Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica
Departamento de Semicondutores
Instrumentação e Fotônica

Dissertação de Mestrado

Proposta de um ambiente de simulação no domínio de fluxo de dados contínuos

Autor**:** **Jaime Alexandre Matiuso**

Orientador: **Prof. Dr. Edson Moschim**

Banca Examinadora:

Prof. Dr. Edson Moschim (DSIF/FEEC/UNICAMP)

Prof. Dr. Fúrio Damiani (DSIF/FEEC/UNICAMP)

Prof. Dr. Mário Jino (FEEC/UNICAMP)

Dr. Alberto Paradisi (Fundação CPQD)

Dr. Mário Tosi Furtado (Fundação CPQD)

Sumário

O objetivo deste trabalho é facilitar o entendimento de sistemas de comunicações ópticos. Para isto, modelos teóricos encontrados na literatura foram codificados e agrupados em uma biblioteca, a qual foi adicionada a um aplicativo desenhado para suportar, de forma simplificada, editoração gráfica e execução de objetos presentes em uma biblioteca associada. Assim, experimentos podem ser construídos de forma rápida, visual e objetiva em uma planilha gráfica, proporcionando agilidade na construção de conexões ópticas e acesso rápido às informações referentes ao sistema em estudo. A ferramenta computacional que suporta essas facilidades é denominada ambiente de simulação, a qual suporta a construções de malhas utilizando elementos modulares presentes na biblioteca, facilitando a visualização do domínio do problema e incrementando a percepção do usuário. Para a construção dessa ferramenta foram especificados os componentes necessários e um conjunto de diretivas gerais de projeto.

Abstract

The objective of this work is to facilitate the understanding of optical communication systems. In order to achieve this goal, theoretic models found in the literature were coded and grouped into a library, which was added to an application designed to support, in a simplified way, graphical editing and object execution from the associated library. As a result, experiments can be created in a fast, visual and objective way directly over a graphical spread sheet, making it easy to construct optical connections and providing quick access to information related to the system under study. The computational tool that supports these functionalities is denominated simulation environment. It allows mesh constructions using modular elements from the library, facilitating the visualization of the problem domain and increasing user perception. To design this tool, all the necessary components as well as a group of general directives were specified.

"... que os nossos esforços desafiem as impossibilidades, pois lembremo-nos que os maiores feitos da história foram obtidos daquilo que parecia ser impossível ..."

Charles Chaplin

Dedicatória

Dedico este trabalho àqueles que se sacrificaram além do possível por mim, em especial, a meus generosos pais, Jaime e Vanda (in memoriam), que me proporcionaram educação para chegar até onde estou.

Às minhas filhas, Lídia e Raquel, que contribuíram com um tempo precioso de sua atenção.

À minha amada esposa Magda, guerreira incansável, sem a qual nada disto seria possível, por sua paciência, amor, compreensão e encorajamento, enfim por estar sempre junto em cada palavra escrita aqui.

Agradecimentos

Agradeço primeiro a Deus, que me proporcionou a vida e forças para superar os momentos mais difíceis.

Aos amigos do CPqD: Dinho, Omar e Salomão, que me encorajaram nesta caminhada, e em especial ao Sandro, que muito me ajudou na compreensão do PcSimFO versão Windows. Ao amigo Raulison que muito me auxiliou na formatação desse trabalho.

Ao meu orientador e amigo, Prof. Dr. Edson Moschim, pela oportunidade de realização deste trabalho.

À minha família, que privou-se de estar comigo durante a realização deste trabalho.

.

Índice Analítico

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

1.1 Considerações Iniciais

Uma soma considerável de esforços tem sido despendida na especificação de ferramentas que facilitem atividades computacionais ligadas à confecção de aplicativos dedicados à prospecção de conhecimento e que possuam as seguintes características: crescente interatividade com o usuário, extensibilidade, facilidade de manutenção, aliada a execução otimizada **[12,34,35]**. Esses aplicativos, denominados ambientes de simulação, prestam-se a suportar o desenvolvimento de outros aplicativos, denominados simuladores, os quais são compostos por modelos que representam um sistema real. O processo de desenvolvimento dos modelos de simulação é um caso especial de desenvolvimento de software, pois o ato de realizar uma simulação está baseado no conceito de *aprender pela experimentação*, onde o aprendizado é obtido através da observação do comportamento do modelo construído. Esta forma de aprendizado facilita a visualização do domínio do problema estudado, na medida em que incrementa a percepção de quem realiza o estudo. A construção de modelos baseia-se no fato de que um problema simulado é observado como um número finito de conceitos, adequadamente agrupados através de afinidade contextual, formando unidades funcionais, as quais são capazes de realizar tarefas específicas.

1.2 Objetivos e Contribuições

Este trabalho tem por objetivo dar continuidade ao projeto denominado PcSimFO, desenvolvido neste departamento. Este projeto consiste em um ambiente de simulação visual projetado para suportar atividades de prospecção em sistemas de comunicações ópticos, desenvolvido através da contribuição de vários trabalhos anteriores,

sendo a última etapa, a construção de uma versão suportada pelo ambiente MS-Windows, a qual trouxe consigo características positivas e outras indesejáveis. O projeto foi desenvolvido com as ferramentas disponíveis, as quais ocasionaram uma forte dependência entre o que foi construído e o ambiente MS-Windows. Esta dependência definiu a forma de construção do ambiente e dos modelos associados, de modo que para manutenir o ambiente ou construi-se um novo modelo exigi-se o conhecimento da estrutura interna de funcionamento do MS-Windows. O foco deste trabalho é cria uma nova versão do ambiente, corrigindo as atuais deficiências e mantendo suas virtudes, através de uma estrutura interna flexível e neutra em relação à plataforma utilizada. As definições do escopo do problema, a abordagem utilizada e a apresentação do projeto PcSimFO serão apresentadas nos próximos capítulos.

1.3 Metas de projeto

As metas estabelecidas para este trabalho foram elaboradas com a finalidade de solucionar as deficiências apresentadas na última versão e adicionar novas características que possibilitem sua expansão. As metas definidas para este trabalho são as seguintes:

1. **Suportar edição e execução de simulações de sistemas ópticos**. O ambiente construído será responsável pela edição de simuladores realizada de forma visual (gráfica) e por sua execução. Os simuladores construídos representarão sistemas de comunicações ópticos do tipo ponto-a-ponto.

2. **Portar o projeto para ambientes do tipo Unix**. Os ambiente do tipo Unix, em especial o Linux, apresentam-se mais adequados ao desenvolvimento de aplicativos que necessitam de alta eficiência em cálculos e ofereçam um ambiente flexível dotado de ferramentas não proprietárias.

3. **Modularizar e formalizar o ambiente**. O ambiente será constituído por um conjunto de módulos funcionais, denominados componentes estruturais, os quais serão planejados para operarem como entes isolados dentro de uma lógica contextual. A

utilização desta sistemática de construção implica na definição formal de interfaces e métodos de acesso para os componentes e proporciona uma arquitetura dotada de alta expressividade sistêmica.

4. **Prover uma arquitetura de ambiente neutra em relação à plataforma**. Os conjuntos de componentes estruturais que irão compor o ambiente constituirão uma arquitetura proprietária capaz de refletir de forma abstrata e transparente, o suporte oferecido pela plataforma utilizada. Dessa forma, a complexidade de acesso à plataforma ficará contida em componentes específicos, o que dará maior condição de portabilidade ao projeto. Um exemplo claro está associado à utilização do suporte visual, que é nativo de uma determinada plataforma e será implementado por um componente.

5. **Formalizar uma metodologia de produção de modelos**. Será desenvolvida uma arquitetura para padronizar a produção de modelos que definirá os recursos disponíveis aos modelos de forma abstrata, apresentando um número reduzido de comandos que facilitem sua utilização. Toda ação que envolva algum tipo de apresentação visual será definida através de uma linguagem formal proprietária, a qual obscurecerá detalhes de execução e comportamento associados.

6. **Definir uma arquitetura portável de biblioteca**. O ambiente terá uma biblioteca de modelos associada, representada por um componente na arquitetura do ambiente, tendo como características: uma arquitetura adequada à integração ao ambiente, será portável para outras plataformas e suportará a atividade de produção de modelos. O processo de criação de modelos será uma atividade realizada a parte, não estando vinculada ao desenvolvimento do ambiente de simulação, de forma que o construtor de modelos necessitará conhecer apenas a estrutura aplicada à biblioteca e as ferramentas de inclusão e remoção de modelos no ambiente. O ato de inclusão e exclusão de modelos na biblioteca será realizado através de um aplicativo apropriado, simplificando esse ato.

7. **Esboçar um plano de atualização do sistema**. A arquitetura adotada para o sistema deverá ser flexível e suportar futuras atualizações e expansões. Qualquer modificação futura deverá constar no documento *plano de atualizações* quando do momento de

definição da arquitetura. Desse modo, a evolução do ambiente deverá estar devidamente documentada.

1.4 Organização deste trabalho

Os capítulos apresentados a seguir apresentam as idéias básicas a respeito da construção de um ambiente de simulação. O capítulo **Revisão Bibliográfica** apresenta conceitos básicos necessários ao entendimento do trabalho. O capítulo **Arquitetura Sistêmica** apresenta a estrutura interna do ambiente. O capítulo **Estudo de Caso** apresenta a construção de modelo pormenorizada e os resultados obtidos com sua execução. O capítulo **Conclusões** apresenta as considerações finais a respeito do trabalho.

Capítulo 2

Revisão Bibliográfica

2.1 O ambiente de simulação PcSimFO versão Windows

Os sistemas de comunicações ópticos apresentam grande dificuldade de avaliação quando se usa apenas métodos analíticos, sendo necessário o uso de técnicas computacionais, semelhantes à simulação, para quantificar o desempenho desses sistemas **[40,41]**. Para a realização dessas tarefas, exigi-se grande capacidade computacional do equipamento utilizado. Vários equipamentos de diferentes classes (estações de trabalho do tipo Unix, supercomputadores, mainframes e microcomputadores) podem ser utilizados. O microcomputador presta-se muito bem a esta tarefa, apresentando baixo custo. Estes conceitos motivaram o projeto PcSimFO (Personal Computer Simulation Fiber Optics).

O PcSimFO é um ambiente gráfico de operação amigável, utilizado para simulações de sistemas de comunicações baseados em fibras ópticas, possibilitando analise e projeto de sistemas. A configuração dos sistemas é realizada através de um diagrama de blocos posicionados em uma planilha gráfica. A execução deste diagrama é realizada no mesmo nível de apresentação, oferecendo interatividade durante esta tarefa e apresentando os resultados em forma gráfica ou textual.

2.1.1 Requisito básico

Um aspecto decisivo na escolha da topologia utilizada para projeto de um ambiente visual é a interface gráfica a ser utilizada. Um aplicativo que tenha como objetivo oferecer facilidades de criação e reconfiguração de topologias de sistemas de comunicações

e prover meios para análise de resultados certamente terão como requisito básico qual interface gráfica será utilizada.

2.1.2 O ambiente de simulação

A última versão do PcSimFO foi escrita na linguagem C para a plataforma MS-Windows e projetado para ser executado em computador IBM-PC ou compatível. A preocupação com uso de uma interface gráfica de operação amigável conduziu a escolha do MS-Windows como padrão gráfico, baseado na facilidade de programação visual e a diminuição de tempo despedido no aprendizado e operação desta ferramenta. Com o objetivo de prover máxima flexibilidade, o ambiente utiliza-se de uma estrutura baseada em quatro componentes, como mostra a figura 1 expressa na linguagem formal UML **[14,15]**. Os componentes básicos são:

- **Biblioteca de modelos**. Este componente armazena os modelos matemáticos previamente codificados representados por blocos funcionais na planilha simulação.

- **Sistema configurador**. Este componente permite especificar o sistema a ser simulado, bem como alterar valores aplicados aos parâmetros do modelo.

- **Executor de simulação**. Este componente supervisiona a execução da simulação e promove: o interfaceamento entra as várias partes do ambiente, a geração de sinalização de controle e armazenamento de sinais de amostragem ao longo da execução do sistema.

- **Pós-processamento**. Este componente possibilita a execução e visualização de resultados originados em processamento eletivo. Este processamento utiliza os resultados obtidos nos processamento principal, gerando informações adicionais.

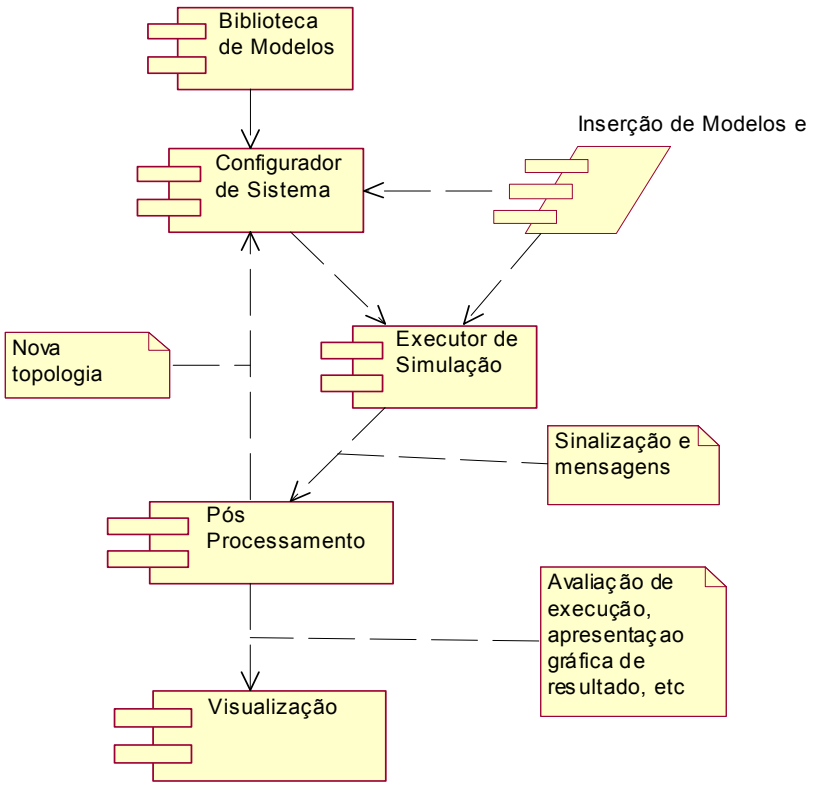

Figura 1: Estrutura funcional do PcSimFO versão Windows

2.1.3 Análise de deficiências

A construção da versão MS-Windows foi um marco importante na evolução do ambiente PcSimFO. A inclusão de facilidades visuais aumentou significativamente a interatividade oferecida pelo ambiente. Este processo demandou grande volume de trabalho, pois as estruturas das versões anteriores não foram planejadas para uma transição para um ambiente visual. O processo de transcrição dessas estruturas para um ambiente visual foi realizado com as ferramentas de desenvolvimento disponíveis na época, as quais estavam num estágio pouco evoluído de abstração funcional. Esta característica veio a contaminar a versão MS-Windows do PcSimFO em diversos aspectos. O ambiente MS-Windows opera através de troca de mensagens entre seus componentes e todo código adicional segue esta mesma orientação. Este relacionamento pode ser observado quando, por exemplo, uma caixa de diálogo é exibida, existe a geração de uma mensagem de

exposição endereçada uma função de resposta, a qual é responsável por atualizar os valores dos atributos apresentados. Quando ocorrer o ocultamento da caixa, será gerada uma outra mensagem para a função de resposta recuperar os valores presentes nos atributos da caixa e armazená-los adequadamente. Esta forma de operação é eficiente, porém inadequada para abrigar estruturas associadas a um ambiente de simulação, na medida em que este arranjo não favorece uma estruturação formal de um aplicativo desenvolvido dessa maneira, pois as funções associadas a um aplicativo não ficam claramente definidas, prejudicando de forma acentuada a modelagem das estruturas de suporte. Essas características constituem a principal fonte de problemas relacionada ao PcSimFo.

A indefinição das funções associadas ao ambiente ocasiona uma baixa abstração funcional, o que dificulta a identificação dos macroblocos sistêmicos, dispersos na estrutura MS-Windows. A dispersão dos macroblocos estende-se até os níveis básicos do ambiente, como exemplo, a geração de caixa de texto, utilizadas para apresentar informações relevantes, é realizada por uma ferramenta diferente daquela utilizada para construção do ambiente. Assim, uma caixa de diálogo é construída à parte e posteriormente adicionada ao conjunto principal. Essa forma de construção constitui uma barreira que impede que consistências sejam elaboradas a fim de tornar o trâmite de dados entre uma caixa de diálogo e a estrutura central mais robusta.

A estrutura imposta pelo MS-Windows ainda vai mais longe, na medida em que dificulta a construção dos modelos utilizados para compor simulações. O código associado ao modelo deve amoldar-se as estrutura imposta, a qual mostrou-se inadequada para produção de modelos e em decorrência disso, o construtor de modelos deve conhecer a arquitetura do MS-Windows, o que se constitui um forte aspecto desestimulador para futuros construtores de modelos adotem o ambiente como ferramenta de desenvolvimento de novos modelos para simulações.

2.2 Conceitos associados a simuladores

Do ponto de vista lógico, um simulador deve representar fielmente uma idéia dentro de um contexto composto por elementos representando aglomerados de conceitos, que unidos, perfazem unidades funcionais. As relações entre as unidades representam informações trocadas ou dependências implícitas. Um simulador deve oferecer uma forma de organização de conceitos semelhante aos estudos preliminares, normalmente expressos em rascunhos organizados em forma de gráfica. Assim, a capacidade de expressão gráfica de um simulador deve possibilitar a tradução de um rascunho. Pela dinâmica exigida em um ambiente de simulação é tido como certo que o conjunto de requisitos estabelecidos inicialmente para o simulador será incompleto, necessitando a inclusão de novos aspectos à especificação inicial. O paradigma adotado para expressar um simulador é a **Orientação a Objetos [7,11]**, através do qual estímulos podem avaliar o desempenho e caracterizar o comportamento de um objeto. A validação é obtida após submeter o objeto a uma quantidade de estímulos suficientes para abranger todo o escopo de utilização do objeto. O modo de acesso aos atributos do objeto deve ser abstrato, não revelando detalhes de operação interna.

2.2.1 Simulação conceitual

Um simulador é uma ferramenta utilizada para exercitar um modelo que represente um sistema real **[1]**. Esta afirmação é o ponto de partida para a definição de alguns conceitos relativos à produção de simulações. Um programa genérico que constitua uma simulação é uma peça computacional desenvolvida para resolver um problema específico, com um conjunto de conceitos identificados da forma rudimentar, com organização deficiente, parâmetros associados obscuros, estrutura implementada de forma elementar, sem nenhuma formalização. Nesta fase, o referido programa representa um conjunto de conceitos denominado **simulação conceitual**, apresentado na figura 2. Um exemplo disto são os métodos de entrada e saída de dados implementados sem nenhuma checagem de consistência. Isto dificulta a reutilização do aplicativo em uma situação onde

se deseja a composição deste modelo com outros. A impossibilidade de reutilização de um aplicativo estabelece uma situação onde a produção isolada não agrega valor, pois utiliza uma arquitetura particular. Como resultado, o conceito contido no modelo anterior será recriado em uma nova estrutura. Este fato define uma região limite entre aplicativos até então chamadas simulações e uma nova arquitetura, onde se podem desenvolver modelos de forma organizada, estável e formal.

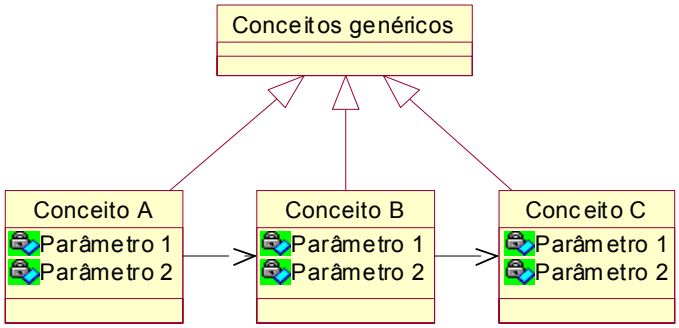

Figura 2: Simulação conceitual

Em uma arquitetura organizada, a criação de modelos é realizada em alto nível de abstração, fornecendo respostas mais precisas na medida em que se aumenta o volume de detalhes, podendo incluir outros modelos, o que define a granularidade que será aplicada ao modelo final. Um aplicativo computacional baseado nesta arquitetura será responsável por prestar suporte às necessidades operacionais relacionadas à execução do modelo, e dessa forma, auxiliará as atividades de pesquisa relacionadas ao modelo desenvolvido, proporcionando um mecanismo de reuso de conhecimentos e formalizando uma postura de produção intelectual.

2.2.2 Metodologia de estudo

O estudo de uma simulação é orientado por uma seqüência de passos que tem por objetivo adotar uma abordagem para a identificação do problema que se deseja resolver **[4]**. As abordagens usualmente encontradas estão baseadas em:

- **Otimização** é o critério para o julgamento da qualidade do sistema baseado em parâmetros observáveis. A maximização de resultados associados aos critérios relevados define a qualidade total.

- **Planejamento** é o procedimento adotado para extrapolar em bases futuras a ocorrência de padrões procedimentais que ilustrem o funcionamento do sistema.

- **Análise** é a possibilidade de entendimento do relacionamento "causa-e-efeito" apresentado pelo sistema.

- **Comparação** é a possibilidade de visualizar um número razoável de comparações em termos de uma definição de um conjunto de métricas.

2.2.3 Abrangência

A abrangência de uma simulação **[4]** é expressa pela definição do problema, a qual descreve os objetivos do estudo e suas implicações. Em decorrência disto, identificam-se às suposições feitas, as variáveis de entrada e saída, os requerimentos para atividades realizadas sobre as coleções de dados, a precisão e resolução dos resultados obtidos nos experimentos realizados através do modelo produzido. As metas de uma simulação definem os objetivos básicos, os quais foram formulados mediante expectativas iniciais, obtidas empiricamente. Se as metas forem muito amplas, o experimento pode tornar-se inexeqüível. Assim, as metas não devem cobrir apenas o que esta sendo modelado, mas

também tornar claro o assunto abordado frente à comunidade a qual será aplicada futuramente.

De uma maneira geral, os modelos são desenvolvidos de modo a obscurecerem seu conteúdo, apresentando apenas um mapeamento que abrange as relações entre os conjuntos de entradas e saídas, planificando o modo de atuação ou efetivamente a relação de transformação. Se o modelo foi projetado para suportar decisões, esta característica estará associada a um conjunto de parâmetros do sistema, os quais tem como responsabilidade, a condução do processamento. O objetivo da simulação neste caso será de maximizar ou minimizar o número e o conteúdo das variáveis, com as quais definem o conjunto de variáveis de decisão.

2.2.4 Análise de sistema

Esta fase é responsável por estabelecer uma base para o desenvolvimento de simulações, onde serão estabelecidos os componentes e a dinâmica do sistema **[11]**. Os componentes podem ser categorizados em entidades, eventos ou recursos. A dinâmica define como os procedimentos adotados serão aplicados e como serão efetuados as interações com outros componentes. Em modelos simples, as entidades são objetos elementares de aspecto homogêneo. Em casos complexos, as entidades podem possuir um conjunto particular atributos que individualizam sua estrutura. Os recursos devem ser dimensionados a priori, pois são objetos de uso compartilhado por outras entidades, definindo o grau de compartilhamento do recurso, sendo seu uso descrito em termos quantitativos e temporais.

2.2.5 Estimação de parâmetros

Para a obtenção de um modelo convincente, os parâmetros que quantificam o efeito de diferentes elementos sobre o modelo devem obedecer à relação de efeito que causariam se fossem aplicados ao sistema real **[4,5]**. Dessa forma, a estimação de

parâmetros pode funcionar como um ajuste compensatório das expectativas iniciais que se configuram como ilusórias ou imprecisas. Este fato é comum no desenvolvimento de modelos, pois a composição está em andamento, e necessita de experimentos que comprovem as proposições teóricas. Para contornar este problema, são relacionados alguns procedimentos úteis como a pesquisa de informações em poder da comunidade alvo, especificações correlatas e planas de estimativas elaboradas anteriormente.

2.2.6 O significado do reuso

Pôr reuso endente-se utilizar objetos já existentes, disponíveis em uma biblioteca, minimizando o desperdício de tempo e contribuindo para a padronização do modo de criação de experimentos **[13]**. Para que o aspecto do reuso seja efetivamente instaurado, é imprescindível uma documentação detalhada a cerca dos objetos utilizáveis, ou seja: conceitos representados no objeto, referências bibliográficas, identificação do objeto, função contida no módulo, aplicabilidade, escopo de validade, definição e variáveis paramétricas. Um exemplo de reuso é apresentado na figura 3.

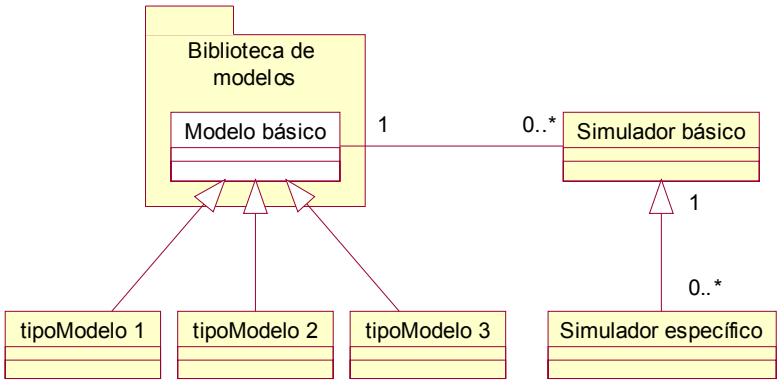

Figura 3: O reuso

2.3 Elementos de um ambiente

Serão apresentados três conceitos utilizados de forma missiva que devem ter seu significado definido: simulação, simulador e ambiente de simulação. **Simulação** é um conceito genérico, onde dada uma necessidade imediata, construía-se um programa suficiente para satisfazer alguns requisitos primários, não aplicando conceitos de estruturação funcional ou reuso de código. Assim, qualquer peça computacional pode representar uma simulação. **Simulador** define uma estrutura computacional para representar um conjunto de modelos conectados e formalizados, para serem utilizados e suportados em um ambiente específico. **Ambiente de simulação** é um aplicativo computacional construído para suportar a execução de estruturas denominadas simuladores, obscurecendo aspectos gerencias e proporcionando comunicação transparente entre elementos.

2.3.1 Requisitos básicos

Por requisitos básicos entende-se o conjunto de requisitos elementares que compõem uma ferramenta para construção de simulações **[34,35,38]**. Estes requisitos expressam a dinâmica exigida de um ambiente e podem ser resumidos nos tópicos a seguir:

- **Modularidade construcional** é a possibilidade de construção de aplicativos com representação funcional, no qual um sistema não monolítico permite elaborar um arranjo estrutural onde as responsabilidades ficam distribuídas, facilitando a avaliação dos requisitos de forma localizada.

- **Aplicação de interfaces** estabiliza os arranjos, de modo que as atualizações realizadas são restritas e preservam as diferenças construcionais e reforça os seguintes aspectos: facilidade de manutenção, ampla conectividade, eleva a coesão funcional, diminui o acoplamento interpartes e individualiza a evolução.

- **Parametrização funcional** são informações relacionadas com a adequação do modelo contido na simulação para uma situação específica.

- **Execução de simulação** é o centro do processo onde são executados os cálculos.

- **Pós – processamento**, é a fase destinada à apresentação e avaliação da documentação resultante.

- **Entidades** representam objetos ativos dentro do sistema, possuindo estreita afinidade com eventos gerados pelo sistema, respondendo a eventos de forma estipulada, desencadeando a geração de eventos futuros. As entidades possuem atributos de estado que representam informações relevantes dos processos realizados, os quais permitem diferenciar uma instância entre várias. A evolução de uma instância pode ser expressa mediante um diagrama de transição de estado.

- **Recursos** são facilidades a disposição do sistema, como ilustra a figura 4. Em sistemas que representam dispositivos reais, um recurso está na forma reusável, podendo ser utilizado repetidas vezes. Independente de sua origem, os recursos compartilham uma característica importante: possuem capacidade limitada. Assim, as preocupações relacionam-se com a sua disponibilidade, quantidade, estado atual. Os recursos possuem grande dinâmica e por esta razão são alocados em tempo de execução em uma simulação. A alocação de recursos está condicionada a uma unidade gerenciadora, que arbitra sua distribuição, condição de acesso e monitora os limites estipulados referentes a sua capacidade.

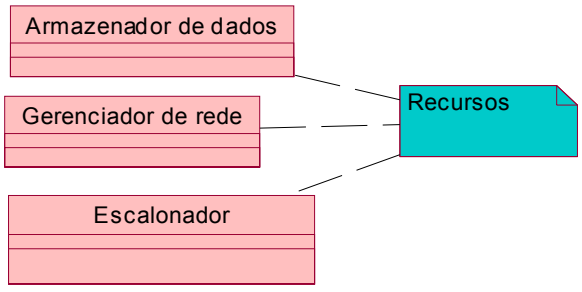

Figura 4: Exemplo de recursos disponíveis

2.3.2 Estruturação formal

A estrutura interna do ambiente segue uma especificação formal **[3,5]**, onde se defini suporte de forma abrangente e suficiente às facilidades de comunicação entre modelos, visualização de resultados, flexibilidade na definição de parâmetros, capacidade de expressão gráfica, gerenciamento geral, recursos de impressão, acesso a dispositivos de memória de massa. Sendo assim, a utilização de um ambiente de simulação sugere a clareza de que **o fato de se estar realizando uma simulação, não implica, necessariamente, que se está executando esta mesma simulação em um ambiente de simulação**. Desse modo, a realização de tarefas básicas deve atender aos seguintes itens:

- **Formalização**, os modelos desenvolvidos serão constituídos por um conjunto definido de métodos, os quais abrigam as funcionalidades básicas válidas para qualquer modelo. Desse modo, o ambiente na sua função de gerenciamento, pode ativar e sincronizar as atividades executadas pelos modelos. Os dados gerados devem obedecer a uma definição genérica de estrutura para comunicação responsável pela troca de informações.

- **Organização de conhecimentos**, um modelo pode ser particionado em modelos menores. Esta divisão deve ser realizada de forma a encerrar conceitos que particularizem uma função, a qual compõem o modelo maior.

- **Facilidade de reuso de conhecimento em outros simuladores**, a organização de conhecimentos deve ser realizada de forma que o nível de particionamento de um modelo original seja feito até o ponto em que os modelos resultantes sejam adequados a serem utilizados para compor outros simuladores.

- **Incentivar a evolução de novos simuladores** significa adquirir conhecimentos mais profundos utilizando toda a base construída, o que minimiza o esforço produtivo.

2.3.3 Domínio de uma simulação

Um conceito importante para caracterizar uma simulação está ligado ao domínio de execução, isto é, qual o modo de execução adotado para avaliar os objetos pertencentes a uma simulação. A seguir, serão apresentados os domínios de execução mais usuais.

