ecai

Universidade Federal de Santa Catarina
Centro Tecnológico
Curso de Engenharia de Controle e
Automação Industrial

ufsc

ControlLab

Um Laboratório Virtual para Sistemas Monovariáveis de Controle

Monografia submetida à Universidade Federal de Santa Catarina

como requisito para a aprovação da disciplina:

EEL 5901: Projeto de Fim de Curso

André Camargo Guedes Rodrigues

Florianópolis, Janeiro de 1998

ControlLab
Um Laboratório Virtual para Sistemas Monovariáveis de Controle

André Camargo Guedes Rodrigues

Esta monografia foi julgada no contexto da disciplina
EEL 5901: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação Industrial

BANCA EXAMINADORA:

Prof. Luiz Fernando Bier Melgarejo
Orientador do Curso

Márcio Quintaes Marchini
Orientador da Empresa

Prof. Augusto Humberto Bruciapaglia
Responsável pela disciplina e Coordenador do Curso

Prof. Eugênio de Bona Castelan Neto
Avaliador

Alessandra d'Aquino
Debatedora

Max Hering de Queiroz
Debatedor

AGRADECIMENTOS

Meus agradecimentos a todos aqueles que me apoiaram na realização deste projeto e do curso de Engenharia de Controle e Automação e Industrial. Em especial a meus pais, Pedro Guedes Rodrigues e Maria Cerly Camargo Rodrigues, que com muito amor e dedicação sempre estiveram ao meu lado, assim como a meus familiares e demais amigos.

Dentre tais amigos não posso deixar de mencionar aqueles que tornaram possível e inesquecível meu período de permanência no Canadá: Luiz Fernando Bier Melgarejo, Márcio Quintaes Marchini, Claudiane Grando, Eduardo José Pereira, Hélio Souza Araújo, Léa Pereira, Silênio Sulivan Quarti e demais pessoas que tive o prazer de conhecer no Laboratório de Software Educacional e na OTI.

Também não posso deixar de mencionar os amigos do curso com os quais compartilhei cinco maravilhosos anos de minha vida – "Amigo é coisa para se guardar...".

Obrigado a todos!

RESUMO

O projeto consistiu no desenvolvimento de um laboratório virtual para sistemas monovariáveis de controle (ControlLab), com ênfase em atividades de pesquisa e ensino. Esta tarefa contou com o apoio da Engenharia de Software, principalmente de técnicas de projeto, tais como padrões de projeto e frameworks.

O ControlLab apresenta um ambiente de modelagem baseado em manipulação direta, através de representações gráficas para os sistemas e de ferramentas e comandos que permitem a sua edição de uma forma amigável.

Os sistemas representados neste ambiente, denominados diagramas, são baseados na semântica de blocos, que representa a forma mais intuitiva para a sua definição, contando com o auxílio de uma ampla biblioteca de componentes que pode ser estendida pelo usuário.

Os diagramas, que apresentam períodos discretos de execução, podem comunicar-se através da Internet permitindo a definição de sistemas monovariáveis distribuídos.

Um diagrama pode ser composto não somente por blocos simples, mas por outros diagramas aninhados, denominados subdiagramas, possibilitando uma melhor organização interna. Sua estrutura pode ser exportada na forma de código para ser utilizada de forma independente ao ControlLab.

ABSTRACT

The project consisted in developing a virtual laboratory for monovariable control systems (ControlLab) with emphasis in research and teaching activities. This task was supported by Software Engineering, mainly by its design techniques, such as design patterns and frameworks.

ControlLab presents a modeling environment based on direct manipulation, through graphic representations for the systems and tools and commands that allow its edition in a friendly way.

The systems represented in this environment, called diagrams, are based in the semantics of blocks. It represents the most intuitive form to define the systems through a wide library of components that can be extended by the ControlLab user.

The diagrams, that present discreet execution periods, can communicate through Internet allowing the definition of distributed monovariable systems.

A diagram can be composed not only for simple blocks, but for other nested diagrams, called subdiagrams, providing a better internal organization. The diagram structure can be exported as code to be used outside ControlLab.

SUMÁRIO

RELAÇÃO DE FIGURAS

1. INTRODUÇÃO

Tradicionalmente, o desenvolvimento de softwares relacionados com a área de pesquisa é realizado através de métodos tradicionais de Engenharia de Software, evoluindo com ênfase nos objetivos a serem atingidos. Esta concepção dificulta a integração de diferentes projetos, assim como a reutilização futura de seus elementos.

[DoD, 1995], em sua discussão sobre engenharia do domínio e da aplicação, exalta a necessidade da análise prévia do domínio. Os projetos pertencentes a determinado domínio necessitariam apenas estender e adaptar-se ao modelo e à arquitetura genérica resultantes desta análise.

Quando este domínio é o Controle de Processos, a situação não é diferente. Em geral, o desenvolvimento de projetos não demonstra preocupação com a integração e reusabilidade de seus elementos em outros projetos da área, resultando na necessidade de, para cada projeto, a criação de um suporte de controle específico para o atendimento dos seus objetivos.

Muitas vezes, softwares comerciais são utilizados como base para este suporte de controle. No entanto, tais softwares, devido a sua natureza comercial, não dispõem de mecanismos totalmente abertos ao usuário, o que representa um fator limitante ao desenvolvimento de uma pesquisa. Além disso, deve-se considerar a possibilidade de que tais softwares não atendam adequadamente os objetivos envolvidos.

Para preencher parte deste espaço, [Quarti, 1997] lançou um *framework*[1] para o desenvolvimento de aplicações relacionadas ao domínio de sistemas monovariáveis.

Seguindo a linha criada por [Quarti, 1997] este projeto objetiva implementar um software de modelagem e simulação de sistemas monovariáveis de controle voltado para a respectiva pesquisa e, consequentemente, ensino.

---

[1] [Johnson et. al., 1990], [Johnson et. al., 1991], [Gamma et. al., 1995]

Para o alcance deste objetivo, o software a ser desenvolvido, um *Laboratório Virtual para Sistemas Monovariáveis de Controle de Processos*, deve apresentar as seguintes características:

- Apresentar um suporte aberto e extensível para a modelagem e simulação de sistemas monovariáveis de controle.
- Apresentar um ambiente de fácil utilização, que possibilite a rápida prototipação de sistemas.
- Ser modular, tornando possível a utilização dos sistemas em outros contextos que não o ambiente de modelagem, como, por exemplo, a aplicação em sistemas embutidos.

Além da sua aplicação na área de controle monovariável de processos, este trabalho também pode ser utilizado no ensino e pesquisa de técnicas relacionadas com a Engenharia de Software.

Na tarefa de descrever este projeto e o seu resultado, o ControlLab, esta monografia apresenta a seguinte distribuição:

- Aplicação da Engenharia de Software – Técnicas de Engenharia de Software envolvidas no desenvolvimento do projeto.
- Framework JHotDraw – Ponto de partida para o desenvolvimento do ambiente de prototipação.
- Framework de Controle de Processos – Suporte de controle do ControlLab.
- Elementos Externos – Características e conceitos do ControlLab que são visíveis ao usuário. Indicado para o leitor que deseje conhecer suas potencialidades.
- Visão Interna – Componentes de projeto e implementação do ControlLab.

2. APLICAÇÃO DA ENGENHARIA DE SOFTWARE

Nesta etapa será analisado o suporte oferecido pela Engenharia de Software utilizado neste trabalho.

Em um primeiro momento, são analisados os fatores desta engenharia relacionados com a execução deste projeto. Em seguida são analisados os elementos envolvidos nas suas fases de análise, projeto e implementação.

2.1 ENGENHARIA DE SOFTWARE

Surgiu em resposta à crise do software que ocorreu no início dos anos 80 [Pressman, 1987], correspondendo à aplicação de técnicas de engenharia no desenvolvimento de softwares, o qual pode ser subdividido em três fases:

- Fase de Definição (Análise) – Corresponde à definição de um conjunto de requisitos que devem ser atendidos pelo software, além de um planejamento para o seu desenvolvimento.
- Fase de Desenvolvimento – Criação de um elemento operacional, o software, baseado nos seus requisitos e no planejamento predefinidos. É subdividido em etapas de projeto, implementação e testes.
- Fase de Manutenção – Inclui as etapas posteriores ao término do desenvolvimento do software, incluindo sua adequação a eventuais alterações de requisitos e a correção de problemas que possam ter escapado da fase de testes.

Estes passos podem ser realizados através da utilização de diferentes paradigmas para o processo de criação de um software. Neste projeto foi utilizado o modelo incremental [Jacobson, 1992], no qual o software evolui ciclicamente, onde cada estágio corresponde à adição de novas funcionalidades predefinidas nos seus requisitos. Deste modo, é possível compreender melhor tais requisitos, realizando, inclusive, eventuais alterações.

A realização deste projeto não requer uma ampla aplicação de técnicas de Engenharia de Software, que normalmente são utilizadas em softwares onde o desenvolvimento envolve um grupo de pessoas e a coordenação de atividades é necessária [Parnas, 1984]. No entanto, o aprimoramento da qualidade não deixa de se constituir em uma necessidade, sendo interessante, portanto, verificar como determinados aspectos de qualidade influenciaram no seu desenvolvimento.

2.1.1 Qualidade de Software[1]

Atualmente, a maior dificuldade no seu aprimoramento está na forma como os desenvolvedores aplicam a Engenharia de Software, que tem a qualidade como o seu principal objetivo. Apesar da existência de uma boa quantidade de fontes relacionadas com técnicas de Engenharia de Software, a efetiva utilização de tais técnicas vem se restringindo a algumas instituições de pesquisa e empresas da área de software.

Além disso, as instituições de ensino não conseguem criar um vínculo adequado entre estas técnicas e o respectivo desenvolvimento de software, o que deveria ser natural. Segundo [Lamb, 1988], isto ocorre devido à sobrecarga de tarefas a que os alunos são expostos, o que dificulta o aprendizado da Engenharia de Software. Acaba-se criando um vácuo entre as técnicas e os softwares nos quais elas deveriam ser aplicadas, fortalecendo-se o conceito de que estes últimos são produtos exclusivos das linguagens de programação.

A norma [IEC9126] indica as características que um software deve cumprir para que seja considerado um software de qualidade. São definidas seis características básicas (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade) que são subdivididas em vinte e uma subcaracterísticas.

Esta classificação, apesar de representar o padrão, não é aqui abordada. Ao invés de discutir cada uma das vinte e uma subcaracterísticas de um software de qualidade, são analisados alguns atributos relacionados que influenciaram o

---

[1] Referência:
Society for Software Quality – http://www.ssq.org/
Software Quality Institute – http://www.utexas.edu/coe/sqi/

desenvolvimento do ControlLab. Os mecanismos utilizados para melhorar estes atributos são detalhados no decorrer desta monografia.

2.1.1.1 CORRETUDE

Fator essencial à qualidade de um software, a corretude define a capacidade com que o software atende os requisitos do usuário. Normalmente tais requisitos não resultam de um único usuário, mas de um domínio de usuários, aumentando a área de atuação do software.

No caso do ControlLab, este domínio é composto pela comunidade de ensino e pesquisa da área de controle de processos monovariáveis, o que foi um fator determinante na definição, além de outros requisitos, de seus objetivos básicos indicados na introdução deste trabalho.

2.1.1.2 AMIGABILIDADE

Define a facilidade com que o software é aprendido e utilizado por seus usuários. Está relacionado diretamente com a interface com o usuário, além de depender dos tempos de execução do software, como no caso do seu tempo de abertura.

Recentemente, com a introdução de soluções gráficas e baseadas em janelas, é possível o desenvolvimento de softwares com melhores resultados em termos de amigabilidade.

A utilização do framework JHotDraw permitiu, ao ControlLab, oferecer a seus usuários uma interface intuitiva e de fácil utilização.

2.1.1.3 ROBUSTEZ

Representa a capacidade de um software de funcionar em condições anormais de execução, que não foram definidas nos requisitos, devendo garantir que tais condições não levem à execução de operações indesejadas. A robustez envolve, por exemplo, a garantia de que o sistema de piloto automático de um avião não irá derrubá-lo em caso de falhas.

A linguagem Java™ permitiu, através do mecanismo de tratamento de exceções [Dony, 1990], que o ControlLab responda adequadamente a situações adversas às definidas em seu conjunto de requisitos.

2.1.1.4 Documentação

Descreve o software, sendo composta de duas partes: documentação externa e interna. A documentação externa é responsável por apresentar as potencialidades do software ao usuário, enquanto a documentação interna é responsável por descrever sua estrutura formalmente.

A documentação externa é composta por manuais, tutoriais, demonstrações e arquivos de ajuda oferecidos ao usuário, afetando diretamente o modo como o software será utilizado pelo mesmo. Além disso, serve como mecanismo de divulgação do produto e permite que se minimizem as linhas de comunicação entre usuário e desenvolvedor, evitando que este último seja sobrecarregado com atividades de auxílio ao usuário.

A documentação interna serve para descrever o projeto do software segundo uma visão de desenvolvimento. Os sistemas não podem ser devidamente compreendidos isoladamente, mas necessitam de um complemento adequado. É comum confundir os comentários aplicados diretamente ao código com documentação interna, mas estas explicações servem apenas para descrever soluções locais, não tratando o software em um nível de abstração adequado. A documentação interna de um software é composta, ainda, pelas descrições das metodologias e ferramentas utilizadas no seu desenvolvimento, por um histórico do mesmo (no caso de softwares mais complexos) e por um conjunto de diagramas que realize a sua formalização em diferentes níveis de abstração.

Uma documentação interna bem implementada diminui a dependência do software para com o seu programador, aumentando a flexibilidade no seu desenvolvimento, que pode ser facilmente delegado a outras pessoas. Além disso, a conseqüente formalização do software produz automaticamente canais de comunicação com metodologias de Engenharia de Software, além de permitir a

compreensão do sistema de uma maneira sistemática, facilitando a sua avaliação e interação com outras soluções.

O documentação do ControlLab é formada pelos documentos relacionados com as técnicas de projeto adotadas, pela documentação do código resultante da implementação em Java™ e por um conjunto de documentos dirigidos ao usuário, incluindo esta monografia.

2.1.1.5 Reusabilidade

Representa a capacidade de software ser reutilizado, totalmente ou em partes, para o desenvolvimento de novas soluções.

O grau de reusabilidade de um software depende principalmente do seu grau de modularidade, ou seja, pelo modo como pode ser dividido em subsistemas, os módulos, que sejam os mais autônomos e funcionalmente bem definidos possíveis.

Um software com maior grau de reusabilidade automaticamente torna-se mais simples e extensível. No entanto, a utilização inadequada da reusabilidade pode gerar interdependências indesejáveis entre os subsistemas, dificultando eventuais alterações no projeto.

A reusabilidade permite que, na busca de novas soluções, os esforços de desenvolvimento sejam aplicados com maior objetividade, explorando-se apenas as partes do problema que nunca foram tratadas anteriormente. Além disso, os componentes reutilizados trazem consigo todo um histórico de aplicação e promovem uma otimização em todo o processo de desenvolvimento.

Atualmente, a reusabilidade não é uma característica tão difundida como deveria. Culturalmente, a programação não é realizada levando-se em consideração utilizações alternativas ao problema original, o que pode ser minimizado, através da utilização de técnicas que promovam a reusabilidade.

A reutilização marcou este trabalho através dos dois frameworks envolvidos, que simplificaram em muito o desenvolvimento do ControlLab.

2.1.1.6 Extensibilidade

Representa o quanto um software está preparado para modificações futuras, baseadas em eventuais alterações de suas especificações. Normalmente, o conjunto de requisitos de um software apresenta uma evolução contínua de acordo com o seu estado atual e com a possibilidade da adição de novas funcionalidades ou melhoria das já existentes, o que demanda da realização de um projeto adequadamente preditivo e extensível.

O software é um elemento naturalmente flexível, através de seu código. No entanto, este tipo de flexibilidade, que é muito utilizado como mecanismo de extensão, é extremamente restrito e pode acarretar uma série de efeitos colaterais à estrutura interna do software, deteriorando a sua qualidade. Esta saída talvez demonstre resultados satisfatórios em pequenos softwares, mas para a obtenção de sistemas adequadamente extensíveis é imprescindível a adoção de técnicas que aprimorem tal atributo.

A utilização da orientação a objetos e de padrões de projeto (*design patterns* – [Gamma et. al., 1995], [Pree, 1994]) permitiu o aprimoramento da extensibilidade do ControlLab.

2.1.1.7 Simplicidade

Representa o quanto simples é a estrutura interna de um software, em função dos seus objetivos e do hardware a que se destina. É essencial, por exemplo, em softwares voltados para a aplicação em sistemas embutidos.

Um software demasiado requintado irá consumir desnecessariamente os recursos do sistema, tais como espaço de armazenamento e de processamento. Além disso, a não existência de um compromisso com a simplicidade tende a resultar em soluções mais complexas do que o necessário (*over-design*).

Por outro lado, sistemas com excessiva simplicidade também falham na solução dos problemas, por descartar mecanismos que, embora sejam mais complexos, sejam os mais adequados.

Este atributo foi auxiliado, neste projeto, pela utilização da linguagem Java™ e das técnicas de projeto empregadas, em especial a de modelagem UML [Booch et. al., 1995].

2.1.1.8 Compatibilidade e Portabilidade

A compatibilidade representa a "facilidade de combinar um sistema com outros" [Meyer, 1986]. Quando um destes sistemas é o sistema operacional, denomina-se de portabilidade. Pode ser alcançada através da padronização ou da abstração.

Com a padronização, através da utilização de mecanismos comuns de representação, é possível otimizar o grau de compatibilidade entre softwares. Um exemplo é o padrão *Initial Graphics Exchange Specification* (IGES), utilizado para a conversão entre diferentes formatos de arquivos de CAD.

As linguagens de programação utilizam, muitas vezes, a abstração de suas estruturas em um nível acima do nível não padronizado para evitar que o código dos softwares deva ser replicados para diferentes casos de aplicação.

