UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JOÃO DANIEL TOGNI

Uma Proposta para Auxiliar a Interoperabilidade entre Ambientes e Ferramentas Independentes de CAD para Microeletrônica

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

Prof. Dr. Renato Perez Ribas
Orientador

Profa. Dra. Maria Lúcia Blanck Lisbôa
Co-orientadora

Porto Alegre, abril de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Togni, João Daniel

Uma Proposta para Auxiliar a Interoperabilidade entre Ambientes e Ferramentas Independentes de CAD para Microeletrônica / João Daniel Togni – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

84 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Renato Perez Ribas; Co-orientador: Maria Lúcia Blanck Lisboa.

1.*Framework*. 2.Interoperabilidade 3.Linguagens *script*. I. Ribas, Renato Perez. II. Lisbôa, Maria Lúcia. III. Titulo.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Profa. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Flávio Rech Wagner
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Aos órgãos CNPQ, Fapergs e Capes, pelo financiamento dos projetos que participei, e que tornaram este trabalho possível. Aos meus familiares e amigos, que compartilharam minha trajetória, oferecendo-me suporte em momentos difíceis e compartilhando os alegres. Aos professores Renato P. Ribas e André I. Reis pela amizade e pela oportunidade que me foi dada por eles de aprofundar meus conhecimentos sobre microeletrônica. A professora Maria Lúcia Lisboa pelo importante auxílio durante o curso de mestrado. E a Simone Bavaresco que tem sido uma companheira fiel e amorosa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
BICO	Basic Interoperability through Components and Objects
BNF	Backus-Naur Form
CAD	Computer Aided Design
CDF	Cell Design Flow
CI	Circuitos Integrados
CIF	Calma Intermediate Format
CLR	Common Language Runtime
CODEC	Coder/Decoder
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DCOM	Distributed Component Object Model
DHTML	Dynamic HyperText Markup Language
DNA	Deoxyribonucleic acid
DTD	Document Type Definition
FSBM	Front-Side Bulk Micromachining
FTP	File Transfer Protocol
GUI	Graphical User Interface
HDL	Hardware Description Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Intergrated Development Environment
IDL	Interface Definition Language
IPC	InterProcess Comunication
LDAP	Lightweight Directory Access Protocol
LVS	Layout versus Schematic
MEMS	MicroElectroMechanical Systems
MOM	Middleware Orientado a Mensagem
MSIL	Microsoft Intermediate Language
OMG	Object Management Group
PDF	Portable Document Format
PI	Procedural Interface
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTL	Register Transfer Level ou Register Transfer Logic
SGML	Standard Generalized Markup Language

SOAP	Simple Object Access Protocol
SPICE	Simulation Program Integrated Circuits Especially
TCP	Transmission Control Protocol
UDDI	Universal Discovery, Description, and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VHDL	VHSIC Hardware Description Language
WfMC	Workflow Management Coalition
WSDL	Web Services Description Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup language
XPATH	XML Path Language
XSLT	Extensible Stylesheet Language Transformations

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Os projetos de CIs (Circuitos Integrados) atualmente compreendem muitas tarefas para sua execução. Durante um fluxo de projeto de CI são necessárias ferramentas que lidam com essas diferentes tarefas. Algumas empresas compilam diversas ferramentas em um único ambiente, ou *framework*, onde tais ferramentas são adaptadas para interagir entre si. O uso desses *frameworks* é suficiente para muitos projetos, porém podem existir requisitos que obriguem a utilização de ferramentas independentes para suprir deficiências dos ambientes, exigindo a utilização conjunta de ferramentas não projetadas para cooperar.

A interoperabilidade entre sistemas computacionais tem se tornado um tópico de extrema importância. Ela possibilita a execução conjunta de ferramentas, diminuindo a necessidade de intervenção humana para tanto. A interoperação entre ferramentas independentes e *frameworks* é importante não somente para facilitar o uso conjunto de ferramentas, mas também permite que outros tópicos sejam explorados. Entre eles estão o trabalho de equipes geograficamente distantes e a possibilidade de trabalho com grandes quantidades de dados, que são duas questões importantes para microeletrônica.

Ainda, a interoperação entre ferramentas independentes e ambientes traz benefícios mútuos: as ferramentas podem utilizar funcionalidades dos ambientes e se adaptar aos fluxos de projeto deles; os ambientes podem ter suas funcionalidades estendidas pela inclusão de novas ferramentas em seu trabalho. Essas questões são especialmente importantes para pequenas empresas ou ferramentas acadêmicas que não têm condições de incorporar em suas ferramentas muitos dos procedimentos que os ambientes oferecem.

Este trabalho apresenta uma proposta para auxiliar a interoperação entre ferramentas independentes e *frameworks* relevantes para a microeletrônica, através de um protocolo inspirado em SOAP (Simple Object Access Protocol), além de oferecer ferramentas de auxílio para a adaptação ao protocolo proposto. A interação com os *frameworks* é feita através de linguagens *script* disponibilizadas por eles. Estudos de caso são apresentados para demonstrar a usabilidade da proposta.

Palavras-Chave: CAD, Microeletrônica, Interoperabilidade.

A Proposal to Aid the Interoperability between Microelectronic CAD *Frameworks* and Independent Tools

ABSTRACT

The IC (Integrated Circuit) project flow encompasses many tasks for its completion. This flow requires tools for each of the involved tasks. Some companies pack together many tools in *frameworks*, adapting the tools to interact with each other. Many projects can be handled with these *frameworks*. However there are some project specifications that require the use of specific tools for supplying some *framework* weakness, leading to the need of using tools not developed to cooperate.

The interoperability among computer systems has become a very important issue. It makes possible the joint execution of tools, reducing the need of human interference. The interoperation of independent tools and *frameworks* is important not only to simplify the joint tool execution, but also allows other issues to be addressed. Among them are the geographically separated team work and the possibility of dealing with a huge amount of data. Both issues are very important in today microelectronic designs.

The interoperation among independent tools and *frameworks* benefits both: the tools, because it can take advantage of the *framework* utilities, being included in their workflow; and the *frameworks* because it has its coverage extended through the inclusion of new tools in the workflow. These benefits are especially important for small companies and academic tools that do not have the possibility of incorporating a huge amount of utilities provided by the *frameworks*, in its tools.

This work presents a proposal for aid the interoperation among independent tools and *frameworks* that are relevant for microelectronics, through a protocol based on SOAP (Simple Object Access Protocol). Besides, tools for aiding the adoption of the protocol are provided. To interact with *frameworks*, their script languages are used. Case studies are presented demonstrating the usability of the proposal.

Keywords: CAD, Microeletronics, Interoperability.

1 INTRODUÇÃO

1.1 Apresentação

A crescente integração de dispositivos microeletrônicos em um único chip, formando sistemas cada vez mais complexos, implica também maior complexidade no projeto e confecção desses dispositivos. As dificuldades dos projetos não são provenientes somente da grande quantidade de elementos de construção (p.ex. transistores). A utilização de diferentes tecnologias (p. ex. *Micro Electro Mechanical Systems* – MEMS (SENTURIA, 2004), sistemas digitais, sistemas analógicos) também resulta em dificuldades e desafios para os projetos. Ainda, o desenvolvimento das tecnologias, que as amadurece até que estas se tornem economicamente e tecnicamente viáveis, também é um desafio à parte que exige grandes investimentos de tempo e dinheiro. Física de semicondutores, física quântica, físico-química, química, matemática e eletrônica são exemplos de áreas envolvidas no desenvolvimento dessas tecnologias e na criação dos dispositivos.

Ferramentas de CAD (*Computer-Aided Design*) tornaram-se indispensáveis para a microeletrônica desde seus primeiros passos (CAMPOSANO, 2000). Elas primeiramente permitiram a automatização de processos repetitivos, facilitando assim, a produção de sistemas com maior número de elementos internos. Atualmente existem ferramentas para automatizar processos com grandes quantidades de dados, que seriam extremamente complexos, e em alguns casos inviáveis, se utilizassem puramente mão-de-obra humana. A utilização de ferramentas que simplificam os processos envolvidos no projeto e desenvolvimento de CIs permite que os processos sejam reavaliados. Como conseqüência, há uma realimentação desses processos com novas técnicas e ferramentas, auxiliando assim, a evolução da área de microeletrônica.

A quantidade e a diversidade de ferramentas de construção necessárias para a obtenção dos sistemas integrados atuais acarreta dificuldades aos projetos. O fluxo de um projeto de CI comum envolve atualmente muitas ferramentas distintas que são utilizadas para projeto, desenvolvimento, validação, verificação, teste e demais etapas que o fluxo contenha. Além de ferramentas independentes, que atuam em nichos específicos, encontram-se disponíveis no mercado soluções integradas que englobam mais de uma área envolvida num projeto de microeletrônica. Essas soluções, também chamadas de *frameworks*, são conglomerados de ferramentas que oferecem facilidades de comunicação entre as mesmas, possibilitam o controle de versões, disponibilizam linguagens *script* para automação de procedimentos, entre diversas outras funcionalidades. *Frameworks* geralmente propõem um fluxo de projeto aos seus usuários, ou seja, além das ferramentas, eles oferecem uma seqüência de

etapas de projeto e desenvolvimento, visando melhorar a produtividade dos usuários e a qualidade dos dispositivos gerados.

Também de grande importância são os CADs para o auxílio ao estudo e simulação dos processos tecnológicos. A busca pela miniaturização implica a necessidade de estudo de efeitos desconhecidos (ou que anteriormente podiam ser ignorados), e mesmo de melhores modelos para representar os processos tecnológicos e os dispositivos construídos neles.

As inovações de software e hardware, especialmente na área de comunicação e rede, apontam para novas formas de utilização e de interação entre ferramentas. A aplicação dessas tecnologias em softwares de desenvolvimento e projeto de CI tornar-se-á cada vez mais importante. As tecnologias de interoperabilidade, interação, cooperação entre softwares, utilizando até mesmo diferentes plataformas de hardware, sistemas operacionais ou linguagens, são exemplos dessa nova tendência.

1.2 Motivação e Objetivos

A produção acadêmica de software de CAD para microeletrônica em grande parte tem como alvo demonstrar as novas tecnologias em desenvolvimento. O foco da produção acadêmica é a tecnologia e não produtos comerciais. Após o desenvolvimento das idéias fundamentais de um projeto, em geral, são desenvolvidos softwares demonstrativos para elas. Visto que as universidades não visam a concretização de produtos, o desenvolvimento dos softwares se faz sob forma de protótipos e, consequentemente muitos deles apresentam deficiências diversas. Entre elas pode-se enumerar:

- Em geral, não são desenvolvidas ferramentas auxiliares, ou seja, a ferramenta funciona separadamente e não compõe ou não se encaixa em um fluxo completo para projeto de CI;
- Em geral, contempla apenas um problema específico de uma área;
- Muitas vezes existe documentação sobre a tecnologia, porém a documentação adequada sobre a utilização da ferramenta é precária.
- Por vezes ao chegar a um resultado suficiente para a demonstração da tecnologia, os softwares têm seu projeto descontinuado, pois a universidade pouco se beneficia com a produção de utilitários.

As universidades continuarão produzindo eficientemente tecnologias, apesar da dificuldade na geração e aproveitamento dos produtos oriundos delas. Como forma de aproveitar com mais eficiência os resultados das pesquisas, este trabalho propõe a integração de ferramentas acadêmicas a ambientes já existentes. Os ambientes escolhidos para o estudo e desenvolvimento do trabalho são o Cadence Design *Framework* II (2005) e o Mentor Graphics Falcon *Framework* (2005), pois estes são dois dos ambientes mais importantes da área. Além do objetivo de integração de ferramentas, a interoperabilidade de *frameworks* também é visada.

Os ambientes Mentor Graphics Falcon *Framework* e Cadence Design *Framework* II são compostos de diversas ferramentas. Além das ferramentas, eles provêem uma estrutura de dados única, controle de versão de projetos, linguagens *script* e interfaces integradas entre as ferramentas que oferecem (HARRISON et al, 1990). Dessa forma, ambos fornecem fluxos de trabalho, que permitem aos usuários obter toda a funcionalidade necessária para a confecção de CI, sem mudar de ambiente e sem problemas de integração entre as ferramentas componentes dos ambientes.

Linguagens *script* são um dos mecanismos propostos para interação com esses *frameworks*. Essas linguagens são importantes, pois permitem a extensão dos *frameworks*. O estudo das linguagens *script* oferecidas pelos *frameworks* (TOGNI, 2005) direcionou as melhores possibilidades de desenvolvimento de uma solução para a interação entre softwares desenvolvidos em linguagens de uso geral e *script*.

A maioria dos trabalhos desenvolvidos no grupo de pesquisa Lagarto (TOGNI, 2002), assim como em outros grupos que desenvolvem CAD para microeletrônica, é desenvolvida em C++ (STROUSTRUP, 1997) e Java (SCHILDT, 2002). Por esse motivo, essas linguagens são também objetivadas neste trabalho. Entre os trabalhos que o grupo desenvolve estão: o Cellplex CDF, um gerador automático de leiautes digitais partindo de equações lógicas; o ET, que é um simulador de corrosão e ferramentas auxiliares; o Gene, que é uma ferramenta de geração automática de leiaute utilizando-se codificação Java; e o Elis, uma ferramenta para mapeamento tecnológico.

Portanto, as linguagens C++ e Java devem ser contempladas para garantir o sucesso da proposta de trabalho. Para obter a interação entre essas linguagens de propósito geral com as linguagens *script* dos ambientes em estudo foram consideradas diversas técnicas como a conversão de código entre linguagens, a utilização de arquivos em formatos padrões e a adoção de meios comuns de comunicação como *sockets*. A área de interação entre softwares foi estudada para auxiliar na decisão e compreensão do estado da arte e soluções possíveis para interoperação dos softwares.

Por fim, o trabalho propõe uma forma de interoperar *frameworks* de microeletrônica com ferramentas independentes, visando incluí-las no fluxo de trabalho dos ambientes de forma simples. Assim, espera-se como beneficio gerado por este trabalho, que ferramentas independentes possam utilizar recursos dos *frameworks* alvos, assim, complementando suas tarefas. Objetiva-se também que ferramentas possam ser integradas ou interoperem com os *frameworks*, entrando em seu fluxo de projeto e desenvolvimento.

1.3 Estrutura da Dissertação

Este trabalho está estruturado em sete capítulos. O capítulo dois apresenta os ambientes (*frameworks*) alvo deste trabalho e suas linguagens *script*, que são utilizadas como forma de customização e integração. O capítulo três apresenta e discute o conceito de interoperabilidade, termos e conceitos relacionados. É de fundamental importância esclarecer os princípios e tecnologias emergentes nessa área, pois a evolução acelerada da área, impulsionada pela evolução das tecnologias de comunicação, constantemente traz inovações importantes e eficazes.

Os capítulos quatro e cinco descrevem a proposta de solução para interoperabilidade, que é o tema deste trabalho. Detalhes de implementação e modelagem são expostos para prover conhecimento de características internas do mecanismo e suas conseqüências, facilitando assim, a extensão do projeto para outras linguagens de programação e outras situações.

O capítulo seis apresenta estudos de caso e avaliações comparativas da proposta com outras soluções para avaliar a validade e eficiência quanto a diversos quesitos. São apresentados exemplos de utilização funcionais que tiram proveito do mecanismo proposto. Finalmente, o capítulo sete apresenta as considerações finais desta dissertação e os trabalhos futuros.

2 AMBIENTES DE CAD PARA MICROELETRÔNICA

2.1 Introdução

Ambientes de CAD de microeletrônica oferecem uma forma muito importante de disponibilização de ferramentas. Como principal objetivo, os ambientes permitem que projetos completos sejam desenvolvidos sem necessitar migrar dados de forma manual. Para tanto, ambientes oferecem mecanismos de integração entre suas ferramentas tais como utilização de formatos padrões, linguagens *script*, base de dados unificada, controle de versões.

Visando esclarecer o surgimento dos ambientes, um breve histórico da evolução das ferramentas é apresentado neste capítulo, assim como seus contextos de utilização. Uma apresentação breve de ambientes de CAD para microeletrônica e suas características mais importantes também é feita. Dois dos principais *frameworks* comerciais são destacados, pois estes são utilizados neste trabalho. Ambos disponibilizam linguagens bastante poderosas, através das quais pode-se customizar as interfaces gráficas e acessar as funcionalidades das ferramentas.

2.2 Evolução das Ferramentas de CAD para Microeletrônica

As ferramentas de CAD para microeletrônica têm um impacto muito importante na produção de CIs. De forma complementar, a evolução das tecnologias requer novas ferramentas que facilitem o projeto e desenvolvimento de CIs, conforme Don MacMillen et al (2000), que traça um histórico sobre a evolução destas áreas. A evolução, segundo este autor, pode ser dividida com base nas tecnologias empregadas, como a seguir resumida.

Até o fim dos anos 60, os projetos de CIs eram feitos manualmente, utilizando-se de desenhos e corte de filmes especiais para o projeto dos circuitos. Minimização de lógica combinacional e álgebra booleana eram as tecnologias empregadas no projeto. No início dos anos 70 surgiram sistemas digitalizados, auxiliando a produção dos desenhos antes executados manualmente. Simulações de circuitos eram feitas com SPICE[1] (Simulation Program Integrated Circuits Especially). As primeiras ferramentas de roteamento automático para placas de circuito impresso também datam desta época. Ainda nesse período o formato GDS II[2] se tornou padrão para descrição de leiautes.

[1] Um programa original da universidade da Califórnia, Berkeley, que provê uma simulação de circuitos integrados com componentes como transistores, diodos, resistores, capacitores, etc.

[2] Um formato de arquivo de foto-plotagem utilizado para arquivos de máscaras de circuitos integrados. Originalmente desenvolvido por GE Calma, uma das primeiras empresas de CAD.

O advento de estações de trabalho para o projeto de CIs e suas interfaces gráficas permitiram que os projetistas desenhassem diretamente a lógica dos circuitos em ferramentas de captura de esquemáticos. Assim os projetos podiam ser simulados e depurados em nível lógico, aumentando o nível de abstração dos testes e por conseqüência a qualidade e complexidade dos circuitos.

A década de 80 foi palco para o surgimento de ferramentas muito poderosas para criação de leiautes de CI: posicionamento e roteamento automático, verificação de regras de desenho, comparação entre leiaute e esquemático, são algumas das tecnologias emergentes da época. Linguagens de descrição de hardware de alto nível como Verilog e VHDL permitiram a adoção de simuladores de lógica e RTL (Register Transfer Logic), facilitando o trabalho com circuitos mais complexos.

Em 1987, as primeiras ferramentas comerciais de síntese lógica são criadas, permitindo o mapeamento de listas de transistores para diferentes bibliotecas de células lógicas. A utilização dessas tecnologias foi amplamente adotada como metodologia de trabalho nos anos 90. Os projetos eram desenvolvidos em RTL e podiam ser transformados em *netlists* e, através do posicionamento e roteamento o leiaute correspondente era gerado. Testes estruturais tornaram-se amplamente adotados no final da década. A síntese de RTL passou a ser utilizada como um meio de verificação de sistemas digitais. A análise estática de *timing* e as verificações formais complementavam a simulação.

Esse breve histórico permite observar que a busca por interoperabilidade tornou-se importante com o aumento da complexidade dos projetos. Abaixo são descritos cinco diferentes cenários relativos às interações entre ferramentas durante essa evolução.

Primeiramente as tarefas necessárias para o projeto de um CI eram totalmente feitas à mão, o que corresponde ao **primeiro cenário** (figura 2.1)**.** Em seguida, alguns procedimentos envolvidos no projeto dos circuitos foram automatizados através de ferramentas. Assim, os engenheiros podiam utilizar o tempo disponível para outras tarefas. Por sua vez novas automações permitiram com que circuitos mais complexos fossem elaborados.

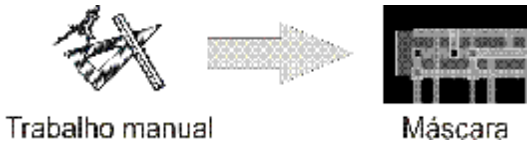

Figura 2.1: Cenário 1.

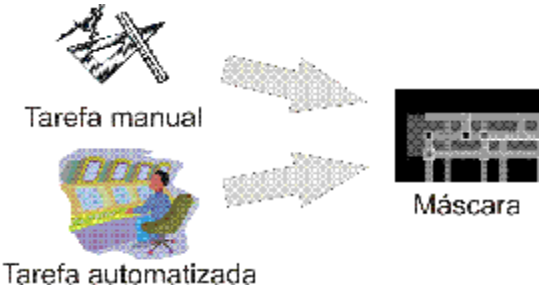

Figura 2.2: Cenário 2.

Porém, esse cenário de ferramentas de automação era composto majoritariamente por ferramentas independentes, ou seja, que não possuíam capacidade de interação com as outras ferramentas envolvidas no fluxo de projeto da época, o que representa o **segundo cenário** (figura 2.2).

Formatos padronizados de arquivos (comunicação por arquivos) e meios de comunicação em rede passaram a ser utilizados para aumentar a produtividade das ferramentas, reduzindo a necessidade de intervenção humana durante as etapas de projeto.

As ferramentas adquiriram a capacidade de receber e enviar dados umas para as outras, de forma automatizada ou semi-automatizada, correspondendo assim ao **terceiro cenário** (figura 2.3).

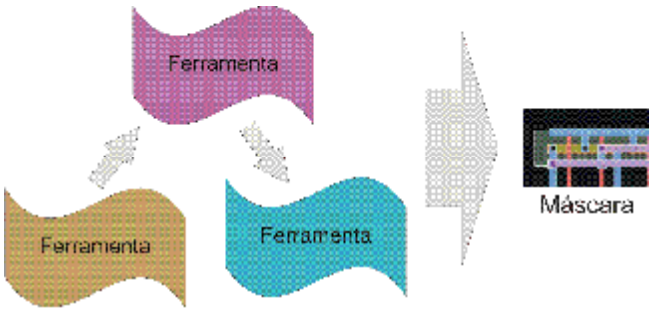

Figura 2.3: Cenário 3.

O mercado de CAD para microeletrônica cresceu juntamente com a produção de *chips*. Grandes empresas passaram a dominar o mercado, oferecendo ferramentas de altíssimo nível. Porém, mais do que apenas de ferramentas independentes, essas empresas tornaram-se capazes de disponibilizar ambientes compostos por diversas ferramentas integradas de forma coesa (HARRISON et al, 1990). Esse é o **quarto cenário** (figura 2.4). Linguagens *Script*, bancos de dados unificados, mecanismos de comunicação e estruturas de dados únicas são exemplos das tecnologias utilizadas para possibilitar a integração das ferramentas dos ambientes.

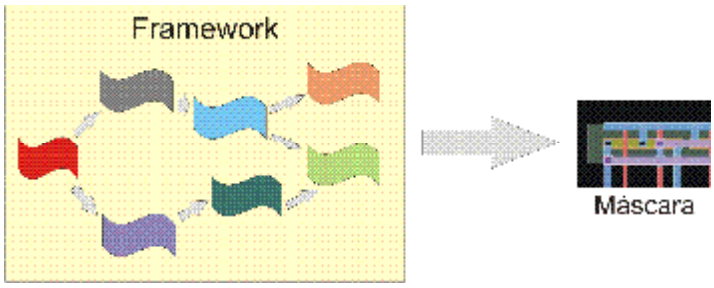

Figura 2.4: Cenário 4.

O atual estado, referente à interoperação de ferramentas, é uma mistura de alguns componentes dos cenários anteriores, adicionando as novas tecnologias de comunicação em rede, especialmente a Internet. A Internet é utilizada principalmente para divulgação das empresas. Porém ela também é palco para a interação de ferramentas através de protocolos proprietários ou de domínio público como HTTP (HyperText Transfer Protocol) e FTP (File Transfer Protocol). A utilização desses protocolos dá origem a mecanismos de comunicação mais completos como, por exemplo, *Web Services*, que surgiram como um novo meio de integrar ferramentas através da Internet. Nesse **quinto cenário** (figura 2.5) *frameworks* e ferramentas independentes comunicam-se de formas variadas para obter os resultados desejados. A utilização de equipes geograficamente distantes e a necessidade de utilização de grandes quantidades de dados são exemplos de dificuldades que podem ser amenizadas com este tipo de interação entre ferramentas.

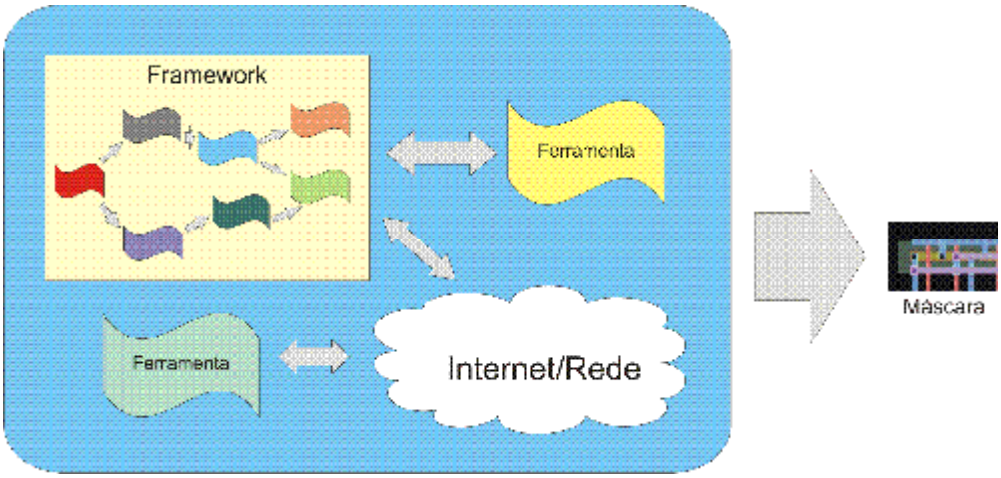

Figura 2.5: Cenário 5.

2.3 Ambientes de CAD

Dentre as diversas soluções propostas para lidar com a diversidade de áreas envolvidas no projeto de CIs, duas categorias de software se destacam: *frameworks* e as ferramentas independentes. *Frameworks* de grande porte apresentam soluções completas para o projeto de CIs, enquanto ferramentas independentes lidam com tarefas específicas. Um *framework* é mais que um simples aglomerado de ferramentas, pois ele provê diversas funcionalidades para auxiliar o fluxo de trabalho dos usuários (HARRISON et al, 1990).

As ferramentas que compõe um *framework* de microeletrônica podem ser classificadas em três categorias: *front-end, back-end* e *environment*. Neste documento considera-se *front-end* como o trabalho com níveis de abstração mais altos como linguagens de descrição de *hardware* de alto nível e descrições lógicas. Por *back-end* entende-se o trabalho com níveis mais baixo como redes de transistores e leiautes. Os *frameworks* a seguir apresentados agregam ferramentas de *front-end* e *back-end*. As funcionalidades que compõe o auxílio à integração das ferramentas, ou que estejam relacionadas ao ambiente e não a microeletrônica são consideradas funcionalidades de ambiente ou *environment.*

Entre as ferramentas de *front-end*, podemos citar a edição, simulação e síntese de HDL (Hardware Description Language), e edição, simulação e síntese de descrições lógicas (equações, tabelas verdade, bdd). Entre as ferramentas de *back-end* pode-se citar a edição, simulação e síntese de redes de transistores (também chamados esquemáticos), mapeamento tecnológico, checagem de regras de desenho, extração de características elétricas, LVS (Layout Versus Schematic), posicionamento, roteamento, teste, entre outros. Entre as ferramentas de ambiente estão: bancos de dados, controle de versão de artefatos de projetos, utilitários internos para unificação de comportamento e aparência (*look & feel*) e linguagens *script*.