2.3.3.1 O domínio SDF

O domínio SDF (Syncronous DataFlow), é uma técnica de avaliação muito utilizada, sendo um modelo bastante maduro, derivado de uma especialização do modelo computacional **DataFlow [30,31]**. Dessa forma, o domínio SDF pode ser identificado dinamicamente por uma entidade depende da aplicação de um fluxo de dados em sua entrada. Por ser um método de avaliação seguro e simples de ser implementado, foi utilizado como base para o desenvolvimento de outros domínios. Aqui a notação de tempo não existe, isto é, o avanço do tempo não está presente, ao invés disto, apenas a passagem contínua do fluxo de dados é percebida, sendo o período compreendido entre o início e fim de uma avaliação completa denominado **iteração**. O SDF é um modelo estático, ou *statically schedule* **[4]**, onde a ordem de avaliação é determinada de forma estática em tempo de compilação. Assim, este domínio toma como princípio a seqüência de avaliação dos objetos como fixa e a resolução dos predicados serão realizados em tempo de compilação. Este sequenciamento é obtido através de algum conceito pré - definido como: a ordem de criação dos objetos, dependências explícitas ou uma ordem de prioridade arbitrária. Este domínio é utilizado para determinar o desempenho problemas estáticos, onde se deseja caracterizar um determinado comportamento. Neste método de avaliação, a interação intermódulo possui importância relativa, pois o objetivo da simulação é

reconhecer o comportamento individual e não o desempenho do conjunto. A forma de acesso aos recursos é apresentada na figura 5.

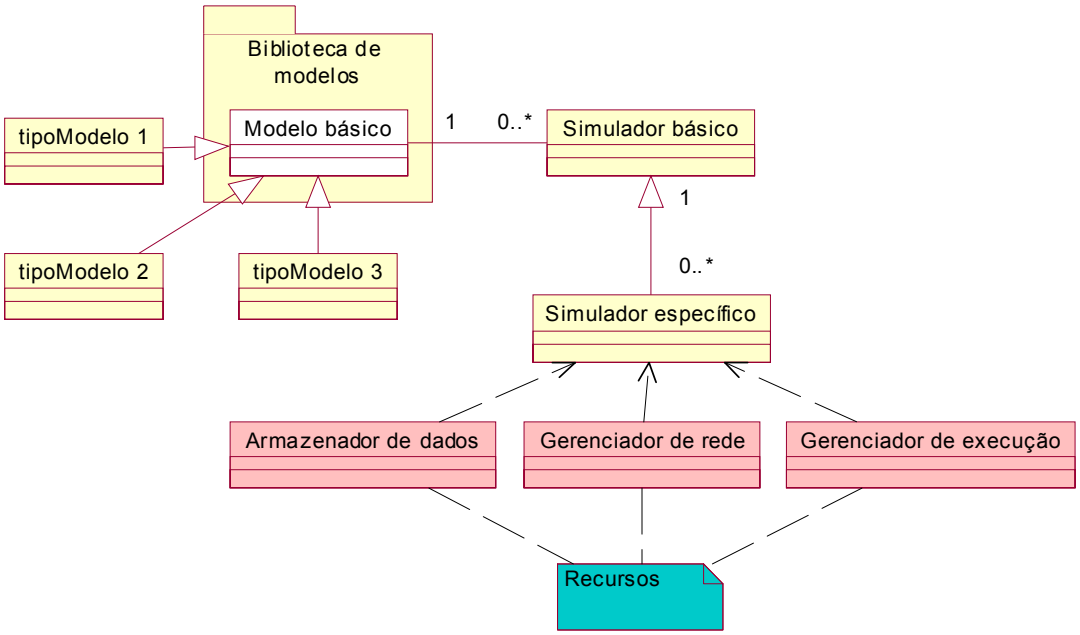

Figura 5: O domínio SDF

2.3.3.2 O domínio DE

O domínio DE (Discrete *Event*) é um método de avaliação utilizado para simulações onde a condição de fluxo de dados contínuo não é aplicada **[2,4,6,32,33,36]**. Neste método a avaliação é baseada na ocorrência de eventos assíncronos, sendo possível à percepção da passagem do tempo. Estas características fazem do domínio DE o preferido em simulações envolvendo problemas de engenharia. O mecanismo de avaliação supõe que os objetos pertencentes a um simulador possuem a capacidade de troca de mensagens, sendo o ato de recebimento de uma mensagem o fato que inicia a avaliação do receptor. O mecanismo DE prove um fila de execução para os objetos selecionados, sendo a execução condicionada a disponibilidade de recursos associados, frente a um cenário dinâmico.

Desse modo, este método de avaliação é indicado para situações onde à dinâmica de contexto é importante. A forma de acesso aos recursos é apresentada na figura 6.

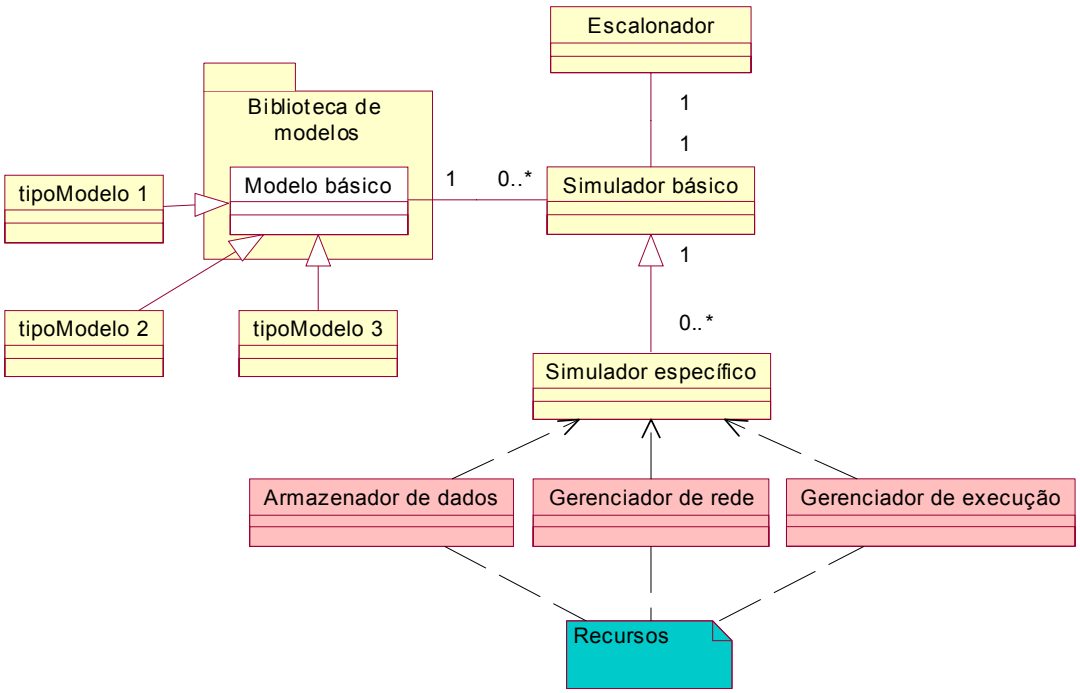

Figura 6: O domínio DE

2.3.3.3 O domínio PN

O domínio PN (Process *Network*) **[30,31]** é uma técnica de avaliação que utiliza conceitos de processamento concorrente, onde os modelos são executados em um mesmo processador, sendo que cada um é executado como se a ele fosse dado um processador exclusivo. Este método engloba o conceito de eventos discretos, com a diferença que o método PN é mais abrangente, pois organiza os objetos pertencentes ao simulador na forma de processos independentes, com capacidade de intercomunicação. Estes processos podem ser gerados através de threads (processos **concorrentes**), que conferem enorme versatilidade ao simulador. Em sistemas reais os processos associados possuem

independência, ocorrendo simultaneamente, de forma que a representação de um cenário semelhante seria necessária uma estrutura computacional com vários processadores. O método PN simula processos através das threads, dando a impressão que possui todos os processadores que necessita. A forma de acesso aos recursos é apresentada na figura 7.

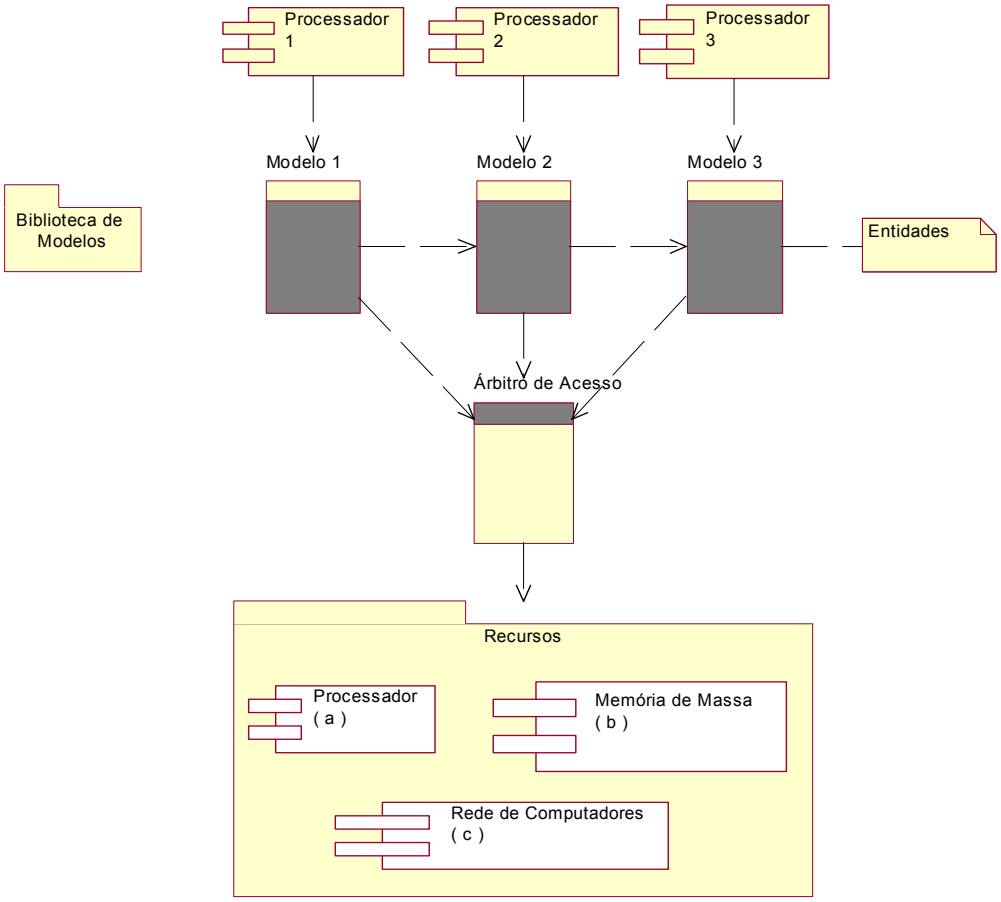

Figura 7: O domínio PN

2.4 Ambientes de simulação pesquisados

O desenvolvimento deste trabalho foi baseado em uma pesquisa realizada em ambientes comerciais e acadêmicos, tendo como objeto de estudo, a identificação de

características funcionais de maior relevância pertinentes ao conjunto de metas estabelecidas no capítulo anterior, atendendo aos seguintes itens:

- O projeto deve ser implementado de forma simples, resguardando funcionalidade suficiente para o desempenho de suas atribuições básicas.

- A estrutura interna será construída de forma simplificada, proporcionando praticidade de uso e fácil aprendizado.

- O ambiente será destinado ao uso acadêmico, de fácil instalação e código fonte aberto.

A seguir serão apresentados os ambientes selecionados para análise de requisitos, e durante esse processo, serão evidenciados os aspectos de maior relevância para o futuro ambiente, os quais juntamente com as metas de projeto, serão a base para a elaboração de uma especificação sistêmica.

2.4.1 BONES

O ambiente de simulação BONES (Block *Oriented Network Simulator*) **[21]** é uma ferramenta construída para auxiliar o desenvolvimento de experimentos relacionados a redes de comunicação incluindo: redes locais e de longa distância, comutação de pacotes e de circuitos, ISDN, redes de pacotes via rádio, barramento de computadores e suas arquiteturas. Esse ambiente é constituído por uma planilha gráfica, onde se desenvolvem análises e projetos. As redes utilizadas nos experimentos são especificadas através de uma topologia de rede, do tráfego aplicado, de estruturas de dados associadas e protocolos utilizados, aplicando uma estrutura hierárquica aos blocos utilizados, provenientes de uma biblioteca de modelos.

A concepção desse ambiente baseia-se na troca de mensagens e pacotes entre blocos, implementando o domínio de eventos discretos como técnica de avaliação de

experimentos. O avanço do relógio interno associado à simulação é controlado pelo ambiente, conferindo flexibilidade e simplicidade ao experimento. Essas características funcionais são construídas segundo uma definição interna de estruturas de dados, suportadas por uma hierarquia incremental de modelos, orientada para minimizar a complexidade total de uma simulação. Esta forma de estrutura é útil quando se deseja construir-se um novo modelo, pois através do mecanismo de herança o novo modelo será definido como um conjunto de característica herdado e alguns poucos métodos. Porém, o gerenciamento de uma estrutura de simulação baseado em hierarquias é uma tarefa complexa, o que ocasiona lentidão na execução da simulação. Dessa forma, o ambiente mostra-se flexível quanto à inclusão de novos blocos, mas é restritivo quanto a sua aplicabilidade.

2.4.2 BOSS

O ambiente BOSS (Block *Oriented Systems Simulator*) **[21]** é utilizado para análise e projeto de sistemas de comunicações, caracterizando-se por construir simulações baseadas em modelos através de uma abordagem hierárquica, executar simulações no domínio do tempo, interação de projeto objetivando otimização de desempenho e visualização de resultados em forma gráfica. A técnica de avaliação empregada nos modelos presentes em um simulador é o fluxo de dados contínuos. Esse ambiente mostra-se muito útil na prospecção de sistemas de transmissão, pois a modelagem de dispositivos físicos como fontes de dados, moduladores, filtros, canais de propagação, não linearidades e outros é realizada de forma simplificada. Sua estrutura é baseada em hierarquias orientadas a prover síntese de modelos complexos a partir de blocos de primitivas.

2.4.3 SPW

O SPW (Signal *Processing Worksystem*) **[22]** é um ambiente de simulação para uso comercial, muito utilizado na área de engenharia de sistemas. Esse ambiente é aplicado

para o desenvolvimento de sistemas e aplicações físicas (hadware), através de uma interface de aplicação dotada de alto nível de abstração, permitindo ao projetista utilizar-se de uma abordagem hierárquica para a criação de blocos constituintes, os quais podem ser construídos diretamente, agregando uma funcionalidade inédita ou através do mecanismo de polimorfismo, onde uma função existente será especializada para representar uma nova forma de atuação. Os blocos criados por quaisquer mecanismos, estarão aptos a serem utilizados em outros arranjos, permitindo que o reuso de código seja utilizado facilmente.

O sistema é constituído basicamente por duas partes: ferramenta visual para edição de blocos e dispositivo para execução de arranjos. Essa forma de construção permite que o sistema seja utilizado de forma remota, de modo que a interface visual seja executada na máquina do usuário, onde estão armazenados os dados do arranjo, e os cálculos realizados em outra máquina, possivelmente mais adequados a esta tarefa, como exemplo um servidor Linux, Solaris ou HP-UX. Desse modo, pode-se estabelecer um ambiente de execução multiplataforma, apoiado por um CVS (Control *Version System*) que gerencia a atualização de blocos em desenvolvimento. Outra característica importante a respeito desse ambiente é a capacidade de importar e executar códigos escritos em MATLAB, C/C++ **[7,8]**, SystemC e Verilog. Isto é possível pelo fato de que a estrutura de um arranjo é organizada através de uma hierarquia de blocos e os blocos utilizados pelo sistema são unidades autocontidas dotadas de portas de comunicação utilizadas para interligar os blocos com um gerenciador de atividade. Quando da importação de um código, o sistema um cabeçalho capaz de conectar o código importado a estrutura interna do sistema. Assim, pode-se executar o arranjo bloco a bloco.

2.4.4 LINKSIM

O LinkSim é um aplicativo para simulação **[12]** de sistemas ópticos que utiliza orientação a objetos como base para topologia sistêmica, análise e construção de dispositivos simuláveis, representados por enlaces ópticos, sendo a metodologia de orientação a blocos é utilizada para expressar a conectividade existente em um sistema

óptico. Esta técnica proporciona a facilidade de representação dos componentes através de ícones presentes em uma planilha gráfica, onde cada ícone nessa topologia está associado a uma classe de componentes, a qual possui parâmetros de execução que podem ser invocados através da planilha gráfica, mediante o acionamento de um botão do mouse sobre um componente.

O ambiente é construído de forma monolítica, onde toda funcionalidade está concentrada em um único aplicativo, o que facilita a instalação e utilização, porém dificulta a inserção de novos blocos na biblioteca de modelos, que é organizada em segmentos associados às classes de utilização. Esta organização é apresentada ao usuário através de uma caixa de diálogo contendo lista de classes facilmente identificáveis. A representação de simulações feita de forma gráfica facilita a construção e faz com que o usuário tenha o sentimento que está realmente analisando um sistema físico. Desse fato, pode-se abstrair que uma interface visual bem construída define um grande atrativo para a utilização desse tipo de aplicativo.

2.4.5 MAISIE

Maisie é uma linguagem de simulação **[28]**, baseada em C, que utiliza a técnica de eventos discretos para avaliar a interação entre os processos executados. Um objeto simulador, também referenciado como um processo físico, é mapeado através de um processo lógico, onde as interações entre processos são identificadas por um rótulo de tempo (timestamp). Esta linguagem é apta a executar simulações no domínio de eventos discretos utilizando diferentes protocolos assíncronos para simulação paralela em várias arquiteturas paralelas, de modo que a definição da linguagem foi estruturada para separar aspectos descritivos de uma simulação e os protocolos de suporte utilizados. Nessa linguagem são definidas entidades, as quais realizam tarefas, e enviam mensagens a outras entidades, através de comandos que estendem a linguagem C. Assim, a ambiência oferecida é puramente a edição de um programa em formato textual, que será compilado e montado a

bibliotecas que conferem sua funcionalidade. Sendo assim, o atrativo oferecido por essa linguagem é a abstração dos recursos paralelizados realizado de forma muito simples.

2.4.6 STER

O STER (Ambiente para desenvolvimento de software Tempo-Real) **[29]** é um ambiente de para aplicações acadêmicas, o qual implementa suporte ao desenvolvimento de aplicações de controle de processos em tempo real. A construção de um aplicativo neste ambiente é composta por duas etapas: a programação de dos módulos que implementam as funções do sistema e a configuração da aplicação a partir dos módulos disponíveis. A programação modular utiliza a Linguagem de Programação de Módulos (LCM), que é uma extensão da linguagem PASCAL que, em adição às construções desta, permite declara portas de entrada e saída, declarar mensagens, enviar e receber mensagens através de portas, suspensão de um módulo por um período de tempo e tratamento lógico de interrupções. Nos módulos escritos em LCM são especificados os módulos que comporão uma aplicação e a maneira como serão ligadas as suas portas de comunicação. A utilização da LCM para construção de aplicações caracteriza-se como um forte atrativo para este ambiente, pois esta linguagem incorpora abstrações interessantes à linguagem PASCAL, a qual é muito conhecida.

2.4.7 OMNeT++

O OMNET++ (Objective *Modular Network Testbed in C++)* **[20]** é um ambiente construído com o propósito de suportar a execução de simuladores utilizando a técnica de eventos discretos através de uma abordagem orientada a objetos. Este ambiente é utilizado para modelagem de: protocolos de comunicação, modelagem de tráfego, redes de computadores, multiprocessamento e sistemas distribuídos, sistemas administrativos e outras aplicações genéricas no campo de eventos discretos. A topologia deste ambiente consiste em uma hierarquia de modelos, a qual permite um número ilimitado de níveis,

permitindo a construção de qualquer estrutura lógica que representante de um sistema físico. A comunicação entre modelos é realizada através de mensagens, as quais podem conter estruturas arbitrárias adequadas ao sistema em estudo, aliadas a existência de parâmetros de controle de modelos, criam uma topologia global flexível.

A descrição da topologia é realizada através da linguagem de descrição de rede (NED), a qual suporta a especificação de uma rede de componentes constituída por: canais de comunicação, módulos do tipo simples e composto. Os arquivos que a descrição de uma rede possuem genericamente o sufixo *.ned.* A rede descrita não será utilizada diretamente, sendo primeiramente transcrita para uma correspondente em linguagem C++ através do compilador NEDC, responsável pela transcrição. O ambiente de simulação é constituído pelos seguintes componentes: descrição de rede, implementação de modelos do tipo simples, núcleo de biblioteca de simulação, interfaces de biblioteca para usuário. A organização destes componentes e´ apresentada na figura **8.**

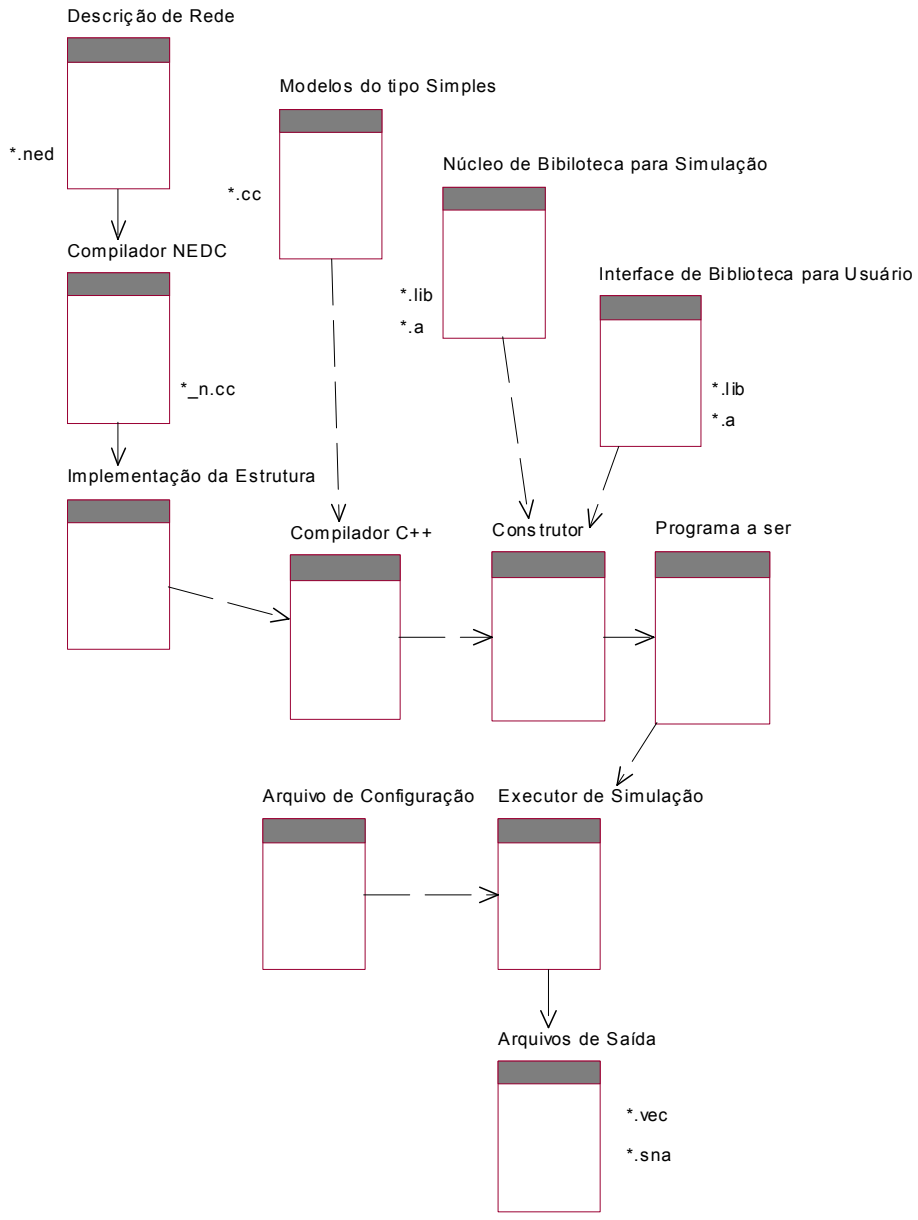

Figura 8: Estrutura do ambiente OMNET++

2.4.8 MERLiN

O ambiente MERLiN (Modeling Evaluation and Reserch of Lightware Networks) **[23]** é um aplicativo de uso acadêmico aplicado ao estudo de sistemas de transmissão ópticos na área de redes WDM (Wavelenght Division Multiplexing), permitindo desenvolvimento e prospecção de algoritmos de atribuição de comprimentos de onda para geração de caminhos ópticos, roteamento óptico, reconfiguração dinâmica e mecanismo de restauração de caminhos. A simulação construída está organizada em nós ópticos, responsáveis pela gerência de atribuição dos comprimentos de onda, constituídos por comutadores ópticos encarregados por encaminhar um comprimento de onda presente em uma porta de entrada para uma de saída.

Sua arquitetura consiste em três blocos principais: núcleo do sistema, modelo de rede e biblioteca de algoritmos, os quais são executados em processos independentes e em máquinas diferentes. O núcleo do sistema realiza gerenciamento global e responde as solicitações provenientes da planilha gráfica, onde são desenvolvidas as simulações. Os modelos de rede são módulos aplicados ao sistema mediante a conveniência do projeto realizado. A biblioteca de modelos está organizada em algoritmos, os quais são ativados mediante solicitação proveniente do núcleo de gerenciamento. As solicitações e mensagens entre blocos são realizadas através do envio de mensagem contendo dados a respeito da topologia utilizada e da configuração dos nós, respeitando uma sintaxe apropriada a o diálogo. A comunicação entre processos é organizada na forma de prestação de serviços quando um bloco solicita uma tarefa a um segundo bloco e este por sua vez gera um processo secundário para efetuar o atendimento do solicitante, estando apto a receber novas solicitações. Esta arquitetura proporciona modularidade e escalabilidade ao ambiente. A figura **9** apresenta a arquitetura desse ambiente.

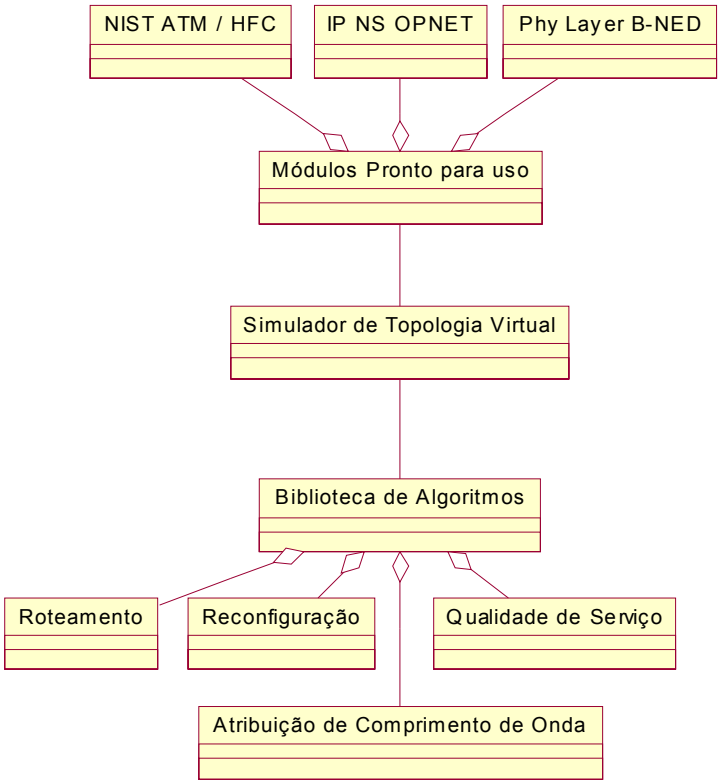

Figura 9: Arquitetura do ambiente MERLiN

2.4.9 PTOLEMY

O ambiente Ptolemy **[30,31]** foi concebido para realizar estudos de modelamento, projeto de concorrências, aplicações em tempo-real e sistemas embutidos, sendo o foco de uso, a montagem de componentes concorrentes. A construção do sistema está fundamentada na definição de um modelo computacional utilizada para gerenciar a interatividade entre componentes, podem existir composições entre diferentes modelos. Neste ambiente são desenvolvidas técnicas heterogêneas para suporte ao modelamento, o que consiste em explorar métodos baseados em fluxos de dados, redes de processos, sistemas orientados a eventos discretos, linguagem reativa, máquinas de estado finito e processos de comunicação seqüencial. Esse ambiente prove um laboratório virtual para realizações de experimentos utilizados em aplicações de processamento de sinais, telecomunicações, processamento paralelo, comunicações sem fio e sistemas de tempo-real.

Nesse ambiente, as simulações são construídas em uma paleta gráfica, onde são posicionados componentes da biblioteca. A interação oferecida pela paleta gráfica não está no estado da arte, pois utiliza um misto de acesso via mouse combinado com comandos de teclado, o que dificulta sua utilização. Por este motivo é considerada uma interface gráfica obsoleta. As simulações construídas, no entanto, são muito versáteis, pois o ambiente possui muitos modelos (ou domínios) de simulação, oferecendo a possibilidade de composição de domínio. Os mecanismos de herança e polimorfismo são características muito exploradas permitindo o desdobramento e especialização dos componentes do sistema, além de contar com um formidável ferramental composição de metacomponentes. Na composição do ambiente são utilizadas as linguagens de montagem C++ e Tcl, aplicadas à semântica de construção de componentes do ambiente, fato este que constitui um ponto desfavorável ao ambiente, pois a pluralidade de linguagem aumenta complexidade do sistema.

2.4.10 KHOROS

O aplicativo Khoros é um ambiente integrado de desenvolvimento **[24]**, dotado de ferramentas de desenvolvimento e depuração, processamento matricial e visualização gráfica dos experimentos realizados. A capacidade de expressão visual está associada à existência de elementos visuais, ou ícones, os quais podem ser manipulados interativamente de acordo com a gramática especificada para um dada simulação. A criação visual dos experimentos é organizada em nós, os quais são representados por ícones, interligados por arcos, os quais representam os caminhos de conexão por onde fluirão os dados, orientados a portas de conexão, de modo que inicia em uma porta de saída de dados e termina em uma outra de entrada. Assim, um experimento é representado por uma rede onde fluirão dados e alongo desta, serão modificados ou criados outros tantos, pelos nós presentes. Controles de estruturas podem ser utilizados, mesclando o uso de controles de fluxo a controles de iteração e laços de realimentação. O controle de estruturas de simulação está dividido em dois grandes blocos:

- **Controle de laços de realimentação** é responsável por realizar o controle de fluxo de dados, de modo a proporcionar a interação das várias seções do experimento quantas vezes for necessário, sendo o controle efetivo baseado no conteúdo de variáveis, as quais determinam quantas vezes um laço deverá ser avaliado.

- **Construções condicionais** permitem criar experimentos muito realísticos possibilitando a elaboração de redes complexas. O fluxo avaliado nesse ambiente traduz um fluxo de dados ou de controle, onde as estruturas de controles presentes no experimento são utilizadas para determinar qual a forma de controle sobre o fluxo de dados que será realizada.

