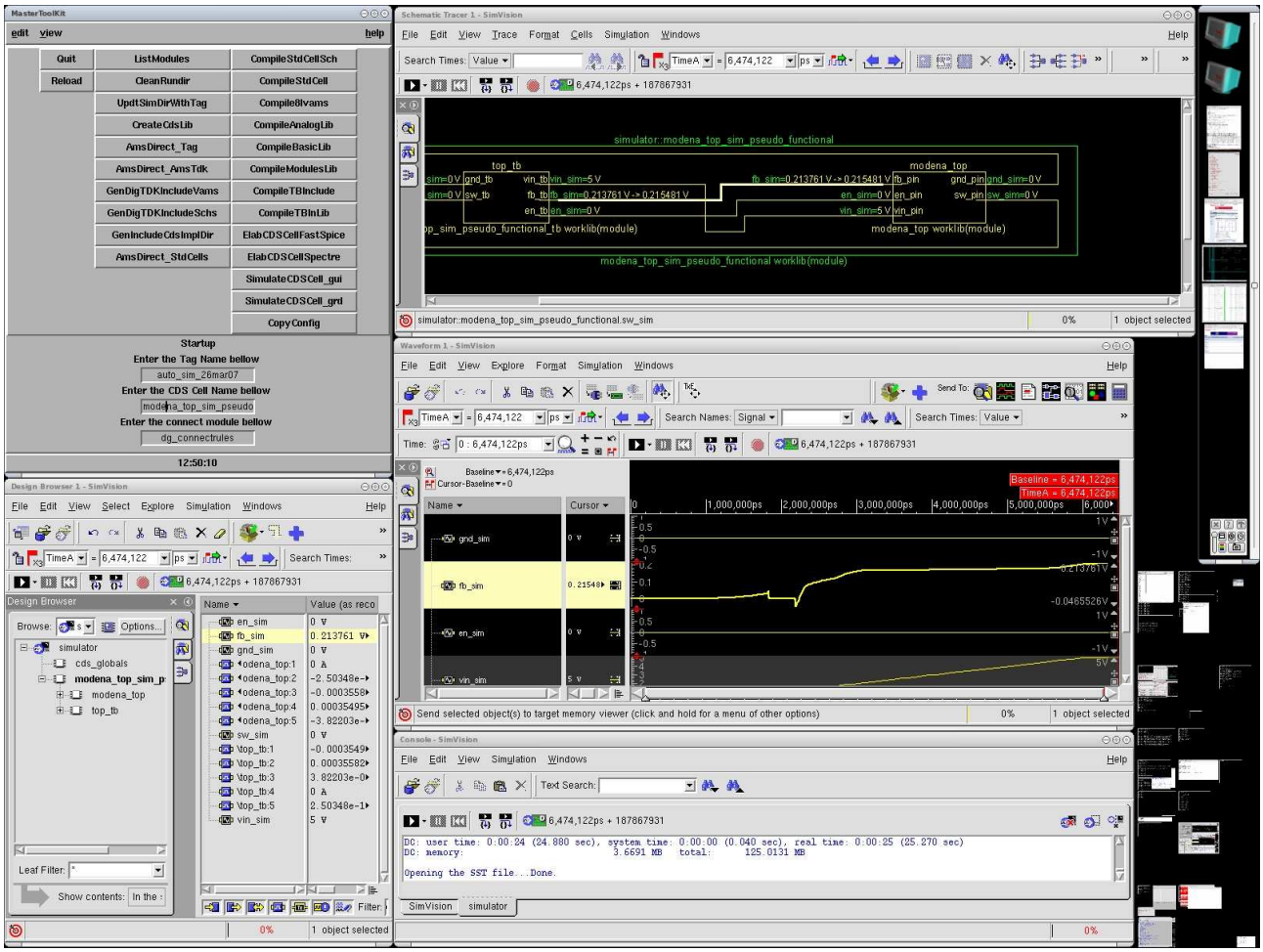


Fig. 4.17: Simulação de topo utilizando a interface gráfica paramétrica.

série de comandos com o objetivo de restabelecer e garantir a sincronização dos arquivos e gerar o pdf final.