Entre os principais ambientes de CAD para microeletrônica estão os das empresas Cadence Design Systems (2005), Mentor Graphics (2005), Synopsys (2005), Quartus Altera (2005) e Symplicity (2005). Os três primeiros possuem maior domínio de mercado. Este trabalho se concentra nos *frameworks* das duas primeiras empresas: Cadence Design

Systems e Mentor Graphics, que serão brevemente apresentados a seguir. Essa escolha deve-se à importância e disponibilidade dos mesmos para estudos e testes.

2.3.1 Cadence Design *Framework* II

O ambiente Design *Framework* II (figura 2.6) é composto de ferramentas para todos os procedimentos envolvidos pelo fluxo completo de projeto de CI.

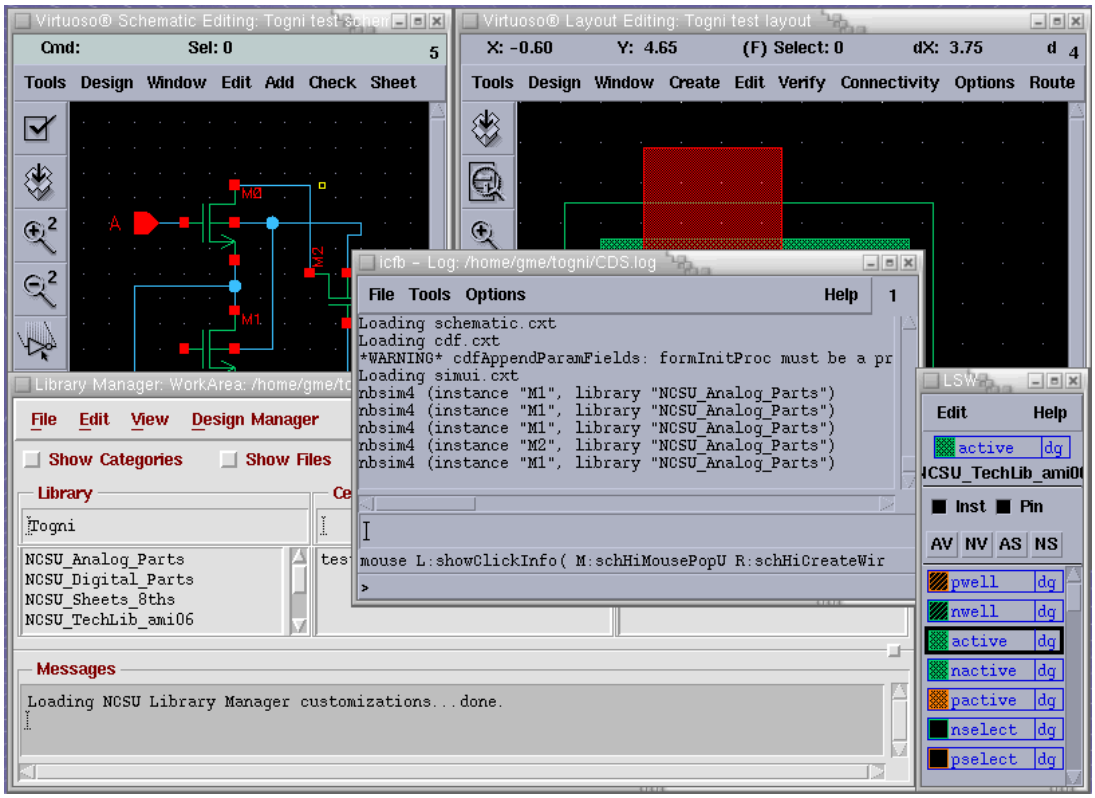

Figura 2.6: Ambiente Cadence Design *Framework* II

A ferramenta Virtuoso Layout Editor é utilizada para edição de leiautes. Ela é integrada a ferramentas de checagem de regras de desenho, extração de parâmetros elétricos, ao banco de dados do ambiente (onde recuperam e armazenam dados sobre o processo tecnológico utilizado e artefatos de projeto). Ela oferece todas as suas funcionalidades através de duas interfaces: a GUI (Graphical User Interface) e através das linguagens Skill e Skill++ (apresentada adiante).

O Virtuoso Schematic Editor é utilizado para a edição de esquemáticos. Ela possui integração com ferramentas de simulação, com o banco de dados do ambiente, e também oferece as duas interfaces para uso da ferramenta, assim como todas as ferramentas disponíveis no *framework*.

A ferramenta Analog Environment permite a simulação de esquemáticos e a visualização dos formatos de onda resultantes da simulação.

Para a edição, simulação e síntese de linguagens de descrição de hardware, o ambiente disponibiliza as ferramentas VHDL Tool Box e Verilog-XL Integration.

O banco de dados do ambiente armazena as informações de artefatos de projeto criados pelos usuários e informações sobre configurações (tecnologia, configurações específicas de ferramentas, etc). Ainda a ferramenta disponibiliza a utilização de um sistema de controle de versões de artefatos de projeto, auxiliando o trabalho de grupos de usuários.

As ferramentas citadas dão uma breve impressão da aparência e funcionalidades presentes no ambiente. Porém, muitas outras ferramentas compõem o ambiente Design *Framework* II.

2.3.2 Mentor Graphics Falcon *Framework*

Assim como o ambiente DFII, o ambiente Falcon *Framework* possui ferramentas para diversas tarefas que fazem parte do fluxo de projeto de CIs. O ambiente da Mentor Graphics, juntamente como o da Cadence e Synopsys, divide a liderança do mercado de CAD para microeletrônica.

A ferramenta IC Station (figura 2.7) é responsável pela edição de leiautes. Esta está integrada a ferramentas de checagem de regras de desenho, ferramentas de LVS, entre outros.

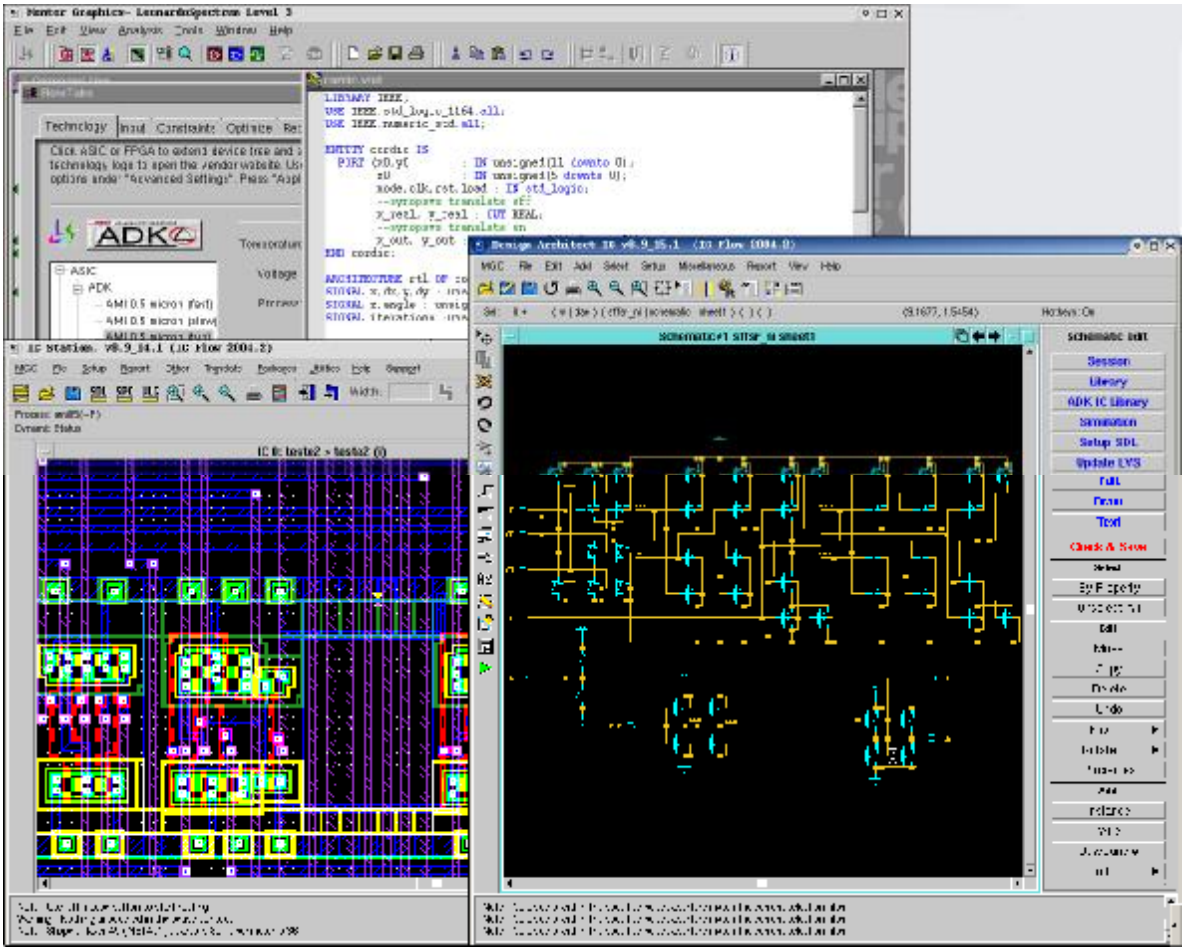

Figura 2.7: Ferramentas Mentor Graphics

A ferramenta Design Architect (figura 2.7) é utilizada para a edição de esquemáticos e VHDL. Integrada a ela estão ferramentas de simulação elétrica. Como em todas as

ferramentas do sistema, as funcionalidades dela e do simulador também são oferecidas através da interface gráfica e linguagem *script*.

Além das ferramentas, o ambiente disponibiliza um banco de dados único para todas as ferramentas. A aparência e usabilidade é semelhante entre as ferramentas do *framewok*. A linguagem script Ample é utilizada para sua customização.

2.4 Customização dos Ambientes

As linguagens *script* são um importante mecanismo de customização e interação automatizada disponível nos ambientes alvo deste trabalho. Dentre suas características principais estão o acesso a todas as ferramentas e funções do ambiente através da linguagem, e a possibilidade de criação de interfaces gráficas (TOGNI, 2005). Apesar de algumas diferenças entre as linguagens oferecidas pelos *frameworks* citados na seção anterior, ambas são bastante poderosas; através delas pode-se customizar e estender os ambientes hospedeiros.

Também chamadas linguagens de comandos, linguagens *script* visam dar controle de forma programática sobre as funcionalidades oferecidas por uma ferramenta. Elas são utilizadas em sistemas operacionais, aplicativos (editores de texto, planilhas eletrônicas), ferramentas para Internet (FTP), etc. Também são bastante populares as linguagens *script* destinadas à programação de páginas Web de conteúdo dinâmico, a exemplo das linguagens JavaScript (GOODMAN, 2004), Perl (DEITEL, 2002) e PHP (MUTO, 2004).

Um exemplo de uso de linguagen *script* que é bastante vinculada a um aplicativo, são macros presentes em diversas ferramentas. Nessas ferramentas pode-se gravar macros, seqüências de digitação e utilização da interface gráfica para futura execução através de teclas de atalho, simplificando a repetição de tarefas.

Existem variações importantes quanto a capacidade de expressão e o controle sobre a ferramenta hospedeira que a linguagem *script* pode oferecer. Por serem atreladas a uma ferramenta hospedeira, em geral as linguagens *script* são interpretadas e não compiladas. Assim, elas permitem a utilização das ferramentas de forma mais flexível do que apenas a utilização das interfaces gráficas da ferramenta, simplificando a execução de tarefas repetitivas e reduzindo o tempo de realização da tarefa ou seqüência de tarefas nessas ferramentas.

A adoção de uma linguagem *script* objetiva oferecer mais recursos aos usuários de uma ferramenta. Tais recursos provêm basicamente de duas fontes. Primeiramente **das características da própria linguagem**, que podem permitir maior controle, organização, expressividade, etc. Por exemplo, existem linguagens *script* orientadas a objetos, estruturadas, funcionais, em sua maioria são não tipadas[3]. Ou seja, as variações possíveis de características de linguagens *script* são importantes e devem ser consideradas. Em segundo lugar **das funcionalidades da ferramenta** que podem ser controladas, adaptadas ou estendidas por meio da linguagem.

2.4.1 Skill++ (Cadence)

O ambiente de desenvolvimento Cadence pode ser estendido utilizando as linguagens Skill e Skill++ (BARNES, 1990) (WOOD, 1986) (CADENCE, 2002) (CADENCE, 2002). A linguagem Skill++ é uma evolução de Skill que adiciona características do modelo de

[3] Linguagens não tipadas são aquelas que não adotam um sistema de tipos para representação de dados

objetos às características funcionais da linguagem original. O ambiente Cadence, além das funcionalidades das ferramentas componentes, também fornece um conjunto de funções que produzem e controlam interfaces gráficas de usuário. Dessa forma é possível criar formulários com telas de aquisição ou exibição de dados, ou até mesmo telas com componentes gráficos desenhados de forma personalizada como elipses, retângulos, *strings*, etc. Utilizando essas funções podem ser criadas novas ferramentas ou módulos, oferecendo uma interface gráfica para o usuário, de forma completamente integrada ao ambiente. A programação em Skill++ permite que sejam automatizadas as tarefas de projeto, como criação de esquemáticos, leiautes, simulação, ou qualquer outra função disponível no ambiente. A grande maioria das funções do ambiente tem acesso disponível através da linguagem, sendo que alguns poucos procedimentos que não estão presentes na linguagem podem ser realizados por chamadas a programas executáveis.

Skill++ é baseado em Lisp (GRAHAM, 1996). Lisp, por sua vez, é a linguagem funcional mais difundida da atualidade. Skill++ é interpretada, característica comum em linguagens *script*. Linguagens interpretadas comumente oferecem funções que permitem a execução de códigos produzidos em tempo de execução e armazenados em *strings* ou arquivos. Essa característica dá ainda mais flexibilidade aos programas dessas linguagens. Skill++ é uma linguagem funcional; ela caracteriza-se por ter funções como elementos de primeira ordem, ou seja, funções podem ser passadas como parâmetros, atribuídas a variáveis ou mesmo utilizadas como retorno de outras funções. A partir de um conjunto de funções primitivas, novas funções podem ser criadas por composição de outras funções, permitindo assim a criação de programas compactos e flexíveis.

2.4.2 Ample (Mentor Graphics)

Ample (MENTOR, 2003-a) (MENTOR, 2003-b) é uma linguagem *script* fornecida como parte do ambiente Mentor Graphics Falcon *Framework*, inicialmente desenvolvida pela empresa Apollo Computers para as estações de trabalho Apollo/Domain. É uma linguagem funcional com sintaxe similar a linguagem C (KERNIGHAN, 1990), que é por sua vez estruturada. Porém existem diversas características de linguagens funcionais (SCOTT, 2000) que não são encontradas em Ample. Na verdade, Ample é basicamente uma linguagem estruturada como C que trabalha com conceitos comuns de linguagens funcionais. Entre eles está a tipagem fraca, a utilização de funções como elemento de primeira ordem e execução dinâmica de código.

Ample dá acesso a todas as funcionalidades do ambiente e, além disso, possibilita a criação de interfaces gráficas para interação com usuários. Dessa forma, o ambiente permite sua extensão de forma programática, podendo automatizar procedimentos complexos. Algumas restrições de Ample a tornam uma linguagem menos poderosa que Skill, porém, para seus objetivos iniciais ela é uma linguagem bastante completa.

2.5 Algumas considerações

Neste capítulo foram apresentados um breve histórico da evolução das ferramentas e ambientes de CAD para microeletrônica. Os ambientes DF-II e Falcon *Framework* foram também brevemente apresentados. Ambos são de grande importância para microeletrônica, pois disponibilizam ferramentas de alta qualidade, que cobrem grande parte das tarefas envolvidas no projeto de CIs. Além disso, eles possuem grande aceitação mercadológica.

Ambos fornecem linguagens *script* específicas como forma de customização e extensão. As linguagens são extremamente poderosas, considerando a sua utilização no ambiente alvo, para a qual foram desenvolvidas.

As linguagens *script* objetivam a simplificação de pequenas tarefas e, devido à simplicidade destas linguagens de uso específico, algumas características importantes em termos de engenharia de software moderna (GAMMA, 1995) não estão presentes nelas, dificultando o desenvolvimento de sistemas complexos. A utilização dessas linguagens para a implementação de algoritmos complexos e de novas funcionalidades para ferramentas de CAD torna-se, portanto, mais dispendiosa. Assim, recomenda-se a utilização das linguagens *script* somente para as funcionalidades que inevitavelmente estarão vinculadas ao ambiente hospedeiro.

Outra consideração importante diz respeito à incompatibilidade entre linguagens de ambientes distintos. Esta dificuldade de portabilidade inibe a utilização de ambientes de desenvolvimento heterogêneos no decorrer de um projeto.

O Capítulo 3 - Interoperabilidade apresenta algumas definições e formas de tornar possível a interação, a cooperação e a interoperabilidade de ferramentas distintas ou de ambientes heterogêneos.

3 INTEROPERABILIDADE

3.1 Introdução

A informatização cada vez maior de tarefas e sistemas complexos das mais diversas áreas motivou o desenvolvimento de inúmeras ferramentas. Muitas delas atendem a objetivos similares ou complementares. Essas ferramentas, porém, não foram projetadas e desenvolvidas visando a sua cooperação na consecução de suas tarefas. Visto que tais ferramentas são geralmente construídas de forma desconexa entre si, não têm por objetivo a troca de informações, por conseqüência atuando de forma independente. Trocar informações entre elas torna-se cada vez mais importante para que a automação se torne mais amplamente usada e a cooperação mais efetiva, reduzindo a necessidade de intervenções humanas.

A cooperação entre sistemas heterogêneos tornou-se possível graças ao desenvolvimento e evolução de padrões de comunicação de dados e de intercâmbio de informações (HTTP, XML (W3C, 2005) – Extensible Markup Language, RMI, CORBA, DCOM), juntamente com a identificação de áreas comuns e a consciência de que essa troca traria benefícios aos sistemas integrados e seus usuários. Impulsionados fundamentalmente pela Internet, protocolos e mecanismos distribuídos que apresentam características mais elaboradas têm conquistado um espaço cada vez mais importante e presente no cenário de projeto auxiliado por computador (CAD – Computer Aided Design).

A integração entre sistemas pode ser obtida pela capacidade de comunicação e, assim, troca de informações entre os sistemas. Os avanços nas tecnologias de comunicação auxiliam apenas a comunicação de sistemas, ou seja, permitem que eles tenham um canal de troca de informações, um meio de transporte. Porém, para que se tire proveito desse canal é necessário transformar os dados transmitidos em informações úteis, alcançando assim a capacidade de interpretar e operar em conjunto, ou interoperar. Sendo assim, a interoperabilidade envolve a habilidade de trocar informações e a capacidade de execução conjunta de tarefas.

Sistemas de maior porte, que oferecem diversas ferramentas integradas em um único sistema, ou *framework*, como Office, Autocad, os ambientes Mentor Graphics Falcon *Framework* e Cadence DFII, oferecem fluxos de trabalho ou *workflows*. Porém, cada fabricante desenvolve um *workflow* diferente, visto que desenvolve sem a preocupação de interoperar com os concorrentes. Para tentar padronizar a terminologia de *workflow*s e possibilitar a interoperabilidade entre sistemas diferentes, a WfMC (Workflow Management Coalition) (WORKFLOW, 2005), fundada em agosto de 1993, elaborou uma

série de documentos que caracterizam, definem e exemplificam conceitos importantes para o entendimento de *workflow*.

3.2 Definições

Diversos termos são utilizados nesse capítulo para descrever as possibilidades de interação entre sistemas. Portanto, antes de prosseguir com as classificações e caracterizações expostas, é necessário diferenciar e esclarecer o significado destes termos.

Interação significa a capacidade de enviar e receber dados de uma ferramenta. Note-se que o termo não implica ou define os resultados obtidos da troca de dados.

O termo **cooperação** significa a possibilidade de trabalho conjunto de ferramentas para atingir um objetivo comum. O termo não impõe formato à interação, ou seja, as ferramentas podem estar sendo executadas em paralelo, por usuários diferentes, ou mesmo estar integradas. Porém, é necessário que as funcionalidades das ferramentas contribuam para um fim comum.

Já por **integração** entende-se a incorporação a uma ferramenta de funcionalidades proporcionadas por outra ferramenta, ou seja, infere-se que uma ferramenta é vista como principal, enquanto a outra torna-se apenas um item ou componente interno da principal.

A **interoperabilidade** é a capacidade de executar serviços de outra ferramenta, ou prover serviços, ou como diversos definem: a habilidade de sistemas, unidades ou forças de prover serviços e utilizar serviços de outros sistemas, unidades ou forças de forma que ambas operem efetivamente em conjunto. Também pode-se definir interoperabilidade como a capacidade que sistemas têm de trocar informações e executar tarefas conjuntamente (LI, 2000) (LEVELS, 1998).

Portanto, a interoperabilidade pode proporcionar cooperação, integração e interação, pois não implica como ela será fornecida. A palavra interoperabilidade também pode ser utilizada para descrever a capacidade que programas têm de ler e escrever formatos de arquivos e protocolos, porém neste trabalho não terá essa conotação.

A utilização de **componentes** para modelar um sistema infere que este é parte de um sistema, ou seja, está integrado e tem capacidade de interoperar com o resto do sistema. Por isso, este termo é também utilizado quando se tem o objetivo de interoperar um sistema independente com demais sistemas, ou seja, transformá-lo em uma parte de um sistema federativo.

Com o intuito de descrever os problemas que levam à falta de interoperabilidade, é necessário definir **heterogeneidade**. Heterogeneidade compreende as diferenças em qualquer nível de abstração ou assunto, que podem incompatibilizar a interoperabilidade de um conjunto de ferramentas.

3.3 Impacto Comercial da Interoperabilidade

Tendo em vista que parte dos sistemas de hardware e de software é desenvolvida independentemente, não prevendo ou planejando a necessidade de interoperar com outros sistemas, diferentes arquiteturas, sistemas operacionais, plataformas de hardware, linguagens de desenvolvimento e modelos de dados foram criados. A interoperabilidade possui, portanto, diferentes facetas que são abordadas adiante no texto.

A interoperabilidade de hardware tem sido obtida através da utilização de protocolos de comunicação comuns, e da integração de protocolos e padrões ao sistema operacional.

Dessa forma, é possível atingir níveis de interoperabilidade, que, atuando como tecnologias de base, facilitam a construção de aplicações distribuídas.

No caso de software, diante dessas diferenças existem duas soluções para obter interoperabilidade: recriar a aplicação, ou adaptá-la. Dependendo do tamanho da aplicação, refazer o desenvolvimento pode ser uma alternativa ideal, pois assim a aplicação será ajustada perfeitamente aos requisitos de interoperabilidade desejados para o sistema. Porém, em geral os sistemas a serem integrados são suficientemente grandes para tornar o custo de recriação proibitivo. Nesse caso, a adaptação das aplicações é a solução mais adequada como forma de lidar com as diferenças dos sistemas, respeitando as limitações temporais e orçamentárias que sistemas complexos impõem.

Motivos financeiros como o domínio de patentes, segredos comerciais, ou *network externalities* (LIEBOWITZ, 1998) podem determinar a escolha por não oferecer interoperabilidade em determinado software. Para proteger tecnologias inovadoras ou promissoras, empresas podem decidir por não disponibilizar formas de interoperação de determinada ferramenta, visando com isso dificultar o entendimento de seu funcionamento e, por conseqüência, a cópia da tecnologia utilizada. Além de proteger o software, empresas podem tentar aumentar sua fatia de mercado obrigando os usuários a utilizar seus sistemas ao não oferecerem formas de interoperação. *Network externalities*, segundo Liebowitz, é o estudo das causas e conseqüências das escolhas do mercado consumidor em relação a novas tecnologias. Por exemplo, se a grande maioria decide por comprar equipamentos VHS ao invés de BetaMax, mesmo sendo superior tecnicamente, o padrão Betamax tende a não se fixar. Logo, é fácil imaginar que a capacidade de interoperar entre tecnologias antigas e novas pode influenciar a decisão de implementar ou não interoperabilidade em novos produtos.

Os resultados econômicos da não utilização de interoperabilidade podem ser desde o monopólio ao fracasso comercial. Por exemplo, supondo que uma empresa possui uma tecnologia inovadora e deseja ser a única possuidora dessa tecnologia, ela pode proteger seu software de várias maneiras, entre elas, não fornecendo interoperabilidade, porém como efeito colateral pode ocorrer a falta de adesão ao padrão imposto, pois outras ferramentas não poderão trabalhar conjuntamente a ela. Na indústria de microeletrônica a empresa Apple® e sua linha de Machintosh são um exemplo de proteção à tecnologia que resultou em perda de mercado, enquanto os PCs tiveram sua arquitetura aberta, permitindo que diversas empresas pudessem fabricar periféricos e componentes que aderiam às suas especificações.

3.4 Causas de Heterogeneidade

O estudo das causas de heterogeneidade é fundamental para a identificação de fatores que geram as diferenças que impedem os sistemas de interagir. Com tal conhecimento se adquire as bases para a formulação de soluções, posto que existem soluções que não resolvem determinados problemas dependendo de suas causas. Entre as soluções possíveis podem estar o combate das causas, ou simplesmente alternativas paliativas, caso não se possa influenciar as causas das heterogeneidades.

Sistemas que são desenvolvidos sem requisitos de interoperabilidade possuem possíveis pontos de divergência, desde as etapas iniciais de projeto até etapas de desenvolvimento. Tais divergências resultam em incompatibilidades que dificultam a sua interoperabilidade. Segundo Young (2003), considerando heterogeneidade representacional, existem três

grandes causas: perspectivas diferentes, construções equivalentes e especificações incompatíveis.

Diferentes perspectivas, ou seja, tipos de necessidades de usuários, gerentes de projeto, e equipes de desenvolvimento levam a diferentes modelos de dados e funcionalidades, mesmo quando a mesma informação está sendo modelada. A etapa de projeto de sistemas computadorizados é de definição e organização subjetiva, sendo assim, é possível fazer a análise de um mesmo sistema de diversas formas, em todas, obtendo um modelo suficientemente completo de forma a cobrir os requisitos funcionais do projeto. Por isso, o mesmo projeto pode ter requisitos similares ou mesmo idênticos e apresentar arquitetura e funcionalidades diferentes.

Por **construções equivalentes** entende-se que modelos de dados e conjuntos de funcionalidades equivalentes podem ser utilizados para modelar a mesma aplicação e informação, combinando de forma diferente os blocos básicos de construção. A utilização de determinada organização interna de dados, funções ou mesmo de tecnologia de implementação geralmente não é uma obrigatoriedade, pois existem possibilidades que alcançarão o mesmo objetivo final.