2.4.11 Conclusões

O estudo dos sistemas mencionados anteriormente foi orientado no sentido de obter informações construcionais e características de maior relevância dentro do espaço definido por cada um dos sistemas, de modo que a soma de todos essas características irão constituir a base funcional e topológica do novo sistema. O resultados serão organizados na forma de uma lista de itens comentados isoladamente sem a intenção imediata de organizar uma estrutura. Alguns dos itens estudados serão utilizados para formar a especificação sistêmica do futuro ambiente, e os que não forem aproveitados de imediato, farão parte do plano de melhorias e evolução. Os itens de destaque encontrados foram esses:

- **Domínio de uma simulação**: define como serão avaliados os componentes presentes em uma simulação. Os domínios SDF e DE mostraram-se muito úteis, sendo opções disponíveis na maioria dos ambientes, podem ser aplicados para controles de simulações estáticas e dinâmicas, respectivamente. O domínio DE mostrou possuir um forte atrativo para o mundo da engenharia, por retratar situações onde o cenário de estudo muda ao longo do tempo. Outra opção muito interessante é a utilização de ambos os domínios em conjunto. Essa combinação é facilmente utilizada compondo-se um cenário voltado em sua macroestrutura aos eventos discretos, tendo como componentes macroblocos capazes de atuar nessa condição,

mas tendo em sua constituição íntima, blocos que operam de modo seqüencial, portanto no domínio SDF, e geralmente ocupam pouco espaço de tempo para sua conclusão. Assim, este cenário composto une o melhor dos dois mundos, tendo a capacidade de retratar de retratar situações dinâmicas, abstraindo etapas intermediárias de avaliação.

- **Organização de uma biblioteca de componentes**: a biblioteca de componentes é o principal elemento de ambiente de simulação, pois nela estará armazenado o conhecimento utilizável. Uma biblioteca deve conter modelos adequados a finalidade que destina o ambiente como um todo, de modo que deve estar apta a corresponder as expectativa de construção de experimentos. O acesso à biblioteca dever construído de forma oferecer o máximo desempenho e facilidade de obtenção de informações. A linguagem utilizada para construção dos componentes deve oferecer alto nível abstração e expressão, capazes ocultando detalhes construcionais e facilidade de arranjo, pois dessas características dependem a versatilidade de um ambiente. A topologia utilizada pode ser autocontida ou organizada em níveis. Na versão autocontida um componente é definido de uma forma compacta, onde todas partes são definidas em conjunto, não restando qualquer dependência externa. Esta versão é facilmente implementável, mas é bastante restritiva quanto ao aspecto de extensibilidade de seus componentes. A versão organizada em níveis, ou hierárquica, defini um componente uma árvore hierárquica, onde cada nível representa uma extensão ou especialização de um componente. Esta versão é mais sofistica que a anterior, permitindo o uso de mecanismo de herança para gerar novos componentes, e associa maior controle a profundidade dos dados associados.

- **Mecanismos de herança**: a possibilidade do uso do mecanismo de herança é um diferencial expressivo, pois sua utilização é faz com que o conhecimento contido nos componentes de sistema, em especial os da biblioteca de componentes, sejam atualizados e expandidos, dirimindo restrições anteriores. Assim, um ambiente de simulação deve utilizar intensamente este recursos.

- **Arquitetura do ambiente**: os ambientes de simulação pode ser construídos de modo monolítico ou distribuído em várias partes. Os sistemas monolíticos são autocontidos, isto é, não necessitam nada para serem executados, sendo muito utilizados como base para aplicativos ainda em fase de estudos ou em versões preliminares. Essa preferência existe pelo fato de que esta arquitetura facilita resolução de um dos requisitos básicos de um aplicativo com vários componentes: a comunicação entre eles. O uso de um aplicativo único facilita a construção de uma topologia onde os vários componentes possam estabelecer comunicação. Quando os sistemas distribuídos são selecionados para suportar a estrutura do ambiente deseja-se dotar o mesmo de grande capacidade evolutiva, pois quando um sistema é estruturado em vários componentes isolados, a evolução individual é o caminho natural para adição de melhorias. A comunicação entre componentes, realizada através do mecanismo de *sockets*, defini uma arquitetura onde os componentes podem executados em um mesmo computador ou não. Dessa forma, podem-se espalhar as partes do ambientes em uma rede de computadores realizando operação distribuída ou remota, maximizando o desempenho global do ambiente.

- **Estrutura de simulação**: é a forma como são construídos os arranjos a serem executados através do suporte do ambiente. Esses arranjos são construídos sob a égide de uma semântica preestabelecida, onde aspectos operacionais devem ser abstraídos. O desempenho na execução do arranjo é freqüentemente medido pelo tempo de despende, na medida em que quanto maior o tempo, menor o desempenho. Um dos muitos fatores associados ao desempenho é a linguagem de montagem utilizada, sendo a linguagem C/C++ **[7,8]** a mais utilizada para esta finalidade. A estrutura do simulador está atrelada à arquitetura do ambiente, de modo que se foi escolhida uma arquitetura monolítica, o simulador será construído dinamicamente no interior do ambiente e posteriormente executado. Se a arquitetura distribuída foi à escolhida, o simulador será montado em um componente do sistema e enviado a outro, para ser executado. A topologia empregada nos componentes e granularidade utilizada variam de uma implementação para outra. Outra característica importante é a oferta controle sobre a execução de um arranjo. A estrutura de suporte do simulador

deve prover meios de controle direcionar o fluxo de execução mediante situações dinâmicas ou parametrizadas, onde pode controlar as interações de um laço, desvios condicionais, escola de protocolos de comunicação entre blocos e organização básica associada à arquitetura aplicada aos dados veiculados.

- **Suporte gráfico**: o suporte gráfico é responsável por realizar interação entre o ambiente e o operador através de uma planilha gráfica dotada de uma interface gráfica orientada a eventos especialmente desenvolvida para este fim, onde serão desenvolvidos os arranjos a serem simulados. Esta interface é organizada na forma de lista de opções e caixas de diálogos contendo mensagens, botões acionadores de comandos, painéis para visualização diversa e uma área destinada à construção de arranjos, limitada à planilha gráfica. A forma de apresentação dos arranjos definida na interface está associada à topologia interna do sistema. Os componentes apresentáveis provem da biblioteca interna e são visualizados como blocos diferenciados sensíveis ao pressionar de botão do mouse. As interfaces gráficas pesquisadas mostraram-se muito intuitivas. Esta condição foi atingida através da organização das informações apresentadas nas listas de opções, as quais estão sempre visíveis, exibindo um conteúdo pertinente ao momento de execução. Essa organização simplifica a operação da interface, possibilitando o uso de apenas um botão do mouse, liberando o operador da necessidade de memorizar a existência de áreas na interface com expressividade diferenciada. Outra facilidade gráfica desejável que é a capacidade de apresentar meta-componente, os quais expressam subarranjos de forma simplificada, estando associada à topologia interna do sistema. A utilização de uma interface visual facilita o uso de aplicativo, porém deve-se ressalta que um determinado gerenciado de suporte gráfico é destinado a operar em uma plataforma específica (hardware mais sistema operacional), o que acarreta associação de um aplicativo computacional a uma plataforma. Este problema pode ser minimiza através de uma arquitetura sistêmica que faça o mínimo de uso das características do gerenciador de visualização, criando em seu próprio âmbito todos atributos necessários ao gerenciamento lógico das atividades de visualização.

- **Portabilidade**: a portabilidade de um aplicativo está associada à capacidade de operação deste ambiente em várias plataformas. Para que isso seja possível, todos as informações manipuladas pelo ambiente sejam neutras em relação à plataforma utilizada. Uma das maneiras verificadas de obter-se esta condição é a utilização de formato texto puro em todos os arquivos salvos e carregados pelo sistema. Esta condição mínima de portabilidade não leva em conta necessidade de produção de uma versão específica do ambiente para várias plataformas. Da análise dos ambientes de simulação pode-se abstrair que quanto maior a independência das estruturas internas em relação à plataforma, maior a flexibilidade construcional e a portabilidade do ambiente.

Capítulo 3

Proposta de Arquitetura

3.1 Definição de domínio do ambiente

O domínio de um ambiente de simulação é representado pela soma das funcionalidades ofertadas, pela ambiência na qual são conduzidas as atividades e pela forma de expressão adotada no trato das informações. Esses conceitos irão descrever a visão externa do ambiente e suas interações com o mundo exterior, representando uma visão de alto nível funcional, onde o sistema será observado com um bloco opaco transparecendo apenas as situações de aplicação, não importando nesse momento compreender como o sistema implementa sua funcionalidade.

O propósito dessa análise é estabelecer os requerimentos funcionais necessários, uma descrição consistente sobre as responsabilidades a serem cumpridas e relacionar possíveis situações de teste. Esta análise será estruturada através de um diagrama de casos de uso, o qual representa um conjunto de seqüências de ações que um sistema desempenha para produzir um resultado observável de valor a um elemento gerador de ações, onde cada caso de uso é uma seqüência completa de cenários de interação mostrando como eventos externos são respondidos. Por sua vez, um cenário é uma narrativa de um comportamento global do sistema e um elemento gerador de ações recebe a denominação ator, o qual representa um agente que interage com o sistema, podendo incluir seres humanos, dispositivos, máquinas ou outros sistemas. O conjunto de casos de uso **[14]** que define o domínio do ambiente será apresentado nas figuras 10, 11 e 12, seguidos por uma descrição formal dos casos de uso apresentados. Na seqüência, os atores que compõem o cenário do ambiente:

- **Usuário** representa é um agente externo ao sistema capazes de iniciar ações primárias que por sua vez irão desencadear alguma função do sistema. O sistema será desenhado para reagir a estímulos provenientes desse agente, não cabendo nenhuma ação primária que se inicie em seu interior, sendo essa condição suficiente para definir o sistema como puramente reativo.

- **Interface Gráfica** é o agente responsável pela captura dos eventos gerados traduzindo-os em ações dentro do escopo do ambiente e pela apresentação de informações. Os eventos mencionados são provenientes de uma planilha gráfica onde será realizada a construção de arranjos através inclusão de blocos funcionais, os quais representam modelos presentes em uma biblioteca de modelos. Os blocos serão interconectados formando uma malha, o que dará origem a uma simulação.

- **Biblioteca de Modelos** é o agente que realiza as transações com uma biblioteca de modelos, apresentando um modelo quando este é solicitado. Essa biblioteca constitui um repositório de blocos funcionais úteis à montagem de experimentos. Quando solicitado esse agente fornece informações necessárias à representação visual bloco e uma referência lógica para uma posterior ativação.

- **Construtor de Agregação** é o agente incumbido de gerar um conjunto de informações necessárias para que um novo modelo seja reconhecido no escopo do ambiente. Esse agente realiza uma coleta seletiva de informações sobre um novo modelo, organizando um apontador, o qual será adicionado a uma tabela de apontadores que referencia todos os blocos utilizáveis.

- **Núcleo de Simulação** é o agente responsável pelas tarefas associadas à execução das simulações construídas. Esse agente recebe informações lógicas relacionadas à configuração do experimento realizado e reproduz essa configuração de uma forma adequada a execução pelo computador.

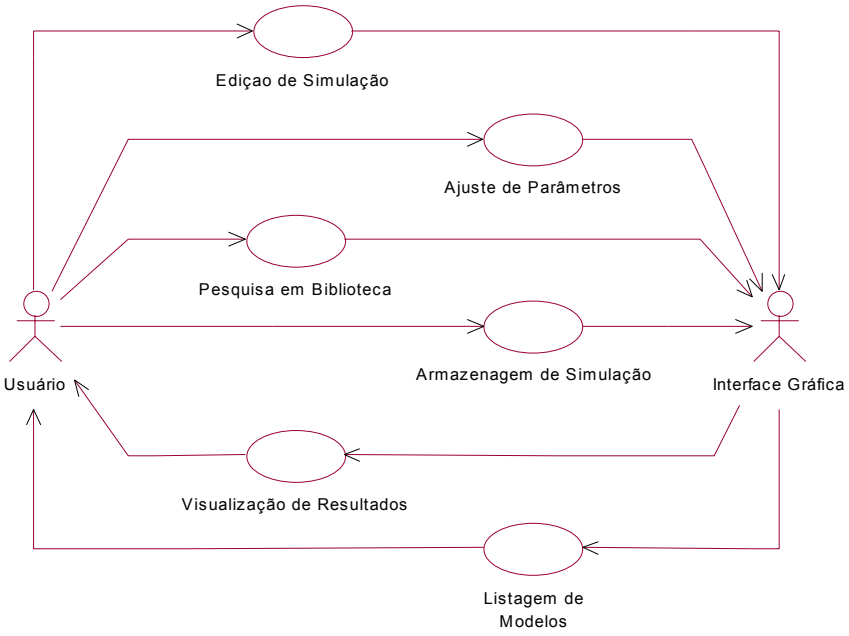

Figura 10: Domínio do ambiente (1)

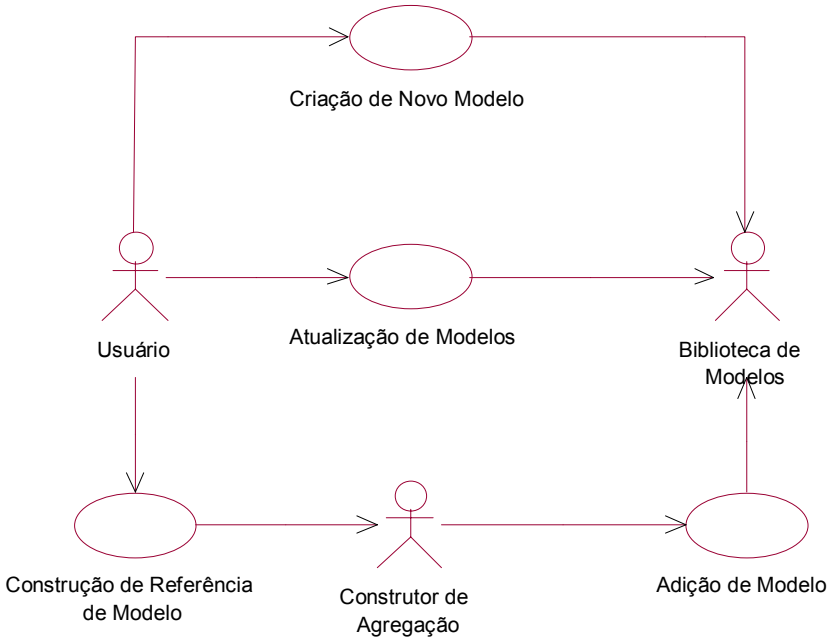

Figura 11: Domínio do ambiente (2)

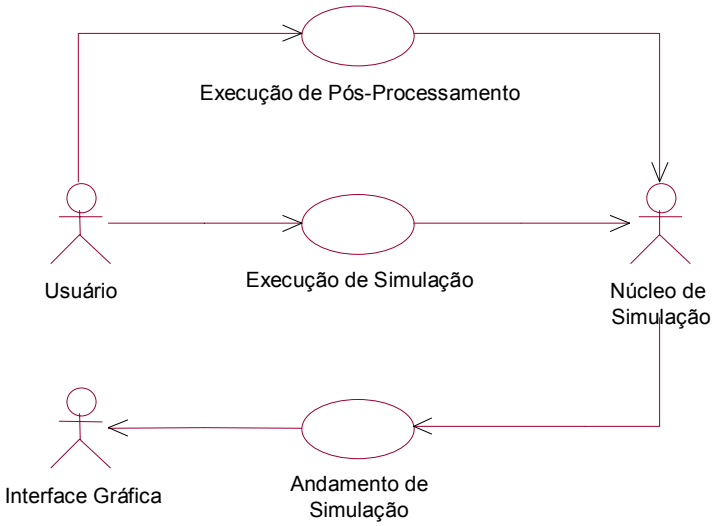

Figura 12: Domínio do ambiente (3)

Descrição dos Casos de Uso			
N. do caso de uso	Quem inicia o caso	Nome do caso de uso	Descrição do caso de uso
1	Usuário	Edição de Simulação	A edição de uma simulação é processo pelo qual é construído um aplicativo na planilha gráfica. A seqüência de ações associadas a essa tarefa é a seguinte: o apontador gráfico é posicionado sobre a planilha e aciona-se um botão do mouse para criar um bloco funcional, em seguida posiciona-se o apontador gráfico sobre o corpo do bloco criado repetindo essa ação mas como apontador sobre outro bloco.
2	Usuário	Ajuste de Parâmetros	O ajuste de parâmetros realiza atualização dos atributos que direcionam a execução interna do bloco. Essa tarefa é invocada quando se posiciona o apontador gráfico sobre o corpo de um bloco e aciona-se um botão do mouse invocando uma caixa de diálogo contendo informações paramétricas.
3	Usuário	Pesquisa em Biblioteca	A pesquisa em biblioteca é o processo pelo qual as informações sobre os componentes presentes na biblioteca são apresentados na forma de uma lista de dados segmentada por assunto.
4	Usuário	Armazenagem de Simulação	O armazenamento de uma simulação é o processo pelo qual um simulador é transferido da planilha gráfica para um arquivo de dados. Neste arquivo estarão todas as informações necessárias as reconstituições da simulação inicial.
5	Interface Gráfica	Visualização de Resultados	Os resultados originados pela execução de uma simulação podem ser apresentados na forma de texto ou gráfica. A forma gráfica

			pode ser aplicada quando o conjunto de dados gerados possuir um apelo visual, como uma curva.
6	Interface Gráfica	Listagem de Modelos	A lista de componentes presentes na biblioteca é construída mediante pesquisa realizada em todo o acervo bibliográfico do ambiente. Essa lista é apresentada através de uma caixa de diálogo, a qual é invocada quando se aciona um botão presente na interface gráfica.
7	Usuário	Criação de Novo Modelo	A criação de um novo modelo para biblioteca significa prototipar um novo componente a partir de um componente básico contendo toda a estrutura formal necessária para tornar o componente acessível ao ambiente.
8	Usuário	Atualização de Modelos	É o processo contínuo de melhoria dos componentes construídos para a biblioteca, o qual é constituído pelo modelamento do componente, codificação, compilação e montagem.
9	Usuário	Construção de Referência de Modelo	Construir uma referencia de um componente significa tornar esse componente visível ao ambiente, através da geração de dados adicionais obtidos a partir de um componente analisado.
10	Construtor de Agregação	Adição de Modelo	Após um novo componente da biblioteca ser construído é necessário torná-lo visível ao ambiente. Este processo é realizado através de uma ferramenta separada do ambiente, a qual pesquisa toda a biblioteca, obtendo informações, e ao final gera uma lista de componentes que será incorporada ao ambiente. O construtor de referências analisa o componente recém criado e obtém informações necessárias à geração de uma entrada em uma matriz de apontadores, a qual disponibiliza todos os elementos da biblioteca.
11	Usuário	Execução de Simulação	Executar um simulador significa iniciar o processo de avaliação dos blocos que compõem o arranjo construído na planilha gráfica. Os resultados deste processo são conjuntos de dados que analisados atestam a condição de funcionamento da simulação.
12	Usuário	Execução de Pós-Processamento	A realização de pós-processamento está associada à produção de informações adicionais à execução principal. Nesta modalidade novos dados são gerados utilizando como base os dados gerados na execução principal.
13	Núcleo de Simulação	Andamento de Simulação	Durante a execução de uma simulação são necessárias informações a respeito de seu andamento, pois uma simulação pode demandar um longo período de tempo para sua finalização, sendo importante informar seu estado.

Tabela 1: Casos de uso para definição de domínio do ambiente

3.2 Especificação sistêmica

O objetivo desta proposta é construir um ambiente de simulação flexível o suficiente que permita futuras modificações. Os elementos que irão compor o ambiente foram planejados para serem independentes, tendo seu código organizado de forma reentrante, e assim poderem operar isoladamente. Esta forma de construção recebe a denominação de **componentização [10]**, e os elementos pertencentes a essa classe serão denominados **componentes estruturais [25,26,27,37,39]**. Na especificação do ambiente de simulação será adotado que um componente será traduzido para uma **oferta de serviço**, a qual representa uma especialidade funcional pertencente à arquitetura do ambiente.

3.2.1 Comunicação entre componentes

Cada componente do ambiente deve interagir com outros através de meios de comunicação explícitos, aqui denominados **canais de comunicação**, implementados através de uma **interface de componente** responsável pela veiculação de informação, a qual estabelece abrangências pública, tornando o componente acessível e atualizável de forma isolada. O conteúdo da troca de informações entre componentes estruturais e entre elementos internos aos componentes será representado pelo o envio de **mensagens**, as quais veiculam informações em diversos formatos, como mostra a figura 13.

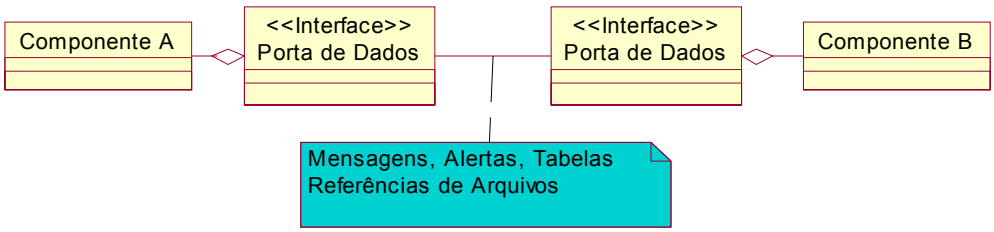

Figura 13: Comunicação entre componentes

3.2.2 Domínio de simulação

O ambiente proposto será utilizado para o estudo de sistemas de comunicações ópticos do tipo ponto-a-ponto, focando a análise na caracterização do comportamento dos elementos que compõem um enlace. Para a realização dessa tarefa estuda-se o comportamento de cada elemento, tendo como cenário um conjunto de dados aplicados por elemento anterior e atributos paramétricos previamente ajustados. Desse modo, a avaliação dos componentes do sistema será realizada de forma seqüencial, não relevando qualquer vinculo temporal, o que assegura que o domínio de simulação SDF é o mais indicado.

3.2.3 Arquitetura do ambiente

O ambiente proposto será construído de forma monolítica, de modo que ao final existirá apenas um único aplicativo computacional a ser executado. Essa arquitetura foi escolhida pela simplicidade de implementação dos serviços necessários como comunicação entre componentes, facilidade de construção de biblioteca de modelos, editoração gráfica integrada ao gerenciador sistêmico entre outras.

3.2.4 Macro estrutura

Uma proposta genérica de estrutura para um ambiente de simulação deve relevar os seguintes aspectos: ativação de módulos que compõem um simulador, o controle geral de fluxo de dados e o suporte visual. Esses aspectos são suficientes para prover a execução de uma simulação, abstraindo o funcionamento do plano de controle e promover boas interações externas, permitindo interferência na condução da execução e apresentação de resultados. Para concretizar essa ambiência foi elaborada uma proposta consistente de um conjunto de componentes capazes de suportar toda funcionalidade necessária é apresentado na figura 14.

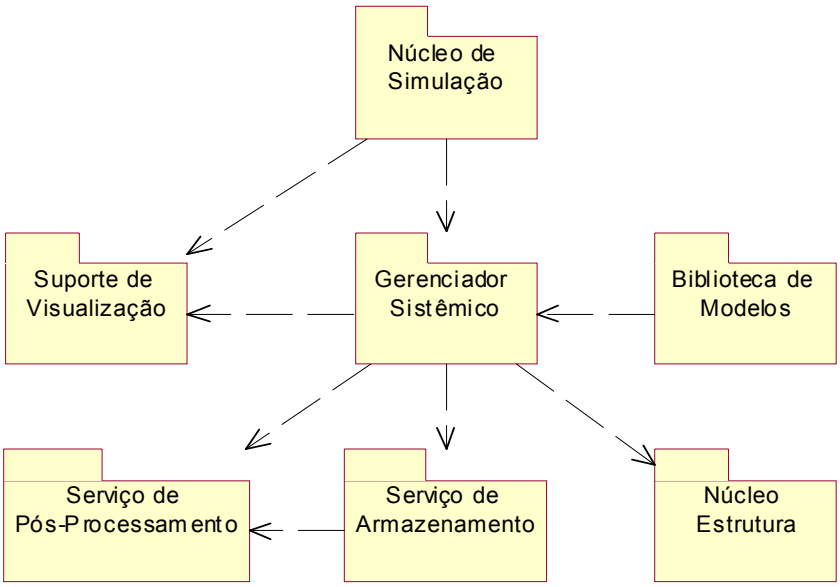

Figura 14: Componentes que integram a proposta de ambiente

O **Gerenciamento Sistêmico** está ligado ao controle geral, representando o nível mais elevado de controle dentro do ambiente, responsável pela coordenação geral das atividades, organização necessária para configurar os recursos que serão disponibilizados ao simulador, como: acesso a arquivos de dados, áreas de comunicação de mensagens, acesso a parâmetros modulares, tradução da organização gráfica do simulador para uma forma estrutural adequada ao entendimento do computador e disponibilizarão acesso a canais de alerta. Esse componente define qual o domínio de simulação adotado.

O **Núcleo de simulação** está ligado à execução de um simulador, sendo o componente responsável pela realização das tarefas de simulação. Este componente efetua a conexão física entre as áreas de mensagens dos módulos, define o número de iterações realizadas, coordena as trocas de mensagens entre os módulos e controlar a ativação do processo de simulação para que este cumpra os compromissos de abrangências definidos no início da simulação.

O **Núcleo Estrutural** está ligado ao suporte gráfico. A representação gráfica de um simulador é construída de forma dual, isto é, existe uma estrutura interna que abriga informações suficientes para a montagem de um ícone representativo de uma função ou serviço. O arranjo interno desta estrutura é responsável em abrigar os objetos e estruturas de dados que suportam a apresentação gráfica de um simulador. O núcleo estrutural também é utilizado como fonte de dados para a tradução do simulador para uma forma adequada ao processamento computacional. A representação gráfica será sempre o ponto de partida para qualquer atividade relacionada ao simulador, pois a representação funcional significa um arranjo construído em tempo de execução adequado ao entendimento do computador.

O **Serviço de Pós-Processamento** apresenta resultados originados em processamento eletivo. A geração desses resultados utiliza como base os dados gerados pelo processamento principal. Esse serviço possibilita a inclusão de atividades que utilizem os dados gerados como fonte de processamento eletivo, realizando de forma secundária, a execução de tarefas relacionadas com análise de dados gerados, produzindo informações complementares.

O **Serviço de Armazenagem** provê meios de armazenamentos de dados, de forma que o simulador não necessite conhecer o formato de gravação, pois este serviço está ligado ao suporte geral do ambiente. Para armazenar dados, um elemento do simulador envia um cabeçalho informativo e em seguida, os dados. Para recuperar os dados, o serviço carrega os dados sem nenhuma formatação e os repassam ao requisitante, responsável em alojar os dados de forma significativa.

O **Suporte de Visualização** é o elemento responsável em prover meios de comunicação gráficos entre o usuário e o ambiente de simulação. A interface gráfica é capaz de manipular os atributos presentes nas estruturas de simulação, gerando modificações visuais. Esse aspecto é fundamental para a proposta de um serviço de apoio visual, na medida em que através do suporte gráfico é estabelecida a expressividade de um simulador. A organização deste serviço compreende a existência de uma interface entre a

porção ligada ao tratamento visual e o processamento numérico, na mediada em que isola a complexidade de cada lado restrito ao escopo de cada componente, não comprometendo a privacidade de cada um. Este arranjo organiza as diretivas de construção gráfica, independendo de qual gerenciador gráfico esteja sendo utilizado, pois a adoção de um determinado gerenciador gráfico define uma associação entre aplicativo e plataforma computacional. Assim, conclui-se que é possível portar o ambiente para uma plataforma diferente daquela utilizada atualmente.

A **Biblioteca de módulos** está ligado ao núcleo de simulação, sendo encarregada do instanciamento dos objetos executáveis associados aos módulos. Nos momentos iniciais do processo de tradução do simulador gráfico é realizado o instanciamento dos objetos modulares, sendo posteriormente repassados ao núcleo de simulação, através de uma interface pública que atende a demanda comunicacional entre biblioteca e núcleo de simulação.

3.2.5 Arquitetura de módulo

O módulo de simulação será o elemento básico de um experimento, resultado da união de um modelo de conhecimento com a interface de integração. Essa interface formalizará um conjunto de facilidades orientadas no sentido de compor um meio de troca de mensagens responsável pela mediação entre um modelo e o ambiente de simulação, realizando a conexão de forma padronizada proporcionando versatilidade e independência funcional, utilizando portas para envio de mensagens, as quais disponibilizam aos módulos, métodos de escrita e leitura de dados, não onerando os módulos quanto à organização da informação no canal de comunicação.

Um módulo será representado por uma classe derivada de uma classe genérica, onde serão definidas as funcionalidades básicas. Este método de construção utiliza uma propriedade denominada **polimorfismo [7]**, que significa a capacidade de um operador executar a ação apropriada dependendo do tipo do operando fornecido. Esta forma de

criação permite que sejam definidas funcionalidades inerentes ao modelo, facilitando o entendimento do domínio do problema, na medida em que organiza, sem prejuízo funcional, métodos com denominação padronizada, os quais podem requisitar ações específicas do módulo através da invocação de um método associado à ação pretendida. Assim, existem métodos que pertencem à interface de comunicação e outros que fazem parte do modelo que está sendo desenvolvido. O módulo deverá ser encapsulado, constituindo um objeto fechado, com funcionamento único, definido por um código de programa na linguagem C++ **[7,8,16,17]**, com área de armazenamento própria e parâmetros restritos, dotado de mecanismos suficientes para suportar e coordenar suas atividades, possuindo uma interface de acesso.

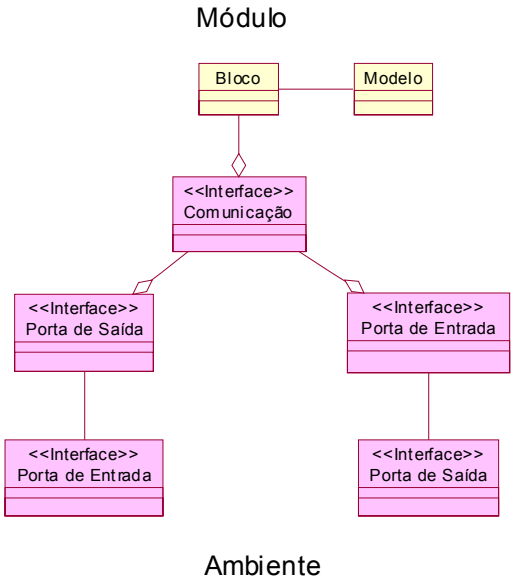

Figura 15: Arquitetura de módulo

3.2.6 Interface gráfica

A aparência dos aplicativos desenvolvidos e demais elementos invocados durante o ato de criação de uma aplicação serão padronizados, estando associados ao gerenciador gráfico Motif **[9]**. Os elementos mais comuns listas de opções, caixas de diálogos contendo botões de ativação e seleção, lacunas para preenchimento e listas de objetos disponíveis, sendo conhecidos como elementos modais de diálogo de uma interface. Estes elementos terão a função de gerenciar e configurar o cenário criado pelo usuário. O acesso aos itens disponíveis nesta ferramenta será realizado através do mouse. O ambiente de simulação está fortemente apoiado na manipulação de uma **planilha gráfica** que está disponível continuamente para realização de modificações. Dessa forma, quando o ambiente é invocado apresenta-se ao usuário uma planilha inicialmente vazia, a espera de inserção de elementos. A tela principal está subdivida em três regiões: a barra de comandos, a planilha gráfica e campo de identificação de simulação, como mostra a figura 15.