Para manter compatibilidade com outras plataformas e também continuar a ter benefício de ferramentas como o corretor ortográfico, controle de revisão e visualização da estrutura do documento, foram criados procedimentos e parâmetros para executar diversas funções. Por exemplo, para verificar diferenças, abrir documentos, converter os arquivos .doc para .tex, utilizando o aplicativo catdoc.

Editar o arquivo com a codificação do “Latex” no Word promove certa flexibilidade e dá a segurança àqueles que não estão habituados com o ambiente “GNU/Linux”. Desta forma, procurou-se demonstrar como algum tipo de interação com os recursos computacionais poderia ser parametrizada e finalmente reutilizada, poupando tempo e tornando as tarefas mais eficientes e acessíveis.

A conexão com o ambiente de projetos de CIs foi feita na medida em que as interfaces gráficas foram integradas. Procedimentos extras foram escritos para rodar simulações de modo autônomo e gerar formas de onda no formato eps, “Encapsulated PostScript”. Monitores, ao identificar mudanças nos arquivos, respondem com a execução dos procedimentos necessários para a atualização do arquivo pdf de modo automático.

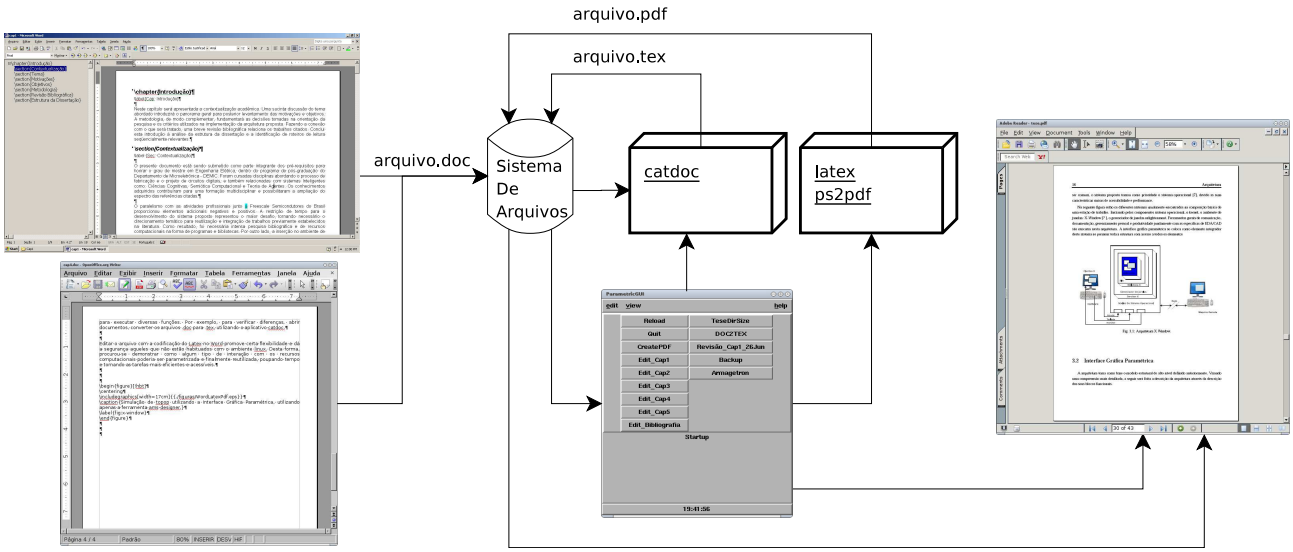


Fig. 4.18: Automação de procedimentos para documentação.