Especificações incompatíveis de projeto podem resultar em diferentes bancos de dados, esquemas e funcionalidades. Considerando projetos similares em termos de objetivo, esses podem apresentar diferentes requisitos, sendo ainda mais provável encontrar diferenças durante o projeto.

3.5 Tipos de Heterogeneidade

O resultado do projeto e desenvolvimento de sistemas independentes é a obtenção de sistemas com divergências. A incapacidade de cooperação entre esses sistemas deve-se a uma ou mais formas de heterogeneidade, que, de acordo com Young, estão divididas em dois grandes grupos e podem ser classificadas como mostrado na tabela 3.1.

O primeiro grupo é o de diferenças em **quais** características do mundo real estão sendo modeladas, chamadas de diferenças de visão. Diferenças de visão compreendem heterogeneidade de escopo, nível de abstração e validade temporal. O segundo grupo engloba **como** essas características são modeladas, chamadas de diferenças de representação. Diferenças na representação compreendem hardware e sistema operacional, modelo organizacional, estruturas, apresentação e significado.

Tabela 3.1: Tipos de heterogeneidades (YOUNG, 2003)

Tipo	**Descrição**
Escopo	É resultado de diferenças na informação utilizada para modelar a entidade real. Pode surgir de diferentes perspectivas de quais atributos da entidade real a aplicação precisa para representar.
Nível de abstração	Resulta de diferenças no nível e grau de agregação de elementos atômicos de dados, ou seja, o quanto da entidade real é necessário ser modelado.
Validade temporal	Resulta de diferenças no tempo utilizado pelos modelos para observar ou gravar o estado da entidade real. Diferenças na quantidade de tempo em que um dado é válido também podem gerar heterogeneidades.
Hardware e sistema	Resultam de diferenças de hardware e sistema operacional que afetam

operacional	a integração de sistemas autônomos. O avanço contínuo do hardware praticamente garante a existência de heterogeneidades desse tipo. Tipos de conectores USB/Serial, diferentes protocolos e tecnologias dependentes de plataforma, são exemplos disso.
Modelos organizacionais	Refere-se a diferenças no modelo conceitual usado por sistemas desenvolvidos autonomamente. Diferenças nos princípios de análise e projeto utilizados, como o uso do modelo de objetos ou análise estruturada.
Estrutura	Variações na estrutura, em como a informação foi arranjada, podem ocorrer em sistemas que usam o mesmo modelo organizacional. Diferenças de composição estrutural, desencontros esquemáticos e variações devido à presença de informação deduzida.
Apresentação	Diferenças de unidades de medida, precisão, tipos de dados, tamanhos de campo, termos da área em questão, diferentes regras de integridade compõem esse item.
Significado	Resulta da natureza imprecisa da linguagem natural quando usada pra caracterizar ou definir entidades do mundo real. Homônimos, sinônimos, e abreviações podem contribuir para esse tipo de diferença.

3.6 Critérios para avaliação das técnicas de interoperabilidade

Existem diversos mecanismos de interoperação que podem ser utilizados, como os descritos na seção 3.11. Entre eles existem diferenças como a técnica empregada, o nível de interoperação adquirido, entre outros. Para poder avaliar esses mecanismos, os critérios propostos por Young (2003) são descritos a seguir. Eles apresentam um conjunto de critérios comuns para comparação de técnicas de interoperabilidade.

Entre os diversos tipos de interoperabilidade, como plataforma de hardware, sistema operacional, linguagem de programação, etc, os mecanismos disponíveis resolvem geralmente uma parte deles, não todos. Portanto, o **primeiro** critério de avaliação são os tipos de heterogeneidade tratados pelo mecanismo em estudo, assim como o grau de eficiência.

A criação de um sistema interoperante requer a utilização de sistemas heterogêneos que se utilizam de modelos de dados que geralmente possuem diferenças nas entidades modeladas. Uma das maiores dificuldades na integração de modelos de dados diferentes é a correlação de entidades, ou seja, a identificação dos dados correspondentes nesses modelos. Li (2000), assim como Murray (1996) ressaltam a dificuldade existente na correlação entre entidades modeladas. A busca pela correspondência entre atributos e funções dos sistemas é, de forma similar, complexa. Dessa forma, a utilização de sistemas automatizados para auxiliar tais tarefas é um fator importantíssimo, pois pode reduzir drasticamente o tempo dessas tarefas, sendo esse o **segundo** critério de avaliação.

Uma das bases de interação entre sistemas é a chamada remota de funções. Uma característica dos mecanismos que se utilizam dessa técnica é a necessidade de conhecimento prévio dessas operações, o **terceiro** critério. Algumas técnicas de RPC (*Remote Procedure Call*) não oferecem meios de descoberta das funções e suas propriedades, sendo necessário que o desenvolvedor saiba antecipadamente quais funções

estão disponíveis, assim como suas características. Em outras propostas são oferecidos meios de descoberta das funções disponíveis e suas características, permitindo que o cliente se adapte ao servidor automaticamente, o que é chamado de ligação postergada.

A quantidade de modificações necessárias em um sistema para que ele interopere com outros é o **quarto** critério. Muitos trabalhos científicos como (SUTTON, 1998) e (BRGLEZ, 2001), são esforços para adaptar sistemas que inicialmente não foram desenvolvidos para interoperar. Existem sistemas que permitem que o programa original a ser adaptado não seja modificado, bastando que ele apresente características que permitam o seu encapsulamento, característica essa que muitas vezes é desejável. Outros mecanismos exigem a alteração de seu código fonte, para que as informações sejam externadas (em caso de encapsulamento), ou sejam adaptados à API (Application Programming Interface) do mecanismo de interoperação.

Para atingir interoperabilidade é inevitável utilizar padrões. A quantidade de conversões entre os protocolos ou padrões é o **quinto** quesito, ou seja, a metodologia de tradução. A existência de formatos intermediários independentes torna possível que o número de traduções necessárias seja reduzido, pois cada sistema terá que converter somente uma vez seus dados para um formato intermediário. Contudo, existem soluções que se utilizam de formatos proprietários, exigindo que sejam feitas mais conversões de dados.

O **sexto** quesito é a existência de mecanismos que auxiliem a criação automatizada de tradutores. Reutilizar traduções comumente usadas ou auxiliar a geração de tradutores para modelos correspondentes são exemplos de automação para tradutores. Assim, reduz-se também o tempo gasto nessa tarefa que tende a causar problemas quando feita manualmente.

Sistemas com interoperabilidade completa podem aproveitar-se das funcionalidades e serviços de outros sistemas. Para tanto é preciso lidar com as heterogeneidades que são naturais de qualquer processo de interação de alguma forma. Saber se o sistema as resolve somente durante a troca de informações, ou se o mecanismo as resolve naturalmente, atingindo assim, a execução conjunta de tarefas é o **sétimo** critério definido por Young.

Alguns desses critérios servirão como base para a comparação entre os diferentes mecanismos de interoperabilidade existentes e o proposto nesse trabalho. Também serão levados em conta o tempo de aprendizagem de um dado mecanismo e a quantidade e dificuldade de intervenção manual. A necessidade de entendimento de conceitos básicos para o mecanismo é também um fator importante, pois pode aumentar bastante o tempo de aprendizado de um mecanismo.

3.7 Características importantes para sistemas distribuídos

Interoperabilidade está diretamente associada com sistemas distribuídos em sua natureza de diversidade de aplicações. Para o correto funcionamento de um sistema distribuído é necessário que as partes interoperem corretamente, ou seja, apresentando características mínimas para tal interação. Tais sistemas podem ser avaliados quanto a sua escalabilidade, disponibilidade, adaptabilidade e robustez, visando determinar quão eficiente é a sua operação (SPILLER, 1997) (EMMERICH, 2000).

Para assegurar que o sistema funcionará corretamente, mesmo que ocorram mudanças de carga, deve-se prever aumentos na carga de execução ou número de usuários, ou a possibilidade de dispor de mais equipamentos de hardware para prover um dado serviço. A possibilidade de expansão de um serviço pode ser fundamental para que este seja provido

de forma eficiente. Quanto mais fácil for adicionar novos equipamentos para distribuição de carga, por exemplo, mais adaptável o serviço será a variações na carga de execução.

Um serviço deve estar disponível nos períodos de tempo que forem determinados, evitando a sua indisponibilidade de forma a respeitar as especificações de uso. O tratamento de falhas que possam degradar o serviço torna-se fundamental para que o mesmo mantenha-se funcionando corretamente e com desempenho aceitável.

Um dado serviço precisa estar preparado para tarefas gerenciais tais como a adição e remoção de usuários, servidores e serviços. A adaptação a essas mudanças deve ocorrer automaticamente e naturalmente no serviço de forma que não seja necessária manutenção ou intervenção durante esses eventos, e as alterações sejam observadas de forma consistente para todos.

Toda falha que possa resultar em perda de informação precisa ser recuperável eficientemente e incrementalmente. As falhas não podem comprometer a segurança dos dados armazenados ou em tráfego.

Ainda é importante que os atributos acima sejam alcançados com custo razoável, viabilizando a utilização da solução e permitindo uma exploração eficiente dos recursos disponíveis.

3.8 Níveis de Interoperabilidade

Todo sistema que pode se beneficiar de interação com outros, tendo sido projetado ou não para interoperar, o faz em determinado nível. Ou seja, sistemas podem não ter nenhuma interação ou ter interação direta, sendo impossível a distinção entre os componentes senão por sua funcionalidade. O documento Workflow Reference Model (1995) discute diferentes níveis de interação que serão descritos nessa seção. Os níveis referem-se a *workflow*s, portanto, a interação entre ferramentas que compõe um fluxo de trabalho.

O nível mais baixo, obviamente, é o de **nenhuma** interação, ou seja, não é possível fazer a comunicação entre os sistemas. É o caso de sistemas que não aceitam entradas ou saídas compatíveis, que sequer possam ser convertidas, e que não ofereçam nenhum outro mecanismo de interação.

Sistemas que funcionam como diferentes partes de um processo, necessitando de um agente humano para efetuar a interação entre eles, são considerados sistemas **coexistentes**. Esses não oferecem nenhum mecanismo nato ou externo para interação entre eles. Esse é o caso de sistemas que não são projetados para interagir e sequer apresentam formatos compatíveis na representação de dados, impossibilitando a troca de informações automatizada.

Um *gateway* é um mecanismo que permite que trabalhos sejam movidos entre os sistemas utilizados, podendo ser parte do sistema ou um mecanismo externo. Se os sistemas oferecem alguma ponte, ou seja, encapsulamento, tradução de formato ou protocolo ou um *gateway* que faça a entrega das operações e informações, ele efetua a interação classificada como interação de ***gateway* único**. Caso o *gateway* ofereça uma API padrão comum, a interação é então classificada como de ***gateway* com API comum**.

A utilização de um subconjunto de API comum compartilhada como forma de permitir acesso aos ambientes é considerada uma interação por **subconjunto de API comum limitada**. Através da API comum, são utilizados formatos públicos de dados de informação e transporte de dados. Assim, funções de tratamento de dados e transporte são

disponibilizadas pelo sistema através da API. A interação é feita diretamente entre os sistemas.

O próximo nível é o de interação por **API completa**. Esse se caracteriza por todas as ferramentas do fluxo de trabalho apresentarem uma única API padrão que dá acesso a todas às funcionalidades de gerenciamento do fluxo de trabalho.

Em termos de gerenciamento dos fluxos de trabalho existem informações de direitos e restrições de acesso que devem ser respeitadas por todas as ferramentas. A capacidade de troca dessas informações através de formatos compartilhados é classificada como interação por **formatos compartilhados**. Considera-se que existem formatos padrões para as estruturas de dados tratadas pelo fluxo. Quando toda a comunicação entre os sistemas cliente/servidor está padronizada considera-se uma interação com **compatibilidade de protocolo**.

Por fim, quando um sistema apresenta, além de todos os itens anteriores, a mesma forma de interação com o usuário, permitindo que o mesmo não perceba a diferença de uso entre os sistemas, considera-se que a interação é de **utilitários de aparência e comportamento comum**. Um exemplo desse nível de integração são as ferramentas do Microsoft Office.

Essa classificação é importante porque esclarece as formas de interoperabilidade que um fluxo de trabalho pode ter.

3.8.1 Categorias de Software

Uma aplicação real compreende não somente sua própria funcionalidade, mas também as funcionalidades providas pelas camadas inferiores de software. Essas camadas são constituídas por softwares que podem ser classificados em categorias. As categorias de software são discriminadas pelo seu objetivo e funcionalidades providas. Segundo Microsoft (2001), as categorias fundamentais são: gerenciamento, aplicações, dados e rede (figura 3.1). Em (SPILLER, 1997) é apresentada uma classificação semelhante (figura 3.2), onde os termos rede e gerência correspondem diretamente às categorias da definição dada pela Microsoft. Já as categorias de visualização, desenho e análise correspondem à soma das camadas de aplicação e dados.

Figura 3.1: Categorias segundo Microsoft.

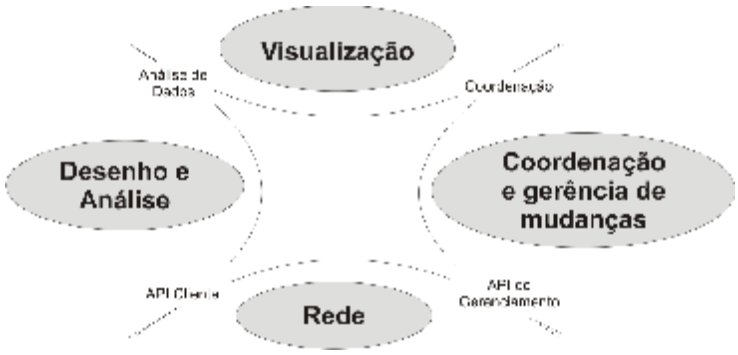

Figura 3.2: Categorias segundo Spiller.

A interoperabilidade de um sistema está geralmente vinculada a propriedades das camadas inferiores, que formam um ambiente de software que provê diversas funcionalidades. Por exemplo, se dois programas são construídos utilizando o mesmo

sistema de arquivos e eles utilizam arquivos como entradas e saídas, estes terão a capacidade herdada do ambiente de trocar informações através de arquivos. Portanto, o conhecimento dessas categorias e suas propriedades podem auxiliar a obtenção de interoperabilidade de forma simplificada, utilizando-se dos recursos já disponíveis ou ubíquos.

3.8.2 Categoria de Rede

A categoria de rede é fundamental para a interoperabilidade entre sistemas. A utilização de protocolos de rede é responsável por permitir a interação entre sistemas operacionais, máquinas e linguagens diferentes, entre outros. A padronização e disseminação dos protocolos de rede garantem a interoperabilidade de diversas plataformas como Unix, Apple Macintosh, Novell Netware, IBM Hosts e Microsoft. Acesso a terminais e serviços de impressão remotos são, também, componentes importantes da categoria de redes. Os mecanismos de interoperabilidade mais elaborados, que serão apresentados no final deste capítulo, utilizam-se de serviços de rede.

3.8.3 Categoria de Dados

A camada de dados é responsável por prover funções de acesso e requisição de informações em dispositivos de armazenamento. O mais antigo e comum sistema de armazenamento é o sistema de arquivos. A heterogeneidade de sistemas operacionais comumente encontrada na atualidade, implica a necessidade de armazenamento de dados distribuído em sistemas operacionais diversos, que utilizam sistemas de arquivos diversos. Para dar acesso aos arquivos remotamente e, possivelmente, utilizando um sistema de arquivos não nativo do sistema operacional utilizado, além do acesso local aos dados, os sistemas de arquivos têm a capacidade de transportar os dados através da rede.

Sistemas de bancos de dados são a base de muitas aplicações comerciais e acadêmicas. Além de permitir o simples acesso e busca de dados, eles oferecem propriedades importantes para a integridade dos mesmos. As propriedades ACID (*Atomicity, Consistency, Isolation and Durability*) são garantidas por modelos transacionais. A interoperabilidade oferecida por bancos de dados é fundamental para a confecção de sistemas distribuídos, que são frequentemente pré-requisitos dos sistemas atuais.

3.8.4 Categoria de Aplicação

A categoria de aplicação oferece uma infraestrutura básica para que novas aplicações desenvolvidas possam interoperar com aplicações existentes. A camada de apresentação de sistemas pode utilizar serviços como os providos pelas linguagens HTML e DHTML (*Dynamic* HTML). Essas linguagens são cada vez mais importantes, pois oferecem uma forma de manter a apresentação do software compatível e uniforme, através de funções de formatação e interação. Atualmente, pode-se encontrar HTML até mesmo em softwares que não foram projetados para operar na Internet. A capacidade de participar em transações (especialmente para bancos de dados) entre sistemas diversos é também um serviço que deve ser oferecido pela categoria de aplicação. Adicionalmente, a utilização da arquitetura de componentes reutilizáveis tem evoluído e ganhado importância para o desenvolvimento de aplicações. Essa arquitetura fornece uma infraestrutura básica através de componentes pré-desenvolvidos que oferecem funcionalidades diversas. Algumas tecnologias são utilizadas para oferecer interoperabilidade entre componentes heterogêneos, por exemplo, DCOM, CORBA, entre outros.

3.8.5 Categoria de Gerência

A categoria de gerência tem o intuito de diminuir a dificuldade de gerenciamento e administração de sistemas múltiplos, particularmente a administração de contas de usuários. O gerenciamento de um sistema compreende lidar com diversos tipos de usuários, que necessitam de recursos específicos disponíveis para eles. O correto acesso, ou seja, gerenciar permissões, assim, oferecendo segurança de dados, é um papel dessa categoria. O correto gerenciamento dos recursos disponíveis, assegurando o desempenho do sistema, gerenciamento de eventos e alertas são também fundamentais.

3.8.6 Utilização de Padrões

A utilização de padrões é essencial para a obtenção de interoperabilidade entre sistemas. A única forma de dois sistemas trocarem informações de forma correta e coerente é através de um formato comum e preestabelecido de troca, um padrão. Porém, padrões de fato são na verdade fruto da adoção desses em diversos sistemas, ou seja, uma definição de um formato, por si só, não pode ser considerada um padrão.

Outras importantes questões devem ser discutidas ao discutir padrões. Primeiramente, definições de padrões dificilmente são completas, geralmente não possuem todas as características ou informações importantes para determinada área. Por exemplo, o padrão HTML sofreu diversas extensões até hoje, e provavelmente ainda sofrerá, pois a evolução das tecnologias que o cercam influencia sua evolução. Ainda, padrões nem sempre oferecem soluções completas para um determinado nicho. Por exemplo, o LDAP (*Lightweight Directory Access Protocol*) (HOWES, 1998) define serviços de diretórios para clientes e pesquisas simples, mas não é otimizado para consultas de servidor, replicação e não define questões de segurança para os diretórios.

Mesmo assim, a utilização de padrões é a única solução para obtenção de interoperabilidade e é utilizada em todas as categorias citadas nessa seção. Padrões são utilizados para estabelecer as linguagens ou formatações comuns para qualquer tipo de interoperabilidade, ou seja, somente a capacidade de escrita e leitura de formatos preestabelecidos torna viável a troca de informações entre sistemas. Utilizam-se padrões em protocolos, linguagens, codificações, formatos de intercâmbio, resumindo é fundamental a adoção de padrões para obter interoperabilidade.

3.9 Cenários de aplicação de Interoperabilidade

A interação entre fluxos ou ambientes de trabalho pode ocorrer de diversas formas. Considerando apenas formas mais simples de interação, como proposto por WfMC (WORKFLOW, 2005), existem quatro cenários onde fluxos trocam itens de trabalho para interagir. Esses cenários auxiliam a compreensão dos requisitos diferentes em termos de mecanismos de sincronização e comunicação.

Primeiramente tem-se o modelo que consiste no encadeamento de processos (figura 3.3), onde um processo envia um item de trabalho (instância de processo ou atividade) para outro fluxo de trabalho. Após a passagem do item de trabalho, os fluxos não têm mais sincronização e continuam suas execuções paralelamente. Para possibilitar o uso desse modelo são necessários conversores para dados, mapeamento de processos e atividades, entre outros, ou é necessário existir uma API padrão entre os ambientes que assegure a correta troca de informações.

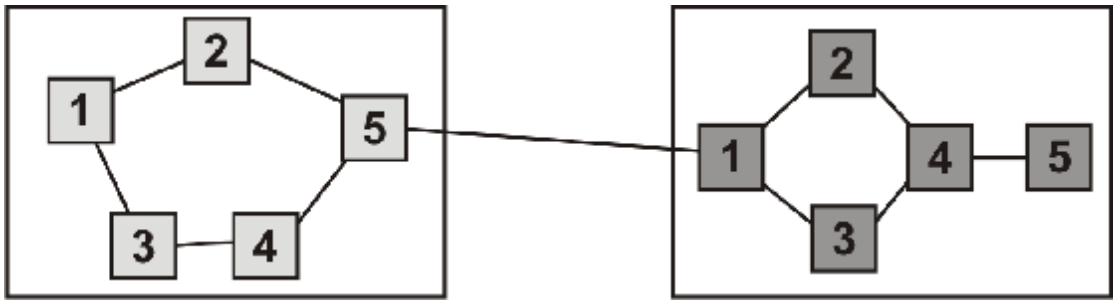

Figura 3.3: Encadeamento de processos.

O segundo cenário corresponde ao caso onde um fluxo é utilizado como um subprocesso de outro (figura 3.4). O *workflow* aninhado é encapsulado como uma única tarefa, onde todo o seu processamento é feito e, só então, o *workflow* chamador retoma a atividade. Existe também a possibilidade de recursão nesse cenário. Devido à relação entre os fluxos de trabalho existente nessa interação, o cenário é classificado como hierárquico ou de subprocesso aninhado.

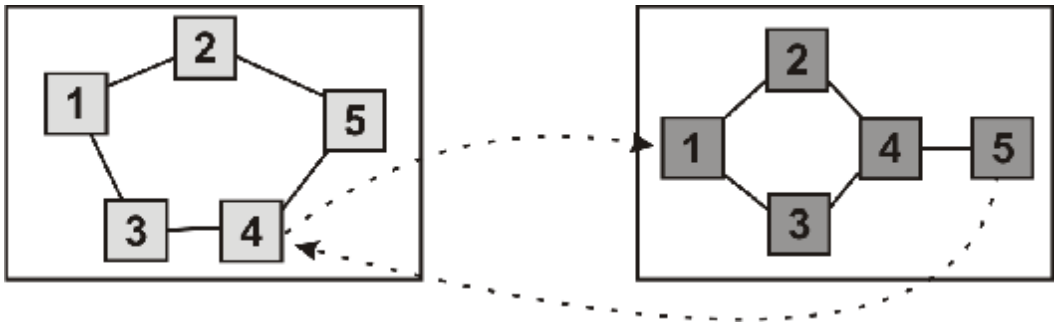

Figura 3.4: Processo atuando como sub-processo.

O cenário chamado de indiscreto conectado ou ponto-a-ponto caracteriza-se por permitir que se crie um ambiente completamente misto, ou seja, as atividades dos *workflow*s são utilizadas intercaladamente em qualquer ordem, tornando a percepção desses *workflow*s como se fosse um só, pois implementam um ambiente único compartilhado (figura 3.5). Para permitir a comunicação entre os processos de forma intercalada, esse cenário requer que os fluxos de trabalho tenham mecanismos de comunicação adequados, como API comum para comunicação e que ambos sejam capazes de interpretar os dados de forma única. No entanto, essa interação só é possível se as heterogeneidades forem tratadas corretamente, de forma a processar as informações corretamente a cada processo.

Por fim, o modelo de sincronização paralela permite que os processos operem independentemente, porém requer pontos de sincronização. Esse mecanismo é usado para facilitar agendamento de processos entre execuções paralelas, pontos de verificação (*checkpointing*) de dados para recuperação, ou transferência de dados relevantes do fluxo entre processos, por exemplo.

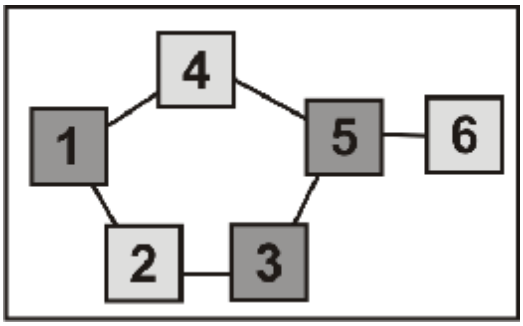

Figura 3.5: Ambiente misto.

3.10 Middleware

Segundo Emmerich (2000), *middleware* (figura 3.6) é uma camada de software que simplifica a interação entre aplicações. Mais especificamente, o termo refere-se às camadas básicas de rede e serviços de gerenciamento, segurança, acesso e troca de informações usadas para permitir que os usuários compartilhem recursos de forma transparente; oferecer ferramentas de colaboração e comunicação efetiva como tecnologias Grid (ABBAS, 2004) e outros serviços avançados; e desenvolver uma arquitetura e metodologia que pode ser estendida para mais usuários de rede local ou Internet.

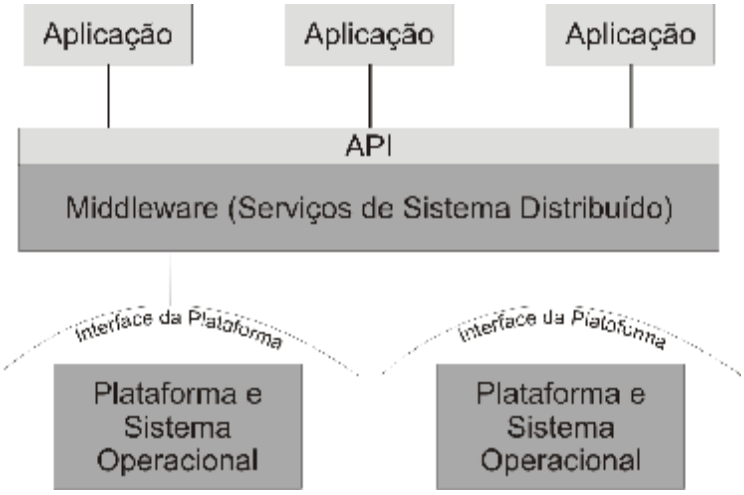

Figura 3.6: *Middleware* (BRAY, 1997)

O termo *middleware* (BRAY, 1997) é utilizado para representar protocolos de maior grau de abstração, residentes entre os serviços básicos de rede e as aplicações. Eles têm o objetivo de simplificar a programação das aplicações que necessitam trocar dados, fazendo isso através da disponibilização de serviços e funções em uma API de alto nível. Sem o auxilio de camadas intermediárias a implementação da comunicação entre softwares seria mais custosa em termos de tempo e recursos.

Assim, os detalhes de baixo nível de programação ficam escondidos como controle de concorrência, gerenciamento de transações, e comunicação em rede, permitindo aos desenvolvedores focarem na aplicação em desenvolvimento. *Middlewares*, portanto,

servem como instrumento de integração e facilitam a obtenção de interoperabilidade. Na verdade, muitos dos assuntos tratados anteriormente nesse capítulo dizem respeito a *middlewares* também.

Existem quatro tipos de *middleware* que oferecem serviços de interação de forma diferenciada. São eles os *middlewares* transacionais, os orientados a mensagens, os procedurais e os orientados a objetos e componentes.