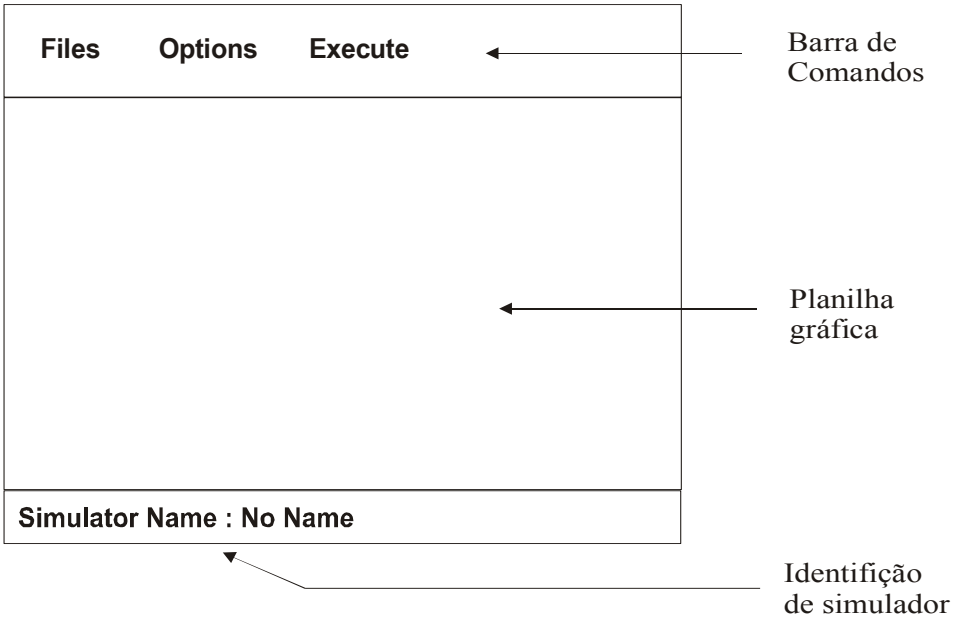

Figura 16: Regiões da tela principal

A **Barra de Comandos** é utilizada para abrigar o topo da hierarquia de comandos possíveis no ambiente. Nesta região estão presentes os comandos disponíveis à operação do ambiente. O acesso será realizado posicionando-se o cursor associado ao mouse sobre algum comando e em seguida aciona-se o botão de seleção (da esquerda) do mesmo. Este ato ocasiona o aparecimento de um menu de opções. Uma vez apresentado o menu pode-se selecionar uma opção de forma análoga à descrita anteriormente. A **Planilha gráfica** é utilizada para a construção dos aplicativos. Esta região é sensível ao acionamento dos botões do mouse, quando este estiver sobre esta região. O resultado desta ação está associado ao modo de edição vigente. A área **Identificação de simulador** é utilizada para associar um nome qualquer de identificação ao aplicativo desenvolvido na planilha gráfica. O acionamento do botão de seleção do mouse sobre o comando **FILES** presente na barra de comandos, ocasiona o aparecimento do menu de opções. A aparência dos menus de comandos e as ações associadas serão apresentadas nas figuras e nas tabelas a seguir respectivamente.

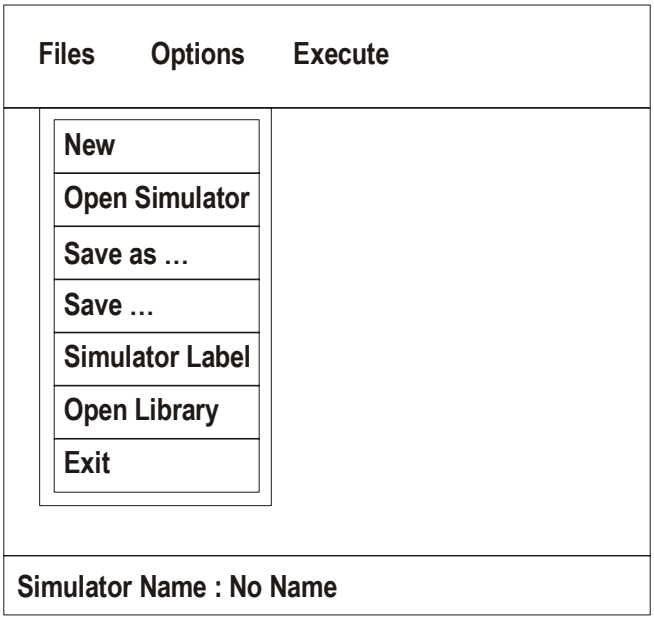

Figura 17: Comando do menu *Files*

Comando	Ação associada
New	Descarta o aplicativo que estava em uso, liberando todos os recursos alocados, apresentando uma planilha vazia.
Open Simulator	Invoca o aparecimento de uma janela de diálogo, onde se pode procurar e selecionar uma aplicação anteriormente salva. Selecionando um arquivo contendo uma aplicação, esta será apresentada na planilha gráfica, trazendo consigo todas as informações e estados a ela associados.
Save as	Invoca o aparecimento de uma janela de diálogo, onde se pode escolher uma localidade designada para abrigar uma aplicação nunca antes salva. Nesta janela deve-se definir o nome real do aplicativo, que não deve ser confundido com aquela apresentado no campo *Simulator Name*. Um nome real será aquele que poderá ser visualizado no sistema de arquivos do computador.
Save	Funcionalidade semelhante a *Save as*, com a diferença de que o acionamento não invoca o aparecimento de nenhuma janela, salvando as modificações realizadas no aplicativo presente na planilha gráfica.
Simulator Label	Invoca o aparecimento de uma janela de diálogo, onde o usuário pode escolher um nome de identificação significativo para o aplicativo.
Open Library	Invoca o aparecimento de uma janela de diálogo, denominada de biblioteca, onde se pode escolher um módulo.
Exit	Esta opção finaliza a utilização do ambiente.

Tabela 2: Opções do menu *Files*

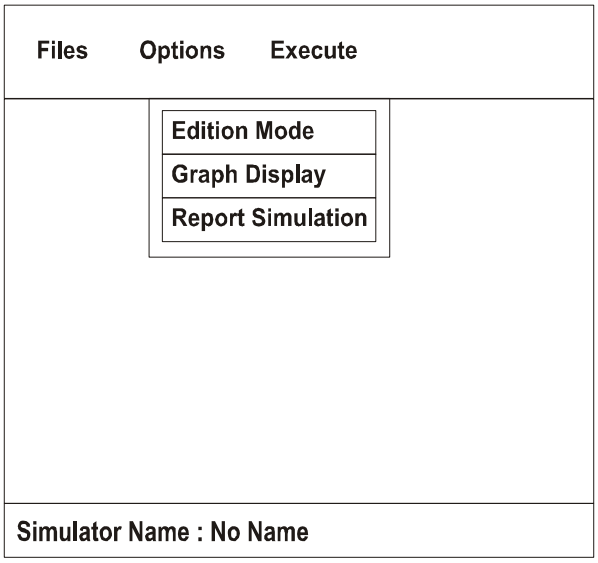

Figura 18: Comando do menu *Options*

Comando	Ação associada
Edition Mode	Invocar o aparecimento de uma janela de diálogo, onde se pode escolher qual o modo edição aplicado sobre os elementos presentes na planilha gráfica.
Graph Display	Invocar o aparecimento de uma janela de apresentação gráfica. O conteúdo apresentado está associado a um resultado obtido com a execução do aplicativo.
Report Simulation	Disparar a geração de um relatório contendo um sumário a respeito do aplicativo que esteja presente na planilha gráfica.

Tabela 3: Opções do menu *Options*

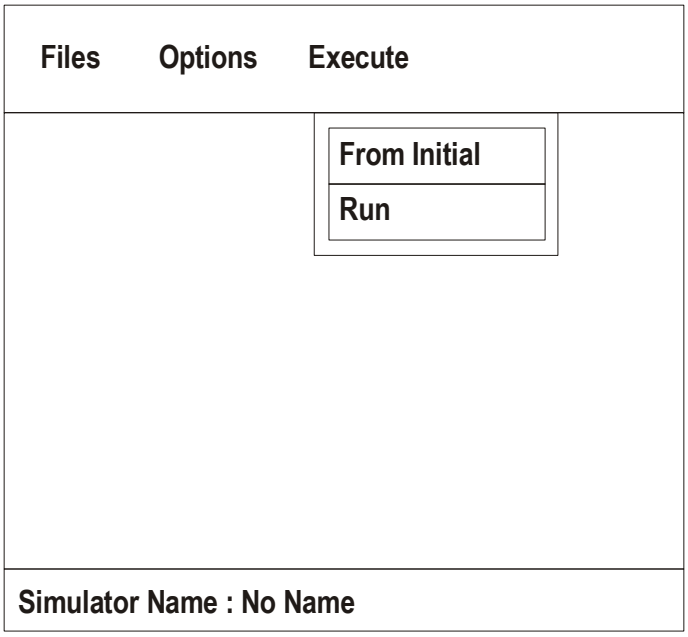

Figura 19: Comando do menu *Execute*

Comando	Ação associada
From Initial	Disparar a execução total do aplicativo presente na planilha gráfica.
Run	Disparar a execução parcial do aplicativo.

Tabela 4: Opções do menu *Execute*

3.2.3 Construção de aplicativos

A rotina de construção de um aplicativo será constituída pela seleção de módulos da biblioteca e posterior inserção na planilha gráfica. Esta rotina será executada até construção final do aplicativo desejado. Uma vez inseridos os módulos, passa-se a conectá-los e como conseqüência natural deste processo será o estabelecimento de uma relação de dependência entre modelos, onde a dependência será suportada pelo

estabelecimento de uma conexão entre portas de comunicações presentes em módulos envolvidos. A conexão de todos os módulos presentes na planilha gráfica significa que uma versão do aplicativo desejado foi construída. O próximo passo será ajustar as informações associadas aos módulos. Um exemplo simplificado de aplicativo é apresentado na figura 19.

O aplicativo construído será representado por um conjunto de módulos interconectados, configurando uma malha de dependências, na qual um módulo será executado apenas quando os outros módulos antecessores tiverem sido executados. A construção da malha de dependências significa proceder à conexão adequada entre os módulos, ligando-se a saída de dados de um módulo à entrada de outro. Informações adicionais associadas aos módulos são inseridas com o objetivo de parametrizar as operações internas realizadas e estão disponíveis ao usuário, mediante o posicionamento do cursor do mouse sobre o ícone representativo do módulo e acionando o botão direito do mouse. Esta ação apresenta ao usuário uma janela contendo uma planilha de parâmetros semelhante à apresentada na figura 20.

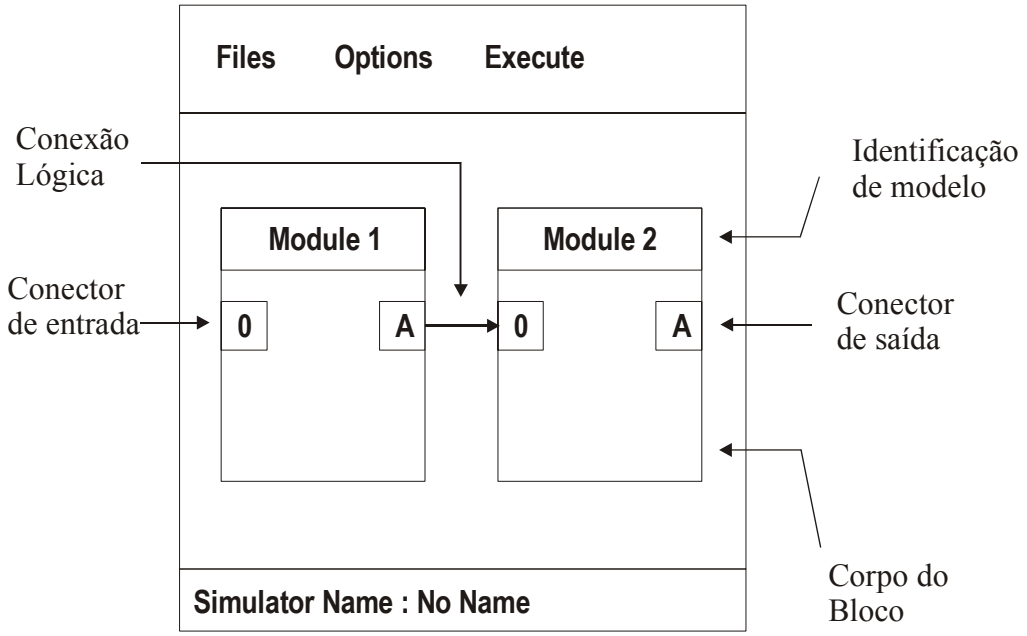

Figura 20: Exemplo de um aplicativo

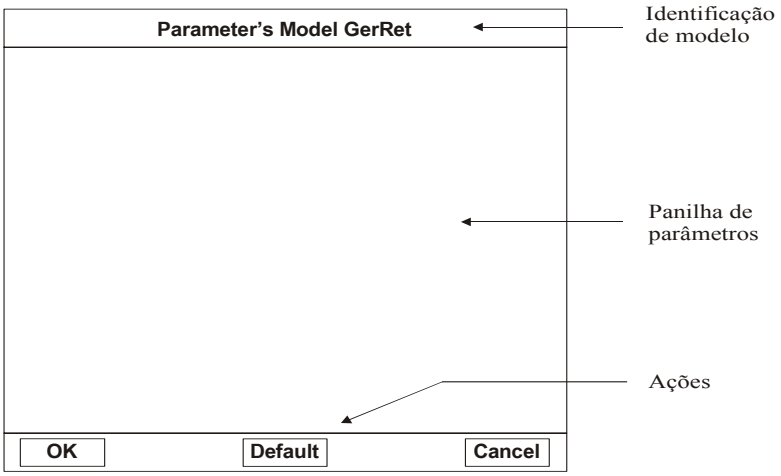

Figura 21: Caixa de diálogo contendo informações paramétricas

O conjunto de informações apresentado na planilha de parâmetros deverá ser ajustado pelo usuário de maneira que represente a forma deseja de funcionamento do módulo. Quando os ajustes estivem concluídos, deve-se ratificar este estado, acionado o botão ***OK***, o que ocasiona o desaparecimento da janela. Quando o conjunto de modificações realizadas não corresponde ao desejado, pode-se reverter à situação para os valores iniciais, acionando-se o botão ***Default***, ou recuperar o conjunto de anterior de modificações, acionando-se o botão ***Cancela***.

Após a execução do aplicativo serão gerados resultados os quais serão armazenados no sistema de arquivos do computador. Estes arquivos conterão informações relativas ao processamento realizado anteriormente, os quais poderão ser apresentados de forma textual (planilhas) ou gráfica (curvas). Estes resultados podem ainda conter dados básicos para outros cálculos realizados de forma eletiva. Este conjunto de cálculos será apresentado ao usuário de forma semelhante ao apresentado para obtenção de informações paramétricas, com a ressalva de que previamente deve-se selecionar o modo de apresentação.

3.3 Implementação funcional

Os conceitos apresentados nos componentes serão utilizados na construção do ambiente através de uma proposta de relacionamento entre os componentes definidos anteriormente, como mostrado na figura 5.1 e 5.2. Todos os componentes que serão utilizados no projeto serão especificados da seguinte forma: identificação do componente, problema solucionado, solução proposta e conseqüência de uso.

3.3.1 Gerenciador Sistêmico

O ambiente de simulação necessita de uma gerência centralizada capaz de coordenar o armazenamento e tramite de informações entre as estruturas de dados. O gerenciador será responsável por ativar as funções do sistema, coletar informações relacionadas à estrutura de simulação traduzindo-as para um formato aceito pelo gerenciador de execução, receber alarmes e os redirecionar para um destinatário, apresentar alertas, obter informações provenientes de um arquivo de script, contendo comandos, que devem ser aplicados ao simulador sem a necessidade de auxílio gráfico e isolar os elementos internos do ambiente da porção dedicada a apresentação gráfica. A centralização da tarefa de gerenciamento proporciona um modo rápido de constituir esse componente, pois agrega versatilidade e facilidade de substituição. A figura 22 apresenta o diagrama de classe referente ao gerenciador sistêmico.

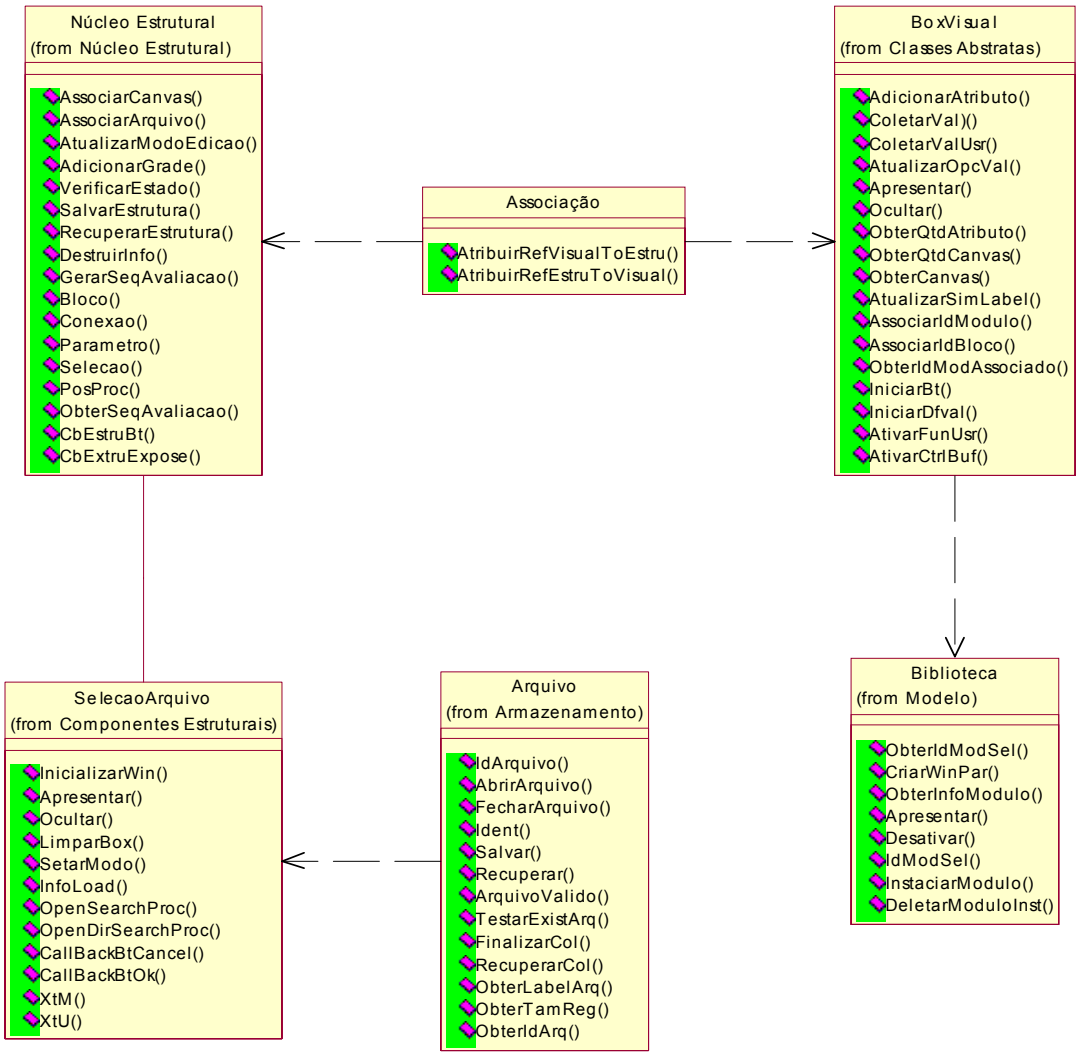

Figura 22: Diagrama de classes do Gerenciador sistêmico

3.3.2 Núcleo de simulação

Para a execução de um simulador é necessário o provimento de uma base de suporte para as tarefas relacionadas à execução dos módulos, pois o núcleo de simulação recebe mensagens contendo a estrutura de um simulador (originalmente em estado gráfico), juntamente com uma mensagem de script, e de posse destas informações, deverá construir uma representação computacional executável. O suporte oferecido baseia-se na organização uma lista de instâncias de módulos, a qual define a seqüência seguida no momento da execução. Isso feito, os módulos são instanciados, executados e os resultados enviados a um componente especializado na guarda de informações e os alertas ocorridos são enviados ao suporte visual. A figura 23 apresenta o diagrama de classe referente ao núcleo de simulação

3.3.4 Suporte Visual

Um ambiente de simulação equipado com um gerenciador gráfico é muito dependente de sua filosofia funcional, a qual determina o modo de operação e apresentação dos objetos visuais envolvidos. A manipulação interna das estruturas de um simulador será realizada em resposta a estímulos gerados pelo gerenciador gráfico, fato este que tende a monopolizar as atenções no ambiente. Por esse motivo, será construída uma interface de comunicação gráfica pública, orientada a mensagens, independente do gerenciador gráfico, na qual os eventos provenientes do gerenciador gráfico disparam procedimentos dentro da estrutura do ambiente, denominados *callbacks* **[9]**. Cada gerenciador gráfico possui uma estrutura evento distinta, ocasionando a necessidade de tradução de formato para uma estrutura de evento padronizada, que é então aplicada ao ambiente. Assim, seja qual for à estrutura do evento gerado, os procedimentos de callbacks sempre receberam os dados da mesma forma, isolando o gerenciador gráfico, e assim podendo substituí-lo. Esta facilidade de adequação faz com que o ambiente seja operado por qualquer gerenciador gráfico, o que implica em portabilidade. A figura 24 apresenta o diagrama de classes abstratas aplicadas

ao suporte visual, a figura 25 apresenta o diagrama de classe referente às ferramentas gráficas e a figura 26 apresenta o diagrama de classe referente às caixas de visualização e diálogo.