Capítulo 5

Considerações finais

Neste capítulo será realizada a síntese dos resultados, conclusões e também trabalhos futuros. Nos resultados serão identificados os elementos novos proporcionados pelo desenvolvimento da pesquisa em questão e seus efeitos. Na conclusão será feita uma correlação entre os resultados obtidos e os objetivos enumerados no início da dissertação. Pretende-se deixar evidentes os pontos onde se obteve o maior sucesso juntamente com aqueles que representaram os maiores desafios. Os trabalhos futuros proporcionam uma visão de longo prazo para a pesquisa realizada, tendo como função indicar um possível direcionamento a ser seguido para aqueles que queiram estender o trabalho. Deve servir de guia para implementações futuras e eventualmente indicar oportunidades para outros pesquisadores.

5.1 Resultados

Dentre os principais resultados obtidos, tendo-se se em vista as aplicações, podemos citar a plataforma autônoma para simulação e o sistema para conversão de documentos. Além disso, pode-se também dar destaque aos elementos gerais como, por exemplo, a concepção e elaboração de métodos sistemáticos para concatenar e automatizar as mais diversas tarefas computacionais envolvidas no desenvolvimento de CIs.

O sistema proposto apresentou características singulares ao proporcionar extrema sincronia entre sua representação gráfica e os comandos elementares do ambiente computacional. Isto proporcionou uma elevada compreensão e domínio do ambiente de projeto, impossíveis de serem alcançadas através de métodos usuais.

Foi possível experimentar maior velocidade na execução das tarefas relacionadas com verificação e implementação de sistemas integrados. A parametrização do fluxo de projeto, através da interface gráfica, permitiu maior ênfase em metodologias "top-down". A possibilidade de criação de vários cenários de simulação, através da replicação de parâmetros, tornou viável a exploração do espaço de projeto e também foi bastante relevante.

Ainda existe pouca divisão formal entre o que é o núcleo do sistema e suas extensões. Apesar de uma divisão natural entre os arquivos de procedimentos, seria interessante elaborar um sistema apenas com funcionalidades básicas. A experiência mostrou como pode ser trabalhosa a integração de diferentes ramificações independentes de desenvolvimento, com o objetivo de atender determinadas aplicações. Desta forma, pode-se ter o desenvolvimento do núcleo e das extensões de forma indepen-

dente. A mesma aplicação poderia, de forma transparente, intercambiar núcleos de versões diferentes. Outro efeito benéfico de uma modularização mais rigorosa está no gerenciamento de dependências específicas como variáveis de ambiente, caminhos absolutos e disponibilidade de ferramentas.

O ponto que deixou o sistema, de certa forma inacessível a uma parcela considerável dos projetistas que tiveram contato com a mesma, se concentrava na necessidade de amplo conhecimento do domínio de aplicação, da interface gráfica e sua arquitetura. A utilização do próprio sistema para a divulgação dos detalhes da execução das ferramentas se mostrou bastante eficiente para a compreensão funcional, mas não de desenvolvimento. Existe a necessidade de uma documentação mais extensa, que sirva de base para estruturação dos conhecimentos, facilitando sua assimilação, desenvolvimento e comunicação. Observa-se, entretanto, que os elementos descritos conservam seus atributos e sua generalidade, tornando possível a qualquer um recriar o sistema proposto em qualquer ambiente computacional. De certa forma este documento deve permitir um grande avanço neste sentido.

5.2 Conclusões

Ao se fazer uma análise crítica a respeito dos objetivos propostos e dos resultados alcançados, acredita-se que houve avanço em diferentes domínios. No âmbito acadêmico foi possível o direcionamento da atenção para literatura mais diversificada, tornando possível a fundamentação do trabalho em termos mais gerais. Na indústria, engenheiros podem reutilizar os conceitos apresentados para construir suas próprias interfaces paramétricas e assim realizar a automação de etapas específicas do fluxo de projeto em diversas áreas. Independente de uma aplicação específica, o uso do ambiente de desenvolvimento proposto pode proporcionar um nível de interação única entre o desenvolvedor e seu objeto de estudo.