Middlewares transacionais suportam a utilização de transações entre os sistemas envolvidos, lembrando que os sistemas podem estar distribuídos em diferentes máquinas de uma rede. Eles utilizam protocolos de execução em duas fases para implementar as transações distribuídas. Produtos como IBM CICS, BEA Tuxedo e Transarc Encina são exemplos dessa classe.

Middlewares orientados a mensagens ou (MOM) permitem a comunicação entre sistemas distribuídos através de trocas de mensagens. IBM MQSeries e Sun Java Message Queue são enquadrados como MOM.

Um dos mais difundidos sistemas de *middleware* procedural é o RPC da Sun Microsystems. Inicialmente foi lançado como parte do sistema operacional e depois foi submetido como um padrão para o X/Open consortium, que o adotou como parte do Distributed Computing Environment (DCE). Atualmente, o RPC está disponível na maioria dos sistemas Unix e também no Windows.

Acompanhando a evolução dos paradigmas de programação que passaram do modelo estruturado para o modelo de objetos, os *middlewares* orientados a objetos evoluíram do RPC. A idéia principal é justamente aproveitar-se dos princípios do modelo de objetos, como herança e referência de objetos, nos sistemas distribuídos. Essa é a classe de *middlewares* que atualmente é a mais difundida. CORBA, DCOM, RMI e J2EE são exemplos de produtos que oferecem suporte a componentes ou objetos de forma distribuída. Alguns desses produtos que são mais importantes para compreensão deste trabalho, serão descritos brevemente na seção 3.11.

3.11 Computação Ubíqua

Middleware é a evolução das tecnologias básicas de rede que permitem um grau de abstração maior para a programação de componentes que tenham interoperabilidade. Ou seja, o desenvolvimento de aplicações comunicantes é simplificado porque os detalhes de comunicação são tratados de forma transparente. A computação ubíqua ou pervasiva é, de certa forma, similar à evolução dos *middlewares*.

A idéia principal é de que a computação estará integrada no ambiente. A interação entre os dispositivos deve-se dar de forma transparente, adaptando-se ao ambiente em que os dispositivos estejam. Os dispositivos se adaptarão ao ambiente em que estiverem delegando computações, transmitindo informações e reconfigurando-se de acordo com os outros dispositivos presentes no ambiente. Apesar da concretização desses princípios ainda estar distante comercialmente, o impacto que essa tecnologia terá na interação entre homem e computador é de extrema importância. Por esse motivo esse tópico é citado no trabalho. A revolução provocada por esses conceitos pode modificar a forma de trabalho e as tecnologias de interoperação.

3.12 Tecnologias

A interoperabilidade entre sistemas é obtida através da utilização de mecanismos que implementam os princípios descritos nesse capítulo. Cada mecanismo apresenta características diferentes, resolvendo tipos de heterogeneidades diferentes, oferecendo um grau de abstração próprio e funcionalidades específicas. A seguir serão descritas algumas das mais importantes e difundidas tecnologias de interoperabilidade, sua origem, algumas características e sua importância.

- **RPC (Remote Procedure Call):** RPC é um protocolo para execução remota de funções. Ele permite que sejam chamadas funções remotas sem a necessidade de lidar com os protocolos de transporte de rede diretamente. As chamadas de função e seus parâmetros são codificadas e transportadas através de protocolos de transmissão de dados como TCP (Transmission Control Protocol) e UDP (User Datagram Protocol). RPC utiliza o modelo de cliente/servidor, onde o cliente é quem efetua as chamadas de função e o servidor quem as responde. As chamadas de funções ocorrem de forma bloqueante, ou seja, o cliente espera até que o servidor processe e devolva o resultado da chamada. RPC é baseado no modelo estruturado, portanto, um servidor pode ter apenas uma instância de uma dada função, e os parâmetros que podem ser passados são tipos simples como números e seqüências de caracteres. RPC é independente de plataforma de hardware e sistema operacional. Os sistemas operacionais que implementam RPC são o Unix e Windows.

- **CORBA (Common Object Request Broker):** Com o intuito de prover um *framework* para aplicações orientadas a objetos, a Object Management Group (OMG) criou a arquitetura CORBA. Ela utiliza uma linguagem intermediária para estabelecer a interface entre componentes, chamada Interface Definition Language (IDL). Serviços de diretórios, *naming*, de persistência de objetos e transações são parte da especificação da arquitetura. Deve-se, no entanto, salientar que CORBA é uma especificação e nem todos os serviços são oferecidos por todas as implementações dela. Existem implementações para diversas linguagens e plataformas (sistemas operacionais e hardware).

- **DCOM (Distributed Component Object Model):** O mecanismo chamado Component Object Model (COM), da Microsoft, é uma arquitetura que permite que aplicações tenham compatibilidade binária, ou seja, possam interoperar, independentemente de quais linguagens foram utilizadas para a construção dos componentes. Distributed COM (DCOM) estende COM, acrescentando a capacidade de interações remotas. A restrição de DCOM em comparação com seu rival CORBA é que este apenas está presente na família de sistemas operacionais da Microsoft (Windows).

- **DotNet:** Essa plataforma tem o intuito de solucionar os problemas de interoperabilidade na Internet. Ela ainda oferece mecanismos de integração com COM e DCOM, porém utiliza-se de um novo conceito de um ambiente de execução único independente de linguagem, o Common Language Runtime (CLR), que é a

evolução da família COM. O CLR possui uma linguagem intermediária (MSIL - Microsoft Intermediate Language) para a qual todas as linguagens são codificadas, permitindo que o código intermediário seja compatível entre elas. O CLR utiliza uma linguagem intermediária para prover instruções independentes de plataforma de hardware, de forma similar à Java Virtual Machine da Sun. Junto com o MSIL, são produzidos meta-dados que descrevem as funções e propriedades dos códigos gerados, eliminando a necessidade de type libraries (utilizados em COM) ou de uma IDL. Uma característica interessante do *framework* é a possibilidade de criação de uma classe em uma linguagem e herdá-la em outra, estendendo ou utilizando os métodos originais.Apesar de ter sido inicialmente desenvolvido somente para os sistemas operacionais Windows, o DotNet é um *framework* aberto, e já possui implementações para outros sistemas operacionais, como é o caso do projeto Mono (2005). Porém, atualmente, existem limitações de linguagens e plataformas. As linguagens disponíveis que já possuem implementação são JScript (GOODMAN, 2004), C++ (STROUSTRUP, 1997), C# (ROBINSON, 2004) e Visual Basic (VB) (DEITEL, 2003), e para Unix somente C#. C# é uma evolução do C++ que tem diversas características em comum com Java, porém apresenta uma sintaxe mais complexa. Outras implementações já existem para outras linguagens (BELL, 2002).

- **RMI (Remote Method Invocation)**: É um mecanismo nativo da linguagem Java que permite a invocação remota de métodos. Pode ser considerado uma implementação de RPC para o modelo de objetos específica da linguagem Java, ou uma versão simplificada de CORBA. Serviços transacionais e outras características não são providas pelo mecanismo, que foca principalmente na execução remota de métodos. Como no RPC, o mecanismo trabalha com a estrutura cliente/servidor, porém mais de uma instância de um objeto servidor pode ser criada em cada servidor. Objetos podem ser passados como parâmetros e seu código executável também pode ser transmitido. RMI permite a interação com CORBA, podendo inclusive utilizar o protocolo CORBA IIOP para efetuar as transmissões de dados.

- **J2EE (Java 2 Enterprise Edition):** O Java 2 Enterprise Edition (J2EE) oferece uma solução para interoperabilidade através de execução remota de métodos, trocas de mensagens, *naming*, busca de recursos e serviços, controle transacional e recursos de bancos de dados. Apesar de a solução utilizar apenas a linguagem Java, ela tem sido amplamente adotada para projetos de sistemas para Internet.

- **XML (Extensible Markup Language):** A Extensible Markup Language (XML) pode ser utilizada como um meio para se atingir a interoperabilidade (DAHALIN et al, 2002). Derivada de SGML (Standard Generalized Markup Language), XML provê um meio para representação de dados. A idéia central da linguagem é prover flexibilidade, para que seja possível representar, verificar e validar dados. XML permite a criação de regras similares ao formato BNF (Backus-Naur Form) para a verificação e validação de documentos através de mecanismos como XML Schema e DTDs (Document Type Definition). Esses mecanismos utilizam-se da formatação XML para a descrição de seus elementos. Além disso, linguagens de transformação (XSLT - Extensible Stylesheet Language Transformations) permitem a conversão de

documentos XML em outros formatos como texto simples, HTML e PDF (Portable Document Format). A utilização de XSLT permite, por exemplo, que sejam separados elementos de dados e formatação, ou outras informações, como utilizado para CSS (Cascading Style Sheets), gerando automaticamente documentos com dados formatados a partir de definições separadas. A localização e referência de elementos dos documentos é facilitada por um mecanismo chamado XPath (XML Path Language). Processadores para a linguagem, que permitem a leitura do formato de forma simplificada, estão presentes em diversas linguagens, inclusive sendo distribuídos como bibliotecas juntamente aos sistemas operacionais. Apesar de não ser uma solução de interoperabilidade, XML é utilizado como tecnologia de base em soluções como *Web Services*. Em resumo, XML define um mecanismo para definição e organização de dados, que pode ser utilizado para codificar formatos intermediários.

- **Web Services:** Segundo (SLEEPER, 2001) são uma forma de interoperação entre sistemas através da Internet, utilizando protocolos simples (p. ex. HTTP). *Web Services* são formados por três outros protocolos. Eles são: SOAP (W3C, 2003) (Simple Object Access Protocol), WSDL (Web Services Description Language) e UDDI (Universal Discovery, Description, and Integration), que são respectivamente, um protocolo para representação e acesso a objetos, um protocolo de descrição de serviços, e um protocolo de descoberta de serviços. Esses são mantidos por grandes empresas de software como Microsoft e Sun Microsystems (2005), o que demonstra a importância que tal solução tem recebido, como citado por Kilgore (2002).

3.13 Algumas considerações

As evoluções tecnológicas especialmente da área de redes de computadores e tecnologias para Internet têm permitido diferentes formas de interoperabilidade entre sistemas. A microeletrônica sofre o impacto dessas formas de interoperabilidade nos processos de desenvolvimento e na necessidade de produção de equipamentos que respeitem os padrões de interoperação. A interação entre CADs para microeletrônica pode interferir diretamente na produtividade desses, podendo determinar o sucesso ou fracasso de novas ferramentas.

Para compreender melhor o que interoperabilidade significa, foram apresentadas causas, critérios de avaliação, características, níveis, categorias de software, cenários e tecnologias que lidam com interoperabilidade. A compreensão da interoperabilidade passa pelos motivos que geram diferenças entre softwares, que por fim, impedem a troca de informações entre eles. Segundo Young (2003), considerando heterogeneidade representacional, existem três grandes causas: perspectivas diferentes, construções equivalentes e especificações incompatíveis.

Critérios de avaliação são expostos para que se possa comparar as soluções existentes e a solução proposta neste trabalho. Entre eles estão: os tipos de heterogeneidade; a utilização de sistemas automatizados para auxiliar a correlação entre entidades modeladas; a necessidade de conhecimento prévio dos serviços disponibilizados por um sistema; a quantidade de modificações necessárias para que um sistema interopere com outros; a quantidade de conversões entre os protocolos ou padrões; a existência de mecanismos que

auxiliem a criação automatizada de tradutores; e finalmente, a possibilidade de execução conjunta de tarefas.

As características que sistemas interoperantes podem ou devem apresentar são escalabilidade, disponibilidade, adaptabilidade, robustez e, para possibilitar sua utilização, um custo-benefício aceitável. Visto que existem diferentes necessidades, foram apresentados diferentes níveis de interoperação, que vão desde a não interação, até a utilização de elementos similares de interface gráfica de usuário.

A categorização de software pode ser utilizada para mapear os diferentes focos e assim, determinar as características de interoperabilidade necessárias para um software de acordo com seu foco. As categorias apresentadas são: rede, dados, aplicação e gerenciamento. A categoria de rede se destaca por ser a base para a interoperação entre sistemas remotos. A utilização de padrões é comum em todas as categorias, pois, só através de padrões sistemas desenvolvidos separadamente, podem trocar informações de forma correta.

Três cenários foram apresentados para ilustrar as situações onde ocorre interação entre ferramentas. São eles: o encadeado, o hierárquico, o ponto-a-ponto e o sincronizado paralelamente.

O termo *middleware* está diretamente vinculado com interoperação porque é utilizado para descrever os serviços de rede de maior grau de abstração, assim oferecendo facilitadores para a obtenção de interoperabilidade. A computação pervasiva foi brevemente introduzida, pois é uma área que pesquisa intensamente formas de interoperabilidade.

E, por fim, algumas tecnologias de importância relevante para este trabalho foram apresentadas.

4 UMA PROPOSTA PARA A INTERAÇÃO ENTRE AMBIENTES E FERRAMENTAS INDEPENDENTES

4.1 Introdução

A busca constante por otimizações, adaptações que suportem as novas tecnologias, ou mesmo novas técnicas de automação, impulsionam a criação de novas ferramentas desenvolvidas nos mais variados ambientes de CAD para microeletrônica. Tanto empresas quanto universidades buscam desenvolver tecnologias revolucionárias e inovadoras para obter melhores resultados em termos de tempo de confecção, eficiência em consumo, maior velocidade de funcionamento, entre outros fatores. Esses esforços muitas vezes geram ferramentas independentes, pois não são projetadas para funcionar em conjunto, que apresentam excelentes resultados em suas tarefas específicas. Assim, diferentemente dos *frameworks*, essas ferramentas não lidam com todo o espectro que é parte de um projeto completo de um chip, mas tratam de problemas específicos de forma superior aos *frameworks*.

A interoperação entre ferramentas pode trazer inúmeros benefícios, aumentando a automação dos processos e por conseqüência diminuindo a intervenção humana em processos repetitivos, e ainda, essa automatização da interação entre ferramentas permite com que se tornem possíveis melhoramentos importantes que aumentam a qualidade do processo. Segundo Goldman (2000), as tarefas de conversão de formatos de uma ferramenta pra outra são propícias a erros, ou seja, beneficiam-se bastante da integração automatizada das ferramentas na medida em que se retira do usuário a tarefa enfadonha e demorada de converter os dados. A interoperação entre ferramentas pode ainda simplificar a adaptação a projetos novos com novas requisições de ferramentas ou mesmo a substituição das ferramentas por outras (decisão tomada por motivos financeiros, temporais, etc).

Motivado pela potencialidade da interação entre ferramentas de CAD para microeletrônica, este trabalho visa apresentar uma solução para a comunicação de ferramentas e *frameworks*. Essa interação dá aos *frameworks* a capacidade de qualificar ainda mais os projetos nele desenvolvidos, e inclui as ferramentas independentes no fluxo de desenvolvimento dos *frameworks*, suprindo as tarefas que não são contempladas por elas.

4.2 Estado da Arte

Como conseqüência da evolução das tecnológicas de redes e as grandes mudanças que elas acarretam, a quantidade de pesquisas feitas na área tem aumentado, juntamente com o surgimento de novos tópicos. Congressos que tratam somente de *middleware* e computação pervasiva foram criados para suprir um espaço adequado para a discussão desses tópicos. A interoperabilidade é um assunto fundamental para a evolução dessas tecnologias, assim como para a evolução dos sistemas automatizados de outras áreas, como a microeletrônica.

Nos últimos anos, diversos pesquisadores da microeletrônica têm dedicado seu tempo para investigar as novidades tecnológicas da área de redes, mais especificamente da Internet, visando obter novas formas de interação entre sistemas através do uso de tais novidades. Muitos trabalhos, como por exemplo (WIRTHLIN, 2002), (FIN, 2000) e (DALPASSO, 1999), visam abordar formas de distribuição e uso de propriedades intelectuais de forma segura e eficiente através da Internet. Outros trabalhos tratam da interoperabilidade de ferramentas legadas, ou que não apresentam meios de comunicação, com *frameworks* ou outras ferramentas. Esses são os dois principais focos de pesquisa na área de interoperabilidade em microeletrônica atualmente encontrados nos principais congressos (SUTTON, 1998) (KAZMIERSKI, 2002) (RETTBERG, 2002) (BRGLEZ, 2001). Além das pesquisas acadêmicas, propostas de mecanismos de integração comerciais e confeccionadas por coalizões são também componentes do cenário atual dessa área de pesquisa (OPENACCESS, 2005) (WORKFLOW, 2005) (ECSI, 2005).

Entre os trabalhos que mais se identificam com a proposta apresentada a seguir, está o de Sutton e Director (1998, que propõe a integração de ferramentas, criando invólucros para *frameworks*. Dessa forma, é possível que os dados de projeto, serviços e ferramentas do fluxo de trabalho sejam utilizados por um gerenciador de processos único. Invólucros são comumente utilizados como forma de implementação de encapsulamento de ferramentas. A idéia central desse trabalho é diminuir as dificuldades geradas pela utilização de diferentes *frameworks* e ferramentas, unificando o gerenciamento de dados e o fluxo de trabalho externamente aos *frameworks*. Assim, o fluxo de trabalho proposto internamente aos *frameworks* é substituído pelo proposto pela ferramenta gerenciadora, no caso, pela ferramenta chamada Minerva II.

Kazmierski e Clayton (2002) propõem um modelo onde um servidor, enquanto serve clientes, atua como cliente de servidores de ferramentas. A técnica tira proveito de tecnologias Java e RMI para prover um *framework* de projetos para microeletrônica baseados em tecnologias Web. Aplicações desenvolvidas em Java encapsulam outras ferramentas, trocando dados com a ferramenta através de entradas e saídas. Essas aplicações fornecem então um comportamento fortemente acoplado com o *framework* proposto.

A ferramenta OmniFlow (BRGLEZ, 2001) faz invocação remota de outras ferramentas, tanto como caixas brancas quanto como pretas. O trabalho é baseado em princípios de fluxos orientados a tarefas, mesclados com conceitos de programação estruturada, descrições de hardware e linguagens de marcação. XML é utilizado para configurar o controle de tarefas como sincronização, execução, repetição e finalização. O projeto oferece a possibilidade de criação de fluxos de projeto (orientados a tarefas) fora de *frameworks*, utilizando uma interface gráfica amigável para configurá-los.

Rettberg (2002) utiliza-se de tecnologias de serviços Web para propiciar um sistema de desenvolvimento para hardware e software embarcados. Este trabalho também aplica

encapsulamento, porém as ferramentas são encapsuladas como serviços Web. A associação ECSI (2005) também lida com sistemas embarcados, visando definir padrões de comunicação entre sistemas de forma a uniformizar e simplificar a comunicação de sistemas embarcados.

Um dos padrões de integração disponível é o OpenAccess (2005). Ele é uma API (Application Programming Interface) disponível apenas para a linguagem C++ que permite integração de ferramentas. Ele baseia-se em uma série de modelos de dados predefinidos e em um banco de dados referencial (que pode ou não ser utilizado). Esse padrão é definido por um consórcio que envolve diversas importantes empresas da área. Além dos modelos de dados, o padrão também estabelece métodos de troca de mensagens entre ferramentas. Apesar da grande importância desse padrão para a indústria de ferramentas de EDA, a dificuldade de aprendizado da ferramenta e de adaptação à mesma é alta, reduzindo a adesão de pesquisas acadêmicas a ele (KAHNG, 2003).

A coligação Workflow Management Coallition caracteriza e define diversos itens que compreendem o compartilhamento de artefatos de trabalho através de um fluxo de trabalho (WORKFLOW, 2005). Em (WORKFLOW, 2001) são detalhados termos e características importantes para a interoperação de ferramentas.

4.3 Tecnologias adotadas

A proposta apresentada neste trabalho tem forte inspiração nas tecnologias chamadas de *Web Services*. Os protocolos mais importantes para a compreensão do trabalho serão apresentados brevemente nessa seção. *Web Services* compreendem três protocolos que são SOAP, WSDL e UDDI, que são respectivamente protocolos para representação de objetos, descrição de serviços e catalogação de serviços. Todos estes protocolos são baseados em XML para a descrição de artefatos e troca de informações. Entre eles, o protocolo mais diretamente relacionado com o trabalho é o SOAP.

4.3.1 XML

Primeiramente XML será descrito, pois este é utilizado como base para os demais protocolos compreendidos por *Web Services*. XML tem merecido destaque por sua grande flexibilidade. Atualmente esta é uma tecnologia suportada por diversas linguagens de programação e sistemas operacionais. Ele é uma linguagem de marcação extensível baseada em SGML (Standard Generalized Markup Language). Ela pode ser estendida, diferentemente de HTML, pois seus elementos não são fixos, somente suas marcações. Um documento XML é considerado válido se respeita os marcadores de início e fim de elementos, atributos, etc, não possuindo erros de construção. Dois mecanismos, XML Schema e DTD (Document Type Definition), são utilizados para auxiliar a validação de documentos XML quanto ao seu conteúdo. Através desses é possível definir regras BNF e validar conteúdos e formatos de dados (inteiros, boleanos, telefones com hífen, etc).

Atualmente, diversos sistemas utilizam-se do XML como base para representar seus dados. Utilizando-se da versatilidade de XML, XSLT (Extensible Stylesheet Language Transformations), uma linguagem de transformação de XML, é empregada para transformar um documento XML para outro formato qualquer (PDF, HTML, e até mesmo outro XML). A variante de XSLT que é utilizada para a geração de HTML é chamada CSS (Cascading Style Sheets). Através dela é possível separar a formatação dos documentos do seu conteúdo, assim permitindo melhor organização das informações. Outro uso de XSLT é

a geração automática de códigos, que tem sido utilizada em sistemas que possuem construções repetitivas e previsíveis como, por exemplo, a representação de entes de bancos de dados através de estruturas em determinada linguagem, ou a criação de esqueletos de *Web Services* através de seus descritores (WSDL).

4.3.2 SOAP, WSDL e UDDI

O protocolo SOAP (Simple Object Access Protocol) utiliza-se da linguagem de marcação XML para representar objetos em formato texto. SOAP permite a representação de estruturas com seus elementos e atributos. É também possível descrever requisições de serviços, respostas às requisições e erros, bem como atributos especiais contidos no cabeçalho.

Utilizando-se dos mecanismos que XML possui para validação (ou verificação) dos conteúdos de documentos, SOAP permite a pré-definição dos formatos das mensagens através de XML Schemas. Assim torna-se possível preestabelecer a estrutura de elementos, quantidade de repetições mínima e máxima, tipos de dados aceitos e parâmetros necessários para cada serviço. Os tipos de dados predefinidos são inteiro, inteiro negativo, números com ponto flutuante, cadeias de caracteres, porém podem-se utilizar outros tipos complexos definidos pelo usuário como enumerações, matrizes de bytes, matrizes de outros elementos e estruturas. Matrizes esparsas e mascaras de validação para valores são contempladas pelo protocolo. SOAP permite que atributos tenham tipos associados e atributos polimórficos, ou seja, sem tipo associado em sua definição.

Para exemplificar alguns dos conceitos citados a figura 4.1 mostra um exemplo de definição de elementos e a figura 4.2 mostra uma mensagem correspondente à estrutura definida.

```
<element name="Book" type="tns:Book"/>
<complexType name="Book">
  <!-- Either the following group must occur or else the
       href attribute must appear, but not both. -->
  <sequence minOccurs="0" maxOccurs="1">
     <element name="title" type="xsd:string"/>
     <element name="firstauthor" type="tns:Person"/>
     <element name="secondauthor" type="tns:Person"/>
  </sequence>
  <attribute name="href" type="uriReference"/>
  <attribute name="id" type="ID"/>
  <anyAttribute namespace="##other"/>
</complexType>

<element name="Person" base="tns:Person"/>
<complexType name="Person">
  <!-- Either the following group must occur or else the
       href attribute must appear, but not both. -->
  <sequence minOccurs="0" maxOccurs="1">
     <element name="name" type="xsd:string"/>
     <element name="address" type="tns:Address"/>
  </sequence>
  <attribute name="href" type="uriReference"/>
  <attribute name="id" type="ID"/>
  <anyAttribute namespace="##other"/>
</complexType>
```

Figura 4.1: Definições no formato SOAP.

MENSAGEN SOAP

```
<e:Book>
  <title>My Life and Work</title>
  <firstauthor href="#Person-1"/>
  <secondauthor href="#Person-2"/>
</e:Book>
<e:Person id="Person-1">
  <name>Henry Ford</name>
  <address xsi:type="m:Electronic-address">
    <email>mailto:henryford@hotmail.com</email>
    <web>http://www.henryford.com</web>
  </address>
</e:Person>
<e:Person id="Person-2">
  <name>Samuel Crowther</name>
  <address xsi:type="n:Street-address">
    <street>Martin Luther King Rd</street>
    <city>Raleigh</city>
    <state>North Carolina</state>
  </address>
</e:Person>
```

Figura 4.2: Mensagem no formato SOAP.

SOAP foi projetado para ser transportado através de protocolos simples de comunicação: todos os dados são descritos como cadeias de caracteres (*strings*). Algumas informações são obrigatórias nas mensagens para o correto funcionamento. O acesso correto aos serviços remotos só é possível através do uso de endereçamento e outras meta informações nas mensagens. As figuras 4.3 e 4.4 mostram a estrutura de uma mensagem SOAP e um exemplo de mensagem SOAP/HTML de requisição respectivamente. Na figura 4.4 podemos observar a presença de diversas URIs (Uniform Resource Identifiers). Primeiramente observamos o cabeçalho HTTP, que vai até a linha que inicia por "*SOAPAction*". O conteúdo SOAP de fato inicia com o elemento XML "*SOAP-ENV*", que contem três URIs: a referente à definição do elemento envelope, a referente à codificação SOAP, e por fim, a referente à definição dos elementos definidos pelo serviço específico.

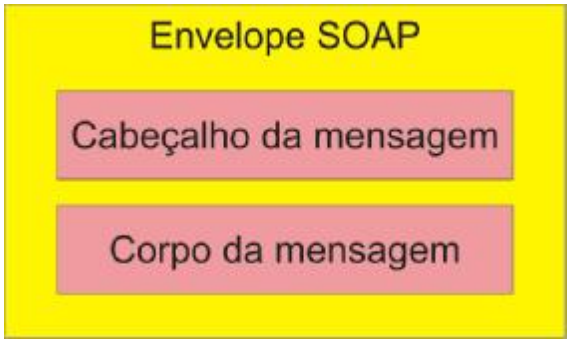

Figura 4.3: Estrutura de uma mensagem SOAP.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:GetLastTradePrice xmlns:m="Some-URI">
         <symbol>DIS</symbol>
      </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 4.4: Mensagem SOAP/HTTP.

A simplicidade de sua formatação, aliada à flexibilidade e disponibilidade do XML, permite que o protocolo seja utilizado para transportar objetos entre sistemas heterogêneos. Uma característica importante é que a utilização protocolos simples como HTTP facilita a transposição de *firewalls*. Um ponto negativo do protocolo é o custo computacional da codificação e decodificação XML, e a sua ineficiência em termos de consumo de espaço.

SOAP, porém, é apenas um dos protocolos utilizado em *Web Services*. Segundo Kilgore (2002), *Web Services* é o nome dado para a utilização conjunta de dos protocolos SOAP, WSDL e UDDI, exemplificado na figura 4.5. WSDL é um protocolo de descrição de serviços, contendo diversas informações sobre ele, tais como: onde ele será provido, quais são os parâmetros de entrada e saída, que protocolo de codificação será usado, etc. UDDI é responsável por catalogar as descrições de serviços (WSDL) de forma a responder a consultas sobre os serviços, distribuindo as descrições WSDL dos mesmos.