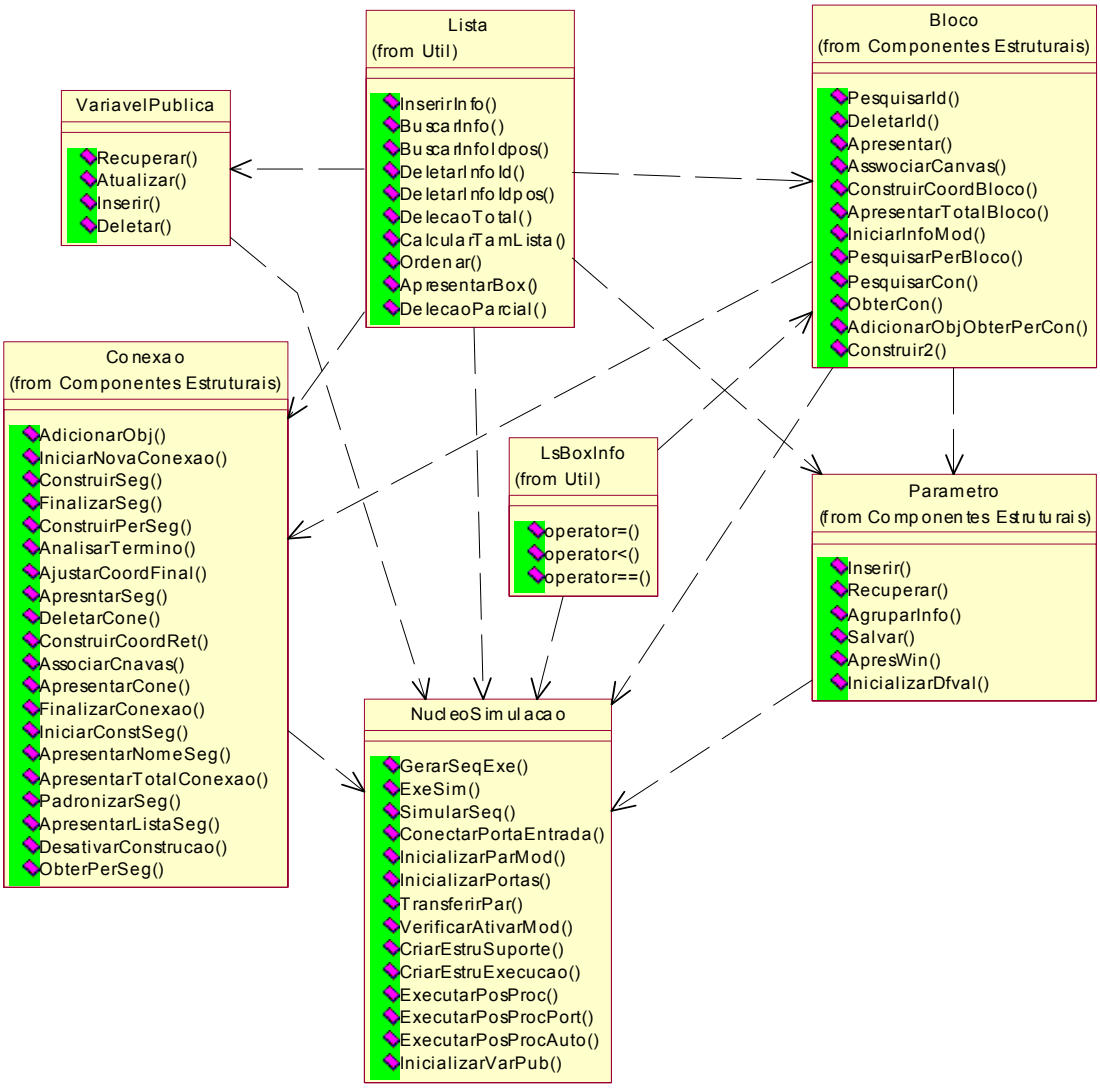

Figura 23: Diagrama de classes do Núcleo de Simulação

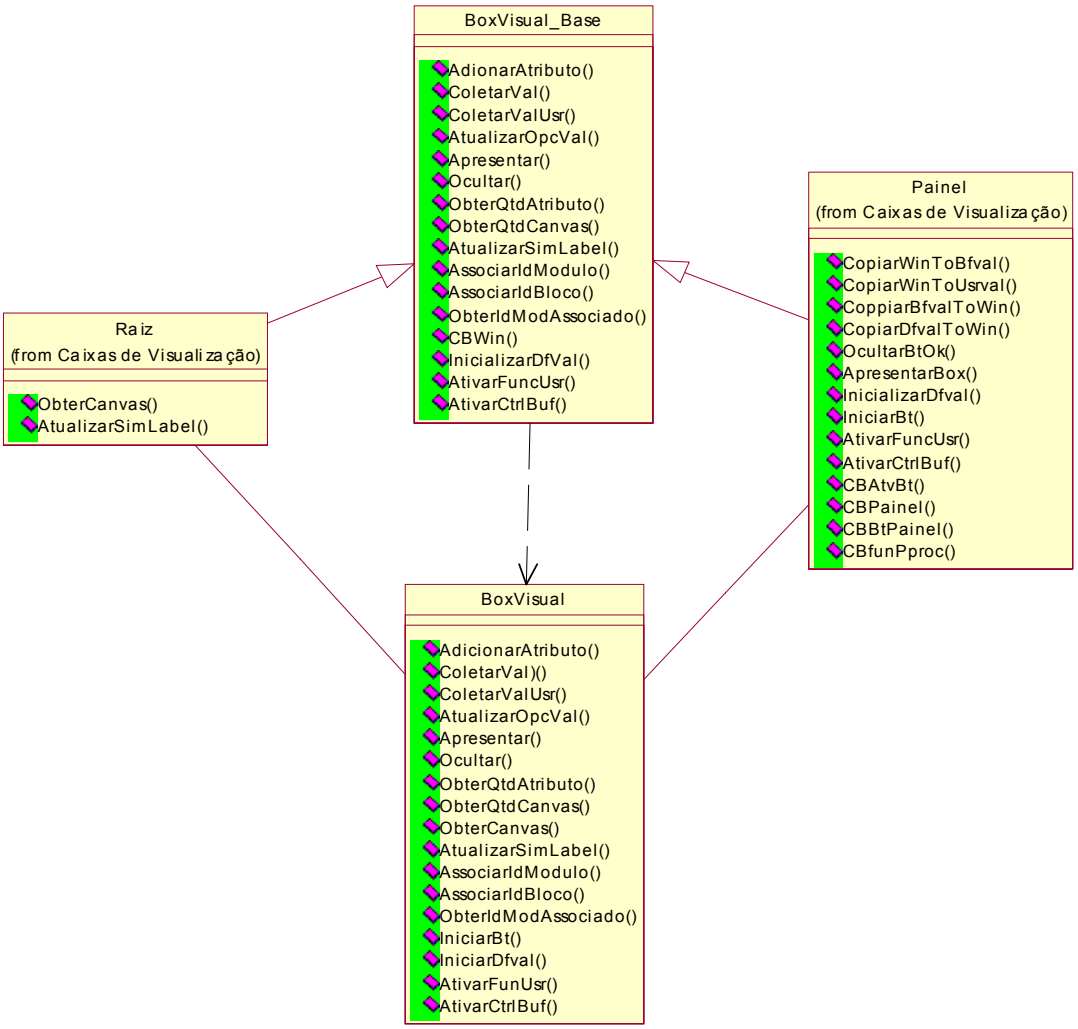

Figura 24: Diagrama de classes abstratas do Suporte Visual

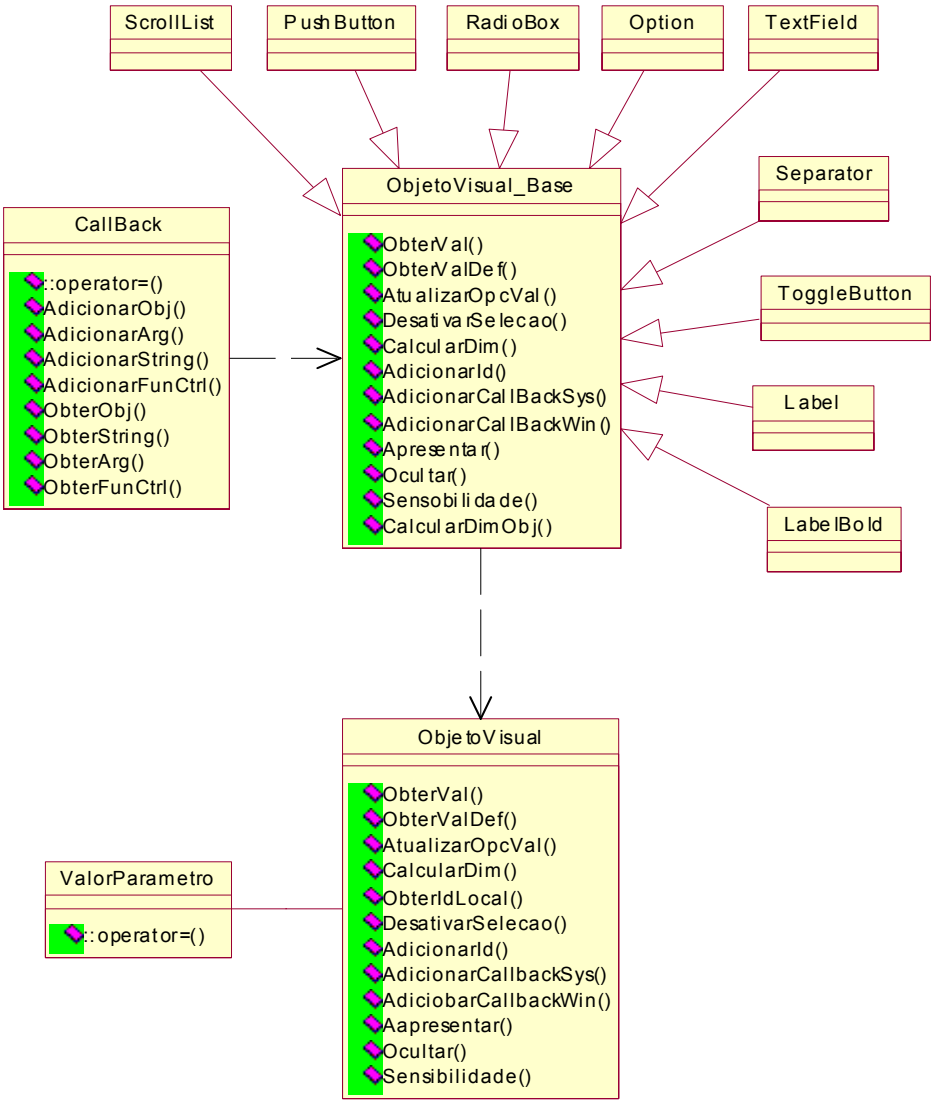

Figura 25: Diagrama de classes das ferramentas gráficas do Suporte Visual

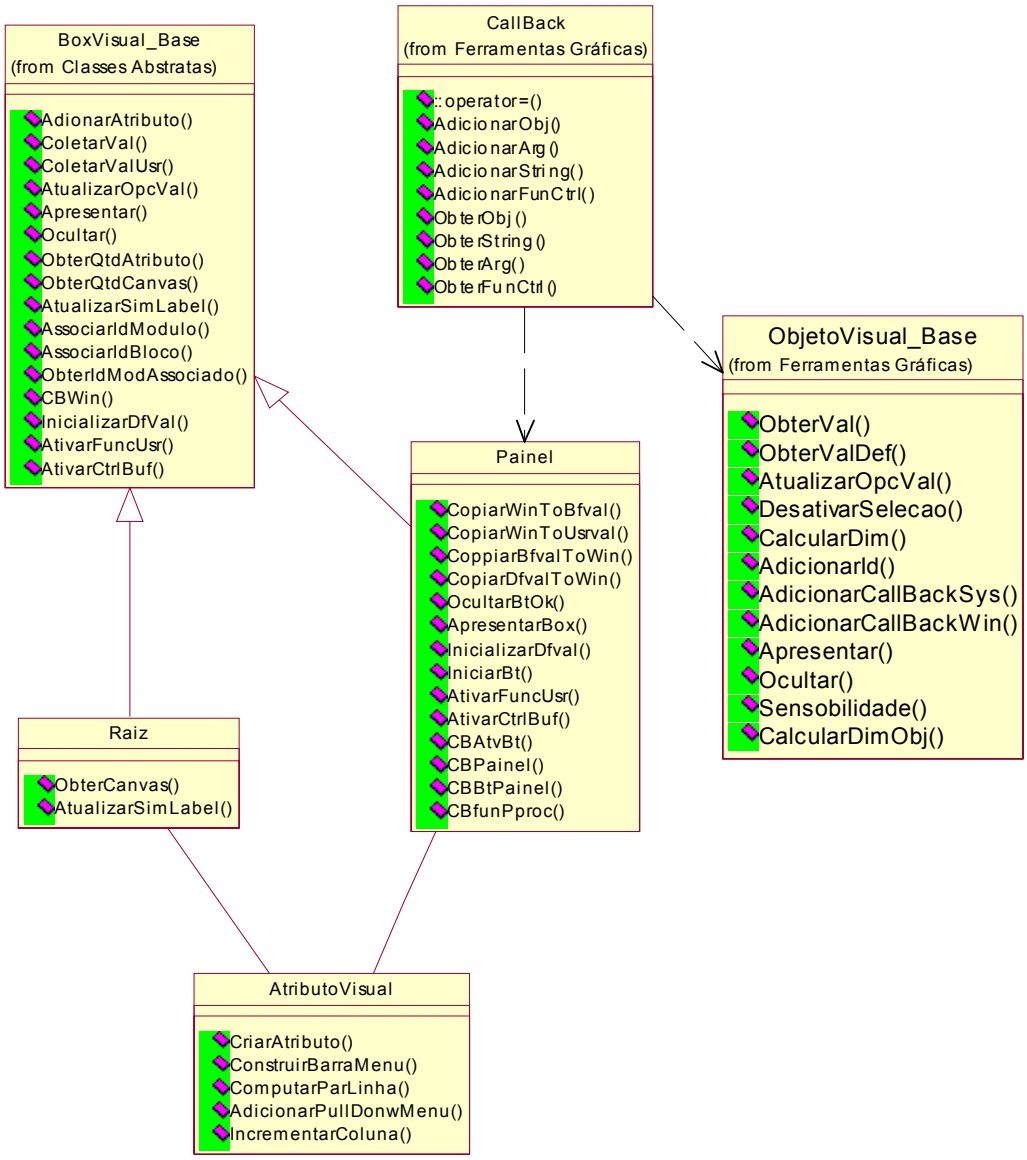

Figura 26: Diagrama de classes das caixas de visualização do Suporte Visual

3.3.5 Núcleo Estrutural

Será um conjunto de estruturas de apoio a edição gráfica onde todas as implicações relativas ao manuseio gráfico serão satisfeitas. Essas estruturas preenchem a necessidade de uma ferramenta de identificação de regiões gráficas no simulador sensível a movimentação do mouse e que representasse um simulador como uma árvore de dependências, a qual seria utilizada para organizar a seqüência de execução do simulador. Dessa forma, poder-se-ia expandir as funcionalidades gráficas de acordo com a necessidade, na medida em que se adiciona nova característica aos módulos, incrementa-se as estrutura de apoio, obtendo flexibilidade de representação visual. A figura 27 apresenta o diagrama de classe do núcleo estrutural.

3.3.6 Serviços de Pós-Processamento

O ambiente de simulação estará apto a realizar tarefas adicionais relacionadas à geração de dados complementares mediante solicitação, sendo gerados a partir dos dados calculados anteriormente no processamento principal. Esse processo será semelhante ao processamento principal, com o diferencial que apenas determinadas coleções de dados serão construídas. A forma de apresentação das várias coleções dados também é alvo de pós-processamento. Os resultados obtidos que possuem significado gráfico serão apresentados nessa forma, e os que não, serão exibidos como relatórios em formato textual. Essa forma de apresentação confere versatilidade de apresentação aos resultados obtidos através do ambiente.

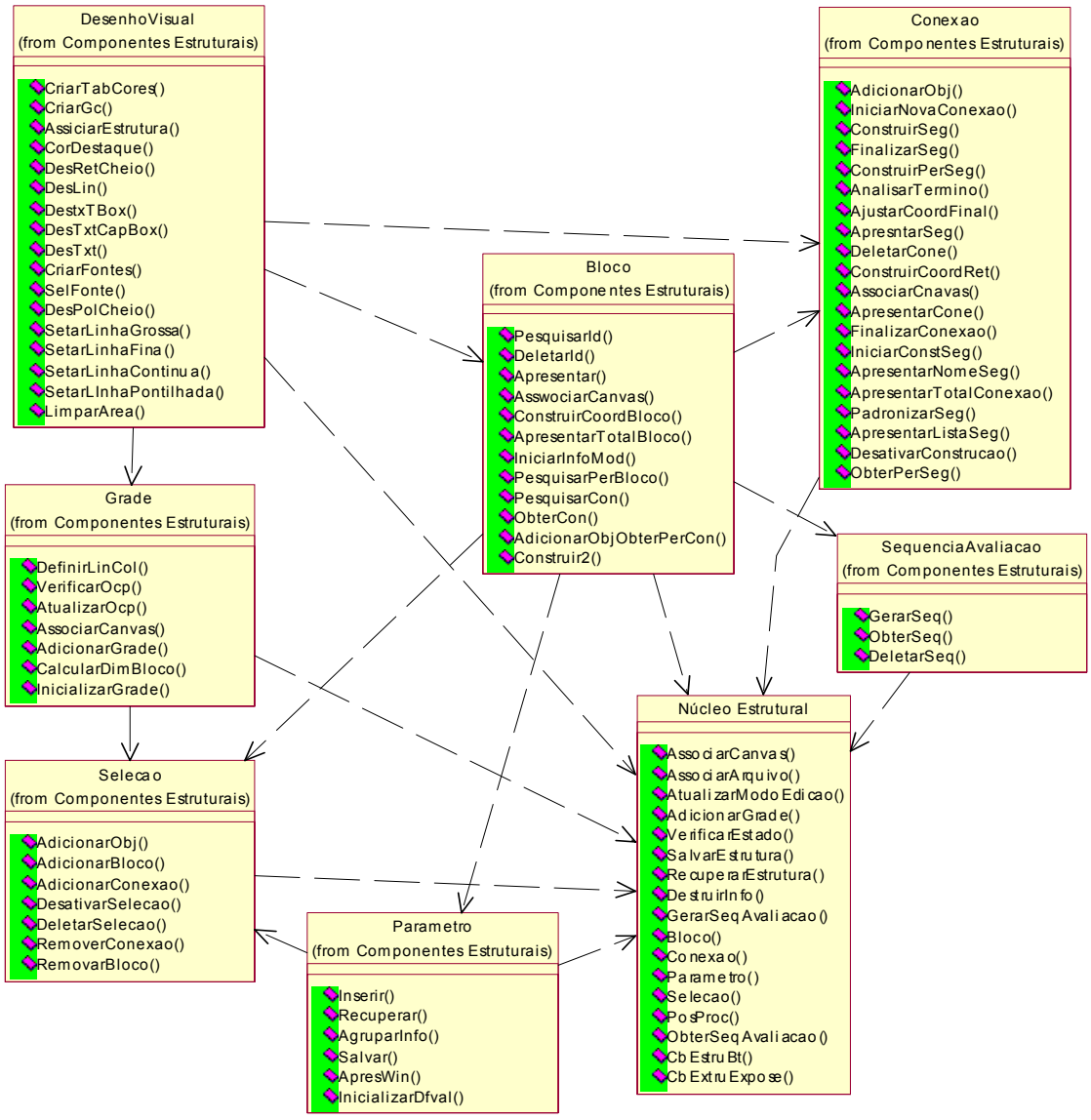

Figura 27: Diagrama de classes do Núcleo Estrutural

3.3.7 Serviços de Armazenamento

Este serviço será composto por um componente especializado em armazenar informações responsáveis por suprir as necessidades de informações disponíveis em meio magnético, armazenadas de forma heterogêneas, constituídas por diversos arquivos com arranjos singulares, implicando na necessidade de gerenciamento de muitos formatos de arquivos. A interface construída para esse serviço suportará a veiculação de diversas estruturas de dados através da solicitação ao cliente do serviço de um cabeçalho descritivo onde constarão informações relativas ao arranjo dos dados. Com estas informações pode-se organizar um arranjo padrão, onde todos os dados são alocados de forma ordenada. Quando do momento de carga destes dados, o cliente requer os dados do arquivo, e terá como resposta um apontador para uma área de dados não formatada. O cliente deverá proceder à transferência dos dados para uma área onde estes dados tenham significado. Assim, pode-se construir um componente que atenda todas as necessidades. A figura 28 apresenta o diagrama de classe do serviço de armazenamento.

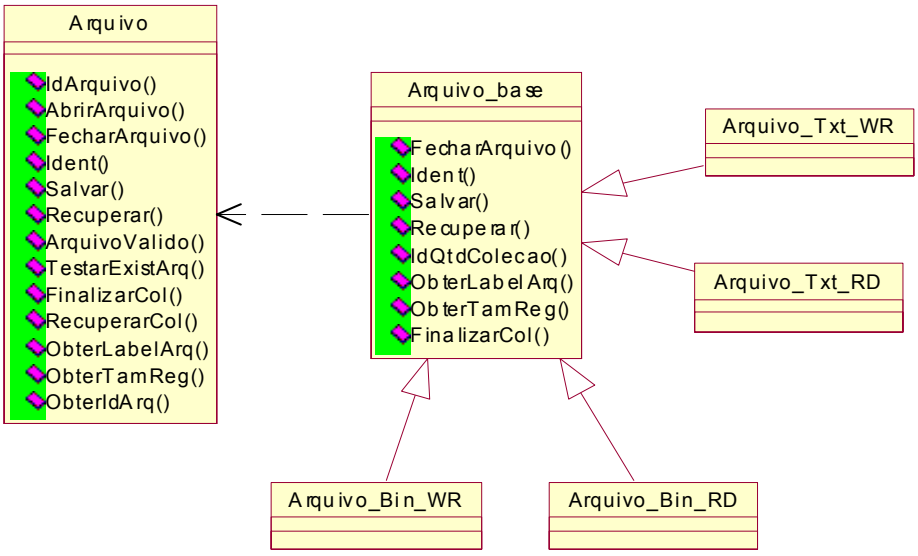

Figura 28: Diagrama de classes do Serviço de Armazenamento

3.3.8 Biblioteca de Modelos

A biblioteca será um repositório de modelos funcionais, representando o histórico de todo o trabalho realizado em termos de produção de conhecimento, disponibilizado em formato adequado à simulação**,** sendo composta por classes de objetos representando os modelos. A manutenção desta biblioteca será feita através de um aplicativo separado, responsável por atualizar a interface de apresentação da biblioteca. Os produtos de manutenção da biblioteca serão agregados ao ambiente em tempo de compilação, tarefa realizada pelo administrador do ambiente. Essa forma de construção simplifica o projeto, pois o ambiente não necessita de elementos externos para seu funcionamento, mas implica na impossibilidade de atualização da biblioteca por parte do usuário final. Sendo assim, a proposta do ambiente é aplicável a situações onde a dinâmica de criação de novos módulos seja em escala moderada. A figura 29 apresenta o diagrama de classe gerado dinamicamente, a figura 29 apresenta o diagrama de classe da Biblioteca de Modelos e a figura 30 apresenta o diagrama de classe do modelo de biblioteca.

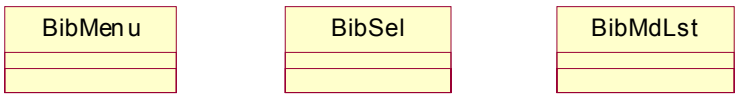

Figura 29: Diagramas de classes geradas dinamicamente

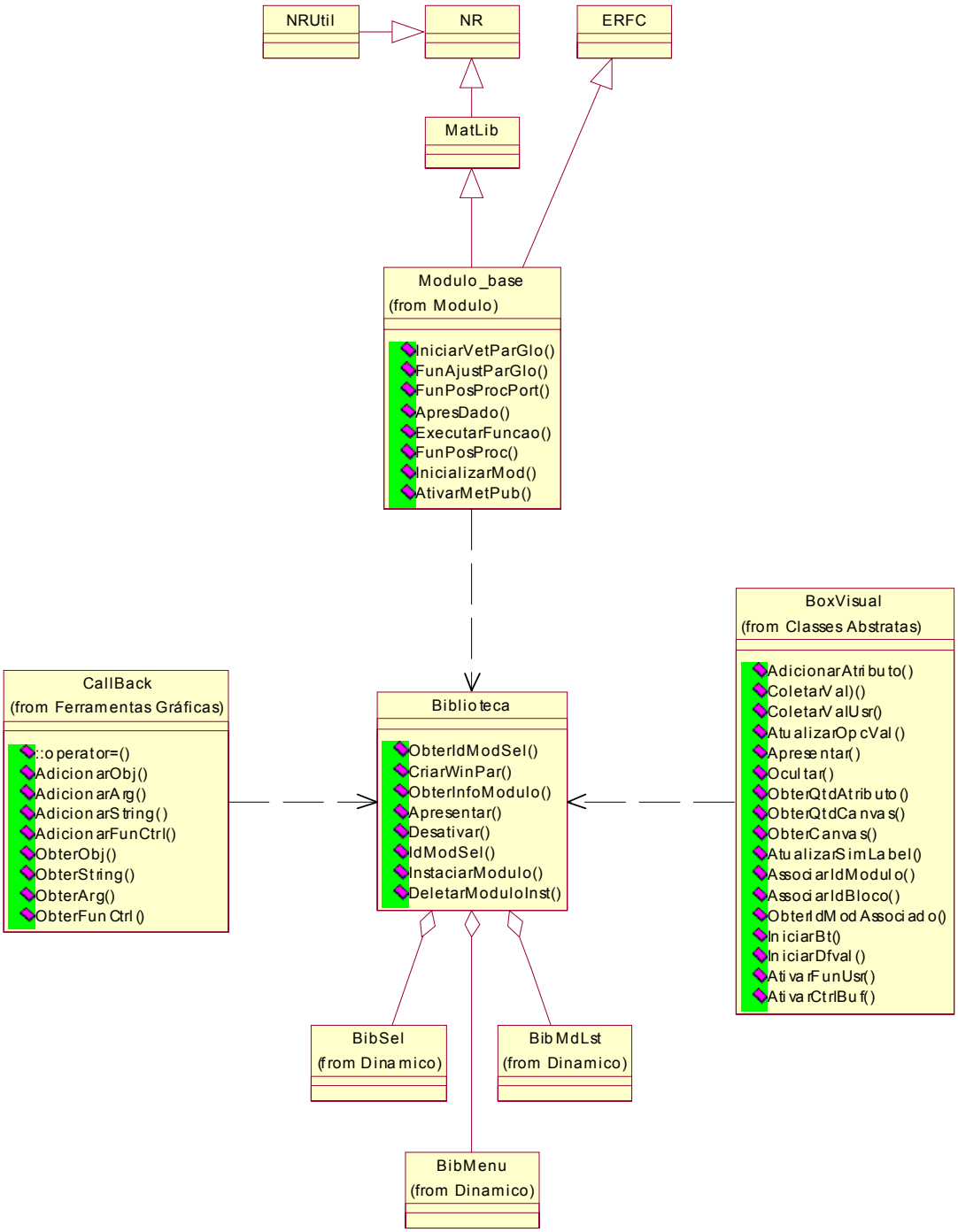

Figura 30: Diagrama de classes da Biblioteca de Modelos

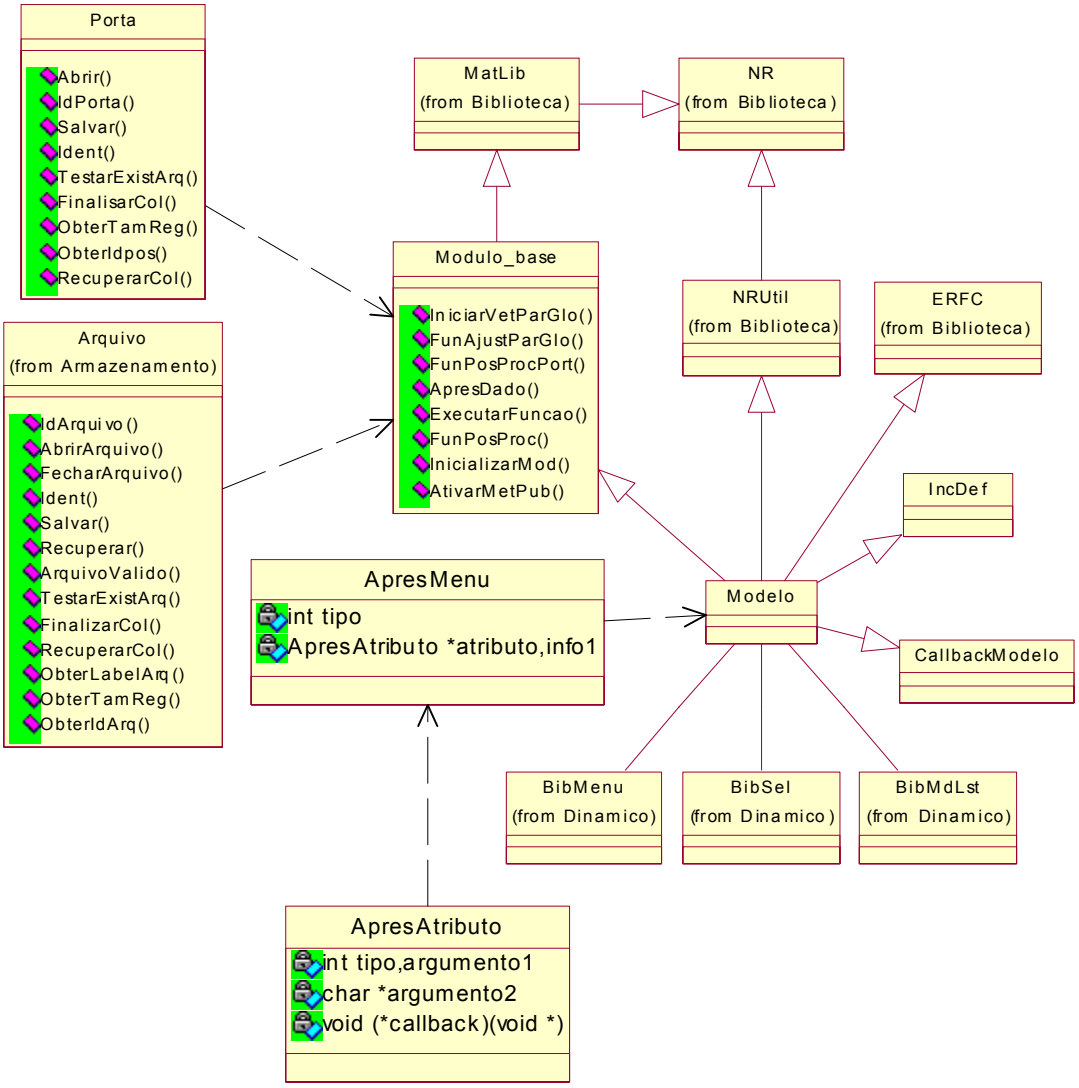

Figura 31: Diagrama de classes de modelo de biblioteca

Capítulo 4

Estudo de Caso

4.1 Definição de abrangência

Neste capítulo serão apresentados os resultados gerados por um simulador construído no ambiente PcSimFO e a potencialidade do ambiente no provimento de facilidades ao ato de projetar e codificar modelos, baseando a construção de simuladores no uso de modelos presentes em uma biblioteca, evidenciando detalhes de conexão e relações de dependência que apresentam o ambiente como uma forma eficiente de acesso ao trabalho desenvolvido. Os modelos utilizados para teste foram obtidos na biblioteca em uso do PcSimFO versão Windows e o estudo desenvolvido tem o objetivo de apresentar as facilidades ofertadas pelo ambiente. Os resultados obtidos foram certificados por experimentos anteriores. A ambiência estabelecida apresenta o simulador na forma visual, com acesso aos parâmetros associados ao modelo através de caixas de diálogos e os resultados de execução em planilhas gráficos adicionais. Além dos resultados principais existem outros, que podem ser solicitados. Dessa forma, pode-se obter variados tipos de informações dos modelos, mediante a necessidade.

4.2 Topologia de teste

A topologia escolhida para teste é um sistema de transmissão óptico operando em banda base, constituído por um gerador de padrão retangular, um *driver* de potência, um laser, uma fibra óptica, um fotodetector, um pré-amplificador, um filtro e medidor de taxa de erro **[18,19]**. O gerador de padrão é responsável por criar seqüências binárias que representam as informações digitais transmitidas. Conectado ao gerador temos um *driver*

de potência responsável pela modulação do laser conectado à sua frente. A saída modulada do laser é aplicada a uma fibra óptica que constitui o meio de transmissão de dados. A outra extremidade da fibra está conectada a um fotodetector responsável pela transformação do sinal óptico em elétrico. O sinal elétrico obtido é aplicado a um pré-amplificador e posteriormente a um filtro. Ao final, o sinal filtrado é aplicado a um medidor de taxa de erro, que avalia a qualidade do sistema de transmissão construído. A figura 32 apresenta o simulador construído.

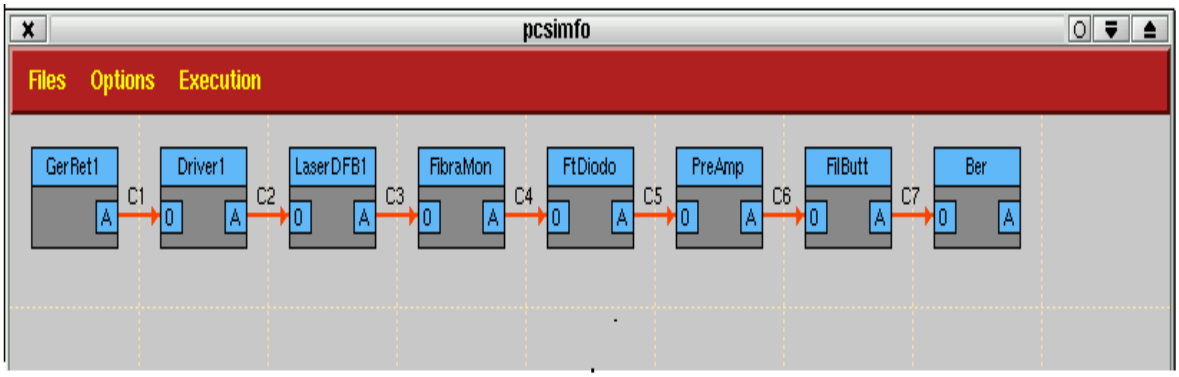

Figura 32: Simulador de um sistema óptico construído para estudo de caso

4.3 Organização de modelo

Os modelos são construídos utilizando-se o paradigma de orientação a objetos, onde são definidos formalmente através de uma classe de objetos contendo atributos de acesso privado e um conjunto de métodos que constituem a interface do objeto, implementados na linguagem de codificação é C/C++ **[7,8]**. Um modelo é constituído por uma definição de classe e um arquivo de inclusão contendo variáveis e estruturas de dados estáticas, onde estão definidas: os símbolos utilizados, a apresentação da caixa de diálogo contendo os atributos associados às informações paramétricas, os valores correntes para cada um dos atributos e a apresentação da caixa de diálogo associadas ao processamento eletivo.

4.4 Parâmetros globais

Existem informações de domínio publicam disponibilizadas a todos os modelos durante a execução de uma simulação, sendo informações relacionadas às diretivas genéricas utilizadas por todos os modelos. Durante o processo de tradução do simulador gráfico para o formato de execução é realizada uma cópia dos parâmetros globais para cada instância de modelo presente no simulador, de modo que esses parâmetros não possam ser alterados durante a execução do simulador. A definição de quais atributos e como será o arranjo de apresentação são definidos internamente ao ambiente não cabendo ao construtor de aplicativos alterá-los. A definição da caixa de diálogo utilizada para apresentação desses parâmetros é realizada de forma análoga aos parâmetros locais. A figura 33 apresenta a atual caixa de parâmetros globais.

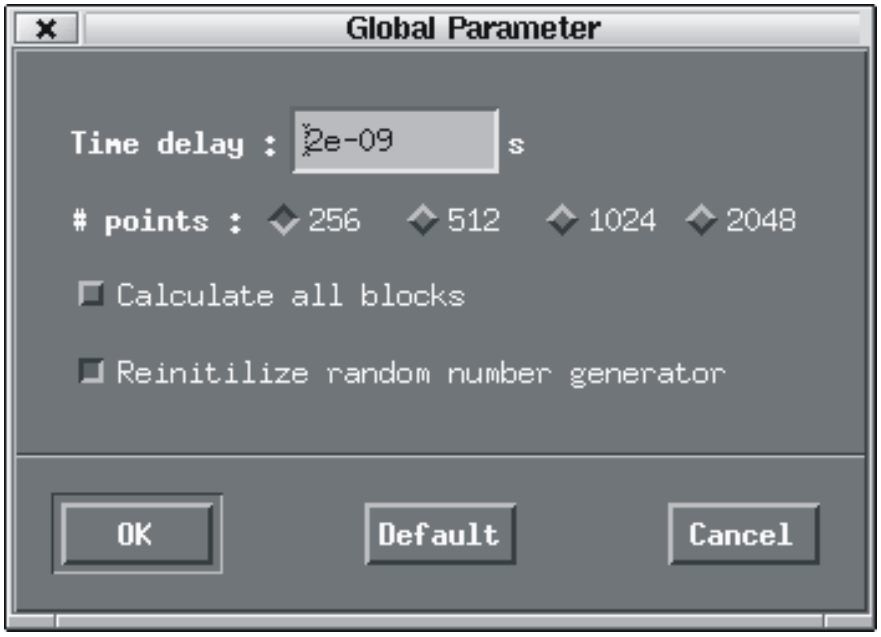

Figura 33: Caixa de diálogo contendo parâmetros globais

4.5 Classe Modelo

A forma adotada para construção de modelos inicia-se pela realização de uma cópia de uma classe modelo, onde estão definidos os métodos padrão e as estruturas de acesso padronizadas. Essa classe foi projetada de forma a herdar todos os métodos presentes nas classes básicas construídas na biblioteca de modelos. A estrutura da classe modelo foi construída em um arquivo com a extensão *modelo.cla* e seu conteúdo será apresentado na seqüência de código a seguir:

```
class MODELO: public MODULO_BASE, public ERFC
{
public:
INFOPARLOC vtploc[x];
MODELO_PLOC stploc;
//------------------------------------------------------------------------
MODELO ();
~MODELO ();
//------------------------------------------------------------------------
// Funções padradonizadas
bool ExecutarFuncao(int,void *),
     FunPosProc(int);
//------------------------------------------------------------------------
//Funções dedicadas
};
```

As estruturas de dados INFOPARLOC e MODELO_PLOC são utilizadas para obter e disponibilizar informações para o processamento realizado internamente ao modelo. Os métodos MODELO() e ~MODELO() serão substituídos pelos construtores da classe definitiva. As funções padronizadas são utilizadas pelo ambiente para ativar determinada parte do código. Para este estágio de desenvolvimento pode-se entender que o modelo possui apenas dois níveis de ativação, isto é, o nível associado ExecutarFuncao() e FunPosProc(). Nada impede que futuras atualizações possam utilizar um número superior de níveis de ativação, desde que sejam formas padronizadas de resposta sejam suportadas para os modelos que não possuam os novos níveis implementados. As funções dedicadas são desdobramentos das funções anteriores. Os métodos de classe foram definidos em um arquivo separado denominado modelo.C, sendo seu conteúdo apresentado nas seqüências de códigos a seguir:

```
#define  LongIdtLst   "Modelo"
#define  ShortIdtLst  "mod1"
#define  ClassIdtLst  "Modelo"
```

O fragmento de código anterior representa o cabeçalho utilizado para de identificação do modelo no momento de sua inclusão na biblioteca de modelos.

```
#include <IncDefMod.h>
#include "Modelo.cla"
#include "ModeloInc.h"
void CbModelo(void *v);
```

O fragmento de código anterior representa a regra de inclusão necessária à definição dos elementos internos ao modelo.

```
Modelo:: Modelo()
{
// parametros locais e globais
// informacoes modulares
// atribuicao de vetores de variaveis
}
```

O fragmento de código anterior representa o construtor da classe onde são disponibilizadas ao ambiente as estruturas de dados declaradas no modelo.

```
Modelo::~Modelo ()
{
}
```

O fragmento de código anterior representa o destrutor da classe onde são finalizadas as atividades associadas ao modelo.

```
bool Modelo::InicializarMod()
{
// inicialização das variáveis paramétricas
return true;
}
```

O fragmento de código anterior representa o estágio onde as variáveis que contém os parâmetros são inicializadas.

```
bool Modelo::IniciarVetParLoc(void *vd,Modelo_PLOC *pl)
{
// atribuição de variáveis paramétricas ao vetor de acesso global
return true;
}
```

O fragmento de código anterior representa o estágio onde as variáveis paramétricas são atribuídas a uma estrutura de dados que será utilizada para acesso ao valor dos parâmetros.

```
bool Modelo::ExecutarFuncao(int idseq,void *ptv)
{
// execução da função principal do modelo
return true;
}
```

O fragmento de código anterior representa o método de acesso à função principal dos modelos, onde são realizadas a maioria das atividades.

```
bool Modelo::FunPosProc(int opc)
{
// ativação das funções eleitivas no pós-processamento
return true;
}
```

O fragmento de código anterior representa o método para ativar as funções eletivas definidas par um modelo.

```
void CbModelo(void *v)
{
// função de callback para a caixa de diálogo contendo os atributos
// representativos dos parâmetros do modelo
}
```

O fragmento de código anterior representa a função de *callback* associada a uma caixa de diálogo, onde podem ser tratados os eventos ocorridos no ambiente gráfico.