Neste sentido, a contribuição deste trabalho está na elaboração de um sistema mais genérico, cuja arquitetura pode ser facilmente replicada por outros pesquisadores em outros contextos. Um trabalho acadêmico, mesmo que seja extremamente pertinente, fica restrito aos seus aspectos conceituais quando elementos específicos de sua implementação, não estão disponíveis para uso imediato. Desta forma, acredita-se que a intensiva utilização de recursos computacionais, caso comum entre programadores, administradores de sistema e projetistas de CIs, têm benefício imediato com a implementação de sistemas como o proposto.

5.3 Trabalhos futuros

Este trabalho está situado de modo singular na fronteira entre a Engenharia de Software e a elétrica. A idéia principal da pesquisa não foi elaborar um extenso programa escrito em TCL/TK, ou desenvolver arquiteturas e topologias comumente encontradas em SOCs. O foco do trabalho foi a automação da interação com o projetista e as ferramentas de projeto através da concepção de uma interface gráfica paramétrica, permitindo de modo rápido e eficiente acesso aos recursos do ambiente computacional através da linguagem TCL.

Uma vez elaborada a arquitetura básica, scripts podem ser agrupados e reagrupados em procedimentos com ênfase na automação do fluxo de projeto completo. O esforço conjunto na criação de topologias canônicas e o mapeamento do imenso conhecimento teórico em estruturas regulares,

irá certamente permitir avanços significativos na exploração de topologias avançadas com extrema eficiência.

Segundo seu autor, o XOTclIDE [27] é um sistema interativo que serve de interface gráfica com o programador e o XOTcl e não deveria ser visto como um gerenciador de "scritps ou procedimentos e nem um editor de código TCL. . Este trabalho pode servir como uma extensão para o XOTclIDE, fornecendo os objetos básicos para a criação de interfaces gráficas paramétricas. Além disso, suas rotinas de bootstrap poderiam ser utilizadas para garantir o funcionamento do sistema mesmo quando objetos do próprio XOTclIDE forem modificados.

Uma vez que se tenha domínio do ambiente de projeto, do ponto de vista das ferramentas e dos recursos computacionais, a implementação de módulos visando obter elementos do sistema para a manutenção de arquiteturas organizacionais, permitiriam o efetivo gerenciamento de projetos e a otimização de resultados. A criação de modelos organizacionais com suporte computacional certamente agregará funcionalidades para lidar com aspectos conceituais de alto nível e suas ambigüidades e dificuldades inerentes. Indo mais além, este trabalho poderia ser estendido através da semiótica em sua vertente organizacional [46].

Referências Bibliográficas

[1] Adel S. Sedra, Kenneth C. Smith. *Microelectronic Circuits* . Oxford University Press, 2003. "http://www.oup.com/us/companion.websites/0195142519/".

[2] Bashir M. Al-Hashimi. *System-on-Chip: Next Generation Electronics*. The Institution of Electrical Engineers, 2006. "http://books.google.com/books?id=NqNvUtZcKA4C&printsec=frontcover&dq=chip".

[3] Synopsys. Synopsys Homepage. Página na internet, Synopsys, julho 2007. "http://www.synopsys.com/".

[4] Cadence . Cadence Homepage. Página na internet, Cadence, julho 2007. "http://www.cadence.com/".

[5] Mentor . Mentor Homepage. Página na internet, Mentor, julho 2007. "http://www.mentor.com/".

[6] M. Tim Jones. *Gnu/Linux Application Programming*. Charles River Media, 2005. "http://books.google.com/books?id=KjEq9Mua5TQC&pg=PR3&dq=gnu+linux&sig=87oMawQRhP5ulrh8YEBju7P-Mzs#PPR7,M1".

[7] Brent B. Welch, Jeffrey Hobbs. *Practical Programming in TCL and TK*. Prentice Hall PTR, 2003. "http://books.google.com.br/books?id=0EB9OIFBnFgC&printsec=frontcover&dq=Practical+Programming+in+TCL+and+TK.&psp=1".