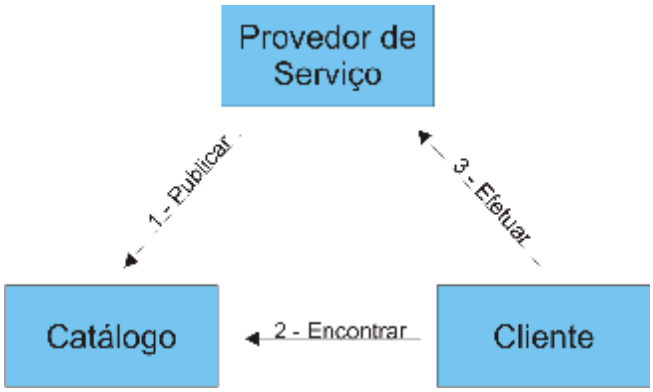

Figura 4.5: *Web Services*.

A utilização conjunta desses três protocolos permite que a chamada de serviços seja feita sem o conhecimento prévio dos serviços e formatação das mensagens. Buscando-se por um

serviço para determinada função ou classificação, obtêm-se um descritor, que dará todas as informações necessárias para construir mensagens de requisições e enviá-las, e enfim permitindo através de SOAP utilizar o serviço de maneira predefinida.

A tecnologia de *Web Services* facilita a integração e interoperabilidade entre sistemas utilizando protocolos e tecnologias que atualmente são padrões universais, utilizados na maioria dos sistemas operacionais e linguagens de programação. Ela também torna possível a reutilização de código, até mesmo através da Internet.

4.4 Protocolo Proposto

Baseado nas tecnologias *Web Services*, mais especificamente em SOAP, é apresentada a seguir uma proposta para facilitar a interoperabilidade entre ferramentas de CAD para microeletrônica independentes e *frameworks*, denominada BICO (Basic Interoperability through Components and Objects). A proposta é composta de uma definição de protocolo. A estrutura de seu codificador/decodificador, ferramentas auxiliares para simplificar a utilização do protocolo proposto e mecanismos auxiliares para o transporte das mensagens são apresentados no próximo capítulo. Esse conjunto de ferramentas permite o transporte de estruturas complexas entre os *frameworks* e ferramentas independentes, diminuindo o esforço necessário para tanto.

De modo similar ao descritor de objetos que trafegam em mensagens SOAP, BICO define estruturas para codificação, transporte e decodificação de informações. Uma estrutura corresponde a uma descrição composta de atributos e um identificador, como visto na figura 4.6.

```
<complexType name="NOME_DA_ESTRUTURA"
             package="PACOTE_DA_ESTRUTURA">
   <element name="NOME_DE_ATRIBUTO"
            type="TIPO_DE_ATRIBUTO"/>
        .
        .
        .
</complexType>
```

Figura 4.6: Estrutura BICO

O protocolo descrito neste capítulo é inspirado nas capacidades de representação e descrição de objetos que SOAP possui. O protocolo não tem o objetivo de apresentar todas as características de SOAP, devido às restrições apresentadas pelas linguagens *script* que são alvo deste trabalho, detendo-se ao que é necessário para transportar estruturas entre as linguagens alvo, assim facilitando a implementação do mesmo. Porém, para manter o máximo possível de compatibilidade com SOAP, o menor número possível de alterações será feito.

4.4.1 Funcionamento Básico

Através de requisições e respostas de serviços, as estruturas podem ser migradas de um sistema para outro. Representações de erros também são contempladas pelo protocolo, que, ao detectar um erro, é capaz de capturar uma mensagem e transmití-la ao requerente do serviço. Dessa forma o requerente de um serviço receberá uma mensagem clara sobre a situação que causou erro.

Como anteriormente mencionado, os serviços também são descritos em XML. Assim como as definições de estruturas, as descrições dos serviços também são transformadas, gerando um esqueleto dos códigos fonte nas linguagens desejadas, onde um método correspondente à execução do serviço deve ser implementado. O objetivo dessas gerações automáticas de código é diminuir ao máximo a necessidade de interação do usuário com o sistema, para que o usuário foque seu trabalho na interpretação dos dados, e não na codificação dos mesmos.

XML possui codificadores e decodificadores em diversas linguagens assim como nas linguagens Java, C e C++, porém isso não é verdade quanto às linguagens *script*. Por esse motivo foi necessário projetar codificadores e decodificadores para o subset de XML utilizado nesta proposta. Analisadores (*parsers*) completos XML com seus adendos XSLT, DTD e XML Schema são bastante complexos por si só para serem implementados em linguagens *script*. O CODEC, portanto, deverá ser implementado em todas as linguagens que precisarem ter suporte ao protocolo BICO, sejam elas *script* ou de propósito geral, incluindo o parser XML se necessário.

4.4.2 Artefatos e Geração Automática de Código

É necessário implementar uma representação das estruturas em cada linguagem alvo, bem como, é necessário implementar o serviço que se quer disponibilizar. É possível gerar automaticamente o código que implementa as estruturas e um esqueleto do serviço através de uma ferramenta auxiliar que utiliza uma descrição XML do serviço ou estrutura como entrada (figura 4.7). Dessa forma, o programador que desejar utilizar o protocolo poderá simplesmente descrever as estruturas e serviços, e após implementar o serviço, diminuindo consideravelmente o trabalho necessário para adaptar seu programa ao protocolo.

A implementação de um único método é necessária para concluir a construção de um serviço. A utilização desse mecanismo de geração do código esqueleto dos serviços não é obrigatória, mas facilita e garante o correto funcionamento do CODEC do protocolo.

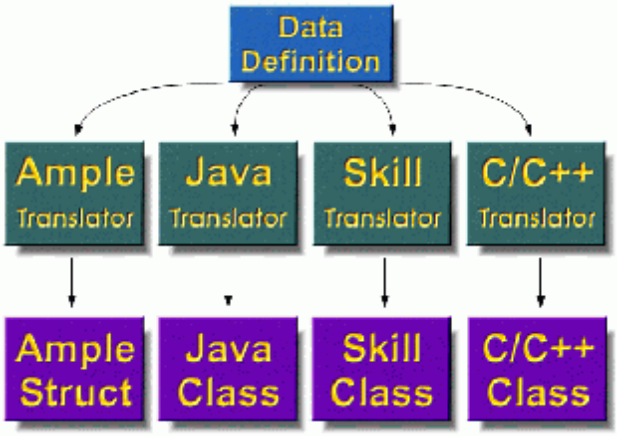

Figura 4.7: Geração automática de código para estruturas e serviços.

A descrição XML de um serviço contém o nome do serviço, o tipo da mensagem de entrada e o tipo de saída. A figura 4.8 é um exemplo de descrição de serviço do protocolo proposto. Uma restrição do protocolo é que as mensagens de entrada e saída são compostas apenas de um objeto. Essa restrição não é severa porque esse objeto pode ser uma estrutura contendo valores simples e outras estruturas, assim formando entradas e saídas com mais de

um valor. A figura define um tipo de serviço chamado CDFGetCIFLayers, que contem mensagens de entrada e saída do tipo CDFGetCIFLayersRequest e CDFGetCIFLayersResponse.

```
<service name="CDFGetCIFLayers">
  <inputMessage type="CDFGetCIFLayersRequest"/>
  <outputMessage type="CDFGetCIFLayersResponse"/>
</service>
```

Figura 4.8: Descrição de um serviço BICO.

Os parâmetros de entrada e saída devem ser definidos como estruturas de dados. As definições de serviços e estruturas podem ser comparadas a IDL (*Intermediate Definition Language*) CORBA, que é uma linguagem intermediaria utilizada para descrever os atributos e seus tipos que comporão as classes e funções. Porém, diferentemente de CORBA as descrições XML utilizadas aqui não permitem que sejam descritas assinaturas de métodos ou funções. Novas funções podem ser adicionadas ao código gerado, desde que o comportamento básico implementado pelo código automaticamente gerado não seja alterado.

As estruturas de dados que trafegarão pelo protocolo são descritas em formato similar aos elementos *complexType* de SOAP. A figura 4.3 mostra um exemplo de definição de estrutura. Uma estrutura tem nome, pacote e atributos.

```
<complexType name="CDFCellGenerationRequest"
        package="lagartools.cdfserver">
  <element name="techId" type="string"/>
  <element name="templateId" type="string"/>
  <element name="logicFunction" type="string"/>
  <element name="generateOptimizedLogicFunction" type="boolean"/>
  <element name="generateNetlist" type="boolean"/>
  <element name="generateOrderedNetlist" type="boolean"/>
  <element name="generateStick" type="boolean"/>
  <element name="generateUnroutedStick" type="boolean"/>
  <element name="generateLayout" type="boolean"/>
  <element name="generateUnroutedLayout" type="boolean"/>
  <element name="logicFamily" type="string"/>
  <element name="diffusion" type="string"/>
  <element name="optimizationMethod" type="string"/>
</complexType>
```

Figura 4.9: Exemplo de descrição de uma estrutura BICO.

O pacote é utilizado com fins organizacionais apenas. Os atributos compõem o conteúdo real do objeto. Um atributo pode conter valores simples como números inteiros, números de ponto flutuante, *strings* e *booleanos*. Um atributo pode também conter valores como matrizes, referências a estruturas ou um valor de qualquer tipo destes, não determinado na inicialização. As matrizes devem ter o número de dimensões determinado na definição (ou polimórfico), porém o tamanho de cada dimensão deve ser dado durante a execução. A utilização de referências a estruturas permite a criação de listas encadeadas, grafos, etc, facilitando assim a representação de estruturas variadas. As estruturas aninhadas são serializadas através das suas referências utilizando uma técnica conhecida como *flattening*, de forma a permitir a remontagem das estruturas. Ou seja, uma estrutura que referencie outra estrutura é serializada de forma não aninhada, sendo a estrutura referenciada serializada anterior ou posteriormente. A estrutura principal conterá apenas uma referência ao identificador da estrutura referenciada. A utilização de tipos não determinados

(polimórficos) permite a utilização de um atributo como receptáculo de diferentes tipos, simplificando situações onde um serviço pode receber ou devolver mais de um tipo através de um mesmo parâmetro. Esses atributos deverão ter seu tipo determinado nas mensagens que trafegarão, ou seja, ao atribuir um valor.

O protocolo BICO não implementa alguns elementos estruturais de SOAP, a saber: *envelope*, *header*, *body* e *fault*. Esses elementos encapsulam uma ou mais requisições ou respostas e também servem para armazenar meta informações sobre o conteúdo da mensagem. O atributo *MUST UNDERSTAND*, enumerações e seqüências não são suportadas. Com enumerações é possível definir novos tipos para uso como valores. Seqüências são utilizadas para forçar a ordem dos elementos internos de uma estrutura. Ainda existem em SOAP atributos que podem ser utilizados para obrigar um número mínimo e máximo de ocorrências de sub-elementos, que também não está presente na versão 1.0 do protocolo proposto. Apesar dessas restrições, é possível representar tipos estruturados, matrizes e referências a outros tipos estruturados, o que é suficiente para representar estruturas complexas como grafos.

Considerando que cada serviço deve seguir uma estrutura predefinida conforme a linguagem, apenas é necessária a implementação do método correspondente ao serviço, que possui uma assinatura padrão. Esse método deve interpretar corretamente os dados de entrada e, após efetuar o processamento adequado, criar uma estrutura correspondente ao retorno. A vinculação do serviço a um servidor é feita através de um registro no servidor. Ao decodificar uma mensagem, o CODEC identifica o nome do serviço requerido, caso o nome do serviço seja encontrado, ele redireciona o objeto de entrada para o serviço registrado. Como a figura 4.10 mostra, o objeto principal e o nome do serviço requerido são fornecidos na mensagem de requisição. A mensagem também contém o objeto de entrada e os objetos referenciados por ele. O CODEC é capaz de lidar com entradas e saídas vazias, ou seja, sem objetos.

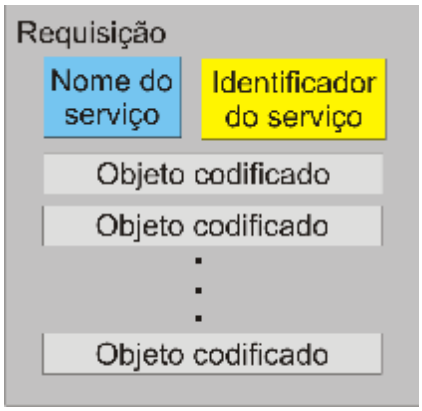

Figura 4.10: Estrutura de uma requisição BICO.

Um elemento de requisição ou resposta pode conter diversas estruturas complexas que formam juntamente com o elemento principal, a requisição ou resposta completa. Por exemplo, um grafo conexo pode ser passado e somente um nodo dado como identificador, porém todos referenciados, direta ou indiretamente, devem ser codificados.

Cada instância criada no cliente ou servidor é mantida em memória até que ocorra o termino da seção. Isso é feito através de um identificador único que cada estrutura recebe ao ser inicializada. Dessa forma é possível referenciar objetos criados ou recebidos anteriormente. Os objetos permanecem em memória pela duração de uma seção, que corresponde à duração da conexão entre o cliente e servidor. O controle de seção é feito modularmente, permitindo que futuras extensões do protocolo adotem outras políticas.

Strings devem ser codificadas de forma a não conterem marcações XML, para tanto, caracteres de escape e seqüências especiais são utilizadas na codificação e decodificação. Por exemplo, o texto "<texto></texto>" seria codificado da seguinte forma: "`<campo>&lt;texto&gt;&lt;/texto&gt;</campo>`".

Para permitir a evolução do protocolo, é importante estabelecer maneiras de recuperar o número da versão do protocolo, garantindo que alterações possam ser feitas no protocolo e seja possível determinar se um par servidor-cliente pode se comunicar. A versão é obtida através de um serviço padrão chamado "ServerInfo", que retornará uma lista de propriedades do servidor, podendo conter diversas informações sobre o servidor além do número de versão. As evoluções e diferenças entre versões, assim, poderão ser tratadas de maneira polida e eficiente.

Outra característica importante do protocolo é a representação de erros. Em caso de falha de qualquer natureza, desde que o funcionamento do protocolo não seja alterado com a falha, todo serviço pode retornar uma estrutura de erro correspondendo ao ocorrido. Para linguagens que suportam tratamento de erros isso é automaticamente controlado pelo CODEC. Assim o cliente recebe uma descrição clara do erro, tornando-se capaz de decidir a ação a tomar de acordo com o ocorrido.

A utilização de elementos de segurança em protocolos de comunicação tem importância relevante nos dias de hoje. Porém a proposta não contempla quesitos de segurança, deixando a cargo de protocolos de mais baixo nível, ou ao usuário do protocolo proposto, a utilização de autenticação, criptografia ou outros mecanismos.

4.5 Conclusão

A proposta aqui apresentada consiste de um protocolo que possui capacidade de representação de requisições e respostas a serviços, formadas por estruturas complexas.

Em resumo, a proposta contempla os itens abaixo:

- Representar estruturas compostas de atributos de tipos básicos e outras estruturas. Os tipos básicos são: *string*, inteiro, booleano e números com ponto flutuante. Permite a criação de atributos com tipos polimórficos, ou seja, ele aceita dados de qualquer tipo. Matrizes de qualquer tipo válido também são construções válidas.
- Representar requisições de serviços.
- Representar respostas de serviços.
- Codificar e decodificar estruturas aninhadas através de referência (*flattening*).
- A identificação das referências é feita através de um identificador de instância.
- Controlar caracteres de escape para *strings*.
- Controlar erros.
- Versionamento do protocolo e disponibilização propriedades do servidor.

Algumas características que não são providas pelo protocolo:

- Não é compatível com envelope SOAP.
- Autenticação e criptografia devem ser tratadas por protocolos de mais baixo nível ou por métodos implementados pelo usuário

Detalhes do funcionamento e implementação são descritos no próximo capítulo para auxiliar a compreensão do mecanismo.

5 ARQUITETURA DO PROTOCOLO BICO

5.1 Introdução

A utilização do protocolo proposto em situações reais é objetivo deste trabalho, bem como ter implementações para as linguagens mais importantes e facilitar seu uso o máximo possível.

A arquitetura do codificador-decodificador, doravante chamado CODEC, é apresentada, permitindo melhor compreensão do funcionamento do protocolo, e também fornecendo o conhecimento básico para a extensão do protocolo para outras linguagens. O CODEC é responsável por traduzir as mensagens recebidas, criando as estruturas transportadas em memória, efetuar as requisições e respostas de serviços, e converter as estruturas em memória em novas mensagens. Portanto, o CODEC não apenas codifica e decodifica mensagens, mas é também responsável pelo controle dos serviços. O entendimento da estrutura do protocolo possibilita alterações como a utilização de outras codificações para as mensagens e adição de funcionalidades ou serviços básicos.

Ferramentas de auxílio e utilitários de comunicação são disponibilizados para facilitar o uso do protocolo. As ferramentas auxiliares reduzem drasticamente a necessidade de intervenção e programação para adaptar as ferramentas. O protocolo não impõe restrição ao meio de transporte a ser utilizado para conectar os entes, bastando que seja uma conexão ponto a ponto com capacidade de transporte de caracteres imprimíveis. Os meios de comunicação fornecidos potencializam a utilização da proposta em diversos cenários e situações.

5.2 Aspectos de Implementação

O protocolo BICO foi projetado em UML para ser inicialmente implementado em Java. As classes responsáveis pelas suas principais funcionalidades estão representadas no diagrama UML da figura 5.1.

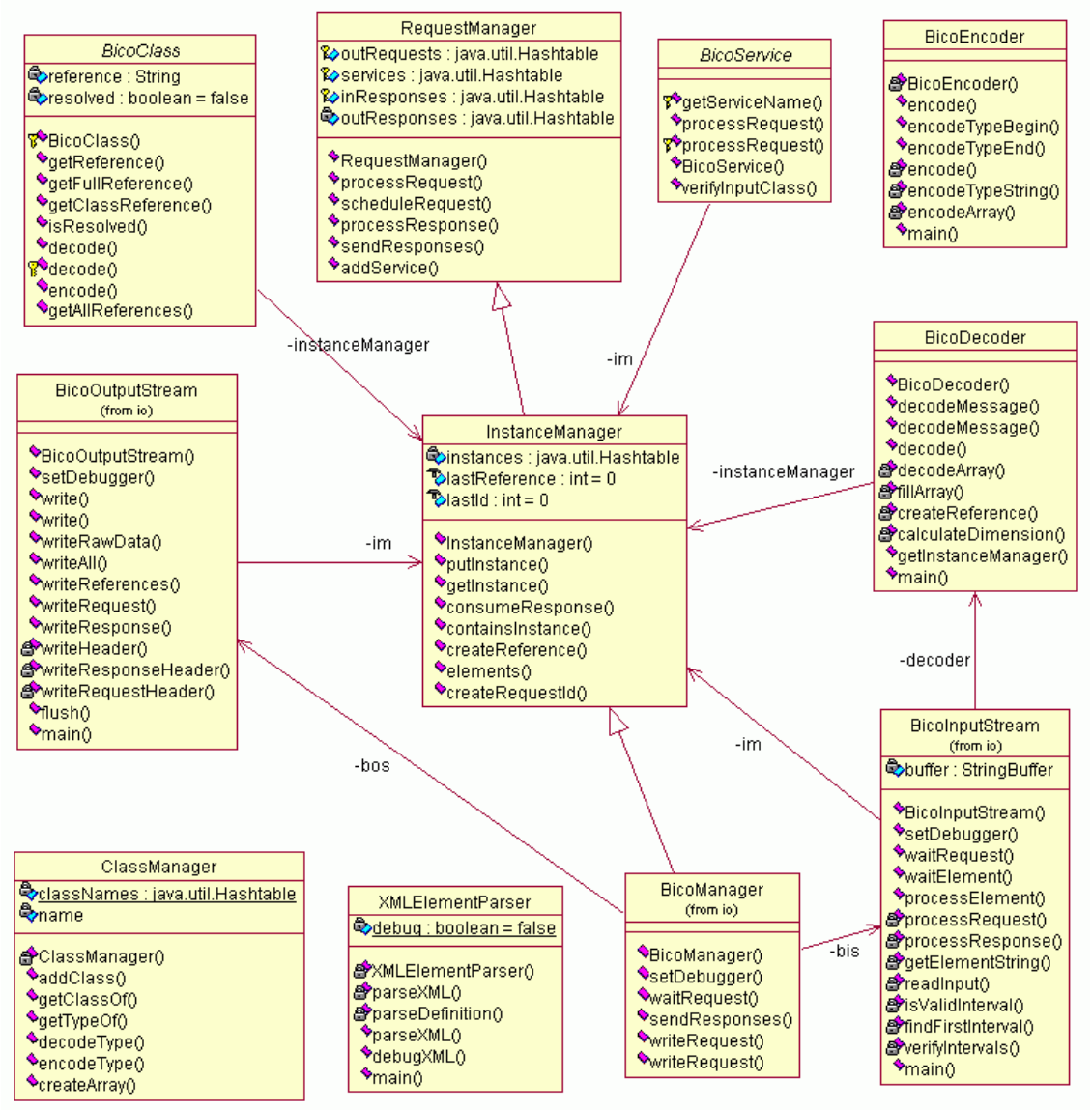

Figura 5.1: Diagrama de classes (UML) do BICO.

Cada uma destas classes é responsável por um conjunto de serviços, mostrado na tabela 5.1.

Tabela 5.1: Funções das principais classes BICO.

Classe	**Função**
BicoClass	Representação básica de uma estrutura, toda estrutura criada deve estender essa classe.
BicoService	Representação básica de um serviço, todo serviço deve estender essa classe.
BicoEncoder	Possui funções para auxiliar a codificação de estruturas em XML.
BicoDecoder	Possui funções para auxiliar a decodificação de estruturas de XML para o formato interno.

XMLElementParser	Transforma um XML em um formato intermediário (*hashtable*).
InstanceManager	Controla as instâncias através de um cadastro dos identificadores de instâncias criadas.
RequestManager	Controla as requisições e respostas.
ClassManager	Auxilia o processo de decodificação através de um cadastro de estruturas válidas.
BicoOutputStream	Serializa os objetos (instâncias de estruturas, requisições e respostas).
BicoInputStream	Deserializa os objetos.
BicoManager	Utilitário composto de um BicoInputStream, BicoOutputStream e InstanceManager, com a finalidade de simplificar a utilização do protocolo.

Esse conjunto básico de classes pode ser ampliado e as classes existentes podem ser estendidas permitindo que extensões do protocolo sejam projetadas e implementadas a partir deste trabalho.

5.3 CODEC – Codificador/Decodificador

Para descrever o funcionamento do CODEC, iniciaremos pela descrição da transformação de estruturas já criadas em XML. Ou seja, a partir de uma estrutura preenchida, deve ser feito o processo de transformação para XML.

O processo de transformação inicia por uma requisição de um objeto BicoManager a um objeto BicoOutputStream e prossegue, conforme o diagrama de seqüência mostrado na figura 5.2 até que seja obtido como resultado uma representação XML equivalente à estrutura de entrada.

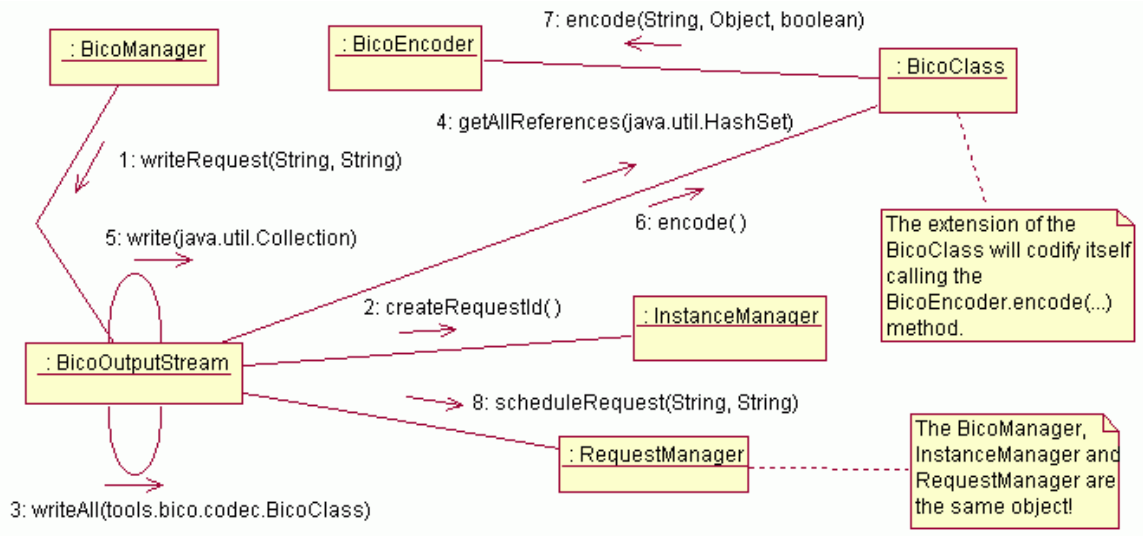

Figura 5.2: Diagrama de seqüência (UML) de uma mensagem de saída.

O CODEC possui diversas funções genéricas de conversão e interpretação de dados, porém parte do processamento é feita por funções da própria estrutura a ser codificada. A função responsável pela codificação em XML utiliza funções genéricas de codificação para

seus atributos. Os tipos simples como inteiro, ponto flutuante, ***string*** e boleano são mapeados para uma representação texto utilizando-se de funções de conversão fornecidas pelo sistema. Caso o atributo seja uma outra estrutura, apenas o identificador desta é codificado. Para que um objeto seja corretamente reconstruído, devem ser enviadas todas as estruturas referenciadas por ele. Ao codificar atributos do tipo '***any***' (polimórficos), o tipo do objeto atualmente atribuído é também codificado. Assim, a codificação é feita pela própria estrutura a ser codificada em conjunto com funções auxiliares providas pelo núcleo do CODEC.

A representação XML pode então trafegar como uma mensagem do protocolo BICO para qualquer destino onde esteja presente um decodificador. A figura 5.3 apresenta um exemplo de mensagem BICO.

```
<request service="EtchSimulation" id="id_0"
         inputId="EtchSimulationRequest#0">
  <BicoPoint id="2"><x>5.0</x><y>5.0</y></BicoPoint>
  <BicoPoint id="4"><x>39.0</x><y>12.0</y></BicoPoint>
  <BicoPoint id="6"><x>14.0</x><y>14.0</y></BicoPoint>
  <BicoPoint id="5"><x>37.0</x><y>4.0</y></BicoPoint>
  <EtchSimulationRequest id="0">
      <polygons><item>1</item></polygons>
      <etchFileName>FeCl3</etchFileName>
      <etchFile/>
      <duration>1.0</duration>
      <visibleSteps>10</visibleSteps>
      <firstVisibleStep>1</firstVisibleStep>
      <internalStepCount>20</internalStepCount>
  </EtchSimulationRequest>
  <BicoPoint id="3"><x>11.0</x><y>20.0</y></BicoPoint>
  <BicoPoint id="7"><x>13.0</x><y>2.0</y></BicoPoint>
  <BicoPolygon id="1">
      <points>
            <item>2</item><item>3</item><item>4</item>
            <item>5</item><item>6</item><item>7</item>
      </points>
  </BicoPolygon>
</request>
```

Figura 5.3: Exemplo de mensagem Bico

O processo inverso, a transformação de XML em estruturas internas é mais complexo, envolvendo os passos esquematizados na figura 5.4.