O ControlLab utilizou o Java™ como mecanismo de portabilidade, e o modelo JavaBeans™ ([DeSoto, 1997], [McIntyre, 1997]) como mecanismo de compatibilidade com outros softwares implementados em Java™.

2.1.1.9 Eficiência

Representa o grau de otimização com o qual são utilizados os recursos do sistema. Este fator sempre recebeu atenção especial por parte dos programadores, de modo que está culturalmente embutido no processo de desenvolvimento de software.

Atualmente, em geral, a eficiência deixou de ser um fator tão crítico. Os primeiros computadores eram demasiado lentos na realização de processamentos e operações em geral, mas o desenvolvimento de computadores mais velozes permitiu que fossem transferidos esforços empregados na otimização da eficiência para a melhor estruturação do software. Deste modo, a redução de custos e melhoria de

qualidade passaram a ser os requisitos mais importantes no desenvolvimento de softwares [Pressman, 1987].

A linguagem Java™ fornece uma boa eficiência para o ControlLab. Além disso, recentes pesquisas têm procurado tornar a linguagem mais eficiente, visando permitir, inclusive, a sua utilização em sistemas embutidos.

2.2 Análise

O principal objetivo da análise de um software é a concepção de seus requisitos e de um planejamento para o seu desenvolvimento. O conjunto de requisitos do ControlLab pode ser sintetizado através dos objetivos descritos na introdução desta monografia, enquanto que o planejamento para o seu desenvolvimento pode ser sintetizado por este capítulo da monografia.

Estes elementos foram amadurecidos através de reuniões realizadas no início do desenvolvimento do projeto. Não foi necessária a utilização de técnicas especiais de análise dado que as pessoas participantes apresentavam conhecimento relacionado tanto com o domínio de aplicação do projeto quanto com as metodologias utilizadas na sua elaboração.

2.3 Projeto Orientado a Objetos

As etapas de projeto do ControlLab foram baseadas na utilização do paradigma da orientação a objetos. Portanto, antes de entrar em detalhes sobre as técnicas de projeto utilizadas neste trabalho, é interessante colocar alguns conceitos que são simples, mas que caracterizam a orientação a objetos como um importante paradigma na Engenharia de Software.

2.3.1 Orientação a Objetos

Segundo [Puttick et. al., 1994], a orientação a objetos melhora as condições para o aumento da produtividade e da qualidade no desenvolvimento de softwares, ao mesmo tempo que permite uma maior aproximação dos sistemas computacionais com o mundo real. A tecnologia de objetos pode ser considerada como um passo em

direção à industrialização do software, que pode ajudar a transformar a programação de uma arte misteriosa em um processo sistemático.

A orientação a objetos é baseada em um conjunto de conceitos que a diferenciam de outras tecnologias aplicáveis ao desenvolvimento de softwares[2].

As definições destes conceitos, apresentadas a seguir, são baseadas nas publicações [Howard, 1988], [Puttick et. al., 1994], [Snyder, 1986] e [Foote et. al., 1991].

2.3.1.1 Objeto

Conceito central do paradigma. Representa uma entidade formada por um conjunto de dados interrelacionados, que formam o seu estado, e por um conjunto de procedimentos, denominados métodos. O modo como os métodos de um objeto determinam o seu estado ou o estado de outros objetos define o seu comportamento.

2.3.1.2 Encapsulamento

Característica através da qual os métodos são o único mecanismo para a alteração do estado de um objeto. Pode ser refinado através da definição de diferentes graus de acessibilidade aos métodos de um objeto.

2.3.1.3 Mensagem

Sinal trocado entre dois objetos requerendo, ao receptor, a execução de um serviço e/ou a alteração do seu estado, através de um de seus métodos.

2.3.1.4 Classe

Representa um modelo para objetos similares, descrevendo como este objetos são estruturados internamente. Objetos da mesma classe, denominados instâncias da classe, têm o mesmo comportamento e as mesmas estruturas de informação que

---

[2] No caso de projeto, podem-se citar, ainda, os projetos orientados a estruturas de dados e a fluxo de dados [Pressman, 1987].

definem seus estados. Uma classe apresenta um conjunto de responsabilidades (o que suas instâncias realizam e seus atributos) e um conjunto de colaborações (com que outras se relaciona para o cumprimento de suas responsabilidades).

2.3.1.5 Herança

Mecanismo de reusabilidade de código que permite a utilização do comportamento de uma classe, denominada superclasse, para a definição de novas classes, denominadas subclasses. Cada subclasse pode estender o comportamento e o estado de sua superclasse, além de poder alterar o comportamento herdado através da sobreposição de métodos. Existem dois tipos de herança: múltipla, quando uma classe pode ter várias superclasses, e simples, quando uma classe pode apresentar apenas uma superclasse.

2.3.1.6 Classe Abstrata

Uma classe que representa um modelo para classes ao invés de objetos, não apresentando instâncias, mas um conjunto de subclasses. É utilizado como mecanismo de projeto para a organização da herança, podendo definir, além dos elementos convencionais de uma classe, um conjunto de métodos abstratos que devem ser implementados por suas subclasses.

2.3.1.7 Interface e Tipo

Uma interface corresponde à especificação de um conjunto de métodos. Objetos que implementam uma mesma interface são considerados como sendo do mesmo tipo.

Segundo este conceito, classes definem tipos, embora ambos apresentem conceitos distintos. Por exemplo, objetos de uma subclasse podem ser consideradas do tipo definido pela respectiva superclasse, apesar de não serem instâncias da mesma. Do mesmo modo, pode-se ter duas classes que não sejam relacionadas por hierarquia, mas que implementem uma interface comum, sendo, portanto, do tipo definido por esta interface.

2.3.1.8 POLIMORFISMO

Característica que indica a habilidade de objetos enviarem mensagens sem conhecerem a classe do objeto receptor, apenas o seu tipo[3].

2.3.2 VANTAGENS

Segundo [Jacobson, 1992], o projeto de softwares pode ser dividido basicamente em métodos orientados a objeto e métodos funcionais. Nestes últimos encontram-se, por exemplo, o SADT (Structured Analysis and Design Technique) [Ross, 1985] e o SA/SD (Structured Analysis and Structured Design) [Constantine et. al., 1979].

Enquanto os métodos funcionais tornam os dados passivos e manipulados por procedimentos ativos, cuja definição é externa aos dados, o método orientado a objetos transforma os objetos em componentes ativos do sistema, responsáveis por manipular seu próprio estado interno [Loomis et. al., 1987].

[Parnas, 1994] complementa, afirmando que um projeto necessita estar preparado para mudanças e que a orientação a objetos, ao promover a modularidade e a independência entre os componentes de um software, reduz o impacto que eventuais mudanças irão ter sobre o mesmo.

Outra característica do projeto orientado a objetos é a sua estruturação em elementos do domínio do problema, que é a maneira mais natural de descrever um sistema. Devido a sua natureza, tais elementos são mais estáveis e tendem a mudar menos.

Além disso, a utilização de metodologias de projeto orientadas a objeto mantém a consistência com a programação orientada a objetos.

---

[3] A linguagem Smalltalk™ [LaLonde et. al., 1990], por exemplo, não requer a identificação de tipos, que estão implícitos na definição dos métodos das classes.

2.3.3 Técnicas de Modelagem

As primeiras iniciativas para a descrição de metodologias de projeto orientado a objetos surgiram no início dos anos 80. Desde então várias técnicas de modelagem foram propostas para auxiliar tais metodologias.

[Ambler, 1997] realiza um apanhado dos principais tipos de técnicas de modelagem orientadas a objetos que podem ser encontradas:

- Modelagem CRC (Class Responsibility Collaborator) – Normalmente empregada aliada ao brainstorming[4]. Utiliza cartões para a definição da estrutura de classes, suas responsabilidades e colaborações.
- Use-Cases – Representam o relacionamento de um sistema com entidades externas, que recebem a denominação de atores. Também define o fluxo de informação envolvido.
- Diagramas Interface-Flow – Representam o comportamento da interface com o usuário, sendo uma alternativa à utilização de protótipos. Definem os diferentes elementos da interface e as ações que o usuário executa para se locomover nos mesmos.
- Diagramas de Classes – Mostram as classes de um sistema e seus inter-relacionamentos.
- Diagramas de Processos – Semelhantes aos diagramas de fluxo de dados, mas utilizados para representar pequenas seções do sistema. Representam processos aliados a fluxos de dados e entidades de armazenamento.
- Diagramas de Dados – Servem como mecanismo de comunicação do projeto orientado a objetos com bancos de dados relacionais, representando-os segundo uma abordagem orientada a objetos.
- Diagramas de Seqüência – Normalmente utilizados em complemento a técnica de Use-Case, representando seu fluxo lógico.

---

[4] Técnica grupal de criatividade.

- Diagramas de Componentes – Representam o sistema em termos de seus subsistemas e seus relacionamentos.
- Diagramas de Desdobramento – Mostram a configuração do hardware durante a execução do sistema. Pode ser utilizado, por exemplo, na representação de um grupo de máquinas servidoras.
- Diagramas de Estados – Representam estados de um objeto, definindo as transações entre os mesmos, além de poder indicar estados de início e fim.
- Diagramas de Colaborações – Mostram o fluxo de mensagens entre objetos e os relacionamentos entre classes.

Tais técnicas são especialmente indicadas no desenvolvimento de sistemas de médio e grande porte.

No entanto, apesar do ControlLab ser um software de pequeno porte, a aplicação de técnicas de modelagem também é importante no intuito de aprimorar a qualidade do projeto. Com este objetivo foram utilizados, como auxilio ao projeto, diagramas de classe e de estados UML, cuja simbologia é descrita no apêndice A.

2.3.4 Padrões de Projeto

Em geral, padrões auxiliam a reduzir a complexidade em muitas situações da vida real. Ao invés de ter que escolher dentre um elevado número de possíveis combinações, padrões permitem a solução de problemas através de combinações já testadas e que funcionam. Eles estão presentes em vários aspectos do cotidiano. Podem ser citados vários exemplos, como padrões de comportamento, de beleza, etc.

No desenvolvimento de software não é diferente. Vários padrões que representam soluções maduras foram desenvolvidos e documentados. Podem ser citados, por exemplo, padrões de ordenamento de listas, de documentação e de estilo de programação.

Baseado nas idéias lançadas em [Alexander et. al., 1977], que define padrões de arquitetura, vem-se desenvolvendo a pesquisa e documentação de padrões presentes em projetos orientados a objeto. Tais estudos envolvem a definição dos

contextos em que os padrões se aplicam e de seus elementos chave. Muitos destes padrões vem sendo catalogados, como, por exemplo, em [Gamma et. al., 1994].

Um padrão de projeto pode ser representado por um conjunto de responsabilidades e colaborações que algumas classes apresentam entre si, servindo para resolver um determinado problema, de modo a aprimorar as características de reusabilidade e extensibilidade.

Os padrões de projeto orientados a objetos apresentam as seguintes vantagens:

- Representam soluções já testadas e que funcionam.
- São acompanhados, em geral, de uma boa documentação.
- Auxiliam na documentação do projeto em que são utilizados.
- Quando utilizados no contexto adequado, automaticamente aprimoram a qualidade do projeto.
- Representam um nível mais alto de abstração do que as classes.

Por outro lado, para que se alcancem tais vantagens com padrões de projeto, é necessário:

- Um profundo conhecimento de orientação a objetos para a sua definição, devendo ser caracterizados precisamente o contexto de sua aplicação e os papéis das classes participantes. No entanto, esta tarefa somente precisa ser realizada uma vez para cada padrão, que se for útil, será empregado inúmeras vezes.
- O seu entendimento para que possa ser utilizado efetivamente. No entanto, uma vez compreendido, transforma-se em um importante componente de projeto, melhorando o nível de abstração envolvido. Além disso, a utilização de uma solução reconhecida é mais desejável do que a necessidade do desenvolvimento de uma solução nova.

Alguns padrões de projeto foram utilizados neste trabalho, sendo citados durante a apresentação do ControlLab.

2.3.5 FRAMEWORKS

Uma das grandes vantagens da orientação a objetos é o seu suporte à reusabilidade, o que pode ser principalmente observado na programação. Um exemplo são os *toolkits* [Gamma et. al., 1994], que são conjuntos de classes relacionadas e reusáveis projetadas para promover funcionalidades de propósito geral[5].

Porém, a longo prazo, a reutilização de projeto é uma condição mais importante do que a de código. Neste contexto, embora as classes abstratas e declarações de tipos forneçam uma maneira de definir o projeto de uma classe ou tipo, respectivamente, tais elementos têm pouco alcance para proporcionar a reusabilidade de projeto a um nível mais abrangente. Para preencher este espaço surgiram os frameworks.

Um framework nada mais é do que um conjunto de classes cooperantes que constituem um projeto reusável para um determinado domínio de aplicação. Este conceito é muito semelhante ao conceito de um padrão de projeto, mas algumas diferenças importantes podem ser observadas:

- Nível de abstração - Um framework pode ser mapeado diretamente em código, o que não ocorre com um padrão de projeto, que o pode somente através de casos de aplicação. Deste modo, os frameworks podem ser executados e reutilizados diretamente, em contraste com os padrões de projeto que têm de ser implementados para cada caso.
- Tamanho - Um framework típico contém muitos padrões de projeto.
- Nível de Especialização - Frameworks sempre têm um domínio de aplicação particular. Por outro lado, os padrões de projeto podem ser utilizados em quase todos os tipos de aplicação.

Pode-se observar, nesta comparação, que os frameworks estão mais próximos das aplicações, como um todo, do que os padrões de projeto. Um framework auxilia, normalmente, na definição de um subsistema de um software, enquanto que um

---

[5] O *package java.io* da linguagem Java™ é um exemplo de *Toolkit*.

padrão de projeto auxilia na resolução de problemas locais. Ambos os conceitos se complementam no projeto orientado a objetos.

Evidentemente não se espera que um framework resolva por completo a problemática relacionada com o domínio de aplicações a que ele se destina, mas que concentre as respectivas decisões de projeto, definindo seu modelo de interação e seu fluxo de controle.

Assim, um framework deve ser extensível para permitir a inserção de elementos específicos de cada aplicação. Esta extensibilidade é fornecida através da utilização de classes abstratas que indicam, ao usuário do framework, como estende-lo.

Muitas vezes, além da definição da arquitetura básica de um subsistema, um framework também fornece uma biblioteca de classes que serve ao mesmo tempo como exemplo e como ponto de partida para a sua utilização.

Dentre as vantagens obtidas com a utilização de frameworks em um projeto orientado a objetos, podem-se destacar:

- Reusabilidade do Projeto – Principal objetivo de um framework, através da definição da arquitetura básica do mesmo para um domínio de aplicações.
- Reusabilidade da Análise – Um framework está ligado a um domínio de aplicações, devendo, para ser projetado, realizar uma ampla análise do respectivo domínio. Uma vez desenvolvido o framework, esta análise é reutilizada cada vez que o mesmo é aplicado.
- Projeto mais Simples – Um framework define o núcleo do projeto de um subsistema, permitindo que seu usuário se concentre em aspectos específicos da aplicação.
- Reusabilidade de Código – Um framework deve concentrar as implementações que são comuns ao domínio relacionado, incentivando a reusabilidade de código. Além disso, eventuais bibliotecas de classes podem fornecer elementos complementares a este domínio.
- Facilidade de Manutenção – A utilização de frameworks tende a facilitar a manutenção de um sistema, dado que as aplicações desenvolvidas são

similares e os frameworks centralizam parte da solução simplificando a análise de eventuais alterações.

- Produtividade – Softwares que utilizem frameworks apresentam um tempo de desenvolvimento menor, devido principalmente as características de reusabilidade envolvidas.

Como pode-se observar, as vantagens da utilização de frameworks são significativas. No entanto, é necessário completar esta análise citando quais são os problemas que a sua utilização pode acarretar:

- Dificuldade de Projeto – Certamente mais difícil de projetar do que uma aplicação, um framework deve ser constituído de forma a se adequar a todas as aplicações do seu domínio. Por esta razão é essencial que seu projeto seja tão flexível e extensível quanto possível.
- Liberdade de Projeto – Com a utilização de frameworks alguma liberdade de projeto é perdida, devendo-se aceitar as decisões de projeto encapsuladas nos mesmos.
- Dependência – Uma aplicação baseada em frameworks depende diretamente dos mesmos, sendo particularmente sensíveis a suas eventuais mudanças.
- Validação – O projeto de um framework deve ser validado, e para tanto, o framework deve ser utilizado em aplicações de seu domínio. Assim, as primeiras aplicações que utilizam um framework podem demonstrar a necessidade de alterações no mesmo, o que pode ocasionar problemas às aplicações que o utilizem.

Os dois frameworks utilizados no desenvolvimento do ControlLab, o Framework de Controle de Processos e o JHotDraw, são apresentados nos capítulos 3 e 4 desta monografia.

2.4 Programação Orientada a Objetos

A programação corresponde à fase de implementação do software, de acordo com as decisões de projeto e com as características da linguagem de programação utilizada.

Quando é orientada a objetos, a programação segue os conceitos definidos pela orientação a objetos, apresentando suas vantagens e, em geral, facilitando a compreensão do código gerado.

Neste contexto, é indicado que o projeto e a linguagem de programação também sejam orientados a objetos, de modo a não ser necessária a tradução entre diferentes paradigmas ou a inclusão dos conceitos da orientação a objetos em uma linguagem não orientada a objetos, respectivamente.

Os tópicos relacionados com o projeto orientado a objetos foram analisados na seção anterior.

Atualmente existem várias linguagens de programação, com diferentes características e limitações. Seria interessante, deste modo, realizar uma comparação entre as diferentes opções para verificar aquela que melhor se adequaria à implementação do ControlLab. Porém, essa escolha recaiu sobre a linguagem Java™ que, além de ter sido utilizada para implementar os frameworks utilizados, mostra alguns pontos positivos que são analisados nesta seção.