[8] Vivek Sagdeo . *The Complete Verilog Book*. Springer, 1998. "http://books.google.com/books?id=-aI0VXNaPQUC&dq=verilog".

[9] Kenneth S. Kundert, Olaf Zinke . *The Designer's Guide to Verilog-AMS* . Kluwer Academic Publishers, 2004. "http://www.designers-guide.org/Books/dg-vams/cover.html".

[10] Prakash Rashinkar, Peter Paterson, Leena Singh . *System-On-A-Chip Verification: Methodology and Techniques* . Kluwer Academic Publishers, 2001. "http://books.google.com.br/books?id=MpzE-kclSMUC&printsec=frontcover&dq=System-On-A-Chip+Verification:+Methodology#PPP1,M1".

[11] Peter Van Zant. *Microchip Fabrication*. McGraw-Hill Professional, 2004. "http://books.google.com.br/books?id=hdThtshYzOEC&printsec=frontcover&dq=Microchip+Fabrication".

[12] Henry Chang . *Winning the Soc Revolution: Experiences in Real Design* . Springer, 2003. "http://books.google.com.br/books?id=uai-YU7hROsC&printsec=frontcover&dq=Winning+the+Soc+Revolution:+Experiences+in+Real+Design".

[13] Dirk Jansen . *The Electronic Design Automation Handbook*. Springer, 2003. "http://books.google.com.br/books?id=mExW4wi2ymcC&printsec=frontcover&dq=The+Electronic+Design+Automation+Handbook".

[14] John D. Cressler. *Silicon Heterostructure Handbook: Materials, Fabrication, Devices, Circuits, And Applications Of SiGe e Si Strained-Layer Epitaxy* . CRC Press, 2006. "http://books.google.com.br/books?id=VzGiXjss-boC&printsec=frontcover&dq=Silicon+Heterostructure+Handbook".

[15] David Chinnery, Kurt William Keutzer. *Closing the Gap Between Asic & Custom: Tools and Techniques for High-Performance Asic Design*. Springer, 2002. "http://books.google.com.br/books?id=WrMD-b9ek1UC&printsec=frontcover&dq=Closing+the+Gap+Between+Asic+%26+Custom:+Tools+and+Techniques+for+High-Performance+Asic".

[16] Dritte Ausgabe. *A Guide to the Project Management Body of Knowledge*. Project Management Institute, 2005. "http://books.google.com/books?id=zsJhAAAACAAJ&dq=pmbok?

[17] Mark D. Birnbaum. *Essential Electronic Design Automation*. Prentice Hall, 2004. "http://books.google.com/books?id=NYnphhDUN5cC&printsec=frontcover&dq=Essential+Electronic+Design+Automation&hl=pt-BR".

[18] Alberto Sangiovanni-Vincentelli. The tides of eda. *IEEE Design and Test of Computers*, 20(6):59–75, 2003. "http://doi.ieeecomputersociety.org/10.1109/MDT.2003.1246165".

[19] *14th International Conference on VLSI Design (VLSI Design 2001), 3-7 January 2001, Bangalore, India*. IEEE Computer Society, 2001.

[20] SI2-ORG. SI2 Home Page. Página na internet, SI2-ORG, julho 2007. "http://www.si2.org/".

[21] Silicon Navigator Inc. . Silicon Navigator Homepage. Página na internet, Silicon Navigator, julho 2007. "http://www.sinavigator.com/".

[22] Mike Murray, Yatin Trivedi, Uwe B. Meding, Bill McCaffrey, Bill Berg, and Ted Vucurevich. Issues and answers in cad tool interoperability. *dac*, pages 509–514, 1996.

[23] Leandro Soares Indrusiak. A Review on the Framework Technology Supporting Collaborative Design of Integrated Systems. Tese de doutorado, Instituto De Informática, Universidade Federal Do Rio Grande Do Sul, Junho 2002.