Figura 5.4: Esquema de decodificação

O processo inicia com a recepção de uma mensagem BICO em formato XML e esta mensagem deve ser então representada como uma estrutura válida no ambiente de destino. A figura 5.4 ilustra o diagrama de seqüência, mostrando o fluxo de requisição de serviços dos objetos envolvidos.

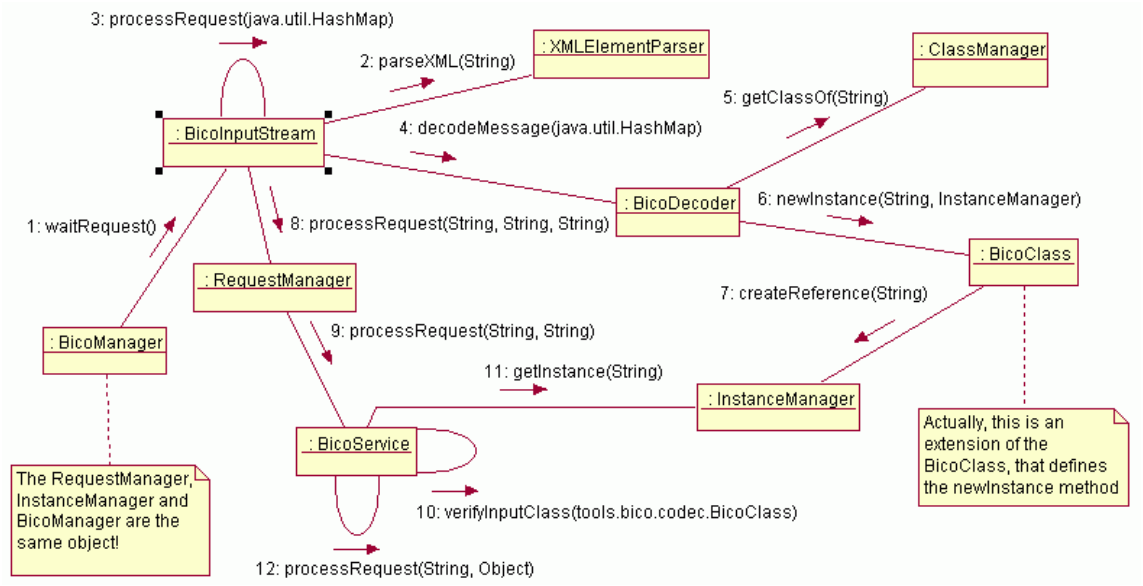

Figura 5.5: Diagrama de seqüência (UML) de uma mensagem de entrada.

Primeiramente a mensagem XML é transformada em um formato interno utilizando mapas encadeados. Dessa forma, isola-se o processamento dependente do XML e suas marcações, do processamento dependente do conteúdo da mensagem. Essa estrutura interna é então interpretada, seus valores são convertidos para os tipos corretos, matrizes são recriadas, etc. Uma estrutura nova (das definidas pelo usuário) é criada para armazenar os valores. Sempre que uma estrutura é criada, uma referência para ela é armazenada em um mapa de identificadores e referências. Caso existam referências a outras estruturas, estas são pesquisadas no registro de estruturas, e se encontradas, o atributo correspondente passa a apontar para ela, caso contrário uma estrutura vazia é criada utilizando o identificador encontrado na mensagem. Assim, estruturas complexas podem ser remontadas pelo protocolo.

No caso de tipos complexos, apenas existirá um identificador da estrutura, que será utilizado para procurar a estrutura existente ou criar uma estrutura vazia, considerando que ela será preenchida num momento futuro. Dessa forma é possível reconstruir estruturas com referências comuns e auto-referências, ou mesmo associar uma nova estrutura com estruturas pré-existentes.

As requisições são geradas utilizando-se um nome de serviço e uma estrutura de entrada. Essa estrutura é codificada e adicionada a um elemento XML que contém como atributos o nome do serviço e o identificador da estrutura principal. Juntamente com a estrutura são codificadas todas as estruturas referenciadas por ela, garantindo que o receptor possa reconstruir a estrutura de dados inicial.

A requisição é interpretada pelo servidor juntamente com seu conteúdo, como descrito anteriormente, ou seja, a requisição é transformada em uma tabela *hash*. O servidor verifica se possui um serviço com o nome requerido e caso não exista, emite uma mensagem de erro ao requerente. Caso exista o serviço, a referência ao objeto que representa o serviço é

recuperada e recebe a estrutura passada como parâmetro na requisição. O método que principal do serviço é então executado.

Podem ocorrer erros de processamento de qualquer natureza, como por exemplo, tipos incorretos (em parâmetros *any*), limites incorretos para processar, ou mesmo erros durante a execução do serviço. Se esses erros não comprometerem o funcionamento do CODEC e corresponderem aos elementos de erro propostos pela linguagem de programação (como Java *exceptions*), eles serão capturados pelo CODEC e darão origem a mensagens de erro com o conteúdo do erro original. Essas mensagens serão utilizadas como resposta e assim o usuário do serviço pode tratar de forma eficiente e polida o erro ocorrido.

Caso não ocorram erros, o serviço deve gerar uma ou mais estruturas (das criadas para o protocolo) para serem enviadas como resultado do serviço. A resposta contém um atributo identificador da requisição e todas as estruturas referenciadas pela estrutura principal da resposta. Desta forma associa-se a resposta à requisição correta.

5.4 Implementações

Alguns detalhes de implementação são descritos nessa subseção, pois a grande diferença entre as linguagens alvo acarreta importantes diferenciações nas implementações e por conseqüência na utilização do protocolo.

Para representar atributos de tipo não determinado (polimórfico) em linguagens que não possuem mecanismos de reflexão para descoberta de tipo, é utilizada uma estrutura paralela aos dados que armazena a informação de tipo. Sabendo que tipos não determinados possuirão identificadores de tipo na própria mensagem, o decodificador trata estes de forma similar aos atributos comuns apenas construindo, se necessário a estrutura de identificação de tipos em linguagens que a necessitem.

As figuras 5.6 e 5.7 demonstram códigos em linguagens variadas as estruturas e serviços que o sistema produz. A figura 5.6 mostra um serviço implementado em Java. Somente o código em negrito é implementado manualmente, o sistema gera automaticamente todo o restante da estrutura necessário para a adaptação ao protocolo. Aconselha-se a não fazer alterações no código automaticamente gerado. Quando necessário, alterações ou extensões devem ser feitas em arquivos separados (figura 5.8).

```
… // imports

public class CDFGetCIFLayersService extends BicoService {
   public static final String defaultName = "CDFGetCIFLayers";

   private CDFGetCIFLayersService(InstanceManager im, String serviceName) {...}
   public static void init(InstanceManager im) {...}
   public static void init(InstanceManager im, String serviceName) {...}
   public void verifyInputClass(BicoClass input) throws Exception {...}

   public String processRequest(String id, BicoClass input) throws Exception {
      CDFGetCIFLayersRequest in = (CDFGetCIFLayersRequest) input;
      CDFGetCIFLayersResponse ret = CDFGetCIFLayersResponse.newInstance(im);

      // INICIO DO CODIGO IMPLEMENTADO MANUALMENTE

      // descobrir que camadas cif podem ser geradas pelo cdf:
      TechFile f = CDFServer.getTechFile(in.getTechId());
      LinkedHashSet<String> lhs = new LinkedHashSet<String>();

      for (int i = 0; i < f.BASIC_LAYERS.length; i++) {
         LinkedHashMap<String, Integer> ll2 =
            f.getCIFRules().getCIFLayers(LayoutLayers.BASIC_LAYERS[i]);
```

```
        lhs.addAll(ll2.keySet());
    }

    String[] r = new String[lhs.size()];
    int i = 0;
    Iterator it = lhs.iterator();
    while (it.hasNext()) {
       r[i++] = (String) it.next();
    }

    ret.setLayer(r);

    // FIM DO CODIGO IMPLEMENTADO MANUALMENTE

    return ret.getFullReference();
  }
}
```

Figura 5.6: Serviço implementado em Java.

```
... // imports

(procedure newCDFCellGenerationRequest(reference instanceManager) ... )

(defclass CDFCellGenerationRequest (BicoClass) ( (techId) ... ))

(defmethod CDFCellGenerationRequest_new ...)

(defun CDFCellGenerationRequest_newInstance ... )

(defun CDFCellGenerationRequest_newInstanceRef ... )

(defmethod CDFCellGenerationRequest_encode ((this CDFCellGenerationRequest))
(prog (buff)
    buff = strcat(BicoEncoder_encodeTypeBegin(
           "CDFCellGenerationRequest" this->reference))

    buff = strcat(buff BicoEncoder_encode("techId" this->techId t))
    ... // encode other fields

    buff = strcat(buff BicoEncoder_encodeTypeEnd("CDFCellGenerationRequest"))
    return(buff)
))

(defmethod BicoClass_decodeObject ((this CDFCellGenerationRequest)
            parameters decoder)
    this->techId = decoder->decode("string" parameters->get("techId"))
    ... // decode other fields
)

(defmethod BicoClass_getAllReferences ... ))
```

Figura 5.7: Exemplo de estrutura gerada para Skill++

```
... ;; imports

(procedure extCDF_CG_Request(cdf_cgr)
   (procedure summary() (CDFCellGenReqEx_summary cdf_cgr->this))
   append(cdf_cgr list( 'summary  summary ))
)

(defmethod CDFCellGenReqEx_summary ((this CDFCellGenerationRequest))
            printf("Logic Function: %s\n" this->logicFunction)
            printf("Logic Family:   %L\n" this->logicFamily )
            printf("Tech ID: %s\n" this->techId)
            printf("Template ID: %s\n" this->templateId)
            printf("Generate:       %s%s%s%s%s%s%s\n"
```

```
                    if(this->generateOptimizedLogicFunction
        "Optimized Logic Function, " "")
                    ... ;; and so on
        )
        ...
)
```

Figura 5.8: Exemplo de extensão para estrutura em Skill++.

5.5 Ferramentas de Auxílio

A ferramenta Project Manager foi desenvolvida com o objetivo principal de facilitar ao máximo a preparação e criação de toda a estrutura necessária para adaptar um projeto ao uso do mecanismo proposto. Para tanto, são fornecidas telas de automação para a criação dos descritores de serviços e estruturas, e podem ser selecionados transformadores de serviços e estruturas para gerar automaticamente os códigos nas linguagens alvo. São fornecidos conversores para as linguagens Java, C++, Skill++ e Ample. Após a seleção de todos as estruturas, serviços e transformadores para ambos, a ferramenta permite realizar a transformação dos artefatos nos códigos fontes desejados. Além da possibilidade de seleção de tradutores previamente disponibilizados, a ferramenta possibilita a edição dos mesmos, visando facilitar a adaptação de novas linguagens ao mecanismo Bico.

A figura 5.9 mostra a tela referente à criação de serviços. A descrição XML correspondente é automaticamente gerada a partir dos dados fornecidos nas caixas de diálogo.

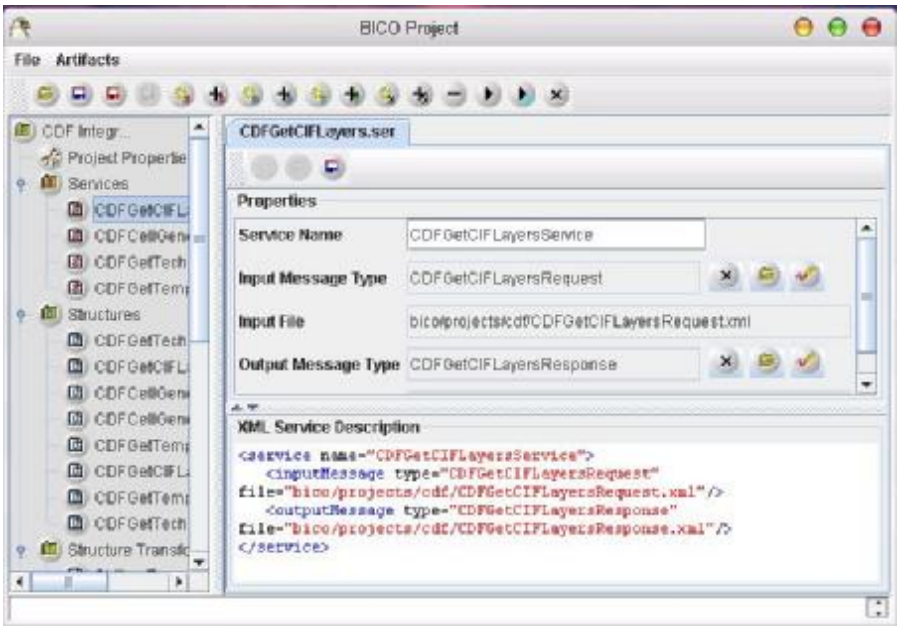

Figura 5.9: Tela principal da ferramenta Project Manager.

O Project Manager também disponibiliza telas para a criação de estruturas. A figura 5.10 mostra o dialogo de entrada de um novo atributo de uma estrutura. A geração do XML que descreve a estrutura ocorre automaticamente (figura 5.11).

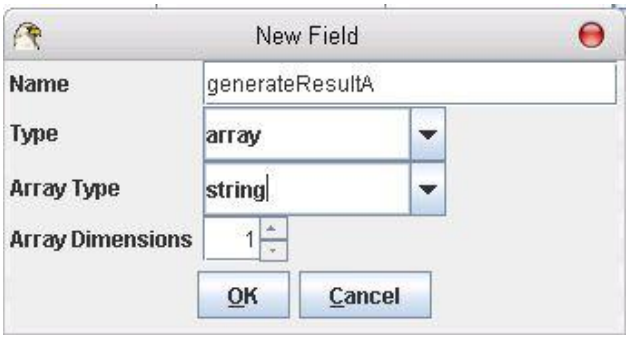

Figura 5.10: Diálogo de criação de campo para estruturas.

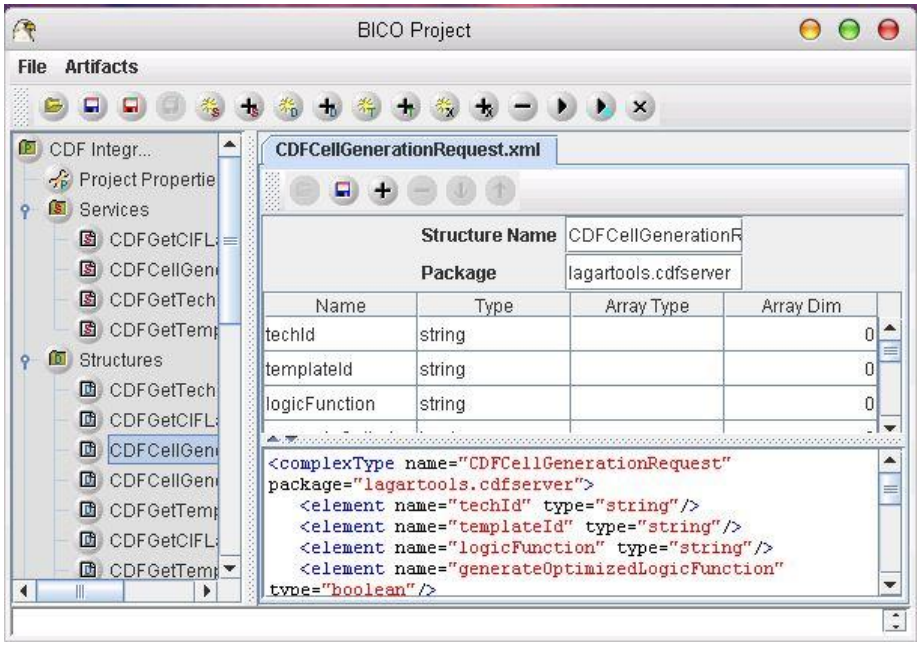

Figura 5.11: Tela de edição de uma estrutura.

Após criar as descrições dos serviços e estruturas que farão parte da integração de determinada ferramenta, deve-se escolher os tradutores (figura 5.12) que serão responsáveis por transformar as estruturas em código fonte nas linguagens escolhidas.

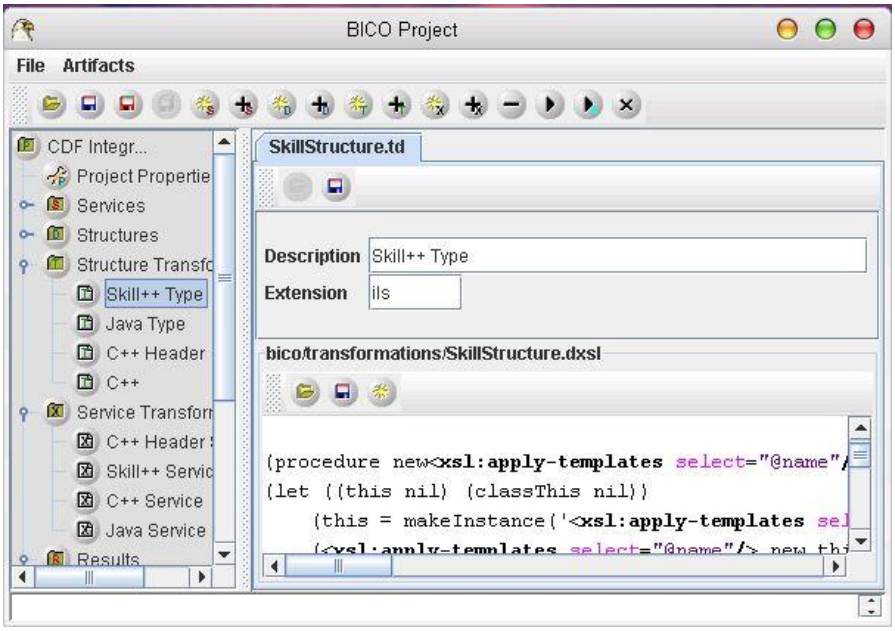

Figura 5.12: Edição de um tradutor.

Escolhidos os transformadores pode-se transformar as definições através da ação rodar (figura 5.13).

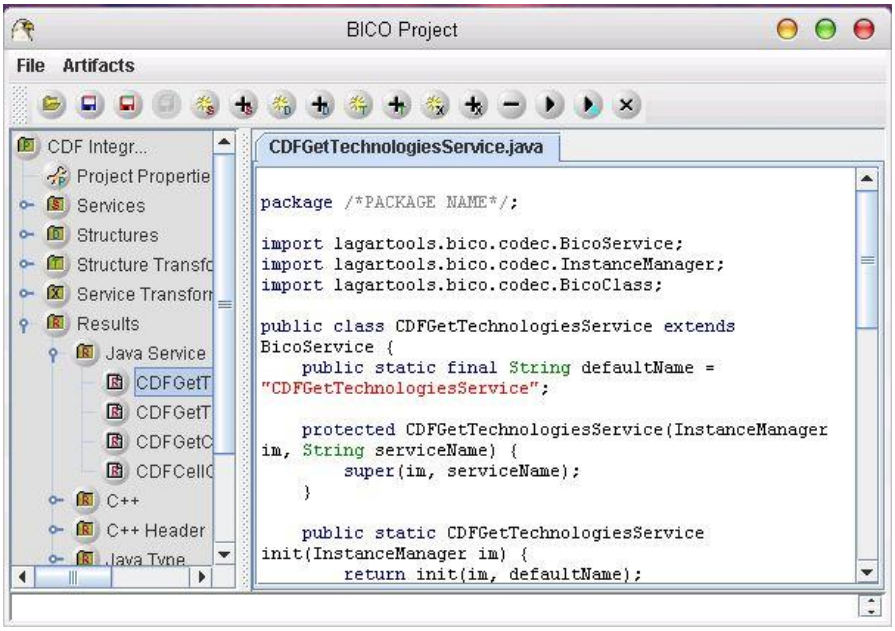

Figura 5.13: Visualização de um código gerado automaticamente.

5.6 Utilização

Este trabalho propõe-se a oferecer uma solução para a integração de ferramentas de CAD independentes e *frameworks* de microeletrônica. Para tornar a utilização da solução proposta mais fácil são disponibilizadas ferramentas para auxiliar a geração de artefatos utilizados pela solução. A ferramenta Project Manager, que foi apresentada é responsável por isso. Essa seção apresenta um esquema de utilização da ferramenta e do protocolo. Além disso, alguns estudos de caso são apresentados, validando e avaliando a utilização da solução em casos reais.

A integração de ferramentas utilizando a solução proposta dá-se em dois passos principais. Primeiramente o passo de definição, onde os serviços e as estruturas de dados são criados, e as linguagens alvo são selecionadas. O segundo passo é a implementação (figura 5.14), onde as funcionalidades do servidor e do cliente devem ser desenvolvidas. Cliente e servidor devem utilizar um meio de comunicação comum entre eles.

```
In = newIPCInputStream(this->process)
out = newIPCOutputStream(this->process)
bm = newBicoManager(in out)

req = CellGenRequest_newInstance(bm)
... // fill the request structure
id = bm->writeRequest2("CellGen" req)

this->bm->waitResponse()
resp = this->bm->consumeResponse(id)
... // process the response
```

Figura 5.14: Exemplo de cliente em Skill++.

O cliente deve executar as chamadas de serviço, coletando dados para os serviços por interfaces gráficas ou parâmetros de linha de comando. O servidor geralmente não possuirá

interface gráfica. Ele é responsável pela execução dos serviços. Os serviços têm sua interface definida no passo de definição e devem ser implementados na segunda etapa.

A etapa de definição compreende a criação das estruturas de dados e dos serviços. Os serviços devem representar cada chamada de função a ser implementada, de forma a permitir ao cliente solicitar informações importantes, solicitar execuções de algoritmos e retornar seus resultados.

As estruturas devem compreender as estruturas de dados envolvidas nas interações entre o cliente e servidor. Elas podem corresponder ou não as estruturas internas dos algoritmos e dados envolvidos, porém deve-se manter em mente que, quanto maior a semelhança com os elementos internos, mais fácil tornar-se-á a programação dos serviços e dos clientes.

Depois de definidos os serviços e estruturas, deve-se escolher as linguagens de programação envolvidas tanto no servidor quanto no cliente. Os serviços devem ser traduzidos, gerando assim o esqueleto do código a ser implementado, para a ou as linguagens que forem utilizadas para servir. As estruturas devem ser traduzidas para todas as linguagens, pois o resultado da tradução é o complemento do CODEC, ou seja, é utilizado conjuntamente nos processos de codificação e decodificação das estruturas.

O código do cliente corresponde à criação do fluxo de dados (meio de comunicação), a criação das estruturas a serem enviadas como parâmetros, e a interpretação das estruturas recebidas como resposta. Após a aquisição dos dados a serem utilizados como parâmetro, deve-se criar as estruturas correspondentes, utilizando-as para efetuar a requisição de serviço. Após a execução do serviço, que atualmente é bloqueante, pode-se obter um erro ou a resposta do serviço. Se houver um erro, pode-se tratá-lo, verificando a mensagem de erro. Se não ocorrerem erros de execução do serviço, a estrutura retornada conterá os resultados do serviço que podem então ser interpretados.

O código do servidor corresponde a inicialização dos serviços e criação do fluxo de dados em modo servidor. Cada serviço deve implementar o método correspondente ao processamento da requisição, interpretando as estruturas de entrada, efetuando a funcionalidade desejada e retornando uma estrutura de resposta. O servidor deve associar nomes de serviço com os objetos responsáveis pelo serviço. O servidor é responsável pelo estabelecimento de um canal de espera de conexões de clientes, ou outra forma de conectar-se com eles. A cada conexão com um novo cliente, uma nova instância do CODEC deve ser criada e associada com o meio de comunicação, interpretando automaticamente as requisições e repassando para os serviços registrados.

Visto que o protocolo não define modelos de dados, mas permite a definição de estruturas ou modelos de dados, tal definição depende do planejamento do usuário. Essa abordagem difere-se da proposta OpenAccess, que fornece modelos de dados para diversas entidades do mundo real. Portanto, o usuário é livre pra escolher e definir os modelos de dados que trafegarao pelo protocolo.

5.7 Mecanismos de comunicação

A comunicação entre ferramentas através do protocolo proposto necessita de um canal de comunicação. Como forma de suprir os usuários do protocolo proposto com canais de comunicação alguns mecanismos de comunicação são fornecidos. Uma característica importante do protocolo é a desvinculação de meio de comunicação permitindo que o usuário escolha o melhor canal de comunicação. Para facilitar a implementação de soluções

que utilizem o protocolo, são oferecidos alguns métodos de comunicação que serão detalhados a seguir.

A forma mais básica de comunicação entre softwares é a utilização das entradas e saídas de dados padrões. Os softwares podem ser conectados através do redirecionamento da entrada e saída padrão. Dependendo da forma de uso, pode tornar-se necessário o uso de buferização, garantindo que os dados não serão perdidos. Essa estratégia, porém, não requer grande esforço e, portanto, não são fornecidos por este trabalho nenhum mecanismo auxiliar para isso.

A ampla utilização de protocolos TCP/IP torna o uso de *sockets* comum e eficiente. *Sockets* podem ser utilizados como canais de conexão entre softwares. Para facilitar a utilização de *sockets* mesmo para as linguagens onde estes não estão disponíveis, foi formulado um mecanismo que faz a interface entre as saídas e entradas padrões para um *socket*. Assim, o programa pode ser invocado por outro software, que deve conectar suas saídas e entradas padrões ao Socket Forwarder. O Socket Forwarder repassa os dados recebidos da entrada para a saída do *socket*, e passa para a saída padrão as entradas recebidas do *socket* (figura 5.15). Clientes e servidores de serviços do protocolo proposto podem utilizar esse mecanismo.

Figura 5.15: Esquema de uso do Socket Forwarder.

Buscando fornecer o uso do protocolo através da Internet, foi formulado um mecanismo de comunicação utilizando como base o protocolo HTTP. O TinyServer é um mini servidor HTTP projetado para encapsular, de forma simplificada, ferramentas que objetivem o protocolo HTTP, transformando-as em servidores. Ele oferece serviços de abertura de arquivo e implementa uma versão simplificada de Servlets para que demais serviços implementados em Java possam ser integrados.

Um Servlet foi especialmente desenvolvido para atuar como um servidor SOAP, permitindo a integração com o protocolo SOAP e a implementação de serviços. Portanto pode-se de forma simplificada integrar ferramentas através de SOAP, ou mesmo utilizando SOAP em conjunto com Bico, ou Bico puramente. No caso do uso de SOAP com Bico, as mensagens bico são inclusas como nodos internos do elemento SOAP_BODY.

5.8 Avaliação da proposta

A avaliação da proposta em termos de suas características quanto a interoperabilidade é fundamental para credenciar o seu uso como solução para problemas reais. Portanto, retomando os tópicos tratados no capítulo três, uma breve avaliação sobre diversos aspectos, características e conseqüências da proposta é apresentada nessa subseção.