4.1 Classe Porta

A classe porta é uma classe de serviço utilizada para prover um meio de comunicação do modelo com o exterior. Os dados que entram e saem do modelo são armazenados em arquivos de dados. Esses arquivos são suportados pela classe de serviço ARQUIVO que oferece uma interface capaz de organizar dados em forma de coleções, de forma ser possível o armazenamento de várias coleções de dados em um único arquivo físico, sendo cada coleção de dados identificada por uma seqüência alfanumérica. Para utilizar uma coleção de dados deve-se fornecer uma identificação e o tamanho dos registros

a serem armazenados. Para recuperar uma coleção deve-se novamente fornecer a identificação de coleção e como resultado será apresentado um apontador para uma área de dados genérica, cabendo ao solicitante a interpretação correta dos dados através de um *cast* para formato adequado. O nome do arquivo físico do arquivo gerado e a quantidade de arquivos necessários a total conectividade do modelo são providos pelo ambiente de simulação, na forma de vetores de portas de comunicação.

```
class PORTA
{
ARQUIVO *arquivo;
int acesso,       // WR, RD
    tipo;         //_BIN,_TXT
//--------------------------------------
PORTA();
~PORTA();
//--------------------------------------
void Fechar();
bool Abrir(),
     IdPorta(int,IDARQ *),
     IdPorta(int,IDARQ *,char *),
     Salvar(void *),
     Ident(char *),
     Ident(char *,int,int),
     TestarExistArq(),
     FinalisarCol();

int  ObterTamReg(),
     ObterIdpos();

char *RecuperarCol();
};
```

4.1 Construção de modelo

O presente estudo de caso irá apresentar a construção de um modelo de uma fibra monomodo. Esse modelo foi escolhido por utilizar todos os recursos disponíveis na interface de modelo. O código original do modelo foi obtido da versão para Windows, apresentada no Anexo A.

4.1.1 Parâmetros

Esse modelo possui o conjunto de parâmetros apresentados na tabela 5, onde constam as variáveis de retorno, o texto de pergunta que irá figurar na caixa de diálogo e os

valores correntes. Na tabela 6 são apresentados às diretivas para acesso ao processamento eletivo.

Identificação de Parâmetro	Texto de Pergunta	Valor Padrão
lambda	Wavelenght, lbda	1550
L	Fiber length, L (Km)	10
D	Dispersion coef., D (ps / nm . Km)	17
S	Diferential disp. coef., S (ps / nm^2.Km)	0.05
Alpha	Atennuation coef., alpha (dB / Km)	0.25
loss	Coupling loss, loss (dB)	0

Tabela 5: Parâmetros do modelo de uma fibra monomodo

Texto de Pergunta
Optical Power, Pf(t)
Chirp, dv(t)
Stationary Noise Dens, Srp(t)
Transfer Function, Hf(t)
Stationary Noise Dens, Srp(t)
Transfer Function, Hf(t)
Results

Tabela 6: Diretivas para processamento eletivo de uma fibra monomodo

4.1.2 Definições de estruturas de dados

As definições sistêmicas associadas a um modelo são agrupadas em um arquivo de inclusão (extensão .h), onde são declaradas as estruturas de dados utilizadas pelo modelo. Toda veiculação de informação no modelo que envolva o exterior será feita

utilizando-se uma estrutura de dados, não importando o formato dessa estrutura, pois a interface de modelo está apta suportar dados de forma heterogênea. A seguir serão apresentados vários trechos de código contendo definições.

```
//###################### simbolos ###################################
#define ID_FIBERPOWER    0
#define ID_CHIRP         1
#define ID_STATNOISEDENS 2
#define ID_TRANSFUNC     3
#define ID_RESULTS       4
```

O fragmento de código anterior apresenta os símbolos que serão utilizados ao longo do código do modelo.

```
//############ quantidade de conectores de dados ########################
QTDCON fbm1qtcon = {1,1,5};
```

O fragmento de código anterior identifica quantidade de conectores lógicos que estarão presentes na representação gráfica do modelo e número de portas que serão alocadas para comunicação física durante o processamento.

```
//#################### parâmetros locais ##############################
char
*fbm1p1310[] = {"0","1310","10","17","0.05","0.25","0"},
*fbm1p1550[] = {"1","1550","10","17","0.05","0.25","0"};
```

O fragmento de código anterior representa os valores correntes das variáveis paramétricas.

```
//################## menu da janela de parametros ####################
STAPRATRI fbmb1rb1[10] = {
{OCNewLine,"F"},
{OVTitulo,"Wavelenght, lbda:"},
{OCCallBack1,"",NULL},
{OCHorizontal},
```

```
{OVDefault,"0",NULL},
{OVLabel,"1310"},
{OVLabel,"1550"},
{OVLabel,"Other: "},
{OCEnd}};
```

O fragmento de código anterior representa a definição de um atributo gráfico presente em um caixa de diálogo. O ato de definir um atributo consiste em atribuir um título, uma possível função de *callback* **[9]**, um valor corrente, associar limites e encerrar formalmente a definição. A existência ou não de um desses itens está associada ao tipo de atributo que esteja sendo definido. A definição é expressa na forma de um vetor do tipo STAPRATRI, onde os diferentes níveis de definição utilizam a mesma estrutura, sendo a distinção realizada internamente ao ambiente. Essa forma de definição é bastante simples e flexível, sendo adequada ao tratamento estruturado em níveis de atributos.

```
STAPRATRI fbm1tf01[8] = {
{OCNewLine,"T"},
{OVTitulo,"  "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[1]},
{VRTPFT},
{OCEnd}};

STAPRATRI fbm1lbsep[3] = {
{OCNewLine,"T"},
{OVTitulo," "},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf1[8] = {
{OCNewLine,"T"},
{OVTitulo,"Fiber length, L ( Km ):                    "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[2]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf2[8] = {
{OCNewLine,"T"},
{OVTitulo,"Dispersion coef., D ( ps / nm . Km ):       "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[3]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf4[8] = {
{OCNewLine,"T"},
{OVTitulo,"Diferential disp. coef., S ( ps / nm^2.Km ):"},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
```

```
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[4]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf5[8] = {
{OCNewLine,"T"},
{OVTitulo,"Atennuation coef., alpha ( dB / Km ):       "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[5]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf6[8] = {
{OCNewLine,"T"},
{OVTitulo,"Coupling loss, loss ( dB ):                 "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[6]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------

STAPRMENU fbm1mploc1[15] = {
{OVRadioBox,fbmb1rb1},
{OVTextField,fbm1tf01},
{OVLabel,fbm1lbsep},
{OVTextField,fbm1tf1},
{OVTextField,fbm1tf2},
{OVTextField,fbm1tf4},
{OVTextField,fbm1tf5},
{OVTextField,fbm1tf6},
{OCEnd}};

STAPRMENU *fbm1vtmploc[3] = {fbm1mploc1,NULL};
```

A última porção do fragmento de código anterior representa o agrupamento das definições de todos os atributos em uma única estrutura que simboliza um conjunto de opções. Essa estrutura será disponibilizada ao ambiente que se encarregará de construir uma caixa de diálogo visual. Essa forma de definição de construções gráficas torna o processo de construção gráfico totalmente transparente ao modelo. A caixa de diálogo resultante é apresentada na figura 34.

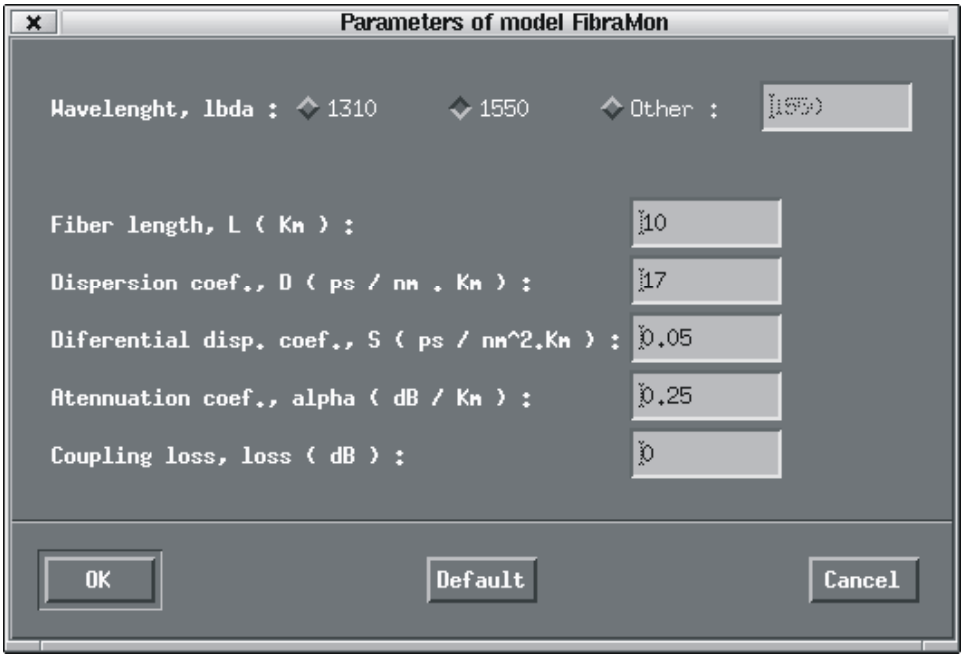

Figura 34: Caixa de diálogo da classe fibra monomodo

```
//################## menu da janela de pos-processamento #################
STAPRATRI fbm1pp1[4] = {
{OCNewLine,"T"},
{OVTitulo,"Optical Power, Pf(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp2[4] = {
{OCNewLine,"T"},
{OVTitulo,"Chirp, dv(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp3[4] = {
{OCNewLine,"T"},
{OVTitulo,"Stationary Noise Dens, Srp(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp4[4] = {
{OCNewLine,"T"},
{OVTitulo,"Transfer Function, Hf(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp5[4] = {
{OCNewLine,"T"},
{OVTitulo,"Results ..."},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
```

```
STAPRMENU fbm1mpproc[5] = {
{OVPushButton,fbm1pp1},
{OVPushButton,fbm1pp2},
{OVPushButton,fbm1pp3},
{OVPushButton,fbm1pp4},
{OCEnd}};
//################## modo de apresentacao de resultados #################
DEFARQ
fbm1mdapr1[2] = {{MDGRAF,_BIN}},
fbm1mdapr2[10] = {{MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                 };

MDAPRES fbm1mdapr = {fbm1mdapr1,fbm1mdapr2};
```

O fragmento de código anterior representa a definição e o agrupamento das definições associadas ao processamento eletivo.

4.1.3 Definição da classe FIBRAMON1

Após a definição das estruturas de dados utilizadas pela nova classe pode-se iniciar o seu desenvolvimento. Para este estudo, a base do desenvolvimento já foi realiza, ficando apenas a adequação do código a nova estrutura de modelo. Sendo assim, a nova classe FIBRAMON1 está definida no fragmento de código a seguir.

```
struct defFBM1PLOC { int lbd; float lambda,L,D,S,alpha,loss; };
typedef struct defFBM1PLOC FBM1PLOC;
//=====================================================================
class FIBRAMON1: public MODULO_BASE, public ERFC
{
public:
INFOPARLOC vtploc[10];
FBM1PLOC stploc;
//---------------------------------------------------------------------
FIBRAMON1();
~FIBRAMON1();
bool ExecutarFuncao(int,void *),
     FunPosProc(int),
     OpticalPower(),
     Chirp(),
     StationaryNoiseDens(),
```

```
     TransferFunction(),
     Results(),
     InicializarMod(),
     IniciarVetParLoc(void *,FBM1PLOC *);

void LinearFiber(float [],float [],int,float,float,float,float,float);
};
```

Os métodos definidos para essa classe serão apresentados nos próximos fragmentos de códigos, sendo comentados e divididos por importância funcional.

```
#define  LongIdtLst   "FbMon1"
#define  ShortIdtLst  "fbm1"
#define  ClassIdtLst  "FIBRAMON"
//---------------------------------------------------------------------
#include <IncDefMod.h>
//---------------------------------------------------------------------
#include "FibraMon1.cla"
#include "FibraMon1Inc.h"
//---------------------------------------------------------------------
void CbFibraMon1(void *v);
```

O fragmento de código anterior representa o cabeçalho de inicialização onde estão definidos quais arquivos de definições serão incluídos e as diretivas de identificação de classe (FbMon1, fbm1, FIBRAMON).

```
FIBRAMON1::FIBRAMON1()
{
// parametros locais e globais
IniciarVetParLoc((void *)&vtploc,&stploc);
IniciarVetParGlo((void *)&vtpglo,&stpglo);
//---------------------------------------------------------------------
// informacoes modulares
infomod.qcon     = &fbm1qtcon;
infomod.mploc    = fbm1vtmploc;
infomod.mpproc   = fbm1mpproc;
infomod.mdapres  = &fbm1mdapr;
infomod.funcbmpp = CbFibraMon1;
//---------------------------------------------------------------------
// atribuicao de vetores de variaveis
ploc = vtploc;
pglo = vtpglo;
strcpy(idclasse,ClassIdtLst);
}
```

O fragmento de código anterior representa o método construtor da classe, responsável em disponibilizar estruturas locais ao ambiente externo. Os comandos IniciarVetParLoc e IniciarVetParGlo expõem as estruturas internas que abrigam os parâmetros reservados e os globais respectivamente ao acesso externo. A estrutura *infomod* presta-se para agrupar informações de origens diversas, utilizadas pelo ambiente para organizar apresentação gráfica dos modelos bem como as caixas de diálogo.

```
//=========================================================================
bool FIBRAMON1::InicializarMod()
{
stploc.lambda = stploc.lambda * 1e-7;
stploc.L      = stploc.L      * 1e5;
stploc.D      = stploc.D      * 1.0e-10;
stploc.alpha  = stploc.alpha  * 1e-5;
stploc.S      = stploc.S      * 1.0e-3;
//-------------------------------------------------------------------------
return true;
}
```

O fragmento de código anterior é utilizado para adequar os valores correntes dos parâmetros às devidas unidades utilizadas nos cálculos necessários ao processamento.

```
//=========================================================================
bool FIBRAMON1::IniciarVetParLoc(void *vd,FBM1PLOC *pl)
{
int i = 0;
INFOPARLOC *v = (INFOPARLOC *)vd;
//-------------------------------------------------------------------------
v[i++].s = (char *)&pl->lbd;
v[i++].s = (char *)&pl->lambda;
v[i++].s = (char *)&pl->L;
v[i++].s = (char *)&pl->D;
v[i++].s = (char *)&pl->S;
v[i++].s = (char *)&pl->alpha;
v[i++].s = (char *)&pl->loss;
//-------------------------------------------------------------------------
return true;
}
```

O fragmento de código anterior apresenta a associação dos atributos de parâmetros locais ao vetor de informações do tipo INFOPARLOC. Esse vetor será utilizado para enviar e receber informações das caixas de diálogo gráficas associadas a também representação gráfica do modelo na planilha de principal.

```
//=========================================================================
bool FIBRAMON1::ExecutarFuncao(int idseq,void *ptv)
{
float *y,*f,*H,*Se;
int n,i,tregn,treg2n;
//-------------------------------------------------------------------------
if(!portain[0].TestarExistArq()) return false;
portain[0].Abrir();

if(!portain[0].Ident("main1x"))  return false;
f = (float *)portain[0].RecuperarCol();
tregn = portain[0].ObterTamReg();
n = (tregn / sizeof(float)) - 1;
```

O fragmento de código anterior é responsável em executar a função principal do modelo. No inicio do fragmento está evidenciado o processo de abertura de uma porta de comunicação (portain[0].TestarExistArq()), que representa o estabelecimento de um canal de dados com o exterior. Após a realização da abertura da porta, pode-se escolher e recuperar uma coleção de dados.

```
if(!portain[0].Ident("main1y"))   return false;
y = (float *)portain[0].RecuperarCol();
treg2n = portain[0].ObterTamReg();
ApresDado(LongIdtLst,"-0",n,2,f,y);
//---------------------------------------------------------------------
H = vetor(2*n);
//---------------------------------------------------------------------
LinearFiber (f,H,n,stploc.lambda,stploc.L,stploc.D,stploc.S,stploc.alpha);
portaout[0].Abrir();
portaout[0].Ident("col1x",tregn,COLECAO);  portaout[0].Salvar((void *)f);
portaout[0].Ident("col1y",treg2n,COLECAO); portaout[0].Salvar((void *)H);
```

O fragmento de código retrata inicialmente a recuperação de uma coleção de dados presente na porta de entrada 0, identificada por main1y, e a criação abertura de uma porta de comunicação de saída 0, onde serão criadas duas coleções de dados *col1x* e *col1y*,

```
ApresDado(LongIdtLst,"-1",n,2,f,H);
//---------------------------------------------------------------------
if(stploc.L > 0) CMultiply (y,H,1,n);
portaout[0].Ident("main1x",tregn,COLECAO);  portaout[0].Salvar((void *)f);
portaout[0].Ident("main1y",treg2n,COLECAO); portaout[0].Salvar((void *)y);

//---------------------------------------------------------------------
// Sinal + ruido do Campo Optico em frequencia
delete f; delete y;
if(!portain[0].Ident("colnx")) return false; f = (float
*)portain[0].RecuperarCol();
if(!portain[0].Ident("colny")) return false; y = (float
*)portain[0].RecuperarCol();
ApresDado(LongIdtLst,"-3.5",n,2,f,y);
if(stploc.L > 0) CMultiply (y,H,1,n);
portaout[0].Ident("colnx",tregn,COLECAO);  portaout[0].Salvar((void *)f);
portaout[0].Ident("colny",treg2n,COLECAO); portaout[0].Salvar((void *)y);
ApresDado(LongIdtLst,"-3",n,2,f,y);
//---------------------------------------------------------------------
// Dens. espectral do ruido estacionario
delete f;
if(!portain[0].Ident("col-x")) return false; f  = (float
*)portain[0].RecuperarCol();
if(!portain[0].Ident("col-y")) return false; Se = (float
*)portain[0].RecuperarCol();
ApresDado(LongIdtLst,"-4",n,1,f,Se);
if(stploc.L > 0)
  {
  for(i=1; i<=n; i++) Se[i]=Se[i]*modulo2(H[2*i-1], H[2*i]);
  }
portaout[0].Ident("col-x",tregn,COLECAO); portaout[0].Salvar((void *)f);
portaout[0].Ident("col-y",tregn,COLECAO); portaout[0].Salvar((void *)Se);
ApresDado(LongIdtLst,"-5",n,1,f,Se);
```

```
//---------------------------------------------------------------------------
portain[0].Fechar(); portaout[0].Fechar();
```

O fragmento de código anterior representa o restante de código pertencente à função principal e ao final, o fechamento da portas de saída, momento o qual será finalizada a operação de escrita.

```
//---------------------------------------------------------------------------
free(H); delete Se; delete y; delete f;
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::FunPosProc(int opc)
{
switch(opc)
      {
      case 0 : OpticalPower();         break;
      case 1 : Chirp();                break;
      case 2 : StationaryNoiseDens();  break;
      case 3 : TransferFunction();     break;
      case 4 : Results();     break;
      }
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::OpticalPower()
{
                    .
                    .
                    .

return true;
}
                    .
                    .
                    .
```

O fragmento de código anterior ilustra como são definidas e ativadas as funções associadas ao processamento eletivo ou posprocessamento. Essas funções utilizam os dados gerados anteriormente como entrada para seu processamento, sendo realizado de forma análoga ao realizado na função principal.

```
//===========================================================================
void CbFibraMon1(void *v)
{
VALOR val;
INFOPARLOC info[10];
FIBRAMON1 fbm1;
```

```
FBM1PLOC stploc;
OBJVIS *ob1;
//-------------------------------------------------------------------------
CALLBACK *cb = (CALLBACK *)v;
PAINEL *painel = (PAINEL *)cb->obj;
```

O fragmento de código anterior representa a função de retroativação dos atributos presentes nas caixas de diálogos. Todo evento gerado por qualquer atributo contido na caixa de diálogo desta classe terá como reposta a ativação dessa função. Como a função não é um método da classe, a mesma função será ativada apara todas as instâncias da classe presentes na simulação. A diferenciação é realizada através do envio de um apontador de identificação como argumento da função, a qual contem as informações necessárias para recuperação da estrutura de dados da caixa de diálogo que originou o evento.

```
//-------------------------------------------------------------------------
painel->ColetarValUsr(&val);
fbm1.IniciarVetParLoc((void *)&info,&stploc);
InicializarPar(info,&val);
```

O fragmento de código anterior representa como é realizada a coleta de informações associadas aos atributos contidos na caixa de diálogo de parâmetros.

```
//-------------------------------------------------------------------------
ob1 = (OBJVIS *)painel->atribox.atributo.vtinfo[1].info;
```

O fragmento de código anterior representa a obtenção de um apontador para um objeto presente na caixa de diálogo associado a um atributo gráfico localizado em um vetor de estruturas adequado a este fim.

```
switch(stploc.lbd)
      {
      case 0: ob1->Sensibilidade(false);
               InicializarVal(&val,fbm1p1310);
               break;
```

O fragmento de código anterior representa uma operação passível de ser aplicada ao objeto recuperado, que por sua vez resultara em uma mudança de estado do atributo gráfico associado.

```
      case 1: ob1->Sensibilidade(false);
               InicializarVal(&val,fbm1p1550);
               break;

      case 2: ob1->Sensibilidade(true);
               break;
      }
painel->AtualizarOpcVal(&val);
}
//=========================================================================
void FIBRAMON1::LinearFiber(float f[],float H[],int n,float lambdac,float
L,float D,float S,float alpha)
{
                     .
                     .
                     .
}
```

O fragmento de código anterior representa a possibilidade de extensão da quantidade de métodos de classe com a finalidade de conferir maior organização ao código gerado. A planilha gráfica identificada na figura 35 apresenta o principal resultado da execução do modelo FibraMon1.

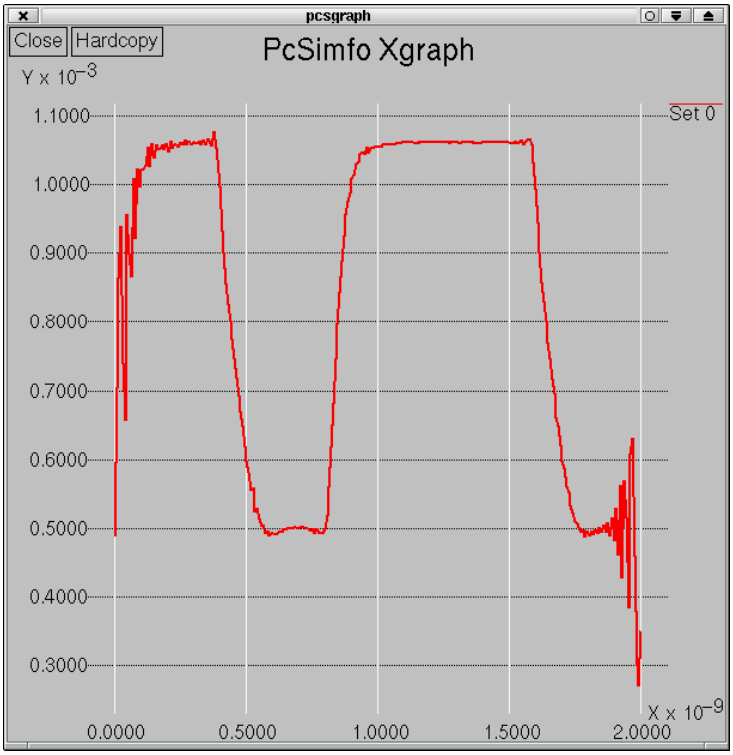

Figura 35 : Principal resultado obtido após execução do modelo FibraMon1.

Capítulo 5

Proposta de melhoria

5.1 Plano de atualizações

A versão do ambiente PcSimFO proposta nesse trabalho foi elaborada de modo a corrigir os problemas encontrados na versão anterior e atender a uma demanda de suporte operacional para a realização de experimentos em uma máquina pessoal ou conectada a uma rede local. O estudo realizado em outros ambientes aplicados ao suporte de simulação demonstrou existirem variações quanto a uso de simulação, formas de construção e topologia funcional. As diferenças encontradas podem ser analisadas através de dois pontos de vista: organização da funcional e as facilidades ofertadas. A organização funcional é estruturada de forma hierárquica, de modo que novas funcionalidades têm o caráter de superconjunto das anteriores, utilizando-as como base funcional para ampliar a abrangência de uso do ambiente. As facilidades associadas ao acréscimo funcional são agrupadas em conjuntos funcionais, os quais podem ser evoluídos de forma incremental, permitindo que o ambiente possa evoluir em diferentes direções de forma simultânea ou não. Isto significa que dependendo da estruturação interna, melhorias podem ser aplicadas em um setor do ambiente sem a necessidade imediata de atualização dos demais, o que permite a elaboração de uma estratégia de evolução.

Esses conceitos foram observados para a elaboração da proposta desenvolvida neste trabalho com a finalidade de proporcionar capacidade evolutiva ao ambiente, formalizada em um plano de atualizações. Esse plano apresenta etapas evolutivas onde novas funcionalidades são agregadas de forma incremental utilizando o legado existem como subconjunto funcional e aplicam gradualmente uma mudam de enfoque de utilização

do ambiente. Os tópicos a seguir apresentarão os estágios propostos para evolução do ambiente.

5.1.1 Conversor de formato

Na implementação do PcSimFO versão Linux existem dois aspectos importantes relacionados a organização de informações identificados quanto ao formato aplicado aos grupos de dados e a necessidade de realização de traduções de formato entre componentes. Os dados associados à representação de uma simulação pode ser armazenados em um arquivo de dados ou encaminhados para serem traduzidos e aplicados ao estágio responsável por executar a simulação. Quando os dados são armazenados em um arquivo é utilizado o formato binário de dados, o qual mostra-se útil e rápido para esse fim, mas impossibilita qualquer acesso ao seu conteúdo a não ser através do uso da interface ofertada pelo sistema. Ato de tradução de informações realizada quando um conjunto de dados é transposto de um estágio para outro configura uma operação vital para a integridade do sistema, pois a vinculação de um formato de dados a apenas um componente do sistema denota isolação funcional entre partes.

Dessa análise pode-se abstrair que será muito útil a existência de um dispositivo que realize a tradução de informações em um formato básico para outros formatos utilizados no sistema. O formato básico da informação poderá constituir um formato padrão de informação, aplicados inclusive para armazenamento de dados em arquivo, e nesse caso, os requerimentos básicos são acesso às informações sem a intervenção da interface do sistema e portabilidade de dados entre plataformas computacionais. Para satisfazer a esses requerimentos foi escolhida a organização em script de dados e o formato texto, os quais podem ser acessados via um editor de texto comum os dados estiverem em um arquivo de dados e sua interpretação é de fácil entendimento. No entanto essas novas características não aplicáveis a estrutura atual do ambiente de simulação devido ao fato de que a tarefa de armazenamento de dados e tradução estão dispersas entre os componente interface gráfica e núcleo estrutural, e por esse motivo existe a necessidade de construção de construção de

um novo componente denominado conversor de formato que será responsável por abrigar toda essa funcionalidade. A nova topologia do ambiente é apresentada figura 37 e as vantagens oferecidas por essa nova topologia serão:

- Estabelecimento de uma forma padronizada para troca de informações entre componentes do sistema, definindo uma interface estruturada para a realização de conversões de formato.

- Facilidade de atualização dos componentes, pois os mesmos possuem alta isolação funcional.

- O arranjo interno do conversor sempre poderá atualizado para representar a estrutura interna do ambiente.

- Meio de isolamento entre o sistema gráfico e o núcleo estrutural, proporcionando facilidade de tradução entre ambos.

- Possibilidade de o sistema operar sem a utilização da interface gráfica, fazendo com que o sistema seja acessado em linha de comando.

- Possibilidade de edição de uma simulação diretamente em arquivo, estando esse arquivo local ou remoto, através de um console textual em qualquer plataforma computacional.

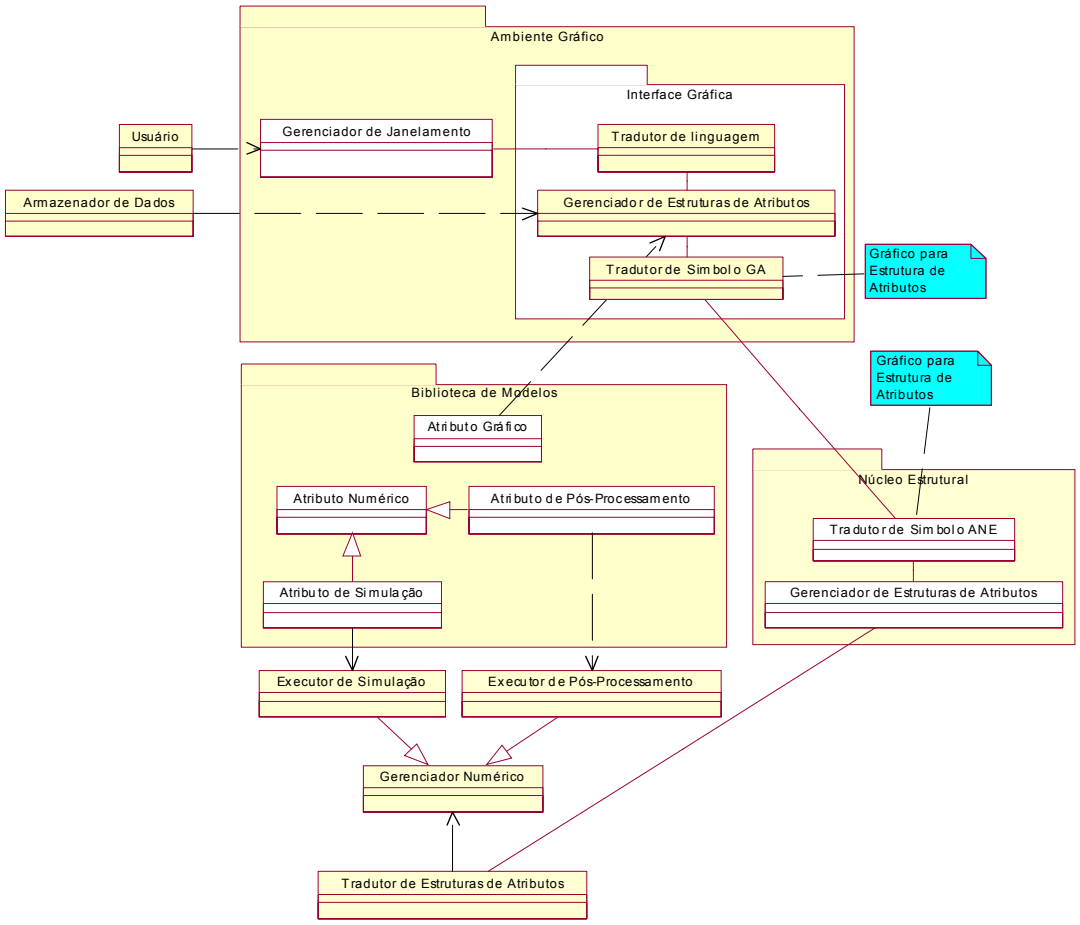

Figura 36 : Conversor de formato

5.1.2 Núcleo poliestrutural

Na proposta de ambiente de simulação implementada as simulações são constituídas por modelos provenientes de uma biblioteca associada ao ambiente, construída em uma única planilha gráfica contida em um único nível de abstração. O desenvolvimento de simulações complexas faz com que a apresentação desse arranjo seja complexa e carregada de muitos detalhes, o que sugere que se a mesma simulação fosse visualizada em diversos níveis de abstração proporcionaria maior conforto ao trabalho e aumento de produtividade. O conceito de níveis de abstração implica que uma simulação poderá ser estruturada em várias subpartes, constituindo uma hierarquia de abstrações e cada nível

será responsável por apresentar apenas uma parte da simulação, dotado de capacidade de troca de informações, pois na atual implementação o núcleo estrutural é composto por um objeto autocontido contendo todas as informações referentes a uma simulação e possui uma planilha gráfica associada responsável em editar o estado das informações contidas no objeto, não necessitando obter informações externas.