[24] Jacob Skolnik . A WAN Based Design Anywhere Environment for Application and Project Data. In *2000 International Cadence Usergroup, ICU 2000 SA SIG Abstracts*, 2000.

[25] Jules P. Bergmann and Mark A. Horowitz . Vex - a CAD toolbox. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 523 – 528, New Orleans, LA, Junho 1999.

[26] Toth, M.S.;Booth, R.V. . A Designer-Customizable Design Environment for Analog/Mixed-Signal Circuit Design . In *O'Reilly Open-Source Convention*, pages 23–27, San Diego CA, July 2001.

[27] Artur Trzewik. XOTclIDE. In *4. European Tcl Workshop 2003 - Nürnberg*, 2003. "http://www.xdobry.de/xotclIDE".

[28] Vineet Sahula and C. P. Ravikumar. The hierarchical concurrent flow graph approach for modeling and analysis of design processes. In *VLSI Design* [19], pages 91–96.

[29] T. B. Tarim. Mixed signal and SoC design flow requirements. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 5974–5977, Los Alamitos, CA, USA, Maio 2005. IEEE Computer Society.

[30] Runtime Design Automation. Runtime Design Automation Homepage. Página na internet, Runtime Design Automation, julho 2007. "http://www.rtda.com/".

[31] Reinaldo Silveira. Diretrizes para uma nova metodologia de projeto digital. Tese de doutorado, Escola Politécnica, Universidade de São Paulo, 2004.

[32] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 2001. "http://books.google.com/books?id=yGFNKDloXq0C&printsec=frontcover&dq=The+Cathedral+and+the+Bazaar&hl=pt-BR".

[33] Adrian Nye . *Xlib Programming Manual*. O'Reilly, 1992. "http://books.google.com/books?id=d8tByjvMmIwC&printsec=frontcover&dq=Xlib&hl=pt-BR".

[34] Enlightenment.org . Enlightenment Homepage. Página na internet, Enlightenment.org, julho 2007. "http://www.enlightenment.org/".

[35] Smalltalk.org. Smalltalk.org homepage. Página na internet, Smalltalk.org, julho 2007. "http://www.smalltalk.org/main/".

[36] Alexander Larsson. DIA GNU Project homepage. Página na internet, Lysator, The Academic Computer Society, Julho 2007. http://www.gnome.org/projects/dia/.

[37] John K. Ousterhout. Scripting: Higher-level programming for the 21st century. *Computer*, 31(3):23–30, 1998.

[38] Tcl Developer Xchange. Tcl Developer Xchange Homepage. Página na internet, Tcl Developer Xchange, julho 2007. "http://www.tcl.tk/".

[39] D. A. Pinhong Chen Kirkpatrick, K. Keutzer. Scripting for EDA tools: a case study . In IEEE, editor, *2001 International Symposium on Quality Electronic Design*, pages 87–93, março 2001.

[40] unixabg. unixabg. Página na internet, unixabg, julho 2007. "http://vtcl.sourceforge.net/".

[41] Graphviz.org . Graphviz - Graph Visualization Software Homepage. Página na internet, julho 2007. "http://www.graphviz.org/".

[42] Roy Sterritt and Dave Bustard. Towards an autonomic computing environment. *dexa*, 00:699, 2003.

[43] Sandklef GNU Labs. Xnee GNU Project Homepage. Página na internet, Sandklef GNU Labs, julho 2007. "http://www.sandklef.com/xnee/".

[44] W. Snyder . Veripool Homepage. Página na internet, julho 2007. "http://www.veripool.com/".

[45] The Meta-Environment. The Meta-Environment Homepage. Página na internet, CWI, julho 2007. "http://www.cwi.nl/htbin/sen1/twiki/bin/view/Meta-Environment/WebHome/".

[46] Ricardo R. Gudwin . *Virtual, Distributed and Flexible Organisations* , chapter Semionics: A Proposal for the Semiotic Modelling of Organisations, pages 15–33. Springer, 2004. "http://www.springerlink.com/content/t477136667j74686".