Primeiramente, visto que o protocolo não define modelos de dados, não cabe um juízo quanto ao tipo de heterogeneidade tratada. A modelagem dos dados é responsabilidade do usuário do protocolo. Assim, não são fornecidos mecanismos que auxiliem a correlação de

dados, pois tal tarefa é de responsabilidade do usuário. Porém, quanto a diferenças de modelagem a proposta (protocolo e ferramentas auxiliares) auxilia o tratamento de diferenças em relação ao hardware e sistema operacional.

A proposta cria uma forma de troca de informações entre diferentes ambientes de CAD e linguagens de programação. Por utilizar Java como linguagem de implementação dos utilitários que criam o canal de comunicação, é possível interoperar entre diferentes plataformas e sistemas operacionais. Por oferecer a possibilidade de utilizar o acoplamento desses utilitários via entradas e saídas padrões, é possível interoperar entre todas as linguagens de programação, mantendo a independência de plataforma. E ainda, como a proposta foi desenvolvida para suprir as dificuldades de comunicação com os ambientes comerciais, ela permite a interoperação entre os ambientes contemplados por este trabalho. Entre todas as linguagens e ambientes é possível transportar estruturas através de requisições e respostas de serviços, fornecendo assim uma forma eficiente em termos de capacidade para trocar informações.

A atual versão da proposta descrita nesse texto não compreende a descrição de meta informações, como as contidas no protocolo WSDL. Porém parte das informações necessárias são obtidas das descrições XML das estruturas e serviços. Assim existe a necessidade de conhecimento prévio das operações dos serviços implementados, porém futuras versões do protocolo, que se identifiquem mais fortemente com SOAP deverão permitir a descoberta em tempo de execução dos formatos das mensagens de entradas e saídas.

Para adaptar os sistemas ao protocolo é necessário adicionar funções que preencham as estruturas de requisição, interpretem as estruturas de resposta, inicializem o sistema e efetuem as requisições. Portanto, o acesso ao código pode ser um fator determinante para a utilização da proposta. Outro fator listado no capítulo três é a quantidade de conversões entre protocolos, que na proposta é basicamente a transformação de e para XML.

Uma característica importante é a possibilidade de atingir execução conjunta, pois a forma de execução requer que ambas as ferramentas participantes estejam em execução, podendo permanecer em execução, ou ainda atuarem ambas como cliente ou servidoras, provendo mutuamente serviços.

6 ESTUDOS DE CASO

Nesse capítulo serão descritas algumas das possíveis utilizações do mecanismo proposto para efetuar interações entre ambientes de CAD profissionais e ferramentas acadêmicas. Apesar da grande cobertura (em termos de áreas abordadas) oferecida pelos *frameworks*, existem áreas que não são exploradas por eles, tornando importante o uso de ferramentas independentes. Devido ao grande leque de possibilidades apenas algumas das possíveis formas de integração possíveis. Porém, com o intuito de demonstrar formas de utilização do protocolo proposto, algumas delas serão descritas e avaliadas a seguir. Os estudos de caso apresentados a seguir demonstram não somente formas de interação, mas também a relevância e potencialidade dos resultados que podem ser obtidos explorando a utilização do protocolo.

6.1 Etch Simulator - E.T.

O primeiro estudo de caso a ser apresentado é o de uma ferramenta de simulação de corrosão utilizada para projetar MEMS (Micro-eletro-mechanical Systems, figuras 6.1 e 6.2).

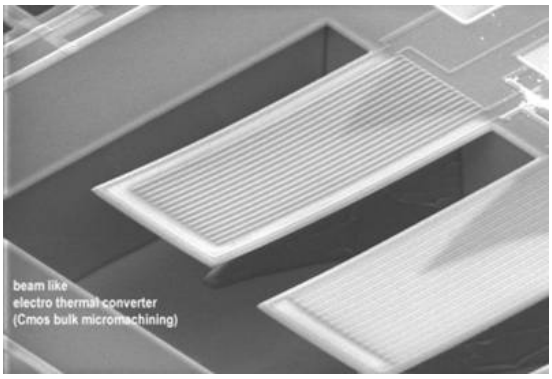

Figura 6.1: Ponte suspensa dupla.

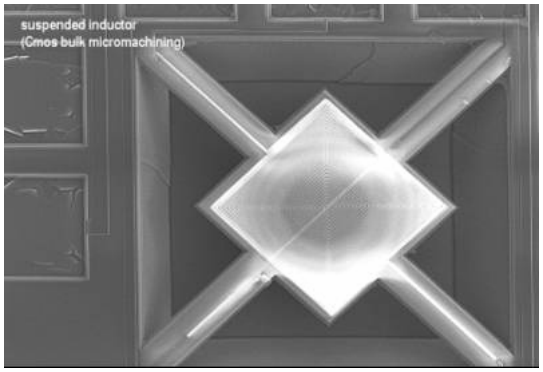

Figura 6.2: Membrana.

MEMS tem ganhado uma crescente importância na microeletrônica nos últimos anos. A evolução tecnológica da fabricação de chips permite que dispositivos compostos por partes ópticas, mecânicas e/ou elétricas fossem compactadas em dimensões microscópicas e, assim, dispostas em chips. O funcionamento dos dispositivos MEMS é fundamentado em efeitos não elétricos. Eles são utilizados principalmente como sensores ou atuadores como acelerômetros, sensores de movimento, sensores de gases (nariz eletrônico), sensores de

temperatura, filtros de freqüência elétrica ou óptica, e até mesmo existem propostas de elementos utilizados para fazer o seqüênciamento de DNA (*Deoxyribonucleic acid*).

A confecção desses dispositivos compostos de partes elétricas, mecânicas e ópticas, ou seja, não puramente elétricos, obriga a utilização de ferramentas especiais. O projeto de dispositivos mecânicos exige que estudos sejam feitos para garantir que os materiais suportaram os movimentos de forma adequada, portanto tais características devem também ser observadas e tratadas. Da forma similar, dispositivos ópticos exigem atenção especial para os parâmetros ópticos, sendo isso válido para qualquer efeito que seja utilizado como base de funcionamento dos dispositivos. Além disso, a forma de projeto destes mecanismos propicia a utilização de ferramentas geradoras de leiautes. As ferramentas MemsCap (2005), Coventor (2005) e Lagarto (TOGNI et al, 2002) são exemplos disso. Essas ferramentas capturam o conhecimento técnico de construção dos dispositivos em algoritmos preestabelecidos, permitindo que os usuários criem dispositivos a partir de alguns parâmetros de construção, sem a necessidade de desenhá-los manualmente.

Duas ferramentas que são de importância relevante para a construção de MEMS são a visualização em perspectivas diferenciadas e a simulação de corrosão. Visto que MEMS são construídos com partes móveis suspensas, a projeção do resultado da corrosão e a visualização das construções são fundamentais para garantir que a fabricação do elemento corresponderá com o resultado esperado. Vistas em corte e 3D auxiliam nas tarefas de verificação visual do resultado da corrosão, enquanto a simulação da corrosão permite que o resultado seja previsto e validado.

A figura 6.3 exemplifica o processo de corrosão utilizando a técnica *Front-Side Bulk Micromachining* (FSBM) (MARTINAZZO et al, 2002) (TOGNI et al, 2004) (RIBAS, 1998). Duas camadas são utilizadas nessa técnica, uma camada que será removida pela corrosão, chamada de sacrificial, e uma camada não removida pelo corrosivo, chamada de camada de proteção. Sobre a camada sacrificial é depositada a camada protetora e, nela abrem-se janelas por onde o corrosivo penetrará removendo seletivamente a camada sacrificial. As janelas em uma camada protetora são utilizadas para gerar as formas geométricas desejadas. Com o passar do tempo a camada sacrificial é removida gradualmente. A velocidade de expansão da corrosão é diferenciada conforme a direção da corrosão, assim podendo-se obter formas resultantes diferentes das janelas de corrosão, como pode ser observado na figura 6.4.

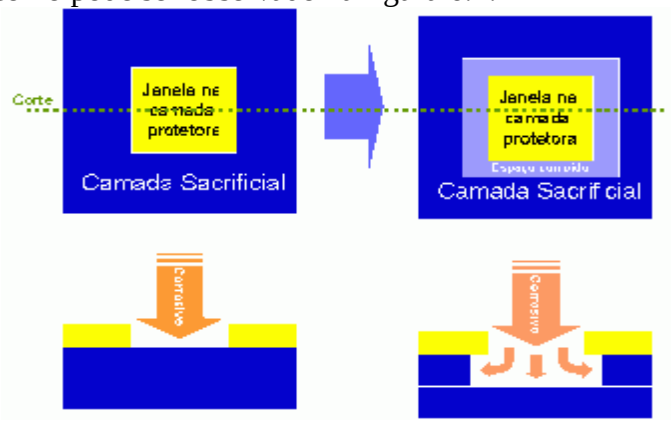

Figura 6.3: Descrição do processo de corrosão.

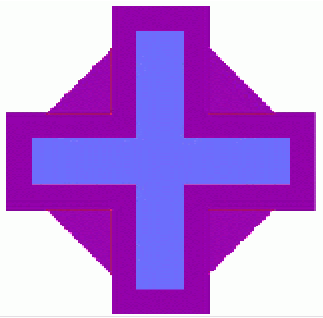

Figura 6.4: Exemplo de resultado de corrosão.

O E.T. (figuras 6.5, 6.6, 6.7 e 6.8) é um ambiente para simulação de corrosão em MEMS. A simulação de corrosão é baseada na técnica de FSBM. Além do simulador são oferecidas ferramentas para edição dos diagramas polares (usados para armazenar as velocidades de corrosão para cada direção), edição de polígonos simples, e visualização dos resultados. O visualização disponível é capaz de gerar visões de corte e 3D além da convencional visão 2D (vista de cima). Dessa forma esse ambiente para corrosão é capaz de agrupar as funcionalidades básicas necessárias para a validação da corrosão de estruturas MEMS.

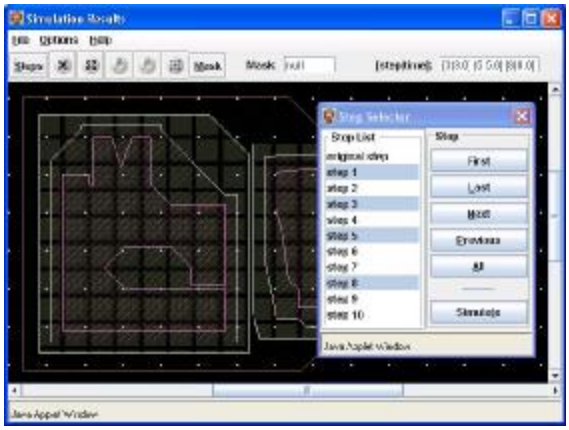

Figura 6.5: Resultado de uma corrosão.

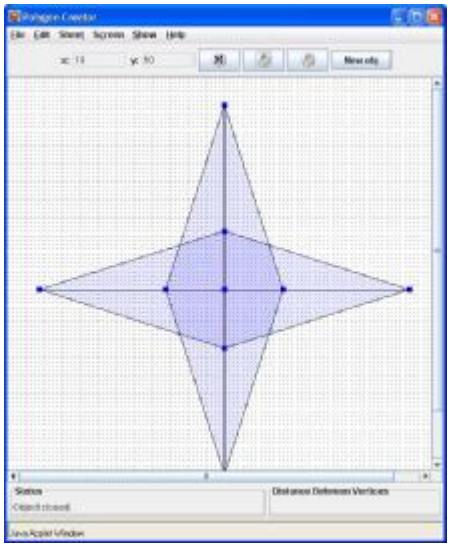

Figura 6.6: Edição de polígonos.

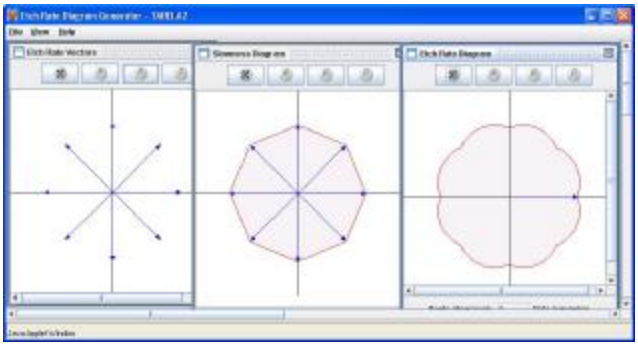

Figura 6.7: Configurador de corrosivos.

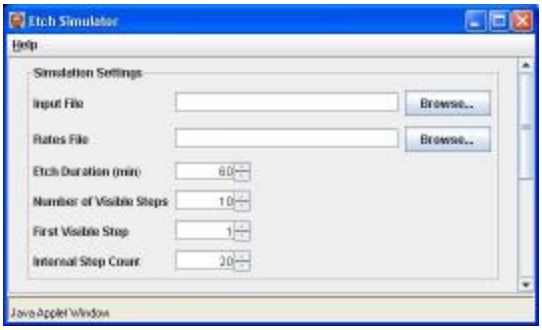

Figura 6.8: Diálogo para execução de simulação.

Ambientes comerciais para CI como os das empresas Mentor Graphics e Cadence abrangem o projeto de circuitos digitais e analógicos, porém não possuem uma combertura tão ampla para a construção de MEMS. Por esse motivo, disponibilizar para estes ambientes as ferramentas utilizadas no projeto de MEMS traz benefícios significativos. Além de simplesmente oferecer as ferramentas para MEMS, o ambiente propiciaria aos seus usuários a comodidade de não necessitar migrar para outras ferramentas para obter um projeto de MEMS. Assim os usuários podem utilizar todas as ferramentas as quais estão

acostumados, reduzindo a necessidade de migração de ambiente durante o desenvolvimento e o tempo de aprendizado, assim facilitando o uso da solução.

Da perspectiva do E.T., o benefício provem das ferramentas maduras e bem elaboradas que tais ambientes oferecem para o projeto de leiautes. A ferramenta E.T. foi desenvolvida para lidar com a simulação de corrosão, não provendo demais ferramentas que podem ser necessárias para um projeto de maior porte. Um exemplo seria a utilização de editores de leiaute dos ambientes ao invés do rudimentar editor de polígonos que é oferecido.

Esse estudo de caso utiliza, então, a ferramenta E.T. (desenvolvida em Java) em conjunto com o ambiente Cadence DFII. O objetivo desse exemplo é disponibilizar a corrosão para polígonos que tenham sido desenhados no ambiente DFII. Estruturas de dados para representar os polígonos e o serviço de corrosão foram definidas e, através da ferramenta Project Manager, automaticamente geradas para Skill++ e Java.

Os serviços responsáveis por efetuar a simulação e listar corrosivos aceitos foram implementados em Java utilizando o esqueleto de serviço gerado automaticamente. O serviço de corrosão compreende um pequeno código que prepara os polígonos na forma esperada pelo algoritmo de corrosão e o executa. O servidor pode utilizar entradas e saídas padrões (*pipe*), *sockets* ou HTTP, isso porque são fornecidas utilitários com o CODEC que implementam essas formas de conexão, sendo mínimas as adaptações necessárias no servidor.

Utilizando a linguagem Skill++ foi criado um menu (figura 6.9), formulários que capturam os dados necessários para fazer a comunicação e configurar os parâmetros necessários para corrosão e acessórios necessários para visualização (figura 6.10).

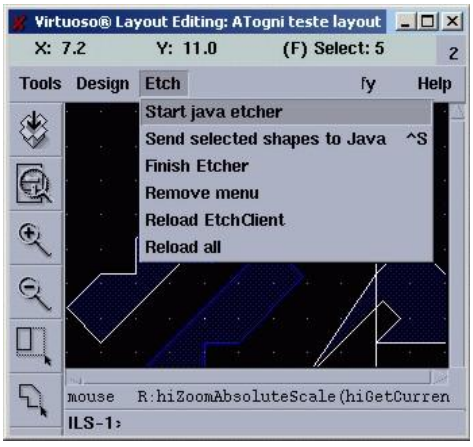

Figura 6.9: Edição de polígonos e menu de execução no Cadence.

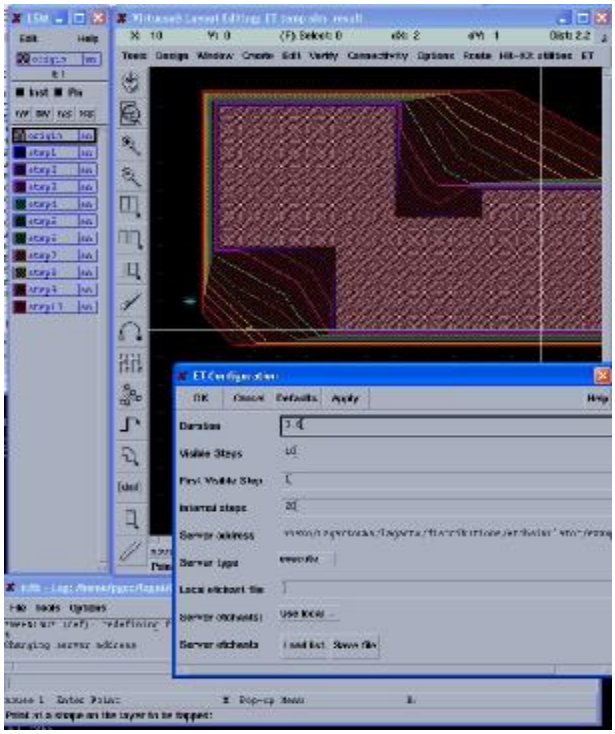

Figura 6.10: Resultado de uma corrosão no Cadence.

Visto que Skill++ não disponibiliza mecanismos de comunicação de rede como *sockets* e HTTP (exceto através de um dispositivo próprio que tem implementação somente em C), o programa executa diretamente o simulador, trocando dados por entradas e saídas padrão, ou executa utilitários que são responsáveis por estabelecer conexões via *sockets* ou HTTP, trocando dados com os utilitários pelas entradas e saídas padrões. Esses utilitários repassam e capturam dados do servidor. Após a execução da corrosão, o resultado pode ser visualizado dentro do Cadence. Para tanto foi necessário desenvolver um gerador de processos tecnológicos, que criam as camadas com suas características visuais, para assim, poder visualizar os polígonos resultantes da corrosão.

Clientes em Java e C++, deste mesmo estudo de caso também foram implementados como forma de validação do mecanismo nas duas linguagens. Porém, tais implementações não se beneficiam de forma igualmente relevante, pois não são vinculadas a um ambiente de CAD, assim não adicionando funcionalidades devidas a interação.

Esse estudo de caso mostra uma situação real de interoperação entre ferramentas independentes e os ambientes de CAD, onde elas podem cooperar e tirar proveito das funcionalidades disponíveis reciprocamente. Portanto, o objetivo de permitir a interoperabilidade entre ferramentas independentes e *frameworks* é atingido, habilitando a inclusão do algoritmo de corrosão ao fluxo de trabalho do *framework*.

6.2 Cell Design Flow - CDF

Um dos fluxos de trabalho mais utilizados para a geração de circuitos integrados é o fluxo *standard cell*. Esse método é baseado na utilização de células, que são pequenas funções lógicas que conectadas compõe o circuito desejado. O leiaute dessas células é projetado para que elas possam ser encaixadas lado a lado, assim oferecendo facilidades para o posicionamento das células e roteamento das entradas e saídas. O fluxo de projeto (figura 6.11) é basicamente composto das etapas de descrição do circuito, mapeamento tecnológico, posicionamento e roteamento de células, considerando que exista uma biblioteca de células previamente definida. A descrição pode ser feita em linguagens de alto nível ou portas lógicas. O mapeamento tecnológico efetua a tradução das equações lógicas de acordo com as células presentes na biblioteca. O posicionamento e roteamento criam o leiaute real do circuito final. Assim, aliando simplicidade e eficiência, o fluxo *standard cell* permite a criação de circuitos integrados e é ainda um dos fluxos de projeto mais utilizados comercialmente.

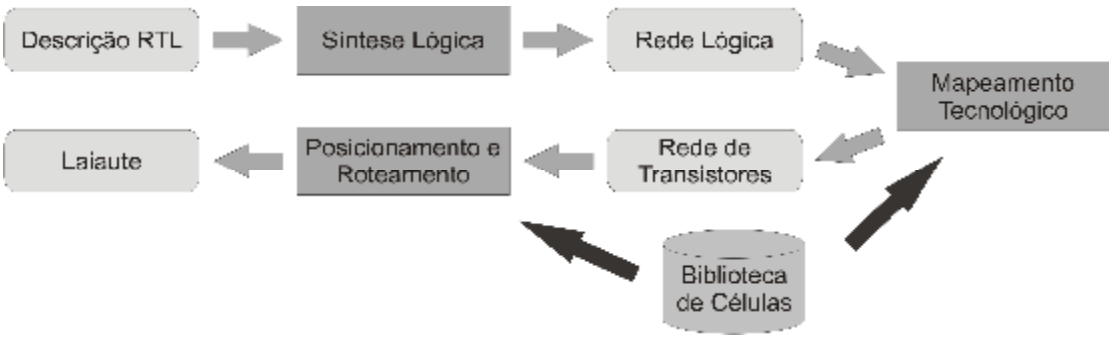

Figura 6.11: Fluxo *standard cell*.

Uma peça fundamental para o fluxo *standard cell* é a biblioteca de células (figura 6.12 e 6.13). Cada célula pode estar repetida milhares de vezes em um circuito, portanto, o projeto de uma célula tem um impacto direto e importante nos circuitos gerados por esse fluxo de projeto. Por esse motivo, o desenho das células é geralmente efetuado por profissionais dedicados somente a essa atividade. A acelerada evolução das tecnologias da microeletrônica impulsiona a evolução dos circuitos. Para poder adaptar um circuito ou parte dele para uma nova tecnologia é essencial criar previamente uma biblioteca para a nova tecnologia.

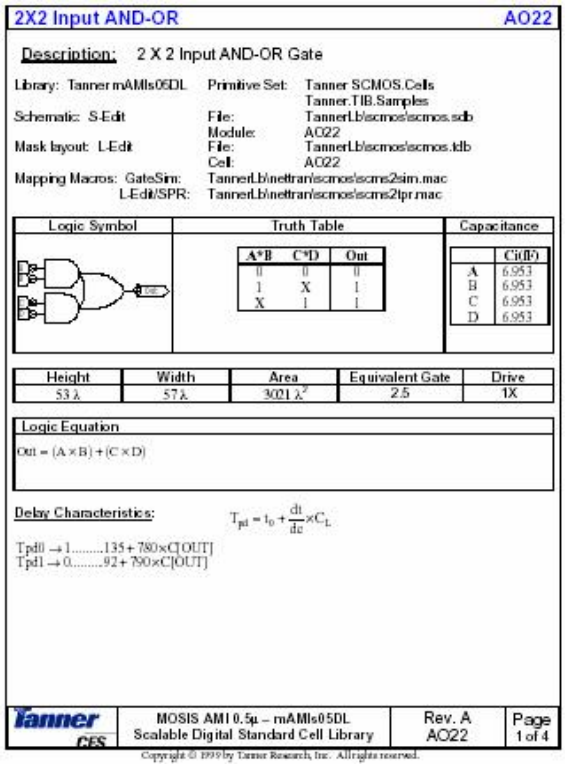

Figura 6.12: Descrição de uma célula (MTSMS035DL, 2005)

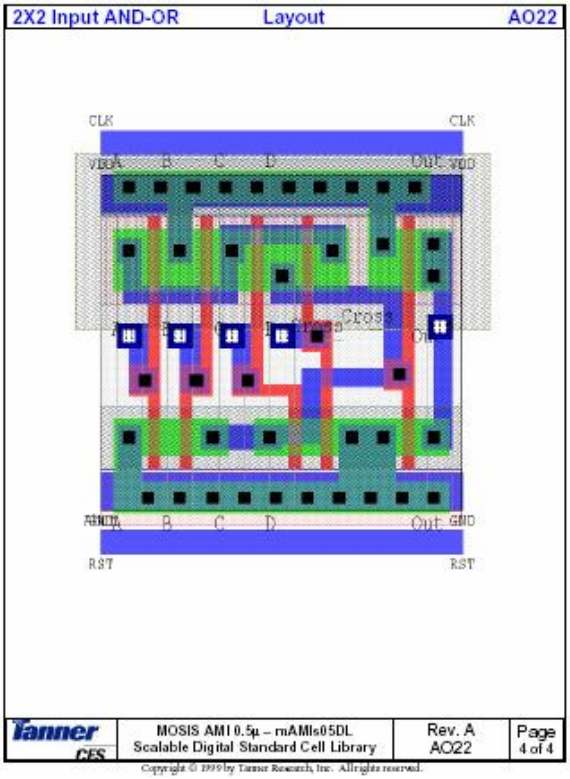

Figura 6.13: Leiaute de uma célula (MTSMS035DL, 2005)

Grosseiramente, pode-se compreender a confecção de uma célula por seu projeto lógico e físico. De forma similar, a automação pode ser obtida através de síntese lógica e síntese física das células. A síntese lógica envolve traduções de representação, minimização e manipulação lógica, entre outros procedimentos que visam obter a melhor representação lógica de determinada célula, ou o melhor conjunto de células. Nessa etapa é importante considerar a tecnologia alvo (p. ex. CMos, NMos, Transistores de passagem, lógicas dinâmicas, *dual rail*, etc), pois a função lógica terá uma representação física com características diferentes conforme a tecnologia.

A síntese física compreende as transformações de listas de transistores até o leiaute propriamente dito. Entre as técnicas envolvidas estão o uso de matrizes simbólicas de representação do leiaute, roteamento, posicionamento, dimensionamento de transistores, estimativas de desempenho, etc. Ferramentas comerciais como Prolific (2005) e Cadabra (2005) dedicam-se a geração de células para bibliotecas.

Visto que a evolução tem sido muito rápida e o desenho de células de biblioteca é, geralmente, feito de forma manual, o esforço para a migração tecnológica torna-se uma tarefa difícil. A ferramenta CDF (Cell Design Flow) (figuras 6.14 e 6.15) tem como objetivo fornecer um fluxo de geração de células que parte da descrição lógica e termina no leiaute da célula, ou seja, efetuando síntese lógica e física. O CDF é capaz de gerar, dada uma equação lógica, minimizações dessa lógica, listas de transistores simples e ordenadas, uma descrição de leiaute simbólico, e uma descrição CIF (Calma Intermediate Format) do leiaute. Essa ferramenta oferece funcionalidades que em geral não são encontradas nos ambientes de grande porte (como os da Cadence, Mentor Graphics e Synopsys). O intuito da ferramenta CDF é demonstrar de forma didática o fluxo de geração de uma célula, podendo ser utilizada também para gerar bibliotecas de forma rápida.

Figura 6.14: Diálogo de criação de célula.

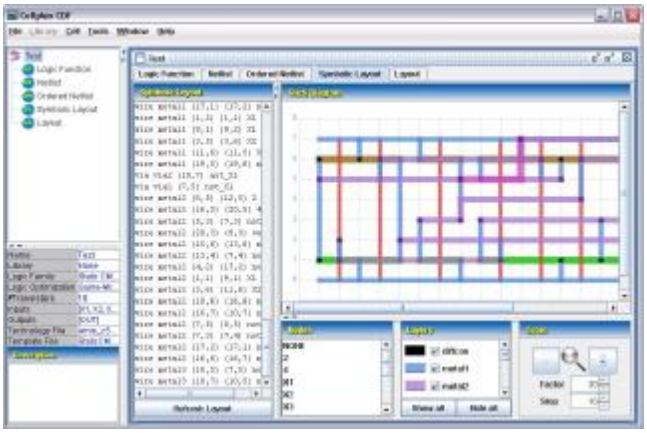

Figura 6.15: Resultado de criação de célula.