A presente proposta de atualização propõe as subpartes de uma simulação sejam associadas a diferentes objetos, e por conseqüência, a diferentes planilhas gráficas. A representação das subpartes de uma simulação será na porção principal será realizada de forma análoga à apresentação de um modelo de biblioteca, dando origem ao aparecimento de uma nova classe de informação denominada metamodelo. Assim uma simulação poderá será constituída por modelos e metamodelos, onde os metamodelos poderão ser editados em uma planilha gráfica adicional e o número de níveis presentes nessa hierarquia não possui limitante lógico dentro do sistema. A estrutura de biblioteca associada também será atualizada para abrigar os metamodelos. Para suportar a nova estrutura hierárquica será construído um gerenciador de poli estruturas encarregado pela ativação funcional dos objetos e pela veiculação de informações entre os mesmos. A nova topologia do ambiente é apresentada figura 38.

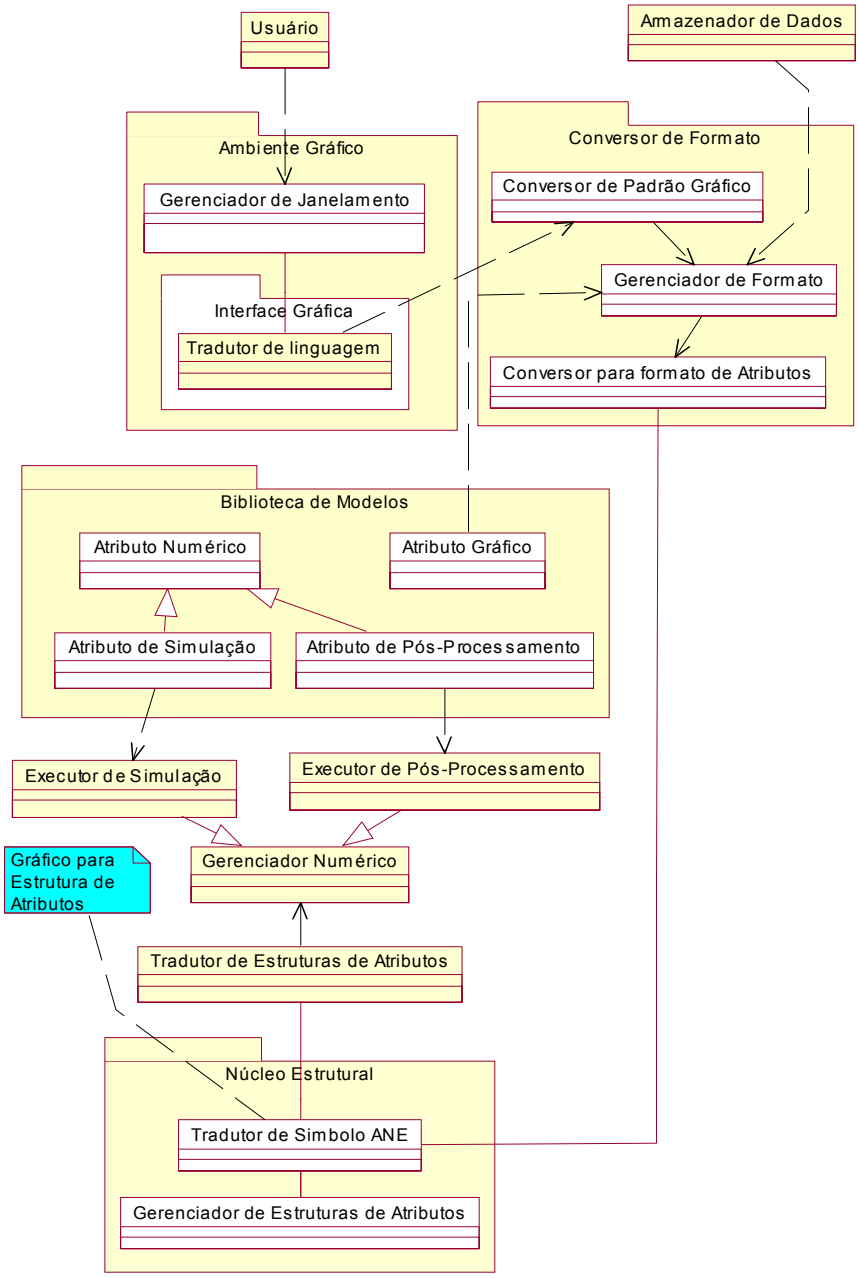

Figura 37 : Núcleo poli estrutural

5.1.3 Multi domínio de simulação

Na proposta de ambiente de simulação implementada as simulações são construídas apenas no domínio SDF, o qual suporta um esquema de avaliação seqüencial. Em muitas aplicações existe a necessidade do uso de eventos discretos, em especial em situações onde se deseja avaliar o desempenho dinâmico de um sistema. Em uma situação de maior generalidades, simulações poderão ser construídas para avaliação de um cenário dinâmico, onde porções desse cenário têm características seqüenciais. Para construção de uma simulação dessa natureza será necessária elaboração de um escalonador de simulação mais sofisticado que o atual, onde modelos criados especial para esse fim e metamodelos são avaliados no domínio de eventos discretos, sendo que os metamodelos são constituídos por modelos construídos para operar de modo seqüencial. Esse arranjo permite que modelos construídos anteriormente sejam reutilizados em novas simulações. A nova topologia do ambiente é apresentada figura 39.

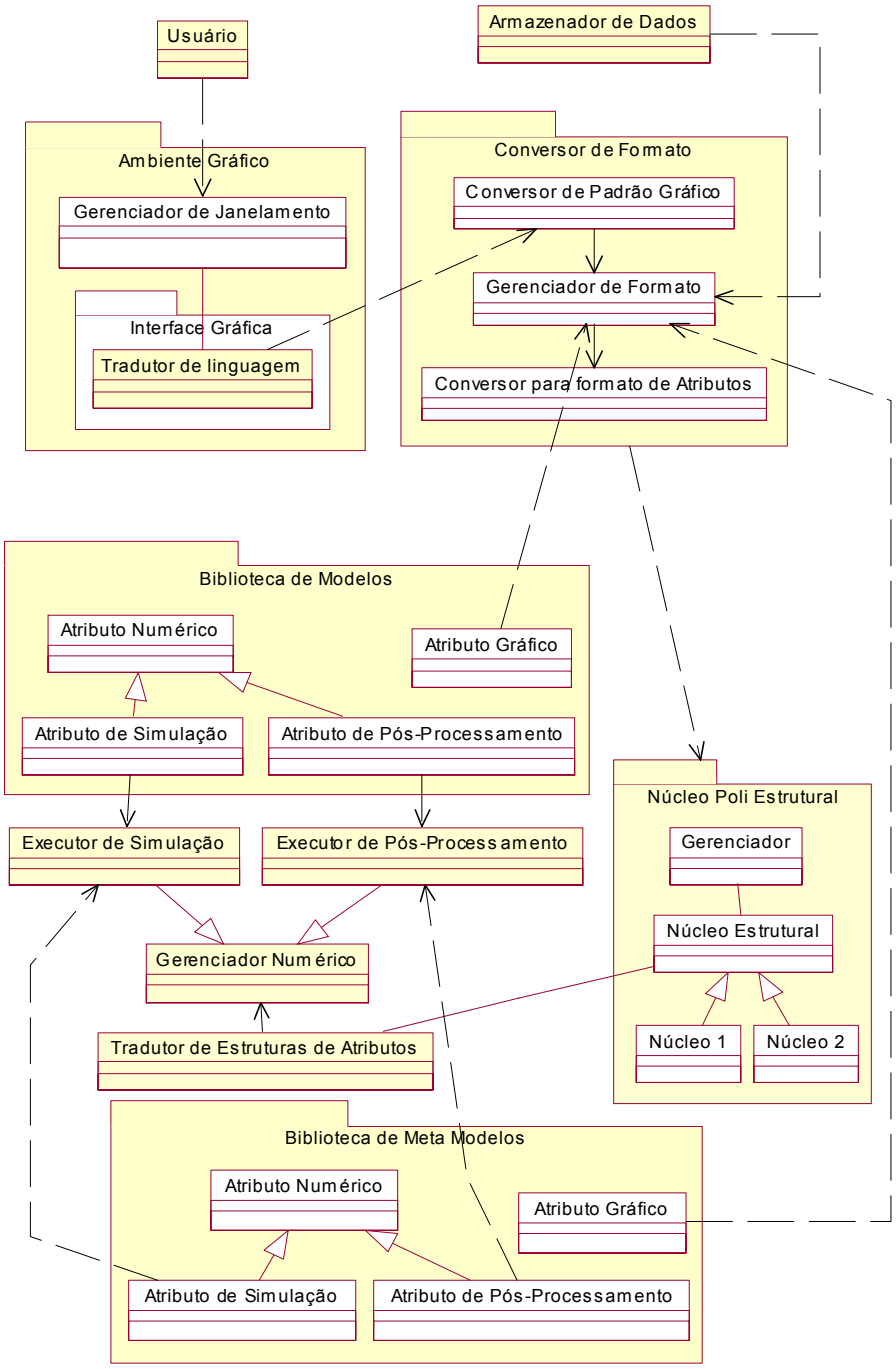

Figura 38 : Multi domínio de simulação

5.1.4 Servidor de serviços de simulação

As sucessivas atualizações proposta anteriormente conduziram a evolução do ambiente simulação por um processo de profunda especialização funcional, onde as classes definidas para compor o conjunto funcional inicialmente tiveram suas fronteiras remodeladas tornado-as coesas e muito especializadas a ponto de poderem operar em contexto diversos do original. A esse processo denomina-se componentização, onde cada componente é uma entidade independente realizando uma função com maior especialização do que a verificada anteriormente, apresenta alta isolação funcional, baixo acoplamento de dados devido à padronização dos dados que transpassam suas interfaces, o que torna transparente qualquer atualização realizada. Em decorrência disso, a harmonização entre componentes se torna muito estável. Uma vez que e a topologia interna do ambiente atingiu esse estado de desenvolvimento existe a possibilidade de se utilizar os componentes identificados para compor aplicativos com finalidade diferente do original, de modo que o conjunto de componentes que compõem o ambiente seja inserido como recursos em um sistema maior, o que atesta a reusabilidade dos componentes desenvolvidos.

Dando seqüência ao desenvolvimento do ambiente observa-se que a flexibilidade alcançada pela estrutura interna permite que seja efetuada uma mudança importante na estrutura do sistema associada à forma de montagem do ambiente, o qual é atualmente composto por um único aplicativo, o que defini a característica monolítica. Pode-se planejar uma modificação no conjunto estrutural, de modo a fragmentar o ambiente em peças não monolíticas. Essa transformação será efetuada de maneira que os componentes associados ao processamento numérico serão agrupados em um aplicativo denominado núcleo de processamento e os componentes associados a interface de edição gráfica serão remodelados para possam ser executados em um *brouser.* A comunicação com esse aplicativo será mediada por um componente denominado adaptador de mídia, que englobará as tarefas anteriormente realizadas pelo objeto conversor de formato. Esse componente possui um árbitro de diálogo responsável por controlar a comunicação entre o núcleo de processamento numérico e o brouser, o que possibilitará ao ambiente de simulação ser transformado em um serviço suportado pela Internet. Esse serviço seria

disponibilizado a comunidade através de uma página de um site residente na universidade, onde os interessados estabeleceriam conexões com o site e receberia uma página inicial onde além da abertura e informações gerais uma planilha apta à edição gráfica de simulações. Após a edição, os dados do arranjo serão enviados ao servidor para serem processados por uma plataforma adequada a esta tarefa. Finalizada essa etapa, os resultados associados a execução da simulação serão enviados ao interessado para serem apresentados. Essa topologia permite que vários tipos de mídia sejam utilizados para a execução das simulações, como exemplo, arquivos em formato texto. A execução das simulações será realizada de modo centralizado, o que possibilita a construção de um cluster para execução de simulações. A verão serviço para Internet é apresentada na figura 39.

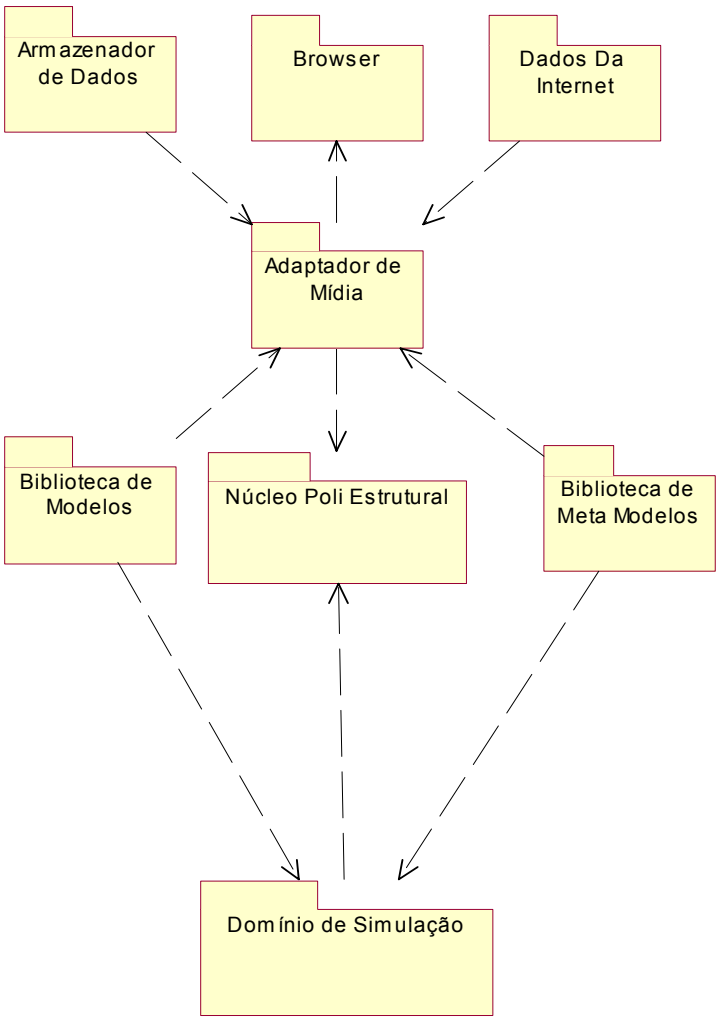

Figura 39 : Servidor de serviços de simulação

Conclusões

O objetivo desse trabalho foi de elaborar uma solução sistêmica para as atuais deficiências do ambiente PcSimFO, bem como adicionar novas características que possibilitem sua expansão, sendo ambos atingidos com êxito. A evolução dos trabalhos realizados foi guiada pelas metas de projeto e para cada item estabelecido foram obtidos os seguintes resultados:

1. O ambiente de simulação foi construído tendo uma planilha gráfica como elemento de entrada de dados e baseada nela, foi desenvolvida uma interface de edição capaz de inserir, deletar e alterar a configuração de modelos, proporcionando a facilidade de composição de arranjos compostos por conexões, execução e obtenção de resultados de forma automatizada

2. O ambiente pode ser executado em qualquer ambiente do tipo Unix. Na versão Linux pode-se observar um melhor desempenho, pois as bibliotecas de suporte gráfico utilizadas foram compiladas para uso em plataformas do tipo Intel, o que adicionou grande eficiência ao desempenho do sistema.

3. A construção do ambiente foi estruturada em blocos funcionais, os quais originaram os objetos que compõem o sistema, os quais foram dotados de interfaces projetadas para tornar o ambiente modular e proporcionar alta isolação funcional ao conjunto.

4. Os objetos associados ao processamento numérico foram agrupados em um conjunto e os relacionados à edição gráfica, agrupados em outro. A comunicação entre ambos é realizada por dois objetos que aplicam uma regra de diálogo padronizada, podendo ser atualizada sem qualquer prejuízo operacional. O conjunto que abriga as funções gráficas foi projetado para subutilizar as facilidades da

plataforma computacional, de modo que as facilidades que poderiam ser utilizadas foram mapeadas no conjunto associadas ao processamento numérico. Dessa forma, a funcionalidade exigida pelo ambiente esta inclusa, o que reduz a dependência de elementos externos.

5. Foi desenvolvida uma metodologia de construção de modelos simplificada, a qual obscurece todos os detalhes de processamento. A construção dos elementos visuais é formalizada através de uma linguagem proprietária, dispensando o conhecimento de regras de construção visual.

6. A biblioteca associada ao sistema foi construída utilizando código reentrante, o que proporcionou ao ambiente a capacidade de troca de toda a biblioteca sem ônus funcional para o ambiente. A linguagem utilizada para codificação dos modelos contidos na biblioteca foi C++. Como a estrutura aplicada à biblioteca é proprietária e a linguagem de codificação amplamente divulgada, pode-se portar a biblioteca para qualquer plataforma computacional.

7. O plano de atualização especificado anteriormente definiu uma seqüência de inovações aplicáveis ao ambiente tornando-o uma ferramenta com múltiplos usos, apto a evoluir para tornar-se um serviço disponível pela rede Internet.

A construção do ambiente foi acompanhada por dificuldades originadas na modelagem do ambiente e de origem operacional. A seguir serão apresentados alguns episódios dessa seara:

- A linguagem de codificação visual utilizada foi a MOTIF, a qual mostrou-se pouco amigável e com baixo nível de abstração funcional. Essas características tornaram o desenvolvimento moroso e adiciona uma complexidade desnecessária. Para futuros desdobramentos desse trabalho, seria aconselhável a utilização de uma ferramenta atualizada, como Kylix, adequada ao ambiente Linux.

- O desenvolvimento do sistema ocasionou a geração de um grande número de arquivos, que a princípio foram organizados em diferentes diretórios classificados por

função, afinidade ou adequação. Essa organização causou, em determinados momentos, dificuldade de localização de um determinado arquivo. Como a compilação era feita de forma automatizada, passado algum tempo sem a necessidade de atualização, era inevitável o esquecimento de onde se encontrava um ente específico. Esse problema seria solucionado com a utilização de uma árvore de diretórios menor.

- Como o sistema foi estruturado em uma árvore de diretórios extensa, houve a necessidade de criação de uma metodologia de compilação e montagem para a criação do aplicativo executável. A dificuldade dessa tarefa consistiu em pesquisar todos ramos dessa árvore de forma sistemática. O tamanho da árvore cresceu a tal ponto que foi necessário a construção de uma ferramenta de manutenção do aplicativo, onde eram realizadas cópias de segurança e compilação geral em vários níveis.

- Com o desenvolver dos trabalhos, a tarefa de construção do ambiente mostrou-se complexa além do esperado, o que sugere que um desenvolvimento melhor estruturado deverá ser realizado por uma equipe. A dificuldade de abstração dos componentes de suporte e o modelamento das classes de suporte visual foram responsáveis pela maioria das barreiras a serem transpostas. Essas dificuldades e o uso modesto de recursos da linguagem C++, em especial as classes template, deixaram marcas no modelamento do sistema, que no futuro deverão ser identificadas e corrigidas.

O plano de melhorias especificado é composto por uma seqüência de atualizações que devem ser realizadas na estrutura do sistema sem a necessidade de remodelamento geral. A ultima atualização aplica uma forte mudança funcional ao sistema sem o ônus de mudanças drásticas em sua estrutura. Isso denota que a modularidade aplicada e o baixo acoplamento funcional fazem do PcSimFO versão Linux um aplicativo computacional de grande utilidade.

Referências Bibliográficas

[1] - Geoffrey Gordon, "System Simulation", Prentice-Hall, 1967.

[2] - Fishman, G. S., "Principles of Discrete Event Simulation", A Wiley-Interscience Publication, 1978.

[3] - Hamdy A. Taha, "Simulation Modeling and SIMNET", Prentice-Hall, 1988.

[4] - Kevin Watkings, "Discrete Event Simulation in C", McGraw-HILL, 1993.

[5] - Averill M. Law, W. David Kelton, "Simulation Modeling & Analysis", McGraw-HILL, 1991.

[6] - Moshe A. Pollatschek, "Programming Discrete Simulation",R&D Books, 1995.

[7] - Herbert Schildt, "The Complete Reference of C++", Trird Edition, McGraw- HILL, 1998.

[8] - Robert B. Murray, "C++ Strategies and Tactics", AT&T Press, 1993.

[9] - Marshall Brain, "Motif Programming", Digital Press, 1992.

[10] - Clemens Szyperski, "Component Software", Addison-Wesley, 1998.

[11] - Roger S. Pressman, "Engenharia de Software", McGraw-HILL, Trird Edition, 1995.

[12] - “LINKSIM Simulation and Analysis for Optical Communication Links”, Optiwave Corp., www.optiwave.com, 1999.

[13] - I. Jacobson, M. Griss, P. Jonsson, “Software Reuse”, ACM Press Books, 1997.

[14] - José D. Fulan, “ Modelagem de Objetos através da UML”, Maron Books, 1998.

[15] - Terry Quatrani, “Visual Modeling with Rational Rose and UML”, Addison-Wesley, 1998.

[16] - Terrence Chan, “Unix System Programminf Using C++”,Prentice Hall, 1997.

[17] - L. L. Arthur, Ted Burns, “Unix shell Programming”, John Wiley & Sons, Inc, 1997.

[18] - Gerd Keiser, “Optical Communications”, McGraw-Hill, 1997.

[19] - Govind P. Agrawal, “Fiber-Optic Communication System”, Wiley-Interscience Publication, 1992.

[20] - András Varga, “OMNeT++ Discrete Event Simualtion System”, Technical University of

Budapest, 1999.

[21] - “BONeS Block Oriented Network Simulator”, Comdisco Sytems, Inc, 1990.

[22] - “SPW Signal Processing Worksystem”, Cadence, www.cadence.com, 1999.

[23] - F. Mouveaux, F. Lapeyrere, G. Golmie, “MERLiN Modelin Evaluation and Research of Lightwave Networks”, National Institute of Standarts and Technology, www.antd.nist.gov, 2000.

[24] - M. Yong, D. Argiro, S. Kubica, “Cantata : Visual Programming Environment for Khoros System”, Khoral Research, Inc, www.khoral.com, 2000.

[25] - E. Ghittori, M. Mosconi, M. Porta, “Desiging and Testing new Programming Constructs in a Data Flow Vl, Dipartimento di Informaticz e Sistemistica – Paiva, www.iride.inipv.it, 2000.

[26] - M. Biersack, V. Friesen, “Towards an Architecture for Simulation Environments”, Computer Simulation Conference (SCSC95), 205-212, 1995.

[27] - E. Hardee, K. Chang, “A CAD-based Design Parameterization for Shape Optimization of Elastic Solids”, Advances in Engineering Software, 30:185-199, 1999.

[28] - R. Bagrodia, “Maisie”, UCLA Parallel Computing Lab, www.may.cs.ucla.edu/projetcs/maisie, 1997.

[29] - Juan M. A. Coello, "Suporte de Tempo Real para um Ambiente de Programação Concorrente", Dissertação de mestrado (FEE), 1986.

[30] - University of California at Berkeley, "Vol. 2 - Ptolemy 0.7 Programmer’s Manual, 1997.

[31] - University of California at Berkeley, "Vol. 3 - Ptolemy 0.7 Kernel Manual, 1997.

[32] - Peter Ball, "Introduction to Discrete Event Simulation", www.strath.ac.uk/Departments/DMEM/MSRG/simulate.html, 1998.

[33] - "Bibliography for Discrete Event Systems Simulation: Optimization and Sensitivity Analysis", http://ubmail.ubalt.edu/~harsham/ref/RefSim.htm.

[34] - Paul A. Fiswick, "Simulation Model Design and Execution", Prentice Hall, 1994.

[35] - Paul A. Fiswick, "A Simulation Environment for Multimodeling", ACM Transations and Modeling and Computer Simulation, 2(1):52-81, 1992.

[36] - J. Darmont, “DESP-C++ : a discrete-event simulation package for C++”, Software-Pratice and Experience, 30:37-60,2000.

[37] - C. Hu, F. Wang, “Constructing an Integrate Visual Programming Environment”, Software-Pratice and Experience, 28:773-798.

[38] - C. J. Coomber, “SCHEMASIM: A Simulation Environment for Real-Time System”, Simulation

Digest, 23:50-63, 1994.

[39] - K. chang, L. Murphy, J.D. Fouss, “Software Development and Integration in a Computer Supported Cooperative Work Environment”, Software-Practice and Experience,28(6):657-679, 1998.

[40] - Sandro M. Rossi, Edson Moschim, "Simulation of High Speed Optical Fiber System Using PC-SIMFO", 47th IEEE Eletronic Component and Technology Conference, San Jose/CA - EUA, Maio/1997.

[41] - Luís C. Kakimoto, Sandro M. Rossi, Edson Moschim, "Simulation of WDM System Using PC-SIMFO", 47th IEEE Eletronic Component and Technology Conference, San Jose/CA - EUA, Maio/1997.