2.4.1 Linguagem Java™

A linguagem Java™[6] foi originalmente desenvolvida pela Sun Microsystems no início dos anos 90. Tornou-se disponível ao público em maio de 1995. Em 1996 havia sido licenciada para mais de 35 companhias, sendo suportada pelos principais sistemas operacionais, incluindo Windows™, Macintosh™ e algumas variantes de UNIX™. Está incorporada em todos os principais *browsers* Internet[7]. Em maio de 1996, uma pesquisa revelou que 62% das maiores companhias já utilizam Java™ para

---

[6] [Gosling et. al., 1996] e [Flanagan, 1996].

[7] Aplicativos que implementam o protocolo HTTP™ na Internet.

algum tipo de desenvolvimento de software, outras 14% planejavam fazê-lo até o fim do mesmo ano. Ne mesma pesquisa 40% das companhias afirmaram que a linguagem Java™ se tornará estratégica em muito breve [Bray et. al., 1997].

Os números impressionam para uma linguagem com menos de três anos de vida. Quais são as razões que motivaram tal divulgação desta linguagem?

Primeiramente, para tentar responder a este questionamento, é interessante descrever suas principais características:

- Orientada a objetos – Java™ implementa todos os conceitos da orientação a objetos, com exceção da herança múltipla, que pode ser representada através da utilização de múltiplas Interfaces.
- Interpretada – Os programas Java™ são compilados em *bytecodes*, uma codificação otimizada, que são executados por interpretadores Java™.
- Multiplataforma – A especificação da linguagem Java™ é independente de plataforma, inclusive seus *bytecodes*. O único mecanismo que depende da plataforma é o interpretador.
- Padronizada – Recentemente, a inscrição de uma especificação pública da linguagem Java™ foi aprovada pela ISO, representando o primeiro passo em direção a sua efetiva padronização. A linguagem foi projetada para ser compatível com diferentes padrões, tais como, por exemplo, o padrão Unicode™ de representação de caracteres.
- Sintaxe próxima à linguagem C – Sintaticamente, a linguagem é muito semelhante ao ANSI C, facilitando a migração de programadores C e C++, além de permitir a utilização de boa parte do conhecimento bibliográfico relacionado.
- Distribuída – Projetada para suportar aplicações em rede, apresenta uma ampla biblioteca de classes para sua implementação. Além disso, sua compatibilidade com diferentes protocolos e com browsers Internet, através

de *Applets*[8], a torna ideal para o desenvolvimento de aplicações na Internet, um importante diferencial com relação a outras linguagens.

- Segura – Fator importante para aplicações distribuídas. Java™ foi projetada para ser imune às atuais formas de ataque a sistemas, através de diferentes mecanismos de verificação de bytecodes e restrições de acesso.

- Robusta – Java™ é uma linguagem fortemente tipada com uma extensiva verificação quando da geração dos bytecodes. Seu gerenciamento de memória é automático, não apresentando ponteiros e dispondo de um coletor de lixo automático que evita o esquecimento da eliminação de objetos. Além disso, Java™ apresenta suporte ao tratamento de exceções, uma funcionalidade muito importante para o tratamento de condições anormais de funcionamento.

- Simples – Sintaticamente semelhante ao C, mas sendo uma linguagem puramente orientada a objetos (C++ é considerada híbrida) e apresentando algumas simplificações (gerenciamento de memória, ponteiros, coletor de lixo e não existência de variáveis globais, de compilação condicional e de herança múltipla), Java™ é uma linguagem relativamente simples e fácil de ser aprendida e utilizada.

- Eficiente – Não é tão eficiente quanto linguagens compiladas, pelo fato de ser interpretada, sendo a interpretação de bytecodes em torno de 10 a 20 vezes mais lenta que a execução de código C compilado, o que é aceitável para a grande maioria das aplicações. Novas versões de Java™ devem apresentar melhores performances. Além disso, pode-se, quando uma alta performance for essencial, compilar os bytecodes em código de máquina, alcançando-se resultados comparáveis ao C.

- Com múltiplos processos – Apresenta o suporte a execução de processos concorrentes com definição de prioridades. Também fornece mecanismo de

---

[8] Pequena aplicações, com restrições de segurança, próprias para serem executadas dentro de *browsers* Internet.

sincronização que evita o acesso simultâneo de diferentes processos a determinados métodos ou variáveis de um objeto.

- Dinâmica – Permite o carregamento de classes dinamicamente em tempo de execução.

Estas são as principais características que definem Java™ no contexto das linguagens de programação, sendo a sua forma de desenvolvimento um fator determinante para a evolução nesse sentido.

Durante os últimos anos houveram duas mudanças significativas na área da informática:

- Temas como padronização e qualidade ganharam relevância.
- A utilização de redes de computadores mudou os requisitos do mercado, intensificando consideravelmente a necessidade de ferramentas com suporte distribuído.

As linguagens de programação não estavam preparadas para a adequação a tais transformações e não podiam ser transformadas para tanto, pois apresentavam um compromisso com aplicações já desenvolvidas.

A linguagem Java™ foi desenvolvida do zero, ou seja, sem nenhum compromisso de compatibilidade com versões anteriores. Deste modo, pôde adequar-se às novas tendências, estando próxima da padronização, apresentando um amplo suporte distribuído e incorporando características que promovem a implementação de softwares de qualidade.

Por outro lado, trata-se de uma linguagem muito nova que ainda está em fase de desenvolvimento. Alguns problemas de segurança, por exemplo, surgiram nas primeiras implementações. O seu coletor de lixo automático pode representar um problema para determinadas aplicações de tempo real [Barry, 1991]. Este problema deve ser solucionado em versões futuras da linguagem.

Atualmente, sua especificação está sendo desenvolvida sob coordenação da Sun Microsystems e com participação aberta a partir do endereço Internet http://java.sun.com. Pessoas e entidades das mais diferentes áreas podem acompanhar

as discussões sobre o desenvolvimento da linguagem e apresentar sugestões, o que tende a fortalecer Java™ como uma linguagem de propósito geral.

Este modelo de desenvolvimento está fazendo com que a linguagem Java™ evolua em direção a novas aplicações, além de sua utilização como linguagem de programação convencional e como linguagem distribuída compatível com TCP/IP:

- Smart Cards – Semelhantes a cartões de crédito, mas com um chip embutido, como, por exemplo, os cartões Visa Cash utilizados nas olimpíadas de Atlanta. Existem vários desenvolvedores com tecnologia proprietária, mas, com a utilização da linguagem Java™[9], pretende-se criar um padrão aberto.
- Sistema Operacional – Foi lançada a arquitetura de um sistema operacional baseado na linguagem Java™, denominado JavaOS™, que objetiva integrar diferentes plataformas diretamente à utilização de Java™. Dividida em diversas camadas, semelhante a uma arquitetura de rede, deve ser licenciada para diferentes fabricantes, como IBM e Toshiba, para a utilização principalmente em pequenos dispositivos, tais como Network Computers e video-games.
- Chips – A empresa Sun Microsystems está desenvolvendo três microprocessadores Java™, um dos quais será licenciado para outros fabricantes, com o objetivo de aplicação em sistemas embutidos.

Deste modo espera-se que a escolha da linguagem Java™ para o desenvolvimento de aplicações de controle seja acertada e que permita a efetiva utilização do ControlLab e de seus resultados no ensino e pesquisa relacionados a sistemas monovariáveis de controle.

[9] Um subconjunto da linguagem já é suportado.

3. JHotDraw

Neste capítulo são abordadas as principais características do framework JHotDraw, sendo realizada uma análise preliminar da sua utilização no desenvolvimento do ControlLab. Exemplos de aplicação do framework podem ser acessados no endereço Internet *http://www.ifa.ch/hjd/hotJavaSamples.html*.

Este framework, implementado em Java™, surgiu de um estudo de caso para um seminário sobre padrões de projeto do IFA (Institut für Informatik-Ausbildung), em Zurique, Suíça. Sua principal função é auxiliar na construção de subsistemas de manipulação direta ([Schneiderman, 1982]), que constituem o seu domínio de aplicação.

A arquitetura do framework foi inspirada no projeto de dois outros frameworks: *ET++* e *HotDraw*.

3.1 ET++

O framework ET++ ([Gamma, 1988], [Gamma, 1994]) foi desenvolvido na Universidade de Zurique, no final dos anos 80, com o objetivo de fornecer uma base para a implementação de interfaces gráficas com o usuário. Implementado em C++, pode ser executado em diferentes plataformas UNIX™, suportando os sistemas X11™, NeWS™ e SunWindows™.

Em seu domínio de aplicação, a implementação de interfaces gráficas com o usuário, podem ser encontradas diversas bibliotecas, denominadas *toolboxes*, que fornecem componentes para a construção de interfaces gráficas. Os toolboxes convencionais, no entanto, não auxiliam na definição de uma estrutura consistente a nível de aplicação.

O framework ET++, por sua vez, procura, além do suporte a criação de interfaces gráficas, criar uma estrutura básica para a implementação de aplicações, que pode ser definida por três elementos principais:

- Aplicação – Realiza o controle de eventos, gerenciando um ou mais documentos.
- Documento – Encapsula uma estrutura de dados ou modelo relacionado a uma aplicação, sabendo como abrir-se e salvar-se.
- Vista – Representação gráfica de um documento. Cada documento pode ser mostrado através de diferentes vistas.

O ET++ ainda apresenta, além destes, outros elementos que podem ser divididos em diferentes grupos: estruturas de dados, elementos de gerenciamento de eventos, metaclasses e elementos da biblioteca gráfica.

Dentre as principais características oferecidas pelo framework ET++ podem-se destacar:

- Existência de um nível de abstração do sistema operacional que encapsula os mecanismo de dependência, permitindo a portabilidade do framework.
- Gerenciamento automático das janelas (movimentação, alteração do tamanho, ativação, tratamento de eventos, etc.) e dos seus conteúdos, controlando, por exemplo, o rolamento dos elementos internos (*scrolling*).
- Manuseio uniforme de arquivos e gerenciamento de caixas de diálogo de abertura, salvamento e impressão de documentos, mantendo a consistência entre diferentes aplicações.
- Compatibilidade com diferentes formatos, tais como o PostScript™.
- Flexibilidade para a atualização das janelas através de eventos de invalidação dos elementos gráficos, permitindo, por exemplo, a atualização através de buffers auxiliares (*double buffering*).
- Possibilidade da construção de elementos de interface compostos, baseados em elementos preexistentes.
- Um ambiente de desenvolvimento criado através do próprio framework.
- Hierarquia extensível de comandos, onde os mesmos possam ser desfeitos (*undo*) e refeitos (*redo*).

Algumas destas características também fazem parte da linguagem Java™, tais como a portabilidade e o gerenciamento de janelas. No entanto, a característica mais importante apresentada pelo ET++ é a arquitetura básica que o mesmo oferece para a definição de aplicações, e que pode ser observada no JHotDraw.

Além da arquitetura a nível de aplicação, outras características do ET++ que foram absorvidas pelo JHotDraw são os mecanismos de atualização de janelas e a definição de comandos.

3.2 HotDraw

O framework HotDraw ([Brant, 1995]) é um framework para a implementação de editores gráficos para desenhos bidimensionais. Tais desenhos são compostos por elementos vetoriais, as figuras, diferentemente, por exemplo, dos editores de imagens.

Sua arquitetura básica segue o modelo do framework de interface com o usuário Model/View/Controller definido no Smalltalk 80™ [Goldberg, 1984], através dos seguintes elementos:

- Editor e Desenho – Representam o *Model*. Um editor é responsável pela inicialização da aplicação e gerenciamento do ambiente de edição, enquanto que um desenho é responsável pelo gerenciamento das figuras.
- Vista – Representa o *View*. Responsável por mostrar seu modelo, o desenho, em uma janela, o editor. Torna-se dependente do desenho, de modo que alterações no mesmo são notificadas para que a vista possa atualizar sua aparência.
- Controlador – Representa o Controller. Responsável por gerenciar as entradas do usuário, representadas por eventos de mouse e de teclado, que são utilizadas na edição dos desenhos e de suas figuras.

As figuras, por sua vez, podem representar graficamente diferentes contextos, dependendo do modo como são implementadas. O HotDraw dispõe de uma biblioteca de figuras básicas, tais como textos, retângulos, elipses, linhas, etc. Também podem ser criadas figuras compostas baseadas em agrupamento de figuras preexistentes

Além desses elementos básicos, o HotDraw dispõe de outros conceitos importantes:

- Manipulador (*Handle*) – Entidades ligadas a figuras que realizam operações de manipulação direta sobre as mesmas. Os manipuladores ligados a um retângulo, por exemplo, permitem a edição de seu tamanho.
- Ferramenta (*Tool*) – Encapsula uma forma de comportamento em resposta a entradas do usuário. Uma ferramenta de criação de retângulos, por exemplo, irá criar um retângulo ao ser pressionado o mouse em uma vista. O editor sempre apresenta uma única ferramenta ativa.
- Animação – O framework dispõe de um mecanismo para a definição de diferentes algoritmos de animação para as figuras.
- Definição de regras (*Constraints*) – A partir de um mecanismo de notificação e dependência entre diferentes figuras pode-se criar regras de relacionamento entra as mesmas. Um conexão entre duas figuras, por exemplo, deve manter-se conectada às mesmas, mesmo durante a sua movimentação.

Todos estes conceitos implementados no HotDraw também estão presentes no JHotDraw, assim como o modelo de comportamento que segue a arquitetura básica do framework de interface com o usuário Smalltalk MVC.

3.3 Características

A arquitetura básica do JHotDraw desenvolveu-se a partir da composição das arquiteturas definidas pelos frameworks ET++ e HotDraw, sendo complementada com os conceitos auxiliares definidos pelo HotDraw.

As classes que formam esta arquitetura básica e seus principais relacionamentos podem ser observados no diagrama de classes[1] apresentado na figura 3.1.

---

[1] Os diagramas de classe apresentados neste trabalho seguem o modelo UML que é tratado no Apêndice A deste trabalho.

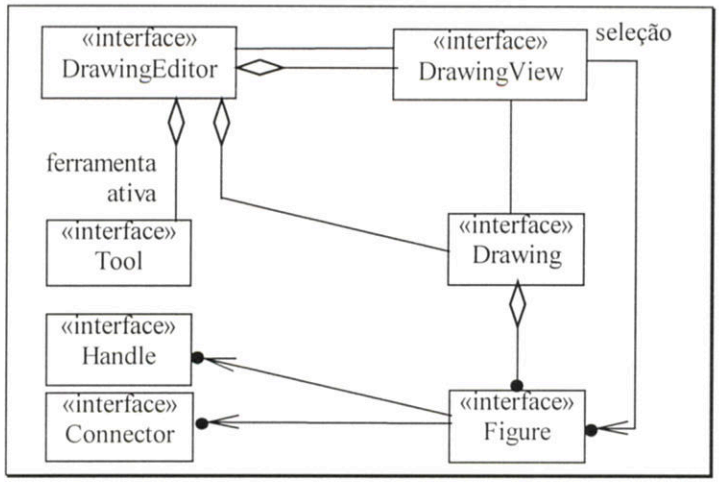


Figura 3.1 - Arquitetura Básica

Pode-se observar, por este diagrama, a principal característica observada em um framework. A arquitetura básica e o correspondente fluxo de controle são definidos através de um conjunto de interfaces, a serem estendidas pelo usuário do framework.

Esta arquitetura é composta pelos seguintes elementos:

- *DrawingEditor* – Apresenta características da aplicação do ET++ e do editor do HotDraw. Sua principal tarefa é intermediar o relacionamento entre a área gráfica de edição (*DrawingView*), o desenho (*Drawing*) e a ferramenta sendo executada (*Tool*), como definido pelo padrão de projeto *Mediator* ([Gamma et. al., 1994:273-282]).
- *DrawingView* – Com responsabilidades semelhantes às das vistas do ET++ e do HotDraw, constitui uma área gráfica que mostra uma visão de um desenho (*Drawing*), gerenciando o mecanismo de atualização gráfica. Também implementa o papel de controlador do HotDraw, realizando o tratamento de eventos do usuário.
- *Drawing* – Pode ser definido como um documento do ET++ que apresenta as características de um desenho do HotDraw. Deste modo, representa o elemento de edição de um DrawingEditor que gerencia um conjunto de figuras (*Figures*).

- *Figure* – Apresenta a mesma semântica de uma figura do HotDraw. Representa a classe mais importante do framework em termos de extensão, pois o mesmo pode ser aplicado como mecanismo de manipulação direta em aplicações de diferentes domínios através da especialização de *Figures*, de modo a representar características inerentes a tais domínios (o ControlLab é um exemplo desta característica). As figuras apresentam, ainda, um mecanismo de notificação de eventos que permite a criação de dependências entre as mesmas e destas com os respectivos Drawing e DrawingView. Este mecanismo, que implementa o padrão de projeto *Observer* ([Gamma et. al., 1994:293-303]), é utilizado, por exemplo, para a notificação da necessidade de atualização gráfica.

- *Handle* – Com a mesma semântica do manipulador do HotDraw, é uma entidade gráfica ligada a uma figura que permite a edição da mesma através de manipulação direta. Como pode ser observado no diagrama, cada figura define seus próprios manipuladores.

- *Connector* – Este elemento, não representado no HotDraw, define um ponto de conexão (ou conector) de uma figura. É utilizado como elemento de referência para figuras de conexão (*ConnectionFigure*). Foram muito importantes para a implementação da semântica de conexões do ControlLab.

- *Tool* – Apresenta a mesma semântica da ferramenta do HotDraw, representando um modo de operação do editor e respondendo aos eventos do usuário. Podem ser definidos diferentes ferramentas, mas apenas uma estará ativa, como pode ser observado no diagrama.

Além destes elementos, outras interfaces definem mecanismos auxiliares do framework:

- *Locator* – Auxilia na localização de pontos de uma figura, tais como, por exemplo, os cantos de um retângulo ou o centro de uma elipse, sendo utilizado por *Handles* e *Connectors*.

- *ConnectionFigure* – Figura que realiza a conexão de figuras, baseada em *Connectors*.

- *Painter* – Encapsula o estratégia de atualização gráfica de um DrawingView, como definido no padrão de projeto *Strategy* ([Gamma et. al., 1994:315-323]).
- *PointConstrainer* – Auxilia a definição de grades para uma DrawingView, através da limitação dos pontos da mesma.
- *FigureEnumertion* – Estrutura de dados que representa um conjunto de figuras.
- *FigureChangeListener* – Objeto que recebe notificações de uma figura através de eventos de modificação (*FigureChangeEvent*).

Além deste conjunto de interfaces, o JHotDraw também dispõe de uma biblioteca de classes concretas que permite a utilização direta do framework. Dentre os mecanismo presentes nesta biblioteca podem-se destacar:

- Suporte à criação de menu gráfico de botões, que permitem a organização das ferramentas em uma estrutura amigável ao usuário.
- Mecanismo de persistência, com suporte a serialização de objetos, permitindo o armazenamento dos desenhos fora do ambiente de edição.
- Hierarquia de comandos que permite a execução de ações atômicas no editor, não suportando, no entanto, ações de undo e redo.
- Menu de comandos que permite a organização dos comandos através de menus.
- Memória auxiliar para o armazenamento temporário de figuras, permitindo a execução de comandos de cópia sobre as mesmas.
- Suporte a animação dos desenhos, de modo semelhante ao HotDraw.
- Classes para o desenvolvimento de implementações mais simples de aplicações normais e *Applets*.
- Conjunto variado de figuras, tais como retângulos, elipses, textos, conexões e linhas.

3.4 APLICAÇÃO

O JHotDraw apresenta a base necessária para o desenvolvimento de subsistemas de manipulação direta. Por esta razão, foi escolhido para realizar a implementação do modelo de edição adotado pelo ControlLab.

Algum nível de adaptação, no entanto, é necessário. Através de uma análise preliminar sobre o framework e sua utilização no contexto deste trabalho, observam-se as seguintes necessidades:

- A implementação de elementos que definam a semântica de controle, devendo ser compatíveis com o Framework de Controle de Processos.
- O JHotDraw, assim como outros editores gráficos, define o agrupamento de figuras. Um agrupamento é um conjunto de figuras que se comporta como uma única figura, permanecendo no mesmo nível das outras figuras. É mais adequada, a implementação do conceito de desenhos aninhados, onde conjuntos de figuras representem diferentes desenhos, seguindo uma organização hierárquica.
- Uma ferramenta que realize a simulação de sistemas.
- A implementação de figuras que possibilitem a definição e o monitoramento de sinais de controle.
- A implementação de mecanismo de undo e redo para os comandos utilizados no ControlLab.
- A extensão do menu de botões de modo que um maior número de ferramentas possa ser utilizado, como no caso do ControlLab, devido a sua extensa biblioteca de elementos de controle.

Os diferentes mecanismos utilizados para estender JHotDraw com o objetivo de atender as necessidades do ControlLab são caracterizados durante sua descrição.

4. Framework de Controle de Processos

O Framework de Controle de Processos foi desenvolvido no mesmo contexto que o ControlLab, como projeto final do curso de Engenharia de Controle e Automação Industrial da Universidade Federal de Santa Catarina [Quarti, 1997]. Tendo como domínio de aplicação os sistemas monovariáveis, com especial ênfase na área de controle, foi utilizado como suporte de controle do ControlLab.

4.1 Análise de Domínio

O Framework de Controle de Processos procura representar de uma maneira simples o contexto de seu domínio, utilizando conceitos similares aos utilizados em sistemas monovariáveis.

Na análise e projeto de tais sistemas é comum a modelagem matemática através de diagramas de bloco. Estes diagramas representam os componentes matemáticos do sistema e o fluxo de dados entre tais componentes.

A utilização desta forma de abordagem, baseada em diagramas de bloco, pelo Framework de Controle de Processos, apresenta as seguintes vantagens:

- Mantém a consistência com a maioria das metodologias de controle de sistemas monovariáveis.
- Facilita a visualização do sistema e de suas características, através da definição de seus componentes e inter-relacionamentos.
- Representa uma melhor abordagem para a indicação do fluxo de dados do sistema real do que uma abordagem puramente matemática.
- Possibilita a avaliação das contribuições individuais dos componentes para o sistema.

4.2 ARQUITETURA BÁSICA

O framework mapea o conceito de bloco em uma classe abstrata, denominada *Block*, que define o comportamento comum dos blocos. Assim, blocos reais são representados por subclasses de *Block*.

Cada bloco, como pode ser observado na figura 4.1, apresenta um determinado número de entradas e de saídas, dispondo de um algoritmo interno para o cálculo dos respectivos valores de saída. Através da criação de novos blocos, podem-se modelar novos algoritmos, limitando-se às características disponíveis pela linguagem Java™.

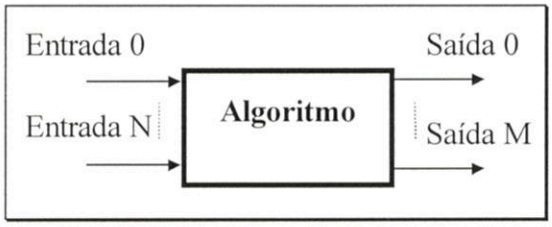


Figura 4.1 - Modelo de um Bloco

O mecanismo de comunicação entre blocos, que indica o seu fluxo de dados é definido através de conexões entre entradas e saídas. Uma conexão determina que os valores de sua saída e de sua entrada são os mesmos.

Com a definição de blocos e conexões o framework permite a criação da topologia de um diagrama de blocos. No entanto, devido à complexidade que sistemas monovariáveis podem alcançar, sua representação pode se tornar igualmente complexa e difícil de ser compreendida, mesmo através da uma abordagem baseada em blocos.

Esta dificuldade é tratada pelo Framework de Controle de Processos pela possibilidade do agrupamento de blocos. Utilizando o padrão de projeto *Composite* [Gamma, 1994:163-173], o framework permite, através da classe *CompositeBlock*, tal estruturação de um modo transparente ao diagrama a que pertencem, ou seja, eventuais agrupamentos de blocos são visto pelo diagrama como blocos simples, mantendo a uniformidade do modelo.

Um exemplo pode ser observado na figura 4.2, onde a implementação da média de dois valores é realizada através da composição de blocos preexistentes.

Neste exemplo, a média também poderia ser calculada pela implementação de um bloco simples com o mesmo algoritmo interno. A escolha depende do compromisso que o usuário do framework apresenta com a complexidade de sua biblioteca de blocos.

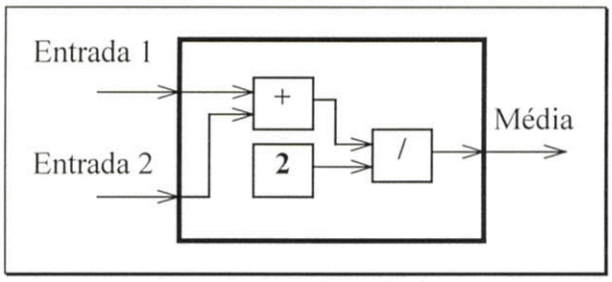


Figura 4.2 - Bloco Composto

Um diagrama de blocos também pode ser visto como um agrupamento de blocos. Além disso, um diagrama não apresenta entradas ou saídas. Este conceito é definido pelo Framework de Controle de Processos, através de uma subclasse da classe *CompositeBlock*, denominada *BlockDiagram*.

É interessante que um diagrama forneça a seus blocos um mecanismo de tempo que possa servir de suporte para seus algoritmos. Um *BlockDiagram* do framework apresenta um relógio, implementado por uma subclasse da classe *Block* denominada *Clock*, que fornece o valor do período inerente ao diagrama e sua integral, que corresponde ao respectivo instante de tempo.

A definição, além do tempo, de um período é muito importante para a implementação e simulação computacional de sistemas monovariáveis, dado que modelos discretos de sistemas reais dependem diretamente do período envolvido.

Um diagrama deve dispor de um mecanismo que gerencia a sua execução. O Framework de Controle de Processos apresenta uma classe abstrata, *BlockDiagramEngine*, que define um agente externo que efetua tal tarefa. O framework também fornece uma subclasse concreta de *BlockDiagramEngine*, denominada *ThreadedEngine*, que dispõe de um mecanismo simplificado para a execução do diagrama. No entanto, o usuário do framework tem a possibilidade de implementar, através de novas subclasses de *BlockDiagramEngine*, algoritmos mais sofisticados para a execução de diagramas, incluindo, por exemplo, mecanismos de

notificação ao usuário, que serviriam para indicar ao usuário eventuais atrasos da execução do diagrama em função do tempo real, entre outras informações.

A função básica de um *BlockDiagramEngine* é acionar a execução do diagrama em intervalos definidos pelo respectivo relógio. Um diagrama, por sua vez, ao ser acionado, aciona todos seus blocos que se executam. Este acionamento recursivo é herdado da classe *CompositeBlock* que também é responsável pelo acionamento de seus blocos.

Este mecanismo de execução de um diagrama, assim como de blocos compostos em geral, apresenta um inconveniente. É necessária a definição da ordem de execução dos blocos, de modo a tornar a execução do diagrama consistente. Esta situação é especialmente observada em sistemas com realimentação.

O Framework de Controle de Processos apresenta uma classe abstrata, *DataFlowSolver*, cujo objetivo é encapsular a estratégia de ordenamento dos blocos de um bloco composto. O framework também fornece duas classes concretas, *AutoSortSolver* e *SequentialSolver* para a execução desta atividade, não impedindo, no entanto, que o usuário implementa outros algoritmos, dependendo de suas necessidades.

Os elementos apresentados nesta seção constituem a arquitetura básica do framework de controle de processos, cuja representação em notação UML de diagrama de classes pode ser observada na figura 4.3.

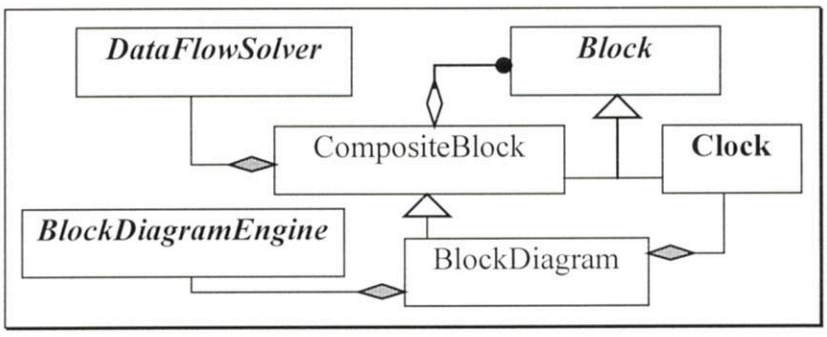


Figura 4.3 - Arquitetura Básica

4.3 Biblioteca

Além das classes relacionadas com sua arquitetura básica, o Framework de Controle de Processos dispões de uma ampla biblioteca de blocos, dentre os quais podem-se destacar:

- Blocos de definição e monitoramento de sinais por parte do usuário, através de campos de entradas de texto e gráficos.
- Principais sinais: constante, degrau, rampa, senoidal e parábola.
- Equação a diferenças.
- Processo de primeira ordem com parâmetros contínuos.
- Controlador PID.
- Bloco para a escrita de dados do diagrama em arquivo.
- Blocos com operações numéricas em geral, incluindo operações trigonométricas.
- Blocos de comparação.
- Blocos booleanos para a definição, em conjunto com o blocos de comparação, de lógicas dentro de um sistema.
- Blocos multiplexador e demultiplexador.

O framework também apresenta uma classe abstrata, *DataAcquisitionBoard*, que representa um elemento de interface com placas de aquisição de dados. Suas subclasses encapsulam o protocolo de comunicação com placas específicas. Dois blocos, *AnalogInputBlock* e *AnalogOutputBlock*, representam, a nível de blocos, o acesso a este mecanismo de troca de informações com placas de aquisição de dados.

4.4 Controle Distribuído

O Framework de Controle de Processos dispões de um mecanismo para a comunicação de diferentes diagramas de blocos através da utilização do suporte distribuído oferecido pela linguagem Java™.

Para que um diagrama de blocos seja acessado remotamente, precisa ser registrado, através de um nome, em um objeto da classe *Registry* que gerencia o acesso ao mesmo. Um *Registry* define a porta a partir do qual os diagramas registrados podem ser acessados remotamente.

Um diagrama registrado pode exportar, através de um nome, os seguintes elementos:

- As saídas de qualquer bloco.
- Blocos que implementem a interface *Value*, através do método *setValue*, cujo parâmetro é um número em ponto flutuante (*float*).

Deste modo, tanto podem ser alterados como observados valores de interesse de um diagrama de blocos remoto.

Por outro lado, para o acesso a tais elementos, o Framework define dois blocos:

- RemoteOutputBlock – Bloco que permite o monitoramento de saídas exportadas por um diagrama remoto.
- RemoteValueBlock – Bloco que permite a alteração de valores de blocos exportados por um diagrama remoto.

Estes blocos precisam conhecer o endereço do diagrama a ser acessado, através de um endereço URL[1], e dos respectivos elementos deste diagrama, através dos nomes pelos quais foram exportados. Este endereço URL deve ser formado pelos seguintes elementos:

- Identificador do framework (*control*).
- Endereço da máquina a ser acessada.
- Porta onde o diagrama está registrado e inicializado.
- Nome do diagrama.

---

[1] *Uniform Reference Locator*

Pode-se, por exemplo, ter-se o seguinte endereço URL de acesso a um diagrama remoto: *control://nomeDaMáquina.ufsc.br:5000/nomeDoDiagrama.*

Cada bloco remoto pode acessar simultaneamente diferentes saídas ou blocos de um mesmo diagrama remoto. É possível ainda definir o tamanho dos *buffers* envolvidos na comunicação.

Este suporte distribuído oferecido pelo framework foi especialmente considerado no desenvolvimento do ControlLab, cujos mecanismos envolvidos são analisados na sua descrição.

4.5 APLICAÇÃO

O Framework de Controle de Processo define um modelo simples e flexível para a modelagem e simulação de sistemas monovariáveis. No entanto, por tratar-se de um projeto novo (o ControlLab é a primeira aplicação a efetivamente utilizar o framework), ainda está fase de amadurecimento e validação.

Deste o modo, o desenvolvimento do ControlLab também serviu como mecanismo de aprimoramento do framework. Dentre as características do framework que foram modificadas em função da experiência resultante do desenvolvimento deste projeto, podem-se destacar:

- Os blocos se tornaram mais flexíveis, de modo a se adequarem a um editor de diagramas. Alguns blocos, por exemplo, passaram a permitir a alteração do número de entradas e saídas, de modo a possibilitar a alteração de tais parâmetros em tempo de edição.
- Os blocos incorporaram o mecanismo de serialização da linguagem Java™ (*Serializable*), que não adiciona qualquer complexidade a classe correspondente.

O ControlLab, por sua vez, foi projetado de modo a não alterar a arquitetura básica de seu subsistema de controle, representado pelo Framework de Controle de Processos. Assim, é possível a utilização dos modelos desenvolvidos no ControlLab em outros contextos, um dos objetivos do projeto, que é detalhado no próximo capítulo.

5. ELEMENTOS EXTERNOS

Os capítulos anteriores abordaram tópicos preliminares que serviram de suporte para a desenvolvimento do projeto. A partir deste ponto é descrito o seu resultado, o ControlLab.

Este programa apresenta um conjunto de características e conceitos visíveis ao usuário que objetivam qualificá-lo como um laboratório virtual para o estudo, modelagem e validação de sistemas monovariáveis de controle. Este capítulo fornece uma visão geral sobre tais elementos.

5.1 ÁREA DE TRABALHO

A topologia da área de trabalho do ControlLab foi elaborada tendo-se como ponto de referência os exemplos fornecidos com o JHotDraw, e apresenta, como pode ser observado na figura 5.1, os seguintes elementos: *Vista do Diagrama*, *Paleta de Ferramentas*, *Linha de Status* e *Barra de Menus*.

5.1.1 VISTA DO DIAGRAMA

Equivalente a uma vista do JHotDraw, representa a área onde é efetivamente realizada a edição de um diagrama (ver seção 5.2 - Diagrama), sendo também responsável por repassar os eventos do usuário para a ferramenta ativa (ver seção 5.3.1 - Ferramentas), assim como as informações do diagrama e a seleção. Atua sobre a seleção através dos eventos de digitação da tecla *Del*, eliminando-a, e das teclas direcionais, deslocando-a.

Seu conteúdo pode ser impresso através do comando *Imprimir* do menu *Arquivos*. Também permite a rolagem da vista por manipulação direta, através do mouse, permitindo a execução de ferramentas entre áreas do diagrama que não estejam aparecendo simultaneamente na vista.

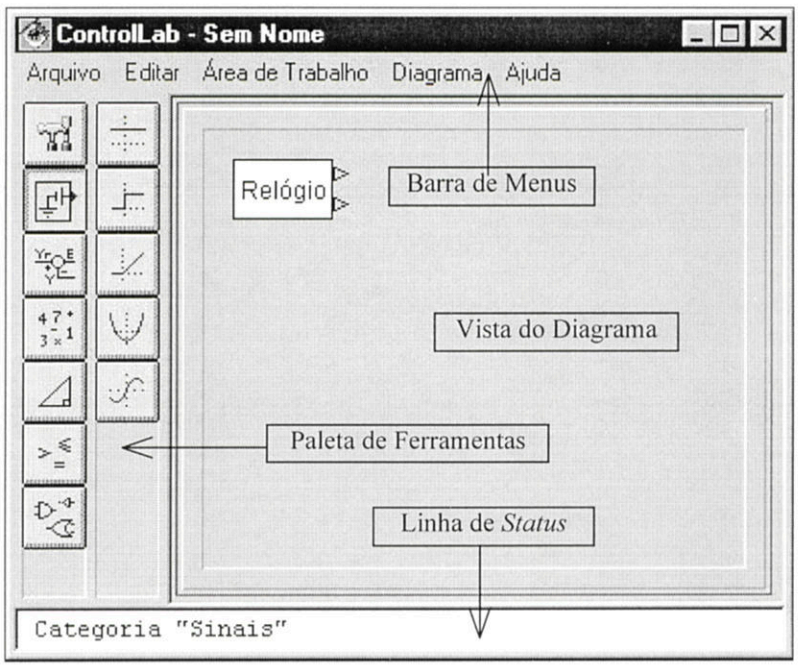


Figura 5.1 - Tela Principal do ControlLab

5.1.2 Paleta de Ferramentas

Elemento responsável por organizar as ferramentas do ControlLab. É composta por duas barras de botões: a barra de categorias e a barra de ferramentas.

A barra de categorias indica os grupos existentes de ferramentas. A primeira categoria, denominado de *Ferramentas Básicas*, reúne ferramentas essenciais para a utilização do ControlLab, enquanto que as outras categorias organizam a biblioteca de figuras de bloco do ControlLab. Já a barra de ferramentas mostra as ferramentas da categoria selecionada.

5.1.3 Linha de Status

Através da linha de status, o ControlLab repassa informações relevantes ao usuário. Normalmente é mostrada função da ferramenta ativa, mas, dependendo do posicionamento do mouse, ainda são mostradas as seguintes informações:

- Função de um bloco.
- Função de uma entrada ou saída.
- Identificação de uma conexão.

- Nome de uma categoria de ferramentas
- Função de uma ferramenta.

Deste modo, o usuário recebe as informações necessárias sobre os elementos sob manipulação. Como será mostrado nas seções 5.4 e 5.5, parte destas informações pode ser definida pelo próprio usuário.

5.1.4 Barra de Menus

A barra de menus organiza os comandos do ControlLab, que correspondem a ações atômicas e são detalhados no decorrer deste capítulo.

5.1.5 Configuração

O usuário pode modificar o área de trabalho do ControlLab através dos comandos do menu *Área de Trabalho*:

- *Editar Paleta* – Permite ao usuário modificar a biblioteca de ferramentas do ControlLab.
- *Opções* – Permite ao usuário alterar certos parâmetros da área de trabalho do ControlLab, através da janela *Opções*, que pode ser observada na figura 5.2.
- *Nova Janela* – Abre uma nova janela do ControlLab permitindo a edição e simulação em paralelo de diferentes diagramas.

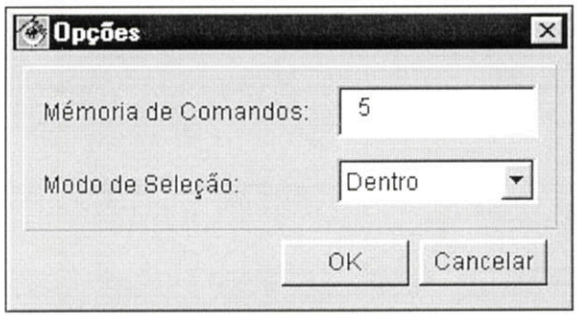


Figura 5.2 - Janela de Opções

5.2 DIAGRAMA

Elemento básico de edição do ControlLab, sendo similar a um *Drawing* do JHotDraw, um diagrama é composto por elementos gráficos, que podem ser mostrados na vista, e por elementos não gráficos que complementam sua definição.

Os elementos gráficos que formam um diagrama, e que podem ser observados na figura 5.3, são a sua borda e as figuras do ControlLab. A borda de um diagrama define seus limites, apresentando duas funções básicas: servir como interface com elementos externos e evitar que figuras sejam perdidas fora da área visível do diagrama. As figuras que podem ser criadas em um diagrama são de três tipos diferentes:

- Figuras de Bloco – Representação gráfica de um bloco do framework de controle de processos, apresentando entradas e/ou saídas para a conexão com outras figuras de bloco.
- Figura de Conexão – Indica graficamente a conexão entre uma entrada e uma saída.
- Rótulo – Para a realização de comentários em um diagrama.

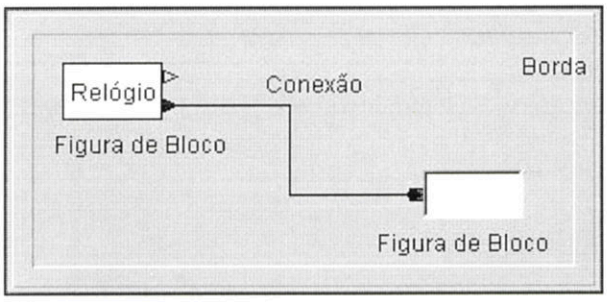


Figura 5.3 - Elementos Gráficos de um Diagrama

Os elementos não gráficos de um diagrama são os seguintes:

- Tamanho – Define a área de um diagrama, exibida graficamente através de sua borda. Pode ser alterado através do comando *Alterar Tamanho* da menu *Diagrama* da barra de menus.

- Informações gerais – Definem o nome e uma descrição para um diagrama, além de indicar seu autor. Podem ser editados na seção *Geral* da janela *Editor do Diagrama Principal*, aberta através do comando *Configurar Diagrama* do menu *Diagrama*, e que pode ser observada na figura 5.4.

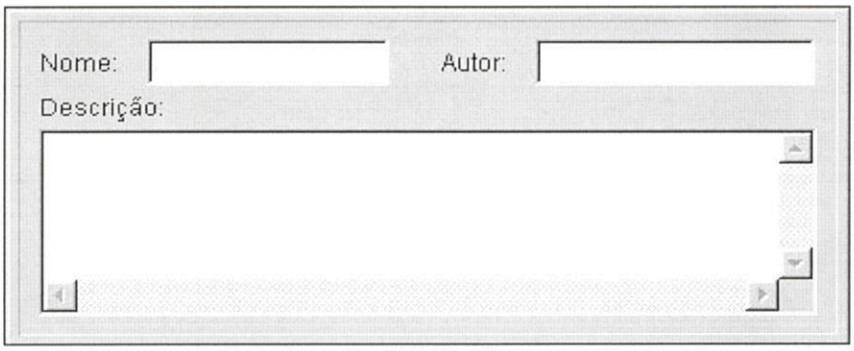


Figura 5.4 - Informações Gerais do Diagrama

- Configuração da Simulação – Define características da simulação de um diagrama (ver seção 5.9 – Simulação).
- Configuração da Grade – Define se um diagrama está utilizando o mecanismo de grade, assim como as respectivas distâncias vertical e horizontal. Podem ser alteradas através do comando *Configurar Grade* do menu *Diagrama*.
- Configuração Remota – Define os elementos necessários para a utilização de um diagrama remotamente (ver seção 5.8 – Execução Remota).
- Grupos – Definem subconjuntos de elementos de um diagrama (ver seção 5.2.2 – Grupos).

Com o objetivo de complementar a semântica de um diagrama, o ControlLab apresenta os conceitos de subdiagrama e grupo.

5.2.1 SUBDIAGRAMA

Tipo especial de diagrama que é utilizado como a mesma semântica de uma figura de bloco, ou seja, como um componente para a composição de diagramas. As configurações de um subdiagrama são herdadas das configurações do diagrama principal do qual o mesmo faz parte.

O ControlLab não limita o número de níveis de subdiagramas que o usuário pode utilizar, ou seja, um subdiagrama pode ser composto por figuras de blocos normais e/ou outros subdiagramas, como pode ser observado no esquema da figura 5.5.

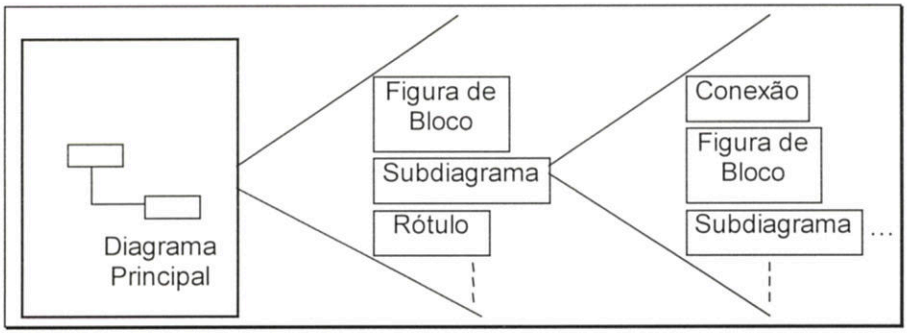


Figura 5.5 - Diagramas e Subdiagramas

O ControlLab indica, através do título de sua janela, qual o diagrama de um diagrama principal que está sendo editado, como na figura 5.6.

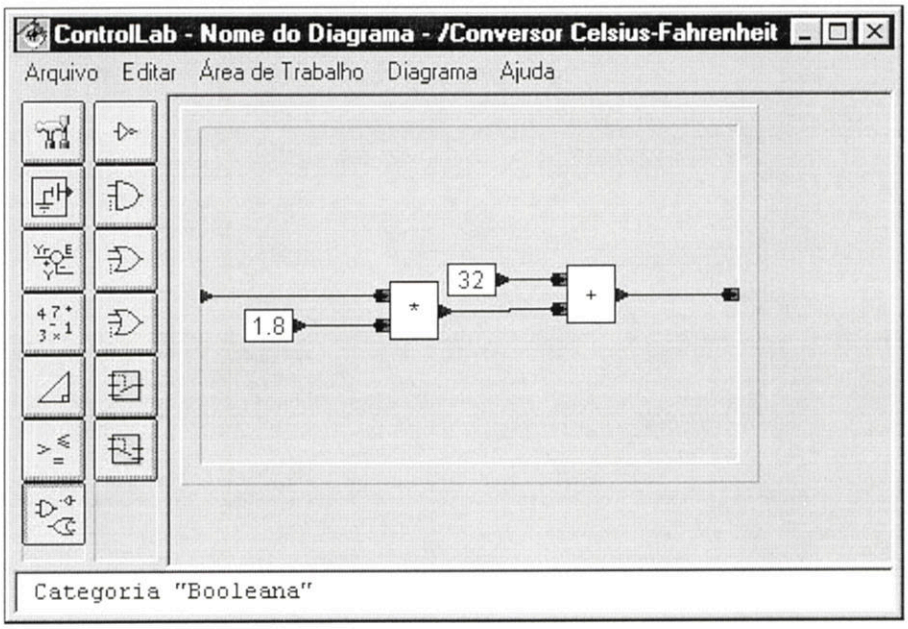


Figura 5.6 - Edição de um Subdiagrama

Um subdiagrama pode ser criado através do comando *Criar Subdiagrama* do menu *Diagrama*. Este comando transforma as figuras selecionadas automaticamente em um subdiagrama.

Para abrir um subdiagrama que compõe um diagrama, o usuário necessita executar um duplo clique do mouse sobre a figura do subdiagrama. Para retornar é necessário executar a mesma ação sobre qualquer área livre do subdiagrama aberto.

5.2.2 GRUPO

Um grupo representa um conjunto de figuras de um diagrama, servindo como um complemento ao conceito de subdiagramas. Apenas figuras do diagrama principal podem ser classificadas em grupos, já figuras de um subdiagrama pertencem ao mesmo grupo a que pertence o subdiagrama. Um exemplo da organização de um diagrama em grupos pode ser observado na figura 5.7.

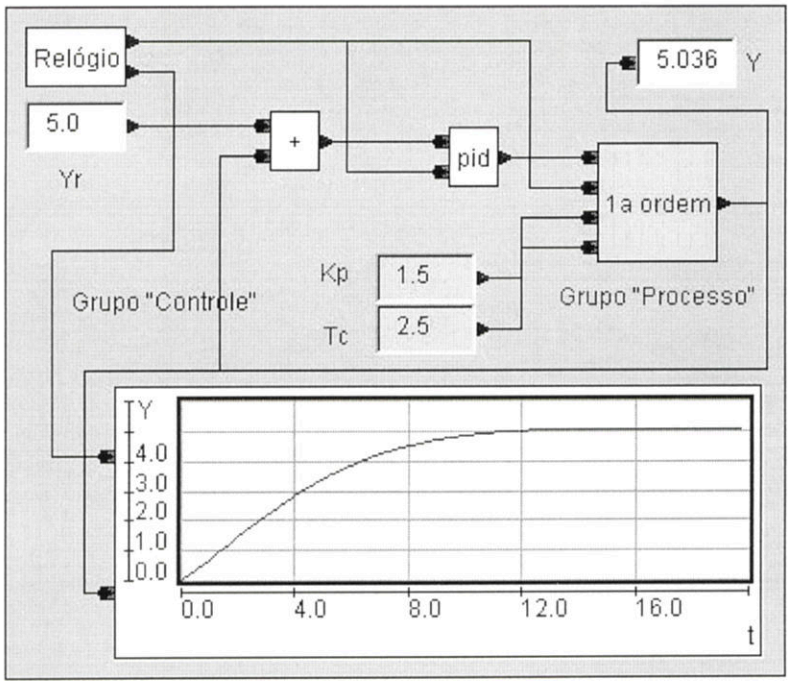


Figura 5.7 - Diagrama Organizado em Grupos

Cada grupo é definido por um conjunto de cores que identificam graficamente suas figuras: *Cor de Fundo*, *Cor da Borda* e *Cor do Texto*. O gerenciamento dos grupos de um diagrama é realizado na seção *Grupos* da janela *Editor do Diagrama Principal*, que é aberta através do comando *Configurar Diagrama* do menu *Diagrama*, e que pode ser observada na figura 5.8.

Figura 5.8 - Configuração de Grupos

5.3 Programação Visual

Segundo [Green, 1995], a programação visual está relacionada com qualquer sistema que permita ao usuário a especificação de um programa em duas ou mais dimensões. Neste grupo não são incluídas as linguagens convencionais baseadas em representação textual, tais como Java™.

Analisando [Quarti, 97] e os exemplo fornecidos de utilização do respectivo framework, pode-se classificá-lo, baseado no conceito acima, como um elemento não visual de programação de diagramas de blocos.

No entanto, os conceitos utilizados pelo framework são relativamente simples e permitem a implementação de um ambiente de programação visual para a implementação dos diagramas, característica implementada no ControlLab.

O exemplo de um processo controlado por um controlador PID fornecido pelo tutorial do Framework de Controle de Processos em [Quarti, 1997], cujo diagrama pode ser observado na figura 5.9, necessita apenas de cerca de vinte linhas de código para definir o seu comportamento básico, através dos blocos que participam do diagrama e o seu inter-relacionamento, além de definir o elemento de execução do diagrama (*DiagramEngine*).

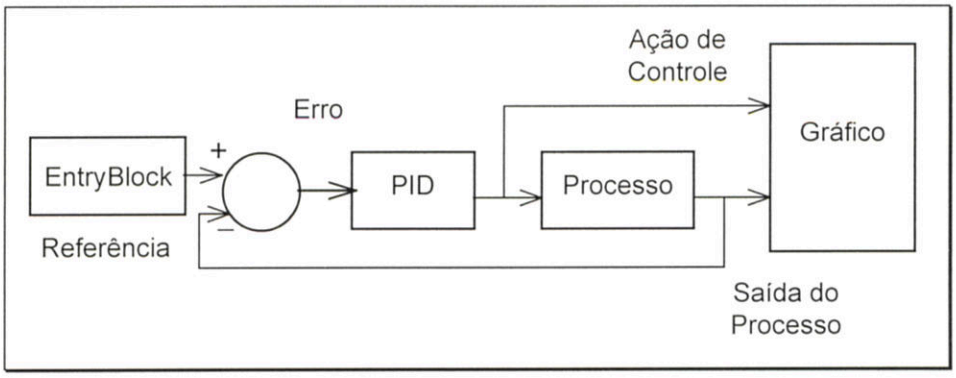


Figura 5.9 - Malha de Controle PID

No entanto, a implementação deste exemplo, que é fornecida pelo framework, utiliza mais de quinhentas linhas de código (desconsiderando os comentários). A simples transcrição do exemplo nesta monografia utilizaria cerca de doze páginas.

Utilizando-se o ControlLab, por sua vez, para a criação de um diagrama equivalente, seria obtido uma representação similar à apresentada na figura 5.7, sem a necessidade da digitação de uma única linha de código. Neste contexto, o ControlLab também executa, através da vista do diagrama, o papel de interface do usuário com o diagrama.

Outro fator importante, além da complexidade envolvida, é a flexibilidade. Um diagrama implementado diretamente sobre o framework deve ser compilado novamente após cada alteração. Já no ControlLab, os diagramas são construídos em tempo de edição de modo transparente ao usuário, cuja única preocupação é definir o diagrama propriamente dito através das ferramentas do ControlLab.

5.3.1 Ferramentas

No ControlLab, de forma semelhante ao JHotDraw, uma ferramenta corresponde a um modo de operação em resposta a eventos do usuário na vista do diagrama. Cada janela do ControlLab apresenta uma única ferramenta ativa.

A figura 5.10 mostra as ferramentas do ControlLab presentes na categoria *Ferramentas Básicas* da paleta de ferramentas, descritas nesta seção.

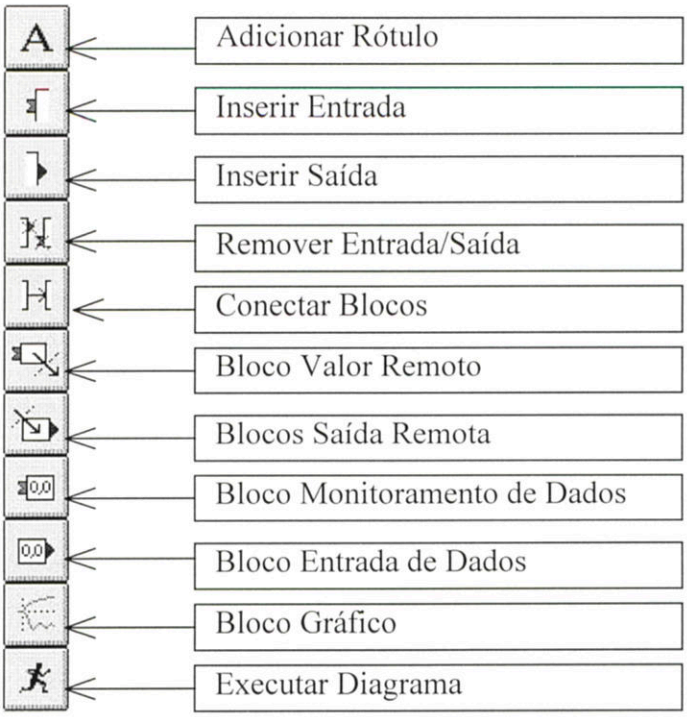


Figura 5.10 - Categoria *Ferramentas Básicas*

Após a execução de uma ferramenta da paleta de ferramentas, esta ferramenta é automaticamente desativada. Caso o usuário deseje continuar utilizando-a, deve manter pressionada a tecla *shift* no momento da sua execução.

5.3.1.1 DEFAULT

Apesar de não fazer parte da paleta de ferramentas, esta ferramenta é muito importante, pois permanece ativa quando nenhuma ferramenta da paleta está selecionada.

Apresenta os seguintes comportamentos:

- Duplo clique do mouse – Sobre a figura de um subdiagrama realiza a sua abertura. Sobre uma figura qualquer ou sobre um subdiagrama (com a tecla *shift* pressionada) executa a abertura da respectiva janela de edição (ver seção 5.5 – Configurando Figuras). Sobre a área livre de um subdiagrama proporciona o seu fechamento.
- Movimentação com o botão pressionado – Se o movimento iniciar sobre uma figura, realiza o seu deslocamento. Caso inicie sobre uma área livre,

serve para executar a seleção de um conjunto de figuras através de um retângulo de seleção. Se por outro lado, o movimento iniciar sobre a entrada ou saída de uma figura tenta criar uma conexão (do mesmo modo que a ferramenta *Conectar Blocos*, descrita na seção 5.3.1.6).

- Ação de pressionar o botão do mouse – Utilizado para realizar a seleção de figuras isoladas. Caso a tecla *shift* esteja pressionada adiciona a respectiva figura para a seleção já existente.

5.3.1.2 Adicionar Rótulo

Adiciona um rótulo ao diagrama no lugar onde o mouse for pressionado, através da abertura de uma área de escrita, como a que pode ser vista na figura 5.11, onde o usuário digita o texto do rótulo.

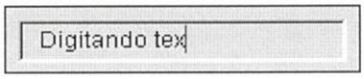


Figura 5.11 - Entrada de Texto de um Rótulo

5.3.1.3 Inserir Entrada

Quando o mouse é pressionado sobre a figura de um bloco que apresente um número variável de entradas, insere uma nova entrada na figura selecionada.

5.3.1.4 Inserir Saída

Quando o mouse é pressionado sobre a figura de um bloco que apresente um número variável de saídas, insere uma nova saída na figura selecionada.

5.3.1.5 Remover Entrada/Saída

Remove, se possível, um conector da figura sobre a qual o mouse foi pressionado.

5.3.1.6 Conectar Blocos

Através da movimentação do mouse com o botão pressionado de uma entrada até uma saída, ou vice-versa, realiza a criação de uma conexão. Pode ser substituída pela ferramenta default, que apresenta o mesmo comportamento.

5.3.1.7 Executar Diagrama

Indica um modo especial onde o diagrama principal está sendo simulado de acordo com a respectiva configuração. Durante a execução desta ferramenta, o usuário não pode modificar o diagrama.

Caso o diagrama apresente figuras desconectadas, o usuário pode executá-lo sem a participação das mesmas. O mesmo não ocorre no caso de uma figura parcialmente conectada, caso onde a execução não é possível.

Se o diagrama apresentar algum problema para ser executado, como, por exemplo, uma figura parcialmente conectada, o usuário é informado.

5.3.1.8 Criar Figura de Bloco

Responsável por adicionar uma nova figura de bloco à vista de um diagrama quando o mouse é pressionado. É utilizada para adicionar as figuras de bloco básicas do ControlLab que estão na categoria *Ferramentas Básicas*, além de ser a única ferramenta presente nas outras categorias.

5.4 Biblioteca de Blocos

Corresponde às figuras de blocos que podem ser criadas a partir da paleta de ferramentas, com exceção das figuras da categoria *Ferramentas Básicas*. O apêndice B apresenta as figuras que compõem a biblioteca de figuras de blocos fornecida com o ControlLab.

No entanto, esta biblioteca não é rígida. O usuário pode modificar inteiramente a biblioteca de blocos. Isto é realizado na janela *Editar Paleta* que é aberta através do comando de mesmo nome do menu *Área de Trabalho*.

A seção *Categorias*, que pode ser observada na figura 5.12, realiza o gerenciamento das categorias da biblioteca, possibilitando a criação de novas categorias, e a eliminação ou modificação de categorias já existentes. Cada categoria é definida por um nome e por um ícone que a identifica na paleta.

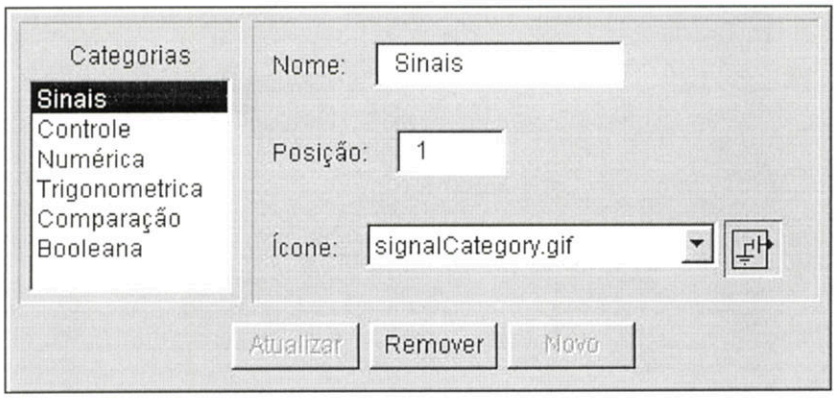


Figura 5.12 - Configuração das Categorias

Na seção *Blocos*, que pode ser observada na figura 5.13, é realizado o gerenciamento dos blocos que pertencem às categorias existentes. Assim como para as categorias, cada figura de bloco apresenta um nome e um ícone que a identifica na paleta de ferramentas.

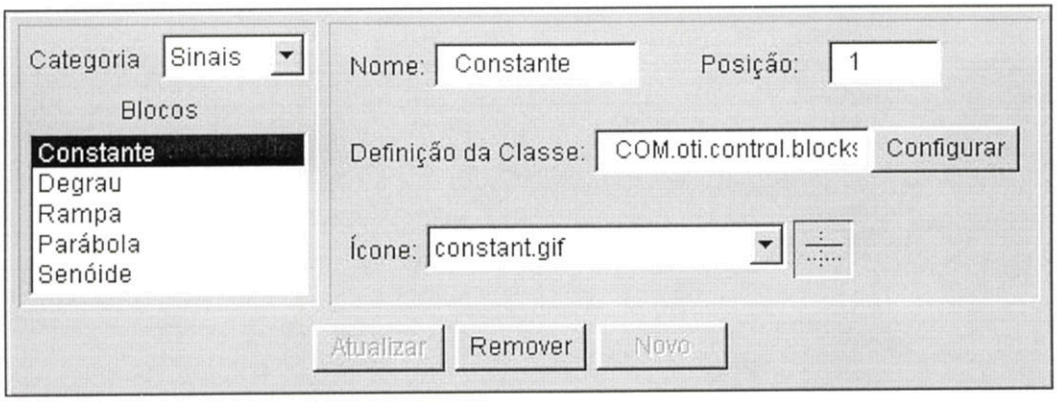


Figura 5.13 - Configuração das Figuras de Bloco das Categorias

Para que uma nova figura de bloco seja criada, o bloco correspondente deve ser válido, apresentando um *constructor*[1] sem parâmetros. As propriedades do bloco de controle da nova figura de bloco que sigam o modelo JavaBeans™ serão

automaticamente reconhecidas pelo ControlLab. Através destas propriedades e de algumas propriedades da figura de bloco em si, podem ser definidas configurações default para as figuras de blocos, como pode ser visto na próxima seção.

Os ícones que aparecem em ambas as seções estão presentes no subdiretório *Icons* do diretório de trabalho do ControlLab (ver apêndice C – Estrutura de Diretórios)[2]. Caso o usuário deseje adicionar novos ícones ao ControlLab, basta copiar os novos arquivos neste diretório.

5.5 Configuração das Figuras

As figuras do ControlLab apresentam dois mecanismos de configuração, a manipulação direta e a janela de configuração. A configuração da figura por manipulação direta ocorre através da ferramenta default, quando alguma ação do mouse é executada sobre os manipulador de uma figura, que aparecem quando a mesma está selecionada. Já a janela de configuração é aberta na ferramenta default, através do duplo clique do botão do mouse sobre a figura.

5.5.1 Rótulo

Os manipuladores desta figura podem ser observados na figura 5.14. O manipulador amarelo é responsável por alterar o tamanho da figura, enquanto que os outros são manipuladores de deslocamento.

Figura 5.14 - Manipuladores de um Rótulo

A janela de configuração de um rótulo, que pode ser vista na figura 5.15, permite definir o grupo a que pertence a figura, caso ela esteja no diagrama principal,

---

[1] Método do Java™ responsável por criar uma nova instância de uma classe.

assim como o seu texto. Caso a tecla *shift* esteja pressionada quando for executado o duplo clique sobre a figura, uma caixa de texto como a da figura 5.11 será aberta permitindo a edição do texto da figura.

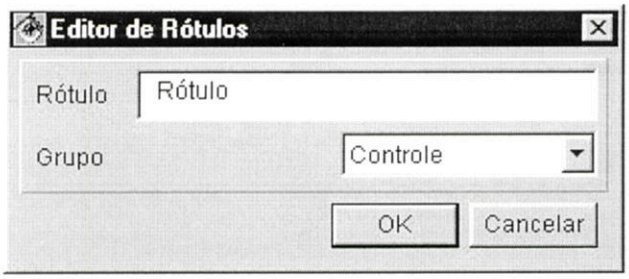


Figura 5.15 - Janela de Configuração de um Rótulo

5.5.2 Figura de Conexão

Os manipuladores de uma figura de conexão podem ser observados na figura 5.16. Os manipuladores amarelos são responsáveis por alterar a posição dos segmentos intermediários da conexão, enquanto que os manipuladores verdes servem para alterar as extremidades (saída e entrada) da mesma. Já a janela de configuração de uma figura de conexão apenas define um nome para a mesma.

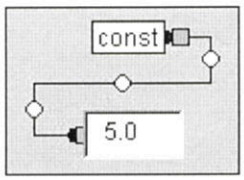


Figura 5.16 - Manipuladores de uma Figura de Conexão

5.5.3 Figura de Bloco

Os manipuladores de uma figura de bloco podem ser de dois tipos: manipuladores de deslocamento como para os rótulos, e manipuladores brancos que servem para alterar o tamanho da figura. Estes último estão presentes apenas nas

---

[2] Ver Apêndice C – Estrutura de Diretórios.

figuras de bloco *Entrada de Dados*, *Monitoramento de Dados* e *Gráfico*. Exemplos podem ser observados na figura 5.17.

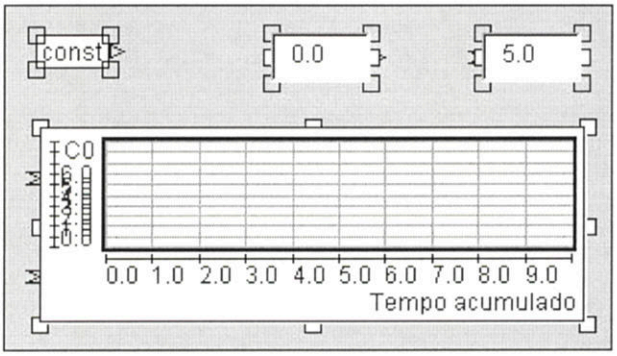


Figura 5.17 - Manipuladores de Figuras de Bloco

Uma figura de bloco pode apresentar uma grande variedade de propriedades para serem configuradas em função do bloco representado. Por este motivo, foi utilizado um modelo baseado em JavaBeans™, que é descrito no próximo capítulo, para a composição da janela de configuração de uma figura de bloco, de modo que, para cada tipo diferente de bloco representado, ter-se-á uma janela de configuração diferente. Esta janela é a mesma que aparece a partir da opção *Configurar* da janela de edição das figuras de bloco de uma categoria da biblioteca de blocos do ControlLab.

No entanto, apesar de cada figura de bloco apresentar uma janela de configuração própria, algumas propriedades são comuns:

- Nome – Nome da figura de bloco.
- Rótulo – Para as figuras que são identificadas graficamente por um texto (todas com exceção das figuras *Entrada de Dados*, *Monitoramento de Dados* e *Gráfico*). Indica o texto de identificação da figura.
- Grupo – Para as figuras que se situam no diagrama principal, no caso de diagramas com diferentes grupos.
- Explicação – Mensagem explicativa da função da figura.
- Explicações das Entradas – Mensagens explicativas das entradas.
- Explicações das Saídas – Mensagens explicativas das funções das saídas.

No caso de figuras de subdiagramas, ainda são definidos os nomes Beans das entradas e saídas utilizados na sua exportação como um componente JavaBeans™. Além destas propriedades, várias outras podem ser encontradas. A figura 5.18, mostra o exemplo de uma janela de configuração de uma figura de bloco somador.

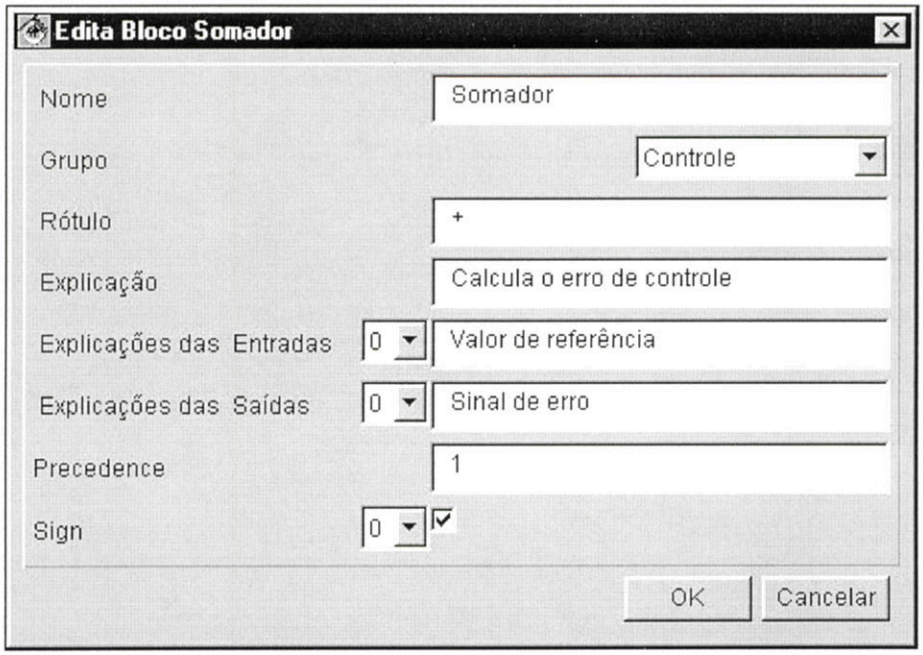


Figura 5.18 - Janela de Configuração de uma Figura de Bloco Somador

A única figura de bloco que não segue este modelo é o *Gráfico* que apresenta uma janela de configuração própria, cujo exemplo pode ser observado na figura 5.19.

5.6 Comandos Complementares

O ControlLab dispõe de comandos complementares à criação e configuração das figuras, os quais podem ser encontrados no menu *Editar*.

5.6.1 Desfazer/Refazer

É possível, através dos comandos *Desfazer* e *Refazer* (undo e redo), que o usuário desfaça e refaça ações (comandos ou ferramentas) sobre o diagrama, através de uma pilha que armazena um histórico destas ações. O tamanho da pilha pode ser modificado na janela de opções do ControlLab (ver seção 5.1.5 - Configuração).

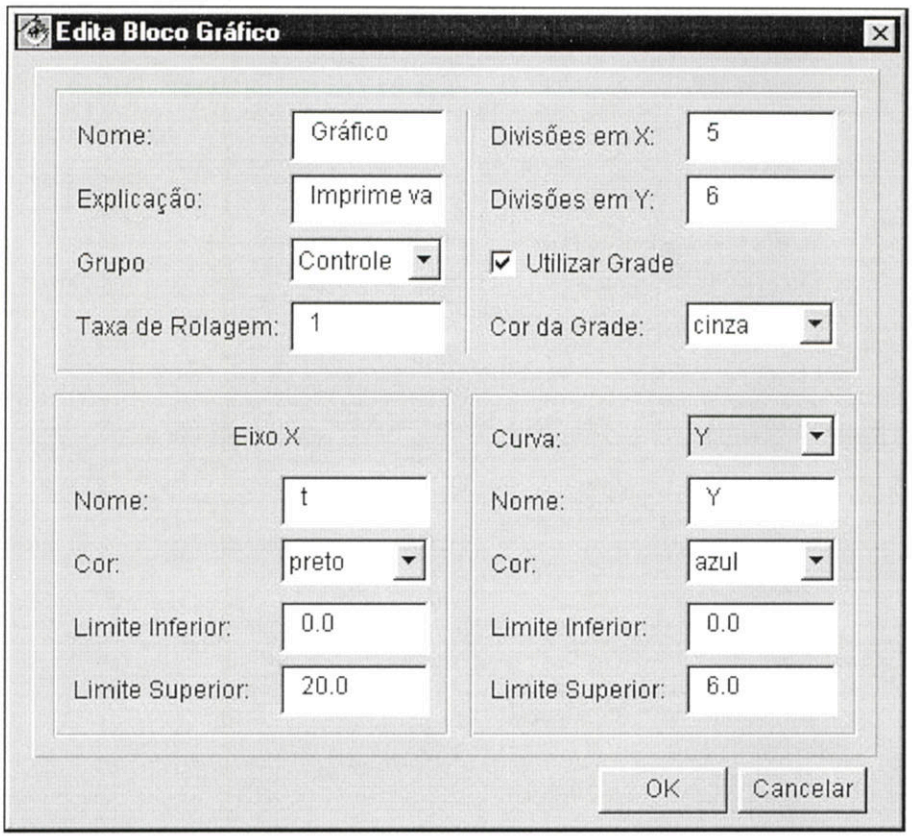


Figura 5.19 - Janela de Configuração de um Gráfico

5.6.2 Área de Transferência

O ControlLab apresenta uma área de transferência própria que permite a cópia de figuras em um mesmo diagrama ou entre diferentes diagramas, através dos seguintes comandos:

- *Copiar* – Copia as figuras selecionadas para a área de transferência.
- *Cortar* – Realiza a mesma ação do comando *Copiar*, removendo as figuras originais.
- *Colar* – Copia as figuras da área de transferência para a vista do diagrama.
- *Duplicar* – Duplica as figuras selecionadas, sem precisar, deste modo, copia-las para a área de transferência.

5.6.3 Alteração de Grupo

O comando *Alterar Grupo da Seleção* permite que o usuário altere o grupo da seleção de figuras sem necessitar alterar uma figura de cada vez.

5.6.4 Ocultamento

Em diagramas mais complexos, é possível o ocultamento de figuras, de modo que apenas as figuras relevantes ao usuário sejam mostradas. Os comandos envolvidos são:

- *Ocultar* – Oculta as figuras selecionadas.
- *Mostrar* – Volta a mostrar as figuras previamente ocultas.

5.7 Suporte de Linguagem

O ControlLab apresenta um suporte de linguagem que permite sua execução em diferentes idiomas. Este suporte é baseado em um mecanismo fornecido pela linguagem Java™, que será descrito no próximo capítulo.

Atualmente o programa apresenta tabelas em inglês e português, ao mesmo tempo que dispõe de um mecanismo para a extensão de tais tabelas pelo usuário. No subdiretório *ls* do diretório de trabalho do ControlLab (ver apêndice C – Estrutura de Diretórios) encontram-se arquivos texto que representam os fontes de tais tabelas. Neste diretório, ainda pode ser encontrado o arquivo *readme.txt* que descreve os passos necessários para a inserção de uma nova linguagem ao ControlLab.

5.8 Execução Remota

A execução de diagramas com possibilidade de acesso remoto representa uma das características mais importantes do ControlLab, vindo a permitir a pesquisa relacionada ao controle distribuído.

Os elementos que constituem o suporte remoto do ControlLab seguem o modelo definido pelo Framework de Controle de Processos (ver seção 4.4 – Controle

Distribuído). Nesse contexto, um diagrama pode desempenhar o papel de servidor ou cliente, ou ainda os dois simultaneamente.

5.8.1 SERVIDOR

Um diagrama torna-se um servidor quando saídas ou blocos do mesmo são exportados permitindo o acesso de diagramas no papel de cliente. O ControlLab permite apenas que elementos do diagrama principal sejam exportados, fato também observado no Framework de Controle de Processos com relação a um *BlockDiagram*.

Uma figura de bloco que pode ser exportada, ou seja, que representa um bloco que implemente a interface *Value* do Framework de Controle de Processos, automaticamente apresenta em sua janela de configuração um campo, denominado *Exportar*, que permite ao usuário exportá-la. A exportação de uma saída, por sua vez, implica na conexão da mesma com a borda do diagrama principal.

Para o devido funcionamento de um diagrama como um servidor, alguns parâmetros devem ser definidos. Esta tarefa é realizada através da seção *Remoto* da janela *Editor do Diagrama Principal* que é aberta a partir do comando *Configurar Diagrama* do menu *Diagrama* e que pode ser observada na figura 5.20.

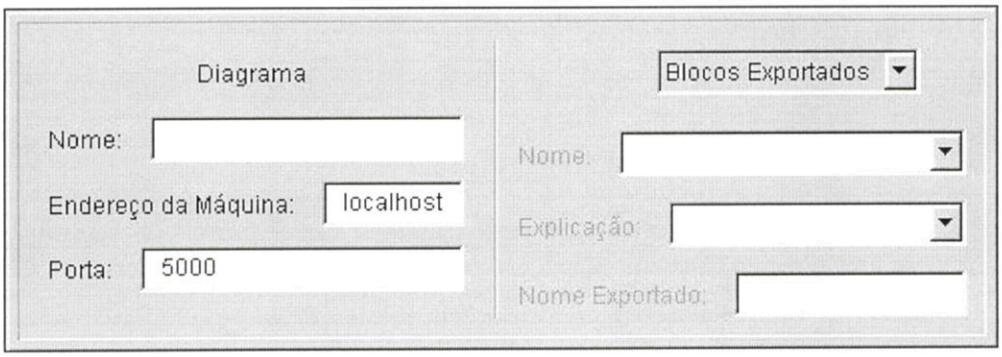


Figura 5.20 - Configuração dos Elementos Remotos

- Nome – Indica o nome com o qual o diagrama será exportado, e que deve ser utilizado para acessá-lo.
- Endereço da Máquina – Indica o endereço da máquina na qual o diagrama será exportado. No caso de os respectivos clientes estarem na mesma

máquina, pode ser utilizado o identificador *localhost* definido pela linguagem Java™.

- Porta – Define a porta onde o diagrama servidor será registrado.

Além destes parâmetros que se referem ao diagrama, também devem ser definidos, na mesma janela, os nomes de cada elemento a ser exportado. Caso algum nome não seja definido o diagrama não poderá ser executado.

5.8.2 Cliente

O ControlLab permite que um diagrama se comporte como um cliente ao dispor das figuras de bloco *Saída Remota* e *Valor Remoto*, que correspondem aos blocos *RemoteOutputBlock* e *RemoteValueBlock* do Framework de Controle de Processos.

Através das janelas de configuração destes blocos são definidos a URL do diagrama, assim como o nome do elemento do diagrama a serem acessados.

Para que um diagrama que seja um cliente possa ser executado é requerido que os servidores a serem acessados já estejam sendo executados.

5.9 Simulação

No ControlLab, a execução (ou simulação) dos diagramas, através da ferramenta *Executar Diagrama*, apresenta algumas opções que são definidas na janela *Configura Simulação* aberta a partir do comando de mesmo nome do menu *Diagrama*, que pode ser observada na figura 5.21.

- Período – Indica qual o período de execução ou passo de simulação do diagrama, representando o mesmo valor que é editado a partir da janela de configuração da figura de bloco *Relógio*.
- Tempo Real / Tempo Simulado – Permite ao usuário indicar uma relação entre o tempo real e o tempo calculado no relógio do diagrama. É bastante útil na simulação de diagramas com constantes de tempo muito altas, evitando que o usuário precise realizar uma simulação muito demorada. Este

parâmetro não influi no resultado da simulação dado que o comportamento do relógio dos diagramas não é afetado.

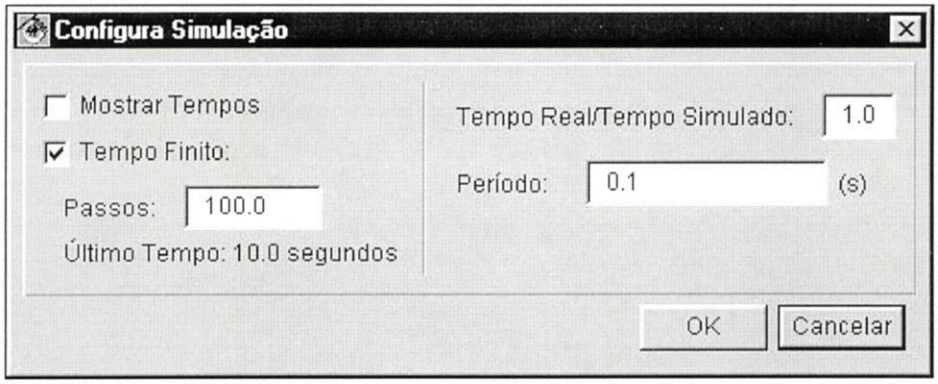


Figura 5.21 - Janela de Configuração da Simulação

- Mostrar Tempos – Opção para mostrar, ao final da simulação, seus tempos real e simulado, como pode ser observado na figura 5.22. É interessante para observar o modo como a simulação de um diagrama acompanha o tempo real, em função de sua complexidade e período de execução.

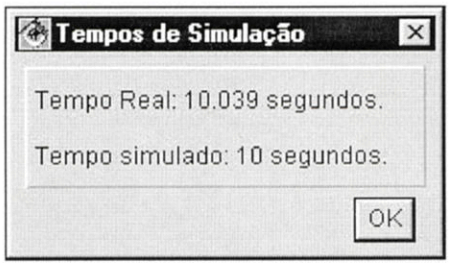


Figura 5.22 - Tempos de Simulação

- Tempo Finito – Opção para definir a execução da simulação em um tempo finito. O valor deste tempo é configurado no campo *Passos*, sendo calculado pela multiplicação do número de passos da simulação com o respectivo período de execução.

Caso queira-se reinicializar o relógio de um diagrama deve-se executar o comando *Reinicializar* do menu *Diagrama*.

5.10 ARQUIVOS

O gerenciamento dos arquivos é realizado pelos comandos do menu *Arquivo*.

Diagramas podem ser salvos em arquivos no formato *.dg* ou *.ser*, que seguem respectivamente os modelos JHotDraw *Storable* e Java™ *Serializable*, descritos no próximo capítulo. Este último apresenta a desvantagem de não permitir a compatibilidade com versões anteriores, motivo pelo qual a abertura e salvamento de arquivos neste modelo estão nos submenus *Importar* e *Exportar* do menu *Arquivo*.

Subdiagramas podem ser salvos no formato *.sdg* que também segue o modelo JHotDraw *Storable*. Assim, é permitido ao usuário a criação de uma biblioteca de subdiagramas, uma alternativa à biblioteca de figuras de bloco do ControlLab. É importante lembrar que uma figura de bloco, mesmo representando um *CompositeBlock* do Framework de Controle de Processos, é vista pelo ControlLab como uma caixa preta, ou seja, o ControlLab não tem acesso ao seu algoritmo interno. Já um subdiagrama é aberto e permite a modificação de seu algoritmo de execução.

Um diagrama principal pode ser exportado como um subdiagrama e vice-versa. Grupos de um diagrama podem ser exportados na forma de subdiagramas ou diagramas principais. Ao exportar um grupo como um diagrama principal, o usuário ainda pode escolher dentre três formas: normal, como servidor ou como cliente. Esta diferenciação resulta das conexões que o grupo apresentar com outros grupos e que podem ser transformados em elementos exportados (servidor), em elementos de acesso a um servidor (cliente) ou podem ser desconsiderados (normal).

O ControlLab procura, em seus algoritmos de transformação, otimizar as características dos elementos criados, de modo que o usuário não necessite realizar maiores ajustes posteriores.

5.11 GERAÇÃO DE CÓDIGO

Além dos formatos de arquivos destinados a seu uso interno, que foram descritos na seção anterior, o ControlLab permite a geração de arquivos na forma de código Java™ (formato *.Java*) a partir de diagramas e subdiagramas. Esta

funcionalidade é especialmente importante para a utilização dos sistemas desenvolvidos no ControlLab fora do ambiente, um dos objetivos iniciais do projeto.

O ControlLab pede ao usuário a definição do arquivo no qual será gerado o código, e cujo nome será utilizado para definir o nome da classe principal a ser gerada. Também pede o nome do *package* Java™ a que irão pertencer as classes a serem geradas. Todos os outros elementos dos arquivos resultantes são gerados automaticamente.

5.11.1 Diagrama

A partir de um diagrama principal pode ser gerado código para os seguintes tipos de arquivo:

- Applet – Para ser utilizado em um *browser* Internet. Os parâmetros de topologia são exportados como constantes da classe gerada, podendo ser ajustados antes da compilação do código fonte.
- Aplicação sem Janela – Uma aplicação Java™ que não apresenta interface com o usuário. Eventuais elementos de interface são substituídos por figuras de bloco sem interface equivalentes. É especialmente útil para a execução de servidores.
- Aplicação com Janela – Aplicação Java™ com características de topologia definidas da mesma forma que para no caso de Applet. Um exemplo pode ser observado na figura 5.23.

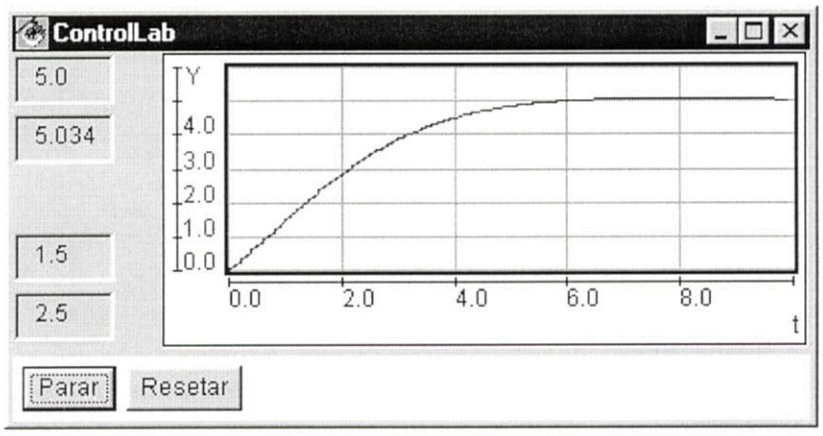


Figura 5.23 - Aplicação Gerada pelo ControlLab

- Diagrama de Blocos – Como uma subclasse da classe *BlockDiagram* do Framework de Controle de Processos. Serve para a utilização dos diagramas gerados no ControlLab em outros softwares que utilizem o mesmo framework.

5.11.2 SUBDIAGRAMA

A partir de um subdiagrama pode ser gerado código na forma de um componente JavaBeans™ ou de uma subclasse da classe *CompositeBlock* do Framework de Controle de Processos:

- Java Bean (ver seção 6.3.2.1 – Informações do Bloco) – Componente de software Java™ que pode ser utilizado em ambientes de programação visual da linguagem Java™, tais como o IBM VisualAge™, cujo exemplo pode ser observado na figura 5.24.

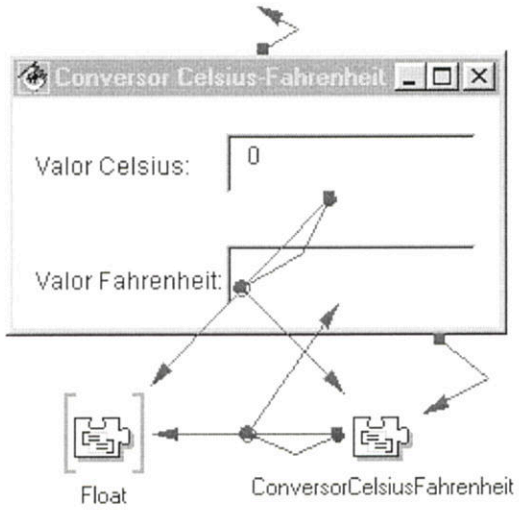


Figura 5.24 - Exemplo de Utilização de um componente JavaBeans™

- Bloco Composto – Serve para a utilização dos subdiagramas gerados no ControlLab em outros softwares que utilizem o mesmo framework, além de permitir que os blocos compostos gerados sejam utilizados na biblioteca de figuras de blocos do ControlLab.

5.12 Suporte ao Usuário

Em complemento ao arquivos que formam o ControlLab, também é fornecido no subdiretório *tutorial* do seu diretório de trabalho (ver apêndice C – Estrutura de Diretórios) um documento que define seu tutorial. Este documento demonstra ao usuário a maior parte dos elementos descritos neste capítulo.

Esta monografia também serve como uma documentação de auxílio a utilização do ControlLab. Além disso, o endereço de correio eletrônico ControlLab@edugraf.ufsc.br pode ser utilizado para o envio de dúvidas, sugestões ou problemas na utilização do ControlLab.

6. VISÃO INTERNA

O capítulo anterior descreveu o ControlLab segundo uma visão de usuário. Este capítulo, por sua vez, objetiva descrevê-lo sob a perspectiva da Engenharia de Software.

6.1 APLICAÇÃO

O primeiro passo no desenvolvimento do ControlLab foi a definição de uma estrutura geral para a aplicação, cujo ponto de referência foi a estrutura definida pelo framework JHotDraw, sintetizada pelas classes *DrawingEditor* e *DrawingView* que podem ser observadas na figura 3.1.

O JHotDraw ainda fornece classe abstratas para a implementação de aplicativos e *Applets* mais simples, que são utilizados no desenvolvimento dos demos do framework, mas que não o foram como base para o ControlLab. Em seu lugar foi projetada a estrutura que pode ser observada na figura 6.1.

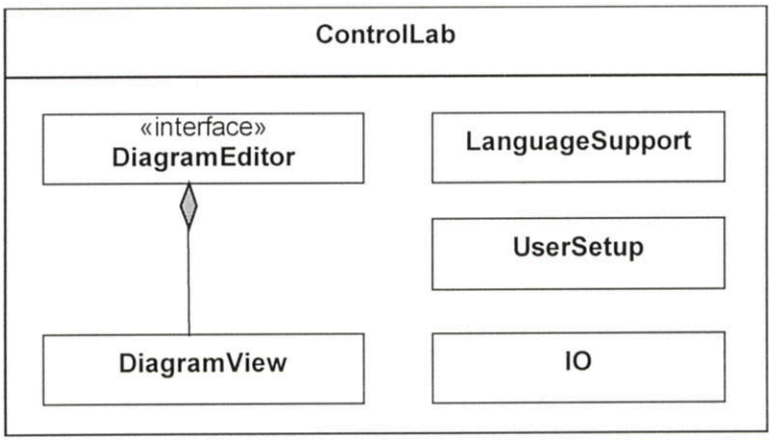


Figura 6.1 - Estrutura Básica do ControlLab

6.1.1 CONTROLLAB

A classe denominada *ControlLab* representa o núcleo da aplicação, sendo composta pelos seguintes elementos: vista de diagrama (*DiagramView*), editor de

diagrama (*DiagramEditor*), suporte de linguagem (*LanguageSupport*), configuração do usuário (*UserSetup*) e interface com o sistema (*IO*).

Apresenta as seguintes funções:

- Sua classe é responsável pela abertura e fechamento da aplicação e de novas janelas, como pode ser observado no diagrama de estados da figura 6.2.

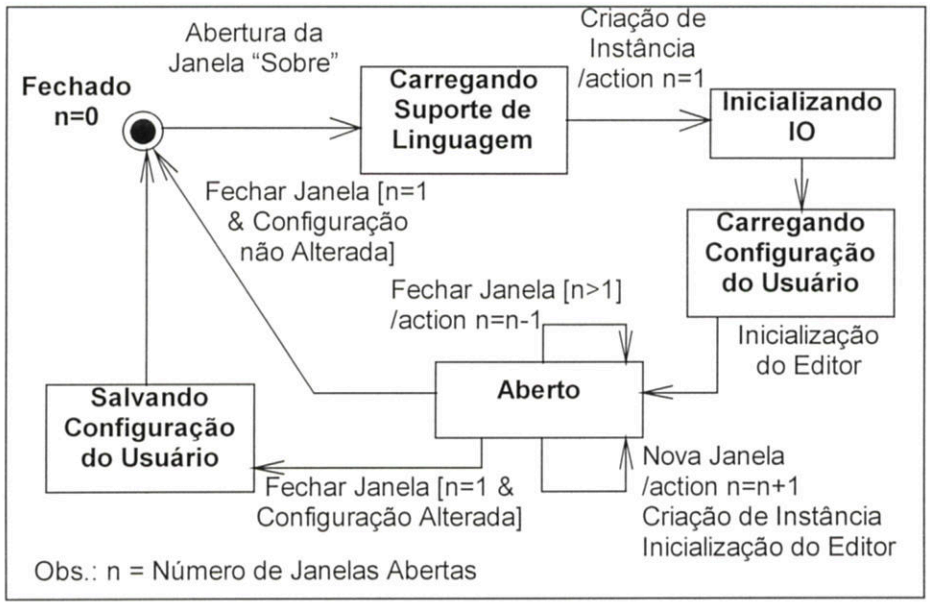


Figura 6.2 - Estados da Classe ControlLab

- A classe também é responsável por gerenciar suas instâncias, além do suporte de linguagem e da configuração do usuário, que são elementos comuns para todas as janelas, representando, portanto, componentes de classe.
- Cada instância, por sua vez, realiza o gerenciamento de uma janela, através do seu editor e de sua vista.
- Realiza a intermediação entre seus componentes, como definido pelo padrão de projeto *Mediator* ([Gamma et. al., 1994:273-282])[1].
- Fornece serviços básicos de aplicação para seus componente, tais como janelas de informação e de questionamento ao usuário.

Além dessas responsabilidades, a classe ControlLab mantém informação sobre sua versão, que é utilizada para a manutenção de compatibilidade com diagramas desenvolvidos em versões anteriores do software.

6.1.2 Editor de Diagramas – DiagramEditor

Esta interface é uma extensão à interface *DrawingEditor* do framework JHotDraw, descrita no capítulo 3. Sua principal responsabilidade, além das responsabilidades herdadas é servir como elemento base para a definição da interface com o usuário. Foi projetada de modo a permitir a implementação de editores na forma de aplicações normais ou de *Applets*. No entanto, sua implementação atual apresenta a versão apenas como aplicação normal (ver seção 6.4.1 - Editor).

Uma das características definidas por esta interface é a apresentação de informações ao usuário, através da interface *Explainable*. Esta interface, implementada por diferentes classes do ControlLab, indica um tipo que apresenta uma explicação sobre si mesmo. A vista do diagrama realiza a seleção do elemento cuja informação será mostrada ao usuário, e repassa-o para o editor de diagramas que a mostra ao mesmo.

6.1.3 Vista do Diagrama – DiagramView

Subclasse da classe concreta *StandardDrawingView* do framework JHotDraw que implementa a sua interface *DrawingView*, é responsável pela visualização do diagrama que está sob edição do respectivo editor, além de implementar algumas funcionalidades não definidas no JHotDraw.

As figuras de bloco de um diagrama podem ser representadas por componentes gráficos Java™, tais como campos de texto (*TextField*) e painéis (*Panels*). No entanto, o JHotDraw não foi projetado para conter tais elementos. A vista de um diagrama, desta forma, implementa um suporte para a edição destes componentes em conjunto

---

[1] Exemplo: O editor de diagrama utiliza o suporte de linguagem através da intermediação da aplicação, sem necessitar desta forma conhecer sua implementação.

com os desenhos próprios do JHotDraw. Este suporte inclui o tratamento de seus eventos, sua atualização, inserção e remoção na vista.

Antes de repassar os eventos de mouse para a ferramenta ativa, a vista realiza uma filtragem sobre os mesmos, de modo que os pontos de ação do mouse não sejam localizados fora da área de trabalho do diagrama sob edição, que corresponde à borda do diagrama e respectivo interior. A vista também verifica se o movimento do mouse não resultou na necessidade de sua rolagem.

O JHotDraw não apresenta o conceito de duplo clique, importante, por exemplo, para a abertura das janelas de configuração. A vista do diagrama apresenta o reconhecimento deste evento que é repassado para ferramentas que apresentem um tratamento para o mesmo. Tais ferramentas implementam a interface *DoubleClickListener*, definida no ControlLab.

A vista do diagrama gerencia o mecanismo de pilha de ações que é utilizado para desfazer e refazer ações do usuário na vista (ver seção 6.7.1 – Suporte para Desfazer/Refazer Ações)[2]. Além disso, a vista apresenta uma variável que indica o arquivo através do qual o diagrama sob edição foi carregado.

6.1.4 Suporte de Linguagem – Language Support

O suporte de linguagem, como afirmado anteriormente, permite a utilização do ControlLab em diferentes idiomas, através de uma representação baseada em tabelas. Estas tabelas são indexadas por chaves utilizadas no código e que apresentam diferentes valores para cada idioma implementado. Por exemplo, a chave *NameDiagram* apresenta o valor *Diagram* para o inglês e o valor *Diagrama* para o português.

A linguagem Java™ permite a implementação das tabelas de duas maneiras: classes ou arquivos, sendo esta última a escolhida por permitir que o usuário realize a extensão do suporte de linguagem para outros idiomas sem a necessidade de alterar o código da aplicação.

---

[2] Ver seção 6.7.1 – Suporte para Desfazer/Refazer Ações

São utilizados no ControlLab as seguintes tabelas:

- About – É aberta antes das outras tabelas, não servindo de suporte para os elementos da aplicação, mas como apoio para a janela *Sobre ControlLab* que é aberta durante a inicialização do aplicativo.
- Labels – Tabela que inclui todos os nomes dos elementos da interface do usuário, inclusive os menus.
- Messages – Tabela contendo as mensagens do ControlLab.

O formato especificado pela linguagem Java™ para a definição de tais tabelas dificulta a inserção de novos idiomas. Cada tabela apresenta um arquivo para cada novo idioma, além de um arquivo com valores default. Neste caso a chave *NameDiagram* seria representada pela entradas *NameDiagram = diagram* e *NameDiagram = diagrama* para os arquivos dos idiomas inglês e português respectivamente, além de uma entrada no arquivo default.

A alternativa adotada foi a implementação de arquivos que servem de fonte para a geração dos arquivos de suporte de linguagem. Nestes arquivos os valores de uma chave para todos os idiomas estão agrupados, facilitando a edição e a utilização de valores previamente definidos para a extensão do suporte de linguagem. O caso anterior seria então representado da seguinte forma:

```
[NameDiagram]
Diagram
Diagrama
```

A conversão dos arquivos fonte é realizada pela classe *LSGenerator* que gera os arquivos Java™ automaticamente e informa eventuais erros ao usuário.

A classe responsável por estas tabelas durante a execução do ControlLab é a classe *LanguageSupport* que é inicializada em função do idioma indicado pela linguagem Java™[3]. Esta classe dispõe de métodos de acesso para as tabelas *Labels* e *Messages* com ou sem a passagem de parâmetros, que podem ser utilizados para definir variáveis de um texto em um idioma, tais como, por exemplo, o nome da figura de bloco a ser criada em uma ferramenta *Criar Figura de Bloco*.

Este suporte é utilizado em várias classes do ControlLab. Para evitar que a classe *LanguageSupport* seja excessivamente referenciada, a única classe que pode acessá-la é a classe *ControlLab*, que serve de intermediador para as outras classes do software.

6.1.5 Configuração do Usuário – UserSetup

O módulo do ControlLab denominado configuração do usuário, cuja estrutura básica pode ser observada na figura 6.3, é responsável por organizar os elementos da aplicação nos quais existe a possibilidade de alteração por parte do usuário:

- Conjunto de atributos de referência do ControlLab, tais como, por exemplo, o nome do diretório de ícones.
- Opções do usuário com relação à área de trabalho.
- Biblioteca de figuras de blocos.
- Biblioteca de resolvedores da ordem de execução de blocos compostos (*DataFlowSolvers*).
- Biblioteca de agentes executores de diagramas de blocos (*DiagramEngines*).

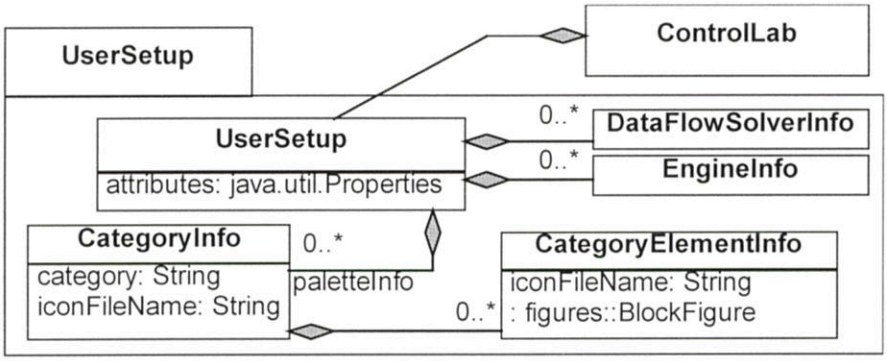


Figura 6.3 - Configuração do Usuário

Como pode ser observado no diagrama, o módulo de configuração do usuário é composto pelas seguintes classes:

---

[3] Através do método *getDefault()* da classe *java.util.Locale* que é responsável por indicar as características de uma localidade, tais como o idioma.

- *UserSetup* – É composto pelos elementos da configuração do usuário, sendo responsável pelo seu salvamento e leitura em disco, através do formato JHotDraw *Storable*.

- *CategoryInfo* – Armazena as informações sobre uma categoria da paleta de ferramentas. Todas as categorias com exceção da categoria *Ferramentas Básicas* são representadas neste formato. É composta por um nome, o nome do arquivo do ícone que a representa e por um conjunto de elementos de categoria.

- *CategoryElementInfo* – Armazena as informações necessárias para a definição de uma instância da ferramenta *Criar Figura de Blocos* de uma categoria da paleta de ferramentas: o nome do arquivo do ícone que a representa e a figura de bloco que serve de protótipo para a ferramenta.

- *DataFlowSolverInfo* – Armazena um elemento para a definição da ordem de execução de um bloco composto do Framework de Controle de Processos. A versão atual utiliza como valor default a classe *AutoSortSolver* fornecida por aquele framework.

- *EngineInfo* – Armazena um elemento da classe *EngineManager* (ver seção 6.5 – Extensão de Controle).

A organização deste módulo realizou o encapsulamento de elementos da aplicação projetados para poderem ser alterados pelo usuário, evitando que se criem dependências com os respectivos mecanismos de edição.

Os mecanismos de edição das opções da área de trabalho e da biblioteca de figuras de bloco, que podem ser observados nas seções 5.1.5 e 5.4 respectivamente, são evocados a partir da classe ControlLab. Os outros ainda não foram implementados, utilizando valores default do ControlLab.

6.1.6 Interface com o Sistema – IO

Este módulo, cuja estrutura básica pode ser observada no esquema da figura 6.4, reúne as classes responsáveis pela interface do ControlLab com o sistema. As classes deste módulo implementam o padrão de projeto *Singleton* ([Gamma et. al.,

1994:127-134]), de modo que apresentam uma única instância que é utilizada por todas as janelas abertas do ControlLab.

Como uma base para as mesmas foi implementada a classe *FileNameBrowser* cujo objetivo é armazenar o contexto em que os arquivos são escritos ou lidos, o que inclui a extensão de arquivos envolvida e o último diretório visitado.

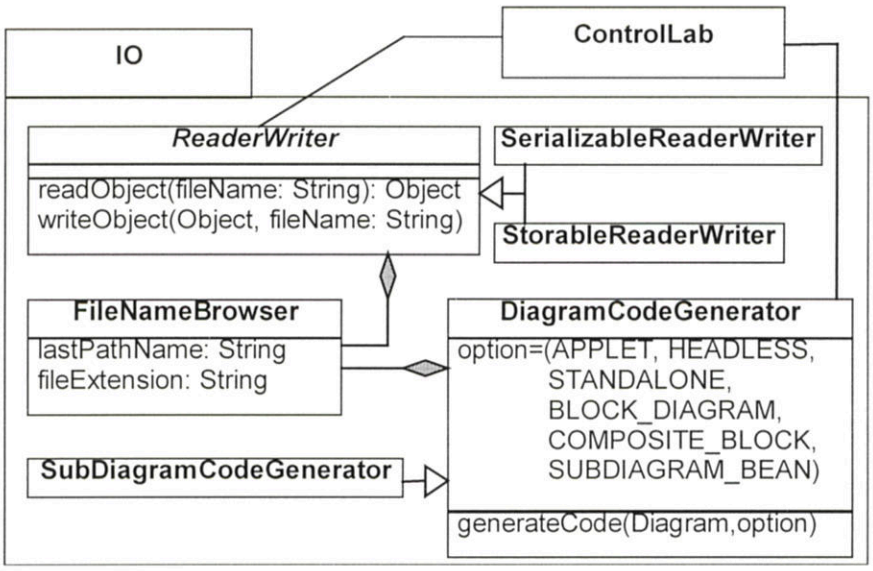


Figura 6.4 - IO

O módulo pode ser dividido em duas partes: salvamento e recuperação de objetos e geração de código. O salvamento e leitura de objetos é evocado através de métodos da classe ControlLab, enquanto que a geração de código é realizada com o auxílio de comandos.

6.1.6.1 Salvamento e Recuperação de Objetos

O salvamento e recuperação de objetos é realizado pela classe abstrata *ReaderWriter* que utiliza o padrão de projeto *Strategy* ([Gamma et. al., 1994:315-323]) para a definição de algoritmos de serialização de objetos, os quais são implementados pelas subclasses *StorableReaderWriter e SerializableReaderWriter* que utilizam os modelos *Storable* e *Serializable*, respectivamente:

- *Storable* – Modelo definido pelo framework JHotDraw que realiza a serialização dos tipos básicos da linguagem Java™, de instâncias das classes

String e *Color*, e de objetos que implementem a interface *Storable*. Estes últimos devem implementar métodos que descrevam sua escrita e leitura.

- *Serializable* – Modelo definido pela linguagem Java™ que realiza a serialização automática de objetos que implementem a interface *java.io.Serializable*[4].

Como afirmado no capítulo 5, o modelo *Storable* é mais adequado por permitir a compatibilidade com a leitura com versões mais antigas de objetos de uma classe. Isto ocorre através do condicionamento dos métodos de leitura à versão de arquivo a ser lida. Para indicar qual é a versão pela qual um arquivo do ControlLab foi gerado, estes arquivos apresentam como primeiro valor a versão do ControlLab em que foram gerados.

6.1.6.2 Geração de Código

A geração de código é executada pela classe *DiagramCodeGenerator* com auxílio da classe *SubDiagramCodeGenerator*. Quando evoca-se a exportação de um diagrama, diagrama principal ou subdiagrama, na forma de código, deve-se fornecer a opção através da qual deseja-se realizar a tarefa:

- APPLET – Diagrama principal como Applet.
- HEADLESS – Diagrama principal como aplicação sem janela.
- STANDALONE – Diagrama principal como aplicação com janela.
- BLOCK_DIAGRAM – Diagrama principal como subclasse de *BlockDiagram*.
- COMPOSITE_BLOCK – Subdiagrama como subclasse de *CompositeBlock*.
- SUBDIAGRAM_BEAN – Subdiagrama como componente JavaBeans™.

Através da utilização do modelo JavaBeans™ (ver seção 6.3.2.1 – Informações do Bloco), o código dos blocos que constituem o diagrama a ser exportado é gerado

[4] Esta interface não apresenta métodos, apenas indica que uma classe foi projetada para a serialização segundo o método *Serializable*.

de modo que suas propriedades sejam as mesmas com as quais os blocos foram definidos no diagrama.

6.2 DIAGRAMA

Um diagrama é uma instância da classe *Diagram*, cuja estrutura básica pode ser observada na figura 6.5. Esta classe é subclasse da classe concreta *StandardDrawing* do framework JHotDraw que implementa a sua interface *Drawing*. Além de representar o elemento de edição de um editor de diagramas e de gerenciar um conjunto de figuras, responsabilidades herdadas do JHotDraw, um diagrama também é responsável por gerenciar a semântica de controle inerente à um diagrama de blocos e fornecer um suporte aos conceitos de subdiagrama e grupo.