A integração do CDF com os ambientes das empresas Mentor Graphics e Cadence pode gerar benefícios para ambas as ferramentas. O CDF beneficiaria-se utilizando, por exemplo, os algoritmos de roteamento de leiaute para rotear os sinais internos de células geradas, podendo oferecer ao usuário uma opção ao roteamento interno disponível no CDF. Os ambientes beneficiam-se por, através do CDF, poderem gerar leiautes e netlists partindo de descrições lógicas. Além disso, o CDF oferece visualizações em corte e 3D, que não são providas pelas ferramentas Cadence e Mentor Graphics.

Este estudo de caso visa a disponibilização da geração automática de células feita pelo CDF dentro da ferramenta Cadence DFII. Para tanto, foram criadas em Skill++ menus, telas de configuração de células e conexão (figuras 6.16 e 6.18), habilitando o usuário Cadence a efetuar a geração de células de forma automática, já incluindo-as como células no formato interno do ambiente (figura 6.17). A adaptação do CDF para atuar como um servidor sem interface gráfica e com acesso via sockets é simplificada pelos mecanismos auxiliares de conexão providos por este trabalho. Como não é feita uma checagem completa de parâmetros, é comum que erros de processamento ocorram devido de entradas inválidas. Quando isso acontece o usuário Cadence recebe uma mensagem de erro que foi gerada no servidor.

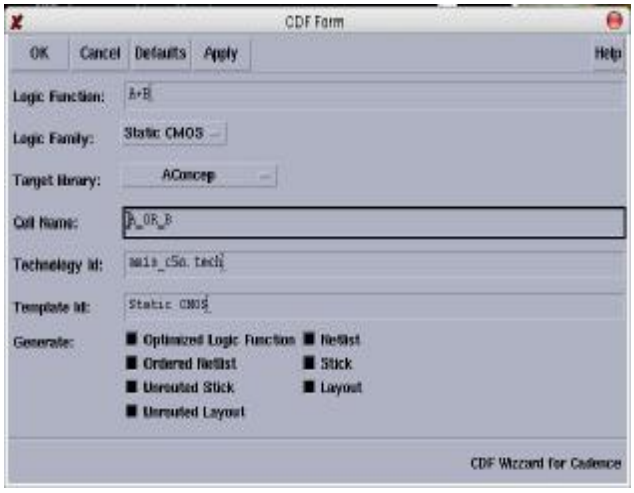

Figura 6.16: Diálogo de criação de célula (Cadence).

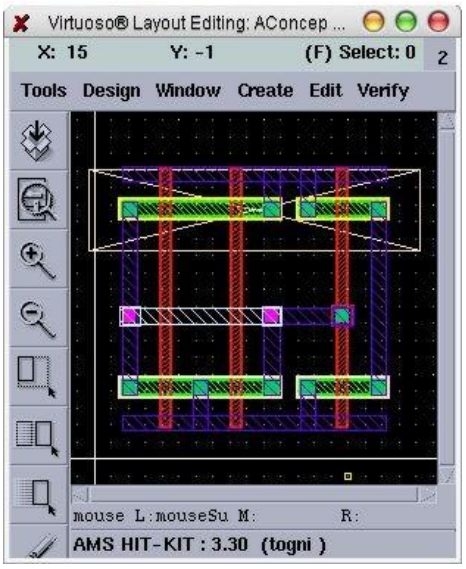

Figura 6.17: Célula gerada automaticamente.

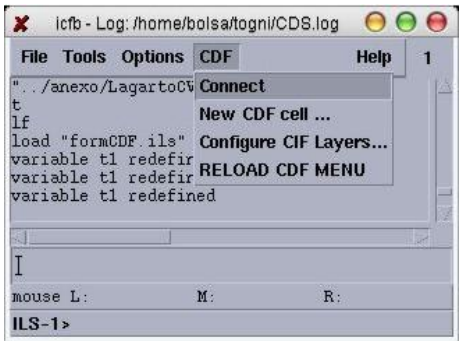

Figura 6.18: Menu de criação de células.

A implementação da parte cliente (Skill++) do estudo de caso é composta por uma interface gráfica amigável para captura de dados, correlação de camadas CIF (Calma Intermediate Format) e pela conversão dos dados para os formatos esperados pelo CDF. Os ambientes estudados neste trabalho oferecem a possibilidade de implementação de interfaces gráficas, esta não é uma tarefa dispendiosa quando se objetivam telas simples. O servidor, graças aos meios de comunicação desenvolvidos como utilitários, tem sua implementação ainda mais simplificada, bastando pequenos ajustes de estruturas de dados e a implementação dos serviços desejados. Foram implementados os serviços de geração do leiaute, mapeamento de camadas para fazer a correlação entre as camadas do ambiente com as geradas pelo CDF, serviço de listagem de tecnologias e modelos (um modelo define informações sobre a matriz simbólica utilizada pelo CDF).

O estudo de caso descrito nessa subseção demonstra a grande potencialidade que a interoperabilidade entre ferramentas especialistas e ambientes de CAD tem. A inclusão das células geradas no CDF ao Cadence permite que as células sejam editadas e possam ser

utilizadas em seu fluxo de geração automático. Considerando os benefícios acrescidos ao ambiente, ele passa a oferecer métodos de síntese lógica e física não presentes a priori.

6.3 Conclusão

Nesse capítulo foram apresentados estudos de caso e comparações do protocolo proposto com outros mecanismos de interoperabilidade. Os exemplos apresentados abordam apenas uma pequena gama de tópicos da microeletrônica, diversos outros trabalhos e ferramentas poderiam ser adaptados ao mecanismo para interoperabilidade.

O resultado obtido na integração das ferramentas estudadas é satisfatório. A utilização do mecanismo facilita a estruturação dos dados a serem trocados e a comunicação entre as ferramentas. Ainda assim, são necessárias adaptações e novas telas para os clientes. Apesar dos exemplos serem simples, já é possível verificar a importância do protocolo para as ferramentas de CAD de microeletrônica.

Os diversos meios de transporte disponibilizados juntamente com o protocolo permitem a utilização das ferramentas de formas muito diversas. A exploração dessas possibilidades tem um valor relevante para a validação e mesmo para exemplificação das formas de utilização do protocolo. A exploração de outros tipos de interação possíveis, ou seja, outros cenários, certamente acarretariam resultados interessantes e importantes para microeletrônica. O foco principal do trabalho é interoperar ferramentas independentes e *frameworks*, porém é possível interoperar entre duas ferramentas independentes e entre dois *frameworks* utilizando o mecanismo.

Em resumo, o protocolo mostrou-se eficiente no cumprimento das tarefas propostas para ele. O protocolo e suas ferramentas auxiliares tornam simples a interação entre ferramentas de CAD. O resultado dos estudos de caso é uma clara demonstração da potencialidade da proposta deste trabalho, permitindo de forma prática e eficiente a interoperação entre ferramentas independentes e *frameworks*.

7 CONCLUSÃO

Este trabalho apresentou uma proposta para a integração de ferramentas independentes e ambientes de CAD para microeletrônica. Conceitos relacionados à interoperabilidade fundamentam o trabalho desenvolvido e fornecem uma visão do estado da arte dessa área. Um protocolo de transporte de objetos entre ferramentas e ambientes de CAD para microeletrônica foi apresentado. Dois dos principais ambientes de CAD para microeletrônica e duas das mais utilizadas linguagens de programação são contempladas pela solução de interoperabilidade proposta. Ferramentas de auxílio para geração automática de artefatos necessários e a disponibilização de meios de transporte para utilização em conjunto com o protocolo também compõem este trabalho. Estudos de caso validam a utilidade e eficiência do protocolo proposto.

O Design *Framework* II e o Falcon *Framework* são dois importantes ambientes de CAD para microeletrônica. Sua importância deriva da ampla cobertura a tarefas envolvidas no projeto de chips, da qualidade das ferramentas providas e da adesão mercadológica que eles possuem. Esses ambientes disponibilizam as suas ferramentas de forma integrada, facilitando o fluxo de trabalho de um projeto de CI.

A evolução das tecnologias de microeletrônica viabiliza a criação de dispositivos de grande complexidade, por outro lado, o desenvolvimento de novas tecnologias de rede (como a Internet), permite novas formas de interação entre softwares. Surge, assim, a necessidade de interação entre os ambientes e ferramentas independentes. Assim, pode-se aproveitar as funcionalidades providas pelas ferramentas nesses poderosos ambientes, e pode-se estender os ambientes com novas funcionalidades através das ferramentas.

A exploração dos novos paradigmas e mecanismos de interoperabilidade é fundamental para tirar proveito real da capacidade de interação atualmente disponível. O protocolo apresentado neste trabalho é um exemplo disso. Embora existam diversos mecanismos de alto nível disponíveis para a simplificação das tarefas de interoperação, eles apresentam características diferentes, cobrindo necessidades diferentes. Critérios para caracterizar as diversas facetas de interoperabilidade foram apresentados, fornecendo ferramentas para melhor compreensão e avaliação das soluções existentes. Assim, uma avaliação das tecnologias atuais foi apresentada, discutindo brevemente suas qualidades e deficiências.

As soluções comerciais atuais para interoperabilidade não se adequam aos objetivos deste trabalho, pois não é possível através delas interoperar com os ambientes de CAD. Já as propostas específicas para microeletrônica possuem também suas deficiências. A principal proposta é a OpenAccess. Essa proposta é baseada em desenvolvimentos comerciais que foram abertos para acesso e desenvolvimento público. Diversas empresas importantes da área adotam o mecanismo proposto, porém duas características dificultam a

adoção acadêmica dele: o custo de aprendizado inicial e a disponibilidade de uma versão apenas em C++.

Uma forma unificada e simples de interação entre ferramentas e ambientes foi proposta através de um protocolo de comunicação e meios de comunicação para ele. Basicamente, o protocolo consiste de uma modificação de SOAP, tendo suas características adaptadas para possibilitar a implementação nas linguagens *script*. A utilização do protocolo é facilitada por ferramentas auxiliares de geração automática de código, diminuindo a quantidade de intervenção humana necessária para a adaptação das ferramentas ao protocolo. Diversos meios de comunicação são disponibilizados, permitindo que os usuários escolham a melhor forma de utilização sem a necessidade de codificação extra para utilizar os diferentes meios de transporte.

Estudos de caso foram utilizados para validar a proposta. Neles é possível observar a importância e relevância do resultado das integrações feitas. Duas ferramentas foram integradas aos ambientes de CAD utilizando o protocolo. Ambas lidam com áreas que não são cobertas pelos ambientes. Os resultados dos estudos são de grande relevância, pois agregam novas funcionalidades ao ambiente e permitem, dessa forma, explorar os benefícios resultantes dessa integração.

Deve-se observar, avaliando o protocolo apresentado, que este não resolve o problema de diferenças na modelagem que é abordado por Goldman (2000) e Young (2003). Tais diferenças são abordadas pela proposta da OpenAccess. O custo de oferecer a solução para algumas diferenças de modelagem é um tempo maior necessário para entender os modelos (curva de aprendizado) e utilizar a solução (KAHNG, 2003). Esse custo torna-se importante, por exemplo, quando o projeto em questão é acadêmico, pois em geral, tem uma grande rotatividade de participantes que, na sua maioria não possui conhecimento prévio dos modelos empregados. Em situações como esta, o protocolo proposto tem características mais apropriadas, apresentando um tempo de aprendizado menor.

Definitivamente, o trabalho apresentado nesse texto não encerra o assunto e discussões sobre interoperabilidade, mas sim, adiciona uma nova possibilidade que, na atual conjuntura da microeletrônica, pode trazer frutos importantes. Como resultado concreto deste trabalho, o grupo de pesquisa que desenvolve o projeto Lagarto já tem a possibilidade de integrar suas ferramentas aos ambientes profissionais de CAD. Outros grupos de pesquisa também podem utilizar a solução, pois a documentação e as ferramentas estão disponíveis publicamente.

Como forma de facilitar o uso do protocolo proposto, todos os códigos fonte referentes ao CODEC, ferramentas, meios de conexão e estudos de caso estão disponíveis para que alterações, aprimoramentos e futuros estudos sejam possíveis.

7.1 Trabalhos Futuros

O protocolo BICO foi planejado de forma a permitir que algumas modificações sejam feitas sem maiores alterações em sua estrutura. Entre elas está a utilização de estruturas não previamente definidas, modificações no sistema de seções (controle de validade de instâncias), diferentes codificações para as mensagens (ao invés de XML), e a extensão para outras linguagens.

A utilização de estruturas não previamente definidas, ou seja, sem possuir uma implementação para ela na linguagem alvo, é aceita em SOAP, porém o protocolo proposto não disponibiliza esse recurso na versão 1.0 por motivos de simplificação. Esse recurso

eliminaria a primeira etapa da utilização do protocolo, pois não seria necessário definir as estruturas utilizadas e por conseqüência não seria necessário definir que estrutura é aceita pelo serviço. Seria necessário, porém, verificar o conteúdo do parâmetro recebido pelo serviço de forma a garantir que este teria todos os dados necessários para sua execução.

A estrutura do protocolo permite que diferentes políticas de duração de seção, ou seja, controle de validade das instâncias de objetos criadas ou recebidas. Assim, o protocolo pode se adaptar de forma simples a outras necessidades.

Levando em conta que XML possui problemas com eficiência porque não é um formato compacto nem seguro, existe a possibilidade de estender o mecanismo para outros formatos de codificação. Assim, pode-se adicionar maior compactação e outras características intrínsecas de outros formatos. Compactação e criptografia não são contempladas pelo formato atual do protocolo, considerando que, no caso de necessidade, serão implementadas por outras camadas da pilha de protocolos ou por uma codificação a ser desenvolvida futuramente.

É também intuito do protocolo tornar possível a sua implementação em outras linguagens. Para facilitar as suas possíveis extensões, o código e a estrutura do protocolo são abertos. Mais do que descrever o funcionamento do protocolo em si e seus itens de trabalho, este texto visa descrever o funcionamento do CODEC permitindo a implementação conforme sugerido nesse trabalho, diminuindo o esforço de projeto do mecanismo.

Outros tópicos podem, também, ser explorados, como por exemplo:

- Definir e implementar limites de tipos e as verificações relacionadas;
- Permitir a utilização de mais de um valor ou estrutura como parâmetro de entrada e saída de uma requisição;
- Estudar detalhes e implementar adequação completa a SOAP;
- Implementar a verificação de elementos com nomes especiais como *request* e *response*;
- Criar mais utilitários e funções para facilitar o uso do mecanismo em cada linguagem, adaptando-se melhor as suas peculiaridades;
- Implementar outros tipos de codificação (não XML) para melhorar a eficiência (em termos de espaço);
- Aplicar a solução para outras ferramentas acadêmicas que possam usufruir dos benefícios da interoperabilidade entre ferramentas e ambientes.
- Estender a utilização do protocolo para outros ambientes e ferramentas importantes de CAD para microeletrônica como as ferramentas da Synopsys e Magic;
- Implementar mecanismos de segurança (autenticação, criptografia, etc);
- Obter licença e estudar profundamente a proposta OpenAccess, de forma a avaliar comparativamente a proposta deste trabalho e da OpenAccess.

REFERÊNCIAS

ABBAS, A. **Grid Computing**: A Practical Guide to Technology and Applications.[S.l.]: Charles River Media: Bk&CD-Rom edition, 2004.

BARNES, T. J. Skill: A CAD System Extension Language. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, DAC, 27.,1990, Orlando. **Proceedings**...New York: IEEE,1990. p. 266-271.

BELL, E. et al. **Fundamentals of Web Applications Using .Net and XML**. [S.l.]: Prentice Hall, 2002. 564 p.

BRAY, M. **Middleware**. 1997. Disponível em: <http://www.sei.cmu.edu/str/descriptions/middleware.html >. Acesso em: abril 2005.

BRGLEZ, F.; LAVANA, H. A universal client for distributed networked design and computing. In: DESIGN AUTOMATION CONFERENCE, DAC, 38., 2001, Las Vegas. **Proceedings**... New York: ACM, 2001. p. 401 - 406.

CADABRA. **Numerical Technologies**. 2005. Disponível em: <http://www.numericaltechnologies.com/ >. Acesso em: abril 2005.

CADENCE DESIGN SYSTEMS, Inc. **Cadence® Design Systems**. Disponível em: <www.cadence.com >. Acesso em: abril 2005.

CADENCE DESIGN SYSTEMS, Inc. **Cadence® SKILL Functions Quick Reference**. United States of America, September 2002.

CADENCE DESIGN SYSTEMS, Inc. **Cadence® SKILL Language User Guide**. United States of America, September 2002.

CAMPOSANO, R.; PEDRAM, M. Electronic design automation at the turn of the century: accomplishments and vision of the future. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v. 19, n. 12, p. 1401 - 1403, Dec. 2000.

COVENTOR**. MEMS Design Software**. 2005. Disponível em: <http://www.coventor.com/ >. Acesso em: abril 2005.

DAHALIN, Z. M.; WAHID, J. Workflow Interoperability Using Extensible Markup

Language**.** In: STUDENT CONFERENCE ON RESEARCH AND DEVELOPMENT, SCOReD, 2002. **Proceedings…**[S.l.]: IEEE Computer Society, 2002. p. 513-516.

DALPASSO, M.; BOGLIOLO, A.; BENINI, L. Specification and validation of distributed IP-based designs with JavaCAD**.** In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION,1999. **Proceedings...**Los Alamitos: IEEE Computer Society, 1999. p. 684-688.

DEITEL, H. M.; DEITEL, P. J.; NIETO, T. R.. **Visual Basic .Net:** como programar. [S.l.]: Makron Books, 2003.

DEITEL, P. J.; MCPHIE, D. C.; DEITEL, H. M. **Perl:** como programar. [S.l.]: Bookman, 2002.

ECSI: European Electronic Chips & Systems Design Initiative.
Disponível em: <http://www.ecsi.org/ >. Acesso em: abril 2005.

EMMERICH, W. Software Engineering and Middleware: A Roadmap. In:
INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 22., 2000. **Proceedings...** [S.l.:s.n.], 2000.

FIN, A.; FUMMI, F. A Web-CAD methodology for IP-core analysis and simulation.In: DESIGN AUTOMATION CONFERENCE, DAC, 37.,2000. **Proceedings**…[S.l.]: ACM, 2000. p. 597-600.

GAMMA, E. et al. **Design Patterns**. [S.l.]: Addison-Wesley, 1995.

GOLDMAN, R.; BARLESON, K. Tool Interoperability is Key to Improved Design Quality**.** In: INTERNATIONAL SYMPOSIUM ON QUALITY OF ELECTRONIC DESIGN, ISQED, 1., 2000, San Jose. **Proceedings...**Los Alamitos: IEEE Computer Society , 2000. p. 407.

GOODMAN, D.; MORRISON, M. **JavaScript Bible,** 5th ed.[S.l.]: John Wiley & Sons, 2004.

GRAHAM, P. **ANSI common lisp**. Englewood Cliffs: Prentice-Hall, 1996. 432 p.

HARRISON, D.S. et al. Electronic CAD *Frameworks*. **Proceedings of the IEEE**, New York, v. 78, n. 2, p. 393 - 417, Feb. 1990.

HOWES, T. **Understanding and Deploying Ldap Directory Services**. [S.l.]: Macmillan Technical Pub, 1998.

KAHNG, A. B.; MARKOV, I. L**.** Impact of Interoperability on CAD-IP Reuse: An Academic Viewpoint. In: SYMPOSIUM ON QUALITY ELETRONIC DESIGN, ISQED, 4.,2003 . **Proceedings...**Los Alamitos: IEEE Computer Society, 2003. p. 208-213.

KAZMIERSKI, T.; CLAYTON, N. A two-tier distributed electronic design framework. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DATE, 2002, Munich. **Proceedings**...Los Alamitos: IEEE Computer Society, 2002. p. 227 - 231.

KERNIGHAN, B. W**. C :** a linguagem de programação : padrão ANSI. Rio de Janeiro: Campus, 1990. 289 p.

KILGORE, R. A. Simulation Web Services with .Net technologies. In: SIMULATION CONFERENCE, 2002. **Proceedings**...Chesterfield, MO, USA: OpenSml & ThreadTec Inc., 2002. v.1, p. 841-846.

LISI. Levels of Information Systems Interoperability, C41SR Arquitecture Working Group, 1998. Disponível em: http://www.c3i.osd.mil/org/cio/i3/. Acesso em: dez. 2004.

LI, W.; CLIFTON, C. SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks. **Data & Knowledge Engineering**, Amsterdam, v.33, n.1, p 49-84, 2000.

LIEBOWITZ, S. J.; MARGOLIS, S. E. Network Externalities (Effects). In: **The New Palgraves Dictionary of Economics and the Law**. [S.l.]:MacMillan, 1998.

MACMILLEN, D. et al. An industrial view of electronic design automation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 19, n. 12, p. 1428-1448, Dec. 2000.

MARTINAZZO, F. et al. Etching Simulator for Bulk Micromachining in Java Platform. In: SBMICRO, 17., 2002, Porto Alegre. **Microelectronics technology and devices**: proceedings. Pennington: Electrochemical Society, 2002. p. 72-80.

MEMSCAP. 2005. Disponível em: < http://www.memscap.com/ >. Acesso em: abril 2005.

MENTOR GRAPHICS CORPORATION, Inc. **AMPLE for IC Flow Reference Manual**. United States of America, 2003.

MENTOR GRAPHICS CORPORATION, Inc. **AMPLE for IC Flow Users Manual**. United States of America, 2003.

MENTOR GRAPHICS CORPORATION, Inc. **Mentor® Graphics Corporation**. Disponível em: < www.mentor.com >. Acesso em: abril 2005.

MICROSOFT CORPORATION. **Why is interoperability important**. 2001. Disponível em: <http://www.microsoft.com/technet/interopmigration/ndam.mspx>. Acesso em: abril 2005.

MONO. **Mono Project**. 2005.

Disponível em: < http://www.mono-project.com/Main_Page >. Acesso em: abril 2005.

TANNER CONSULTING & ENGINEERING CENTER. **MTSMS035DL Digital Low Power Standard Cell Library For TSMC 0.35u Sub-micron Process.** 2005. Disponível em : < http://www.mosis.com/cell-libraries/scn035-std-cells/mtsms035dl.pdf >. Acesso em: abril 2005.

MURRAY, M. et al. Issues and Answers in CAD Tool Interoperability. In: DESIGN AUTOMATION CONFERENCE, DAC, 33.1996 . **Proceedings** ... New York: ACM, 1996. p. 509-513.

MUTO, C. A. **PHP e MySQL:** Guia Avançado.[S.l.]: Braspot. 2004.

OPENACCESS. 2005. Disponível em: < http://www.si2.org/openaccess/index.html >. Acesso em: abril 2005.

PROLIFIC. 2005. Disponível em: <http://www.prolificinc.com/ >. Acesso em: abril 2005.

QUARTUS II. **Altera**. 2005.
Disponível em: < http://www.altera.com >. Acesso em: abril 2005.

RETTBERG, A.; THRONICKE, W. Embedded system design based on webservices. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DAC, 2002, New Orleans. **Proceedings**... New York: ACM, 2002. p. 232 - 236.

RIBAS, R.P. **Etude et Conception de Microsystèmes Micro-usinés par la Face Avant en Utilisant des Technologies Standards des Circuits Intégrés sur Arséniure de Gallium**, 1998. PhD. Thesis, TIMA Laboratory, INPG-UJF-CNRS, Grenoble-France.

ROBINSON, S. et al. **Professional C#** (Programmer to Programmer).[S.l.]: Wrox, 2004.

SCHILDT, H. **Java 2:** The Complete Reference. 5th ed. [S.l.]: Osborne Media: McGraw-Hill, 2002.

SCOTT, M. L. **Programming Language Pragmatics.** San Francisco: Morgan Kaufmann, 2000. 856 p.

SENTURIA, S. D. **Microsystem Design**. [S. l.]: Springer, 2004.

SLEEPER, B. **Defining Web Services**. The Stencil Group. 2001.
Disponível em: < http://www.perfectxml.com/Xanalysis/TSG/WebServices.asp>.
Acesso em: abril 2005.

SPILLER, M.D.; NEWTON, A. R. EDA and the network. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1997. **Proceedings**...Los Alamitos: IEEE Computer Society, 1997. p. 470-476.

STROUSTRUP, B. **The C++ programming language**. [S.l.]:Addison-Wesley, 1997. 911 p.

SUTTON, P.R.; DIRECTOR, S.W. Framework encapsulations: a new approach to CAD tool interoperability. In: DESIGN AUTOMATION CONFERENCE, DAC, 35., San Francisco,1998 **Proceedings**..., New York: ACM, 1998. p. 134 - 139.

SYMPLICITY Inc. 2005. Disponível em: < http://www.symplicity.com >. Acesso em: abril 2005.

SYNOPSYS Inc. 2005. Disponível em: < http://www.synopsys.com >. Acesso em: abril 2005.

TOGNI, J. D. et al. Automatic Generation of Cell Libraries. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 15., 2002. **Proceedings**... Los Alamitos : IEEE Computer Society, 2002. p. 265-270

TOGNI, J. D. ; RIBAS, R. P. ; LISBOA, M. L. **Estudo de Linguagens Script e sua aplicação em microeletrônica**. Universidade Federal do Rio Grande do Sul , Instituto de Informática. Porto Alegre, Brasil, 2005. A ser publicado.

TOGNI, J. D.; KLOCK, C.; REIS, A. I.; RIBAS, R. P. E.T. – CAD Tool: new version of the 2D anisotropic etching simulator for front-side bulk micromachining in Java plataform. In: CONGRESSO IBEROAMERICANO DE SENSORES, 4., 2004, Puebla, México. **Proceedings**...[S.l.s.n.], 2004.

W3C: World Wide Web Consortium. **Extensible Markup Language (XML)**. 2005**.** Disponível em: < http://www.w3.org/XML/ >. Acesso em: abril 2005.

W3C: World Wide Web Consortium. **SOAP Version 1.2**. 2003. Disponível em: < http://www.w3.org/ TR/2003/REC-soap12-part1-20030624/ >. Acesso em: abril 2005.

WIRTHLIN, M. J.; MCMURTREY, B. **IP delivery for FPGAs using applets and JHDL.** In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DAC, 2002, New Orleans. **Proceedings**... New York: ACM, 2002. p. 2-7.

WOOD, G.; LAW, H. S. **Skill - An Interactive Procedural Design Environment**. [S.l.]: IEEE, 1986. p. 544-547.

WORKFLOW Management Coalition. 2005. Disponível em: <http://www.wfmc.org/>. Acesso em: abril 2005.

WORKFLOW Management Coalition. **The Workflow Reference Model**. 1995. Disponível em: <http://www.wfmc.org/standards/docs/tc003v11.pdf>. Acesso em: abril 2005.

YOUNG, P. et al. Evaluation of Middleware Architectures in Achieving System Interoperability. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, 14., 2003. **Proceedings**...Los Alamitos: IEEE Computer Society, 2003. p. 108-116.