[42] - “LINKSIM”,

Anexo A

A.1 Código fonte

A.1.1 Fibra Monomodo versão Windows

```
#include "pcsimfo.h"
#include "const.h"

#define ID_FIBERPOWER   0
#define ID_CHIRP  1
#define ID_STATNOISEDENS 2
#define ID_TRANSFUNC    3           // Amplitude e fase
#define ID_RESULTS 4

#define IDLOADPARAM     3

static char *DefParam[4] =
      {"0.65e-4 -250e-10 0.162e-3 10e-5 1.473",
       "0.85e-4 -100e-10 0.138e-3 3e-5 1.468",
       "1.31e-4 0 0.085e-3 0.4e-5 1.462",
       "1.55e-4 17e-10 0.08e-3 0.25e-5 1.462"};

typedef struct tagPARAM
{
      float lambda,L,D,S,alpha,loss,ng; //,sigma;
//    int source;
      HMENU hmenu;
      HWND janGraph[6];
      char FName[6][40];
      BOOL bDataModified, bCalculated;
} PARAM;

#define ID_LASER        0
#define ID_LED                1

static PARAM *param;
static char szNomeBloco[] = "FIBRA_MONO";
static float lambda,L,D,S,alpha,loss,ng; //,sigma;

extern float delay;
extern HANDLE hCop;
extern FARPROC wndProcBloco;
extern char szNomeProj[];
```

```
extern Colors[];

long FAR PASCAL _export ProcFibraMono (HWND hjan, UINT mensagem, UINT
wParam, LONG lParam);
FAR PASCAL _export DlgFibraMono (HWND hDlg, UINT mensagem, UINT wParam,
LONG lParam);
long ShowGraphFibraMono (HWND hjan, UINT id);
BOOL CalcFibraMono (HWND hjan, char *szNomeArqIn, char *szNomeArqOut,
BOOL bInputModified);
void LinearFiber (float f[], float H[], int n, float lambdac, float L,
float D,
      float S, float alpha, float ng);

long FAR PASCAL _export ProcFibraMono (HWND hjan, UINT mensagem, UINT
wParam, LONG lParam)
      {
      int i,j,num;
      param = (PARAM *) GetWindowLong (hjan, 0);
      if (param != NULL)
            {
            lambda = param->lambda;
            L= param->L;
            D = param->D;
            S = param->S;
            alpha = param->alpha;
            loss= param->loss;
            ng= param->ng;
            }

      switch (mensagem)
            {
             case WM_CREATE:
                  param = (PARAM *) malloc ((size_t) sizeof(PARAM));
                  param->lambda=1.55e-4;// cm
                  param->L=10e5; // cm
                  param->D=17e-10; // s/cm.cm
                  param->S=0.05e-3; // s/cm^2.cm
                  param->alpha=0.25e-5; // dB/cm
                  param->loss=0; // dB
                  param->ng=1.462;
                  param->hmenu = LoadMenu (hCop, szNomeBloco);
                  ResetAllGraph (param->janGraph, ARRAY_SIZE(param-
>janGraph));
                  param->bDataModified = FALSE;
                  param->bCalculated = FALSE;
                  SetWindowLong (hjan, 0, (LONG) param);
                  return 0;

            case WM_DESTROY:
                  CloseAllWindow (param->janGraph, ARRAY_SIZE(param-
>janGraph));
                  DestroyMenu (param->hmenu);
                  free (param);
                  SetWindowLong (hjan, 0, 0L);
                  return 0;

            case WM_PARENTNOTIFY:
                  if ((wParam == WM_DESTROY) && (HIWORD (lParam) ==
ID_GRAPH))
```

```
                        ResetThisGraph (LOWORD (lParam),param->janGraph,
ARRAY_SIZE(param->janGraph));
                  break;

            case WM_LBUTTONDBLCLK:
                  EditParam (hjan, szNomeBloco, DlgFibraMono);
                  return 0;

            case WM_RBUTTONDOWN:
                  ShowMenu (hjan, lParam, param->hmenu, param->janGraph);
                  return 0;

            case WM_COMMAND:
                  return ShowOutput (hjan, wParam, param->janGraph,
ShowGraphFibraMono);

            case WM_PAINT:
                  {
                  HDC hdc;
                  PAINTSTRUCT ps;
                  RECT rc;
                  int dx, dy;

                  hdc = BeginPaint (hjan, &ps);
                  SaveDC (hdc);
                  SelectObject (hdc, Colors[IDC_OPTICAL]);

                  GetClientRect (hjan, &rc);
                  Rectangle (hdc, rc.left, rc.top, rc.right, rc.bottom);

                  dx = rc.right/10;
                  dy = rc.bottom/10;
                  SelectObject (hdc, CreatePen (PS_SOLID, 1, RGB
(255,0,0)));
                  SelectObject (hdc, GetStockObject (NULL_BRUSH));
                  Ellipse (hdc, 3*dx, 3*dy, 7*dx, 8*dy);
                  Ellipse (hdc, 4*dx, 3*dy, 8*dx, 8*dy);
                  MoveTo (hdc, 2*dx, 8*dy);
                  LineTo (hdc, 9*dx, 8*dy);
                  DeleteObject (SelectObject (hdc, GetStockObject
(BLACK_PEN)));

                  if (!param->bCalculated)
                        GrayRect (hdc, rc.left, rc.top, rc.right,
rc.bottom);
                  RestoreDC (hdc,-1);
                  EndPaint (hjan, &ps);
                  return 0;
                  }

            case CM_INIT:
                  return 0;

            case CM_LOAD:
                  fscanf ((FILE *) lParam, "%g %g %g %g %g %g\n", &param-
>lambda, &param->L,
                        &param->D, &param->S, &param->alpha, &param-
>loss);//, &param->ng);
                  param->bDataModified = param->bCalculated = FALSE;
                  fscanf ((FILE *) lParam, "%i", &num);
```

```
                  for (i=0;i<num;i++)
                        {
                        fscanf((FILE *) lParam, " %i", &j);
                        ShowGraphFibraMono(hjan, j);
                        }
                  fscanf ((FILE *) lParam, "\n");
                  return 0;

            case CM_SAVE:
                  fprintf ((FILE *) lParam, "%g %g %g %g %g %g\n", param-
>lambda, param->L,
                        param->D, param->S, param->alpha, param->loss);//,
param->ng);
                  param->bDataModified = FALSE;
                  num=0;
                  for (i=0;i<ARRAY_SIZE(param->janGraph);i++)
                        if (ExistWindow(param->janGraph[i]))
                              num++;
                  fprintf ((FILE *) lParam, "%i", num);
                  for (i=0;i<ARRAY_SIZE(param->janGraph);i++)
                        if (ExistWindow(param->janGraph[i]))
                              fprintf((FILE *) lParam, " %i", i);
                  fprintf ((FILE *) lParam, "\n");
                  return 0;

            case CM_CANCLOSE:
                  return !param->bDataModified;

            case CM_CALC:
                  {
                  char szNomeArqOut[80];
                  BOOL result;

                  if (wParam)
                        {
                        param->bCalculated = FALSE;
                        InvalidateRect (hjan, NULL, TRUE);
                        UpdateWindow (hjan);
                        }
                  result = CalcFibraMono  (hjan, (char *) lParam,
szNomeArqOut, wParam);
                  strcpy ((char *) lParam, szNomeArqOut);
                  InvalidateRect (hjan, NULL, TRUE);
                  return result;
                  }
            }
      return CallWindowProc (wndProcBloco, hjan, mensagem, wParam,
lParam);
      }

BOOL FAR PASCAL _export DlgFibraMono2 (HWND hDlg, UINT mensagem, UINT
wParam, LONG lParam)
      {
      switch (mensagem)
            {
            case WM_INITDIALOG:
                  CheckRadioButton (hDlg, IDD+1, IDD+3, IDD+1);
                  return 1;

            case WM_COMMAND:
```

```
                  switch (wParam)
                        {
                        case IDOK: {
                              int opc = GetRadioButtonChecked (hDlg,
IDD+1, IDD+4)-(IDD+1);
                              sscanf (DefParam[opc], "%g %g %g %g %g\n",
&param->lambda,
                                    &param->D, &param->S, &param->alpha,
&param->ng);
                              PostMessage (GetParent (hDlg),
WM_INITDIALOG, 0, 0L);
                              EndDialog (hDlg, wParam);
                              return 1;
                              }

                        case IDCANCEL:
                              EndDialog (hDlg, wParam);
                              return 1;
                        }
            break;
            }
      return 0;
      }

FAR PASCAL _export DlgFibraMono (HWND hDlg, UINT mensagem, UINT wParam,
LONG lParam)
      {
       switch (mensagem)
            {
            case WM_INITDIALOG:
                  SetDlgItemFloat (hDlg, IDD+1, "%.7g",param->
lambda/1.0e-7);
                  SetDlgItemFloat (hDlg, IDD+2, "%.4g",param-> L/1e5);
                  SetDlgItemFloat (hDlg, IDD+3, "%.4g",param-> D/1.0e-
10);
                  SetDlgItemFloat (hDlg, IDD+4, "%.4g",param-> S/1.0e-3);
                  SetDlgItemFloat (hDlg, IDD+5, "%.4g",param-> alpha/1e-
5);
                  SetDlgItemFloat (hDlg, IDD+6, "%.4g",param-> loss);
                  return TRUE;

              case WM_COMMAND:
                  switch (wParam)
                        {
                        case IDLOADPARAM: {
                              char szTemp[20];

                              strcat (strcpy (szTemp, szNomeBloco),
"ADV");
                              DialogBox (hCop, szTemp, hDlg,
DlgFibraMono2);
                              return 1;
                              }

                        case IDOK: {
                              int i;
                              for (i=IDD+1;i<=IDD+6;i++)
                                    if (CheckDlgItemFloat (hDlg, i))
return 1;
```

```
                              param->lambda = GetDlgItemFloat (hDlg,
IDD+1)*1.0e-7;
                              param->L = GetDlgItemFloat (hDlg,
IDD+2)*1e5;
                              param->D = GetDlgItemFloat (hDlg,
IDD+3)*1.0e-10;
                              param->S = GetDlgItemFloat (hDlg,
IDD+4)*1.0e-3;
                              param->alpha = GetDlgItemFloat (hDlg,
IDD+5)*1e-5;
                              param->loss = GetDlgItemFloat (hDlg, IDD+6);
                              param->bDataModified = TRUE;
                              param->bCalculated = FALSE;
                              InvalidateRect (GetParent(hDlg), NULL,
TRUE);
                              DestroyWindow (hDlg);
                              return TRUE;
                              }

                        case IDCANCEL:
                              DestroyWindow (hDlg);
                              return TRUE;
                        }
                  break;
            }
      return FALSE;
       }

long ShowGraphFibraMono (HWND hjan, UINT id)
      {
      char szArq[80];
      float *x, *y;
      char Title[100], buffer[100];
      int i,n;

      GetWindowText (hjan, Title, sizeof(Title));
      GetMenuString (GetSubMenu (param->hmenu, 0), id, buffer,
sizeof(buffer), MF_BYCOMMAND);
      strcat (strcat (Title," - "), buffer);
      switch (id)
            {
            case ID_FIBERPOWER:
                  {
                  MakeOutputFileName (hjan, szArq, szNomeProj, '0');
                  n=ReadDataFromFile (szArq, &x, &y, 0, 2);

                  cfft (y,n,-1,1.0/delay);
                  for (i=1;i<=n;i++)
                        {
                        x[i]=((i-1)*delay/(n-1));
                        y[i]=modulo2 (y[2*i-1], y[2*i]);
                        }

                        MakeOutputFileName (hjan, param->FName[id],
szNomeProj, 'P');
                  WriteDataToFile (param->FName[id], x,y,n,1);

                  free (y);
                  free (x);
```

```
                  ShowJanGraph (&param->janGraph[id], Title, "Time (s)",
"Optical power (W)",
                        param->FName[id], 1.0, 1.0, hjan);
                  break;
                  }

            case ID_CHIRP:
                  {
                  float aux,ymed;

                  MakeOutputFileName (hjan, szArq, szNomeProj, '0');
                  n = ReadDataFromFile (szArq, &x, &y, 0, 2);
                  cfft (y,n,-1,1.0/delay);
                  for (i=1;i<=n;i++)
                        {
                        x[i]=((i-1)*delay/(n-1));
                        y[i]=fase(y[2*i-1],y[2*i]);
                        }
                  Differ (x, y, n);
                  for (i=1;i<=n-2;i++)
                        {
                        ymed=(y[i]+y[i+2])/2.0;
                        if (fabs(y[i+1]) > 10.*fabs(ymed))
                              y[i+1]=ymed;
                        }

                  aux=1.0/(2.0*M_PI);
                  for (i=1;i<=n;i++)
                        y[i]=aux*y[i];
                        MakeOutputFileName (hjan, param->FName[id],
szNomeProj, 'C');
                  WriteDataToFile (param->FName[id], x,y,n,1);

                  ShowJanGraph (&param->janGraph[id], Title, "Time (s)",
"Chirp (Hz)",
                        param->FName[id], 1.0, 1.0, hjan);

                  break;
                  }

            case ID_STATNOISEDENS:
                  {
                  MakeOutputFileName (hjan, szArq, szNomeProj, '-');
                  n=ReadDataFromFile (szArq, &x, &y, 0, 1);

                        MakeOutputFileName (hjan, param->FName[id],
szNomeProj, 'T');
                  WriteDataToFile (param->FName[id], x,y,n,1);
                  free (y);
                  free (x);

                  ShowJanGraph (&param->janGraph[id], Title, "Frequency
(Hz)", "Stationary noise dens. (W/Hz)",
                        param->FName[id], 1.0, 1.0, hjan);
                  break;
                  }

            case ID_TRANSFUNC:
                  {
                  float *y1;
```

```
                  MakeOutputFileName (hjan, szArq, szNomeProj, '1');
                  n=ReadDataFromFile (szArq, &x, &y, 0, 2);
                  y1=vector(n);
                  for (i=1;i<=n;i++)
                        y1[i]=modulo (y[2*i-1],y[2*i]);

                        MakeOutputFileName (hjan, param->FName[id],
szNomeProj, 'Y');
                  WriteDataToFile (param->FName[id], x,y1,n,1);

                  ShowJanGraph (&param->janGraph[id], Title, "Frequency
(Hz)", "|H[f]|",
                        param->FName[id], 1.0, 1.0, hjan);

                  for (i=1;i<=n;i++)
                        y1[i] = fase (y[2*i-1], y[2*i]);
                  UnwrapPhase (y1, n);
                  for (i=1;i<=n;i++)
                        y1[i] = y1[i]/M_PI;

                        MakeOutputFileName (hjan, param-
>FName[ID_RESULTS+1], szNomeProj, 'F');
                  WriteDataToFile (param->FName[ID_RESULTS+1], x,y1,n,1);
                  free (y);
                  free (x);
                  free (y1);

                  ShowJanGraph (&param->janGraph[ID_RESULTS+1], Title,
"Frequency (Hz)", "Phase (x180°)",
                        param->FName[ID_RESULTS+1], 1.0, 1.0, hjan);

                  break;
                  }

            case ID_RESULTS:
                  {
                  float *z, Imed=0., Nmed=0.;
                  int ii;
                  FILE *arq;

                  MakeOutputFileName (hjan, szArq, szNomeProj, '0');
                  n = ReadDataFromFile (szArq, &x, &y, 0, 2);
                  z = vector(2*n);
                  if (n != 0)
                        {
                        cfft (y,n,-1,(x[n]-x[1])/(n-1));
                        MakeOutputFileName (hjan, szArq, szNomeProj, 'N');
                        ReadDataFromFile (szArq, &x, &z, n, 2);
                        cfft (z,n,-1,(x[n]-x[1])/(n-1));
                        for (i=1;i<=n;i++)
                              {
                              ii=2*i;
                              z[ii-1] = z[ii-1] - y[ii-1];
                              z[ii] = z[ii] - y[ii];
                              Imed = Imed + modulo2(y[ii-1], y[ii]);
                              Nmed = Nmed + modulo2(z[ii-1], z[ii]);
                              }
                        Imed = Imed/n;
                        Nmed = Nmed/n;
```

```
                        }

                  free (z);
                  free (x);
                  free (y);

                        MakeOutputFileName (hjan, param->FName[id],
szNomeProj, 'R');

                  arq = fopen (param->FName[id], "wt");
                  fprintf(arq,
                        "Pmed(signal) = %4.4f dBm (%.4g mW)\n"
                        "Pmed(noise) = %4.4f dBm (%.4g mW)\n"
                        "SNR = %4.4f dB\n",
                        dB(Imed,1e-3), Imed/1e-3,
                        dB(Nmed,1e-3), Nmed/1e-3,
                        dB(Imed,Nmed));
                  fclose (arq);

                  ShowJanResult (&param->janGraph[id], Title, param-
>FName[id], hjan);
                  break;
                  }
            }
      return 0;
      }

BOOL CalcFibraMono (HWND hjan, char *szNomeArqIn, char *szNomeArqOut,
BOOL bInputModified)
      {
      char szArq[_MAX_PATH];
      float *y, *f, *H, *Se;
      int n, i;

      MakeOutputFileName (hjan, szNomeArqOut, szNomeArqIn, '0');
      if (!bInputModified && param->bCalculated) return 0;

      n = ReadDataFromFile (szNomeArqIn, &f, &y, 0, 2);

      H = vector (2*n);

      LinearFiber (f,H,n,lambda,L,D,S,alpha,ng);

      MakeOutputFileName (hjan, szArq, szNomeArqIn, '1');
      WriteDataToFile (szArq,f,H,n,2);

      if (L > 0) CMultiply (y,H,1,n);

      MakeOutputFileName (hjan, szArq, szNomeArqIn, '0');
      WriteDataToFile (szArq,f,y,n,2);

      szArq[strlen (strcpy (szArq, szNomeArqIn))-1]='N';
      ReadDataFromFile (szArq, &f, &y, n, 2);

      if (L > 0) CMultiply (y,H,1,n);

      MakeOutputFileName (hjan, szArq, szNomeArqIn, 'N');
      WriteDataToFile (szArq,f,y,n,2);

      Se = vector (n);
```

```
      szArq[strlen (strcpy (szArq, szNomeArqIn))-1]='-';
      ReadDataFromFile (szArq, &f, &Se, n, 1);

      if (L > 0)
            for (i=1; i<=n; i++)
                  Se[i]=Se[i]*modulo2(H[2*i-1], H[2*i]);

      MakeOutputFileName (hjan, szArq, szNomeArqIn, '-');  // dens.
espectral de ruido estacionario
      WriteDataToFile (szArq,f,Se,n,1);

      free (H);
      free (Se);
      free (y);
      free (f);

      for (i=0;i<ARRAY_SIZE(param->janGraph);i++)
        if (ExistWindow (param->janGraph[i]))
                  ShowGraphFibraMono (hjan,i);
      param->bCalculated = TRUE;
      return 1;
      }

void LinearFiber (float f[], float H[], int n, float lambdac, float L,
float D,
      float S, float alpha, float ng)
      {
      int i,ii;
      float *w;
      double M_2PI,arg,a,fc,beta1,beta2;
      double beta3;

      if (L > 0.0) {
            float aux= 0.0;
            fc=c0/lambdac;
            a=pow(10,-(alpha*L)/20.0);                // = sqrt(10^(-
alpha*L)/10);  atenuacao em Campo Eletrico
            w = vector (n);
            M_2PI=2*M_PI;
            if (f[1] <= 0.0)  // Banda base
                  for (i=1;i<=n;i++)
                        w[i] = M_2PI*f[i];
            else                    // Com portadora
                  for (i=1;i<=n;i++)
                        w[i] = M_2PI*(f[i]-fc);
                  beta2= -0.5*D*sqr(lambdac)/(M_2PI*c0);
                  beta3=
lambdac*sqr(lambdac/(M_2PI*c0))*(lambdac*S+2.0*D)/6.0;
                  for (i=1;i<=n;i++)
                        {
                        ii = 2*i;
                        arg= -w[i]*(w[i]*(beta2 + w[i]*beta3))*L;
                        H[ii-1] = (float) (a*cos(arg));           // Real
part
                        H[ii] = (float) (a*sin(arg));         // Imaginary
part
                        }
            }
      else
            for (i = 1;i<= n;i++) {
```

```
                H[2*i-1] = 1.0;
                H[2*i] = 0.0;
                }
    }
```

A.1.2 Modelo de Classe versão Unix

```
class MODELO: public MODULO_BASE, public ERFC
{
public:
INFOPARLOC vtploc[x];
MODELO_PLOC stploc;
//---------------------------------------------------------------------
MODELO ();
~MODELO ();
//---------------------------------------------------------------------
// Funções padradonizadas
bool ExecutarFuncao(int,void *),
     FunPosProc(int);
//---------------------------------------------------------------------
//Funções dedicadas
};
```

A.1.3 Modelo para confecção de modelos de biblioteca versão Unix

```
#define  LongIdtLst   "Modelo"
#define  ShortIdtLst  "mod1"
#define  ClassIdtLst  "Modelo"
//---------------------------------------------------------------------
#include <IncDefMod.h>
//---------------------------------------------------------------------
#include "Modelo.cla"
#include " ModeloInc.h"
//---------------------------------------------------------------------
void CbModelo(void *v);
//#####################################################################
FIBRAMON1:: Modelo()
{
// parametros locais e globais
//---------------------------------------------------------------------
// informacoes modulares
//---------------------------------------------------------------------
// atribuicao de vetores de variaveis
}
//=====================================================================
Modelo::~Modelo ()
{
}
//=====================================================================
bool Modelo::InicializarMod()
{
```

```
// inicialização das variáveis paramétricas
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool Modelo::IniciarVetParLoc(void *vd,Modelo_PLOC *pl)
{
// atribuição de variáveis paramétricas ao vetor de acesso global
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool Modelo::ExecutarFuncao(int idseq,void *ptv)
{
// execução da função principal do modelo
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool Modelo::FunPosProc(int opc)
{
// ativação das funções eleitivas no pós-processamento
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
void CbModelo(void *v)
{
// função de callback para a caixa de diálogo contendo os atributos
// representativos dos parâmetros do modelo
}
```

A.1.4 Classe Fibra Monomodo versão Unix

```
 struct defFBM1PLOC
       {
       int lbd;
       float lambda,L,D,S,alpha,loss;
       };
typedef struct defFBM1PLOC FBM1PLOC;
//===========================================================================
class FIBRAMON1: public MODULO_BASE, public ERFC
{
public:
INFOPARLOC vtploc[10];
FBM1PLOC stploc;
//---------------------------------------------------------------------------
FIBRAMON1();
~FIBRAMON1();
bool ExecutarFuncao(int,void *),
     FunPosProc(int),
     OpticalPower(),
     Chirp(),
     StationaryNoiseDens(),
```

```
     TransferFunction(),
     Results(),
     InicializarMod(),
     IniciarVetParLoc(void *,FBM1PLOC *);

void LinearFiber(float [],float [],int,float,float,float,float,float);
};
```

A.1.5 Fibra Monomodo versão Unix

```
#define  LongIdtLst   "FbMon1"
#define  ShortIdtLst  "fbm1"
#define  ClassIdtLst  "FIBRAMON"
//---------------------------------------------------------------------
#include <IncDefMod.h>
//---------------------------------------------------------------------
#include "FibraMon1.cla"
#include "FibraMon1Inc.h"
//---------------------------------------------------------------------
void CbFibraMon1(void *v);
//#####################################################################
FIBRAMON1::FIBRAMON1()
{
// parametros locais e globais
IniciarVetParLoc((void *)&vtploc,&stploc);
IniciarVetParGlo((void *)&vtpglo,&stpglo);
//---------------------------------------------------------------------
// informacoes modulares
infomod.qcon     = &fbm1qtcon;
infomod.mploc    = fbm1vtmploc;
infomod.mpproc   = fbm1mpproc;
infomod.mdapres  = &fbm1mdapr;
infomod.funcbmpp = CbFibraMon1;
//---------------------------------------------------------------------
// atribuicao de vetores de variaveis
ploc = vtploc;
pglo = vtpglo;
strcpy(idclasse,ClassIdtLst);
}
//=====================================================================
FIBRAMON1::~FIBRAMON1()
{
}
//=====================================================================
bool FIBRAMON1::InicializarMod()
{
stploc.lambda = stploc.lambda * 1e-7;
stploc.L      = stploc.L      * 1e5;
stploc.D      = stploc.D      * 1.0e-10;
stploc.alpha  = stploc.alpha  * 1e-5;
stploc.S      = stploc.S      * 1.0e-3;
//---------------------------------------------------------------------
return true;
}
//=====================================================================
bool FIBRAMON1::IniciarVetParLoc(void *vd,FBM1PLOC *pl)
```

```
{
int i = 0;
INFOPARLOC *v = (INFOPARLOC *)vd;
//---------------------------------------------------------------------------
v[i++].s = (char *)&pl->lbd;
v[i++].s = (char *)&pl->lambda;
v[i++].s = (char *)&pl->L;
v[i++].s = (char *)&pl->D;
v[i++].s = (char *)&pl->S;
v[i++].s = (char *)&pl->alpha;
v[i++].s = (char *)&pl->loss;
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::ExecutarFuncao(int idseq,void *ptv)
{
float *y,*f,*H,*Se;
int n,i,tregn,treg2n;
//---------------------------------------------------------------------------
if(!portain[0].TestarExistArq()) return false;
portain[0].Abrir();

if(!portain[0].Ident("main1x"))  return false;
f = (float *)portain[0].RecuperarCol();
tregn = portain[0].ObterTamReg();
n = (tregn / sizeof(float)) - 1;

if(!portain[0].Ident("main1y"))   return false;
y = (float *)portain[0].RecuperarCol();
treg2n = portain[0].ObterTamReg();
ApresDado(LongIdtLst,"-0",n,2,f,y);
//---------------------------------------------------------------------------
H = vetor(2*n);
//---------------------------------------------------------------------------
LinearFiber
(f,H,n,stploc.lambda,stploc.L,stploc.D,stploc.S,stploc.alpha);
portaout[0].Abrir();
portaout[0].Ident("col1x",tregn,COLECAO);  portaout[0].Salvar((void *)f);
portaout[0].Ident("col1y",treg2n,COLECAO); portaout[0].Salvar((void *)H);
ApresDado(LongIdtLst,"-1",n,2,f,H);
//---------------------------------------------------------------------------
if(stploc.L > 0) CMultiply (y,H,1,n);
portaout[0].Ident("main1x",tregn,COLECAO);  portaout[0].Salvar((void
*)f);
portaout[0].Ident("main1y",treg2n,COLECAO); portaout[0].Salvar((void
*)y);

//---------------------------------------------------------------------------
// Sinal + ruido do Campo Optico em frequencia
delete f; delete y;
if(!portain[0].Ident("colnx")) return false; f = (float
*)portain[0].RecuperarCol();
if(!portain[0].Ident("colny")) return false; y = (float
*)portain[0].RecuperarCol();
ApresDado(LongIdtLst,"-3.5",n,2,f,y);
if(stploc.L > 0) CMultiply (y,H,1,n);
portaout[0].Ident("colnx",tregn,COLECAO);  portaout[0].Salvar((void *)f);
portaout[0].Ident("colny",treg2n,COLECAO); portaout[0].Salvar((void *)y);
ApresDado(LongIdtLst,"-3",n,2,f,y);
```

```
//---------------------------------------------------------------------------
// Dens. espectral do ruido estacionario
delete f;
if(!portain[0].Ident("col-x")) return false; f  = (float
*)portain[0].RecuperarCol();
if(!portain[0].Ident("col-y")) return false; Se = (float
*)portain[0].RecuperarCol();
ApresDado(LongIdtLst,"-4",n,1,f,Se);
if(stploc.L > 0)
  {
  for(i=1; i<=n; i++) Se[i]=Se[i]*modulo2(H[2*i-1], H[2*i]);
  }
portaout[0].Ident("col-x",tregn,COLECAO); portaout[0].Salvar((void *)f);
portaout[0].Ident("col-y",tregn,COLECAO); portaout[0].Salvar((void *)Se);
ApresDado(LongIdtLst,"-5",n,1,f,Se);
//---------------------------------------------------------------------------
portain[0].Fechar(); portaout[0].Fechar();
//---------------------------------------------------------------------------
free(H); delete Se; delete y; delete f;
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::FunPosProc(int opc)
{
switch(opc)
      {
      case 0 : OpticalPower();         break;
      case 1 : Chirp();                break;
      case 2 : StationaryNoiseDens();  break;
      case 3 : TransferFunction();     break;
      case 4 : Results();     break;
      }
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::OpticalPower()
{

//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::Chirp()
{

//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::StationaryNoiseDens()
{

//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::TransferFunction()
{
```

```
//---------------------------------------------------------------------------
return true;
}
//===========================================================================
bool FIBRAMON1::Results()
{

//---------------------------------------------------------------------------
return true;
}
//===========================================================================
void CbFibraMon1(void *v)
{
VALOR val;
INFOPARLOC info[10];
FIBRAMON1 fbm1;
FBM1PLOC stploc;
OBJVIS *ob1;
//---------------------------------------------------------------------------
CALLBACK *cb = (CALLBACK *)v;
PAINEL *painel = (PAINEL *)cb->obj;
//---------------------------------------------------------------------------
painel->ColetarValUsr(&val);
fbm1.IniciarVetParLoc((void *)&info,&stploc);
InicializarPar(info,&val);
//---------------------------------------------------------------------------
ob1 = (OBJVIS *)painel->atribox.atributo.vtinfo[1].info;
switch(stploc.lbd)
      {
      case 0: ob1->Sensibilidade(false);
               InicializarVal(&val,fbm1p1310);
               break;

      case 1: ob1->Sensibilidade(false);
               InicializarVal(&val,fbm1p1550);
               break;

      case 2: ob1->Sensibilidade(true);
               break;
      }
painel->AtualizarOpcVal(&val);
}
//===========================================================================
void FIBRAMON1::LinearFiber(float f[],float H[],int n,float lambdac,float
L,float D,float S,float alpha)
{
int i,ii;
float *w,aux= 0.0;
double M_2PI,arg,a,fc,beta1,beta2;
double beta3;
//---------------------------------------------------------------------------
if(stploc.L > 0.0)
  {
  fc=c0/lambdac;
  a=pow(10,-(stploc.alpha*stploc.L)/20.0);
  w = vetor(n);
  M_2PI=2*M_PI;
  if(f[1] <= 0.0){ for(i=1;i<=n;i++) w[i] = M_2PI*f[i]; }
    else{ for (i=1;i<=n;i++) w[i] = M_2PI*(f[i]-fc); }
```

```
  beta2= -0.5*D*sqr(lambdac)/(M_2PI*c0);
  beta3= lambdac*sqr(lambdac/(M_2PI*c0))*(lambdac*S+2.0*D)/6.0;
  for(i=1;i<=n;i++)
     {
     ii = 2*i;
     arg= -w[i]*(w[i]*(beta2 + w[i]*beta3))*L;
     H[ii-1] = (float) (a*cos(arg));
     H[ii] = (float) (a*sin(arg));
     }
  free(w);
  }
  else for(i = 1;i<= n;i++){ H[2*i-1] = 1.0; H[2*i] = 0.0; }
}
```

A.1.6 Arquivo de definições padrão para Fibra Monomodo versão Unix

```
//######################### definicoes ###################################
#define ID_FIBERPOWER    0
#define ID_CHIRP         1
#define ID_STATNOISEDENS 2
#define ID_TRANSFUNC     3
#define ID_RESULTS       4
//############ quantidade de conectores de dados ########################

QTDCON  fbm1qtcon = {1,1,5};

//###################### parametros locais ###############################
char
*fbm1p1310[] = {"0","1310","10","17","0.05","0.25","0"},
*fbm1p1550[] = {"1","1550","10","17","0.05","0.25","0"};

//#################### menu da janela de parametros ######################
STAPRATRI fbmb1rb1[10] = {
{OCNewLine,"F"},
{OVTitulo,"Wavelenght, lbda:"},
{OCCallBack1,"",NULL},
{OCHorizontal},
{OVDefault,"0",NULL},
{OVLabel,"1310"},
{OVLabel,"1550"},
{OVLabel,"Other: "},
{OCEnd}};
//----------------------------------------------------------------------
STAPRATRI fbm1tf01[8] = {
{OCNewLine,"T"},
{OVTitulo,"  "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[1]},
{VRTPFT},
```

```
{OCEnd}};

STAPRATRI fbm1lbsep[3] = {
{OCNewLine,"T"},
{OVTitulo," "},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf1[8] = {
{OCNewLine,"T"},
{OVTitulo,"Fiber length, L ( Km ):                    "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[2]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf2[8] = {
{OCNewLine,"T"},
{OVTitulo,"Dispersion coef., D ( ps / nm . Km ):      "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[3]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf4[8] = {
{OCNewLine,"T"},
{OVTitulo,"Diferential disp. coef., S ( ps / nm^2.Km ):"},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[4]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf5[8] = {
{OCNewLine,"T"},
{OVTitulo,"Atennuation coef., alpha ( dB / Km ):      "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[5]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRATRI fbm1tf6[8] = {
{OCNewLine,"T"},
{OVTitulo,"Coupling loss, loss ( dB ):                "},
{OCCallBack1,"",NULL},
{OVTxtVis,"10"},
{OVTxtLen,"10"},
{OVDefault,fbm1p1310[6]},
{VRTPFT},
{OCEnd}};
//-------------------------------------------------------------------------
STAPRMENU fbm1mploc1[15] = {
{OVRadioBox,fbmb1rb1},
{OVTextField,fbm1tf01},
```

```
{OVLabel,fbm1lbsep},
{OVTextField,fbm1tf1},
{OVTextField,fbm1tf2},
{OVTextField,fbm1tf4},
{OVTextField,fbm1tf5},
{OVTextField,fbm1tf6},
{OCEnd}};

STAPRMENU *fbm1vtmploc[3] = {fbm1mploc1,NULL};
//################# menu da janela de pos-processamento #################
STAPRATRI fbm1pp1[4] = {
{OCNewLine,"T"},
{OVTitulo,"Optical Power, Pf(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp2[4] = {
{OCNewLine,"T"},
{OVTitulo,"Chirp, dv(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp3[4] = {
{OCNewLine,"T"},
{OVTitulo,"Stationary Noise Dens, Srp(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp4[4] = {
{OCNewLine,"T"},
{OVTitulo,"Transfer Function, Hf(t)"},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRATRI fbm1pp5[4] = {
{OCNewLine,"T"},
{OVTitulo,"Results ..."},
{OCCallBack1,"",NULL},
{OCEnd}};
//-----------------------------------------------------------------------
STAPRMENU fbm1mpproc[5] = {
{OVPushButton,fbm1pp1},
{OVPushButton,fbm1pp2},
{OVPushButton,fbm1pp3},
{OVPushButton,fbm1pp4},
{OCEnd}};
//################## modo de apresentacao de resultados #################
DEFARQ
fbm1mdapr1[2] = {{MDGRAF,_BIN}},
fbm1mdapr2[10] = {{MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                  {MDGRAF,_SDF},
                 };

MDAPRES fbm1mdapr = {fbm1mdapr1,fbm1mdapr2};
```

A.1.7 Classe Modulo Base versão Unix

```
#include <ParGlo.h>
//---------------------------------------------------------------------------
class MODULO_BASE: public MATHLIB
{
public:
char modnome[20],idclasse[20];

STPGLO      stpglo;
INFOPARLOC  vtpglo[4],*ploc,*pglo;
PORTA      *portain,*portaout,*portapproc;
INFOMOD     infomod;
VARPUB     *vp;
IDARQ       idarq;
//---------------------------------------------------------------------------
void IniciarVetParGlo(void *,STPGLO *),
     FunAjustParGlo(),
     FunPosProcPort(int),
     ApresDado(char *,char *,int,int,float [],float []);

virtual bool ExecutarFuncao(int,void *){ }
virtual bool FunPosProc(int){ }
virtual bool InicializarMod(){ }
virtual bool AtivarMetPub(){ }
};
```

A.1.8 Classe Porta versão Unix

```
class PORTA
{
ARQUIVO *arquivo;
int acesso,       // WR, RD
    tipo;         //_BIN,_TXT,_SDF

public:
int idpos,        // ident. do bloco possuidor da porta
    idmodante,    // ident. do modulo onde parte a conexao (out)
    idmodprox;    // ident. do modulo onde chega a conexao (in)
IDARQ idarq;

PORTA();
~PORTA();
void Fechar();
bool Abrir(),
     IdPorta(int,IDARQ *),
     IdPorta(int,IDARQ *,char *),
     Salvar(void *),
     Ident(char *),
     Ident(char *,int,int),
     TestarExistArq(),
     FinalisarCol();
```

```
int  ObterTamReg(),
     ObterIdpos();

char *RecuperarCol();
};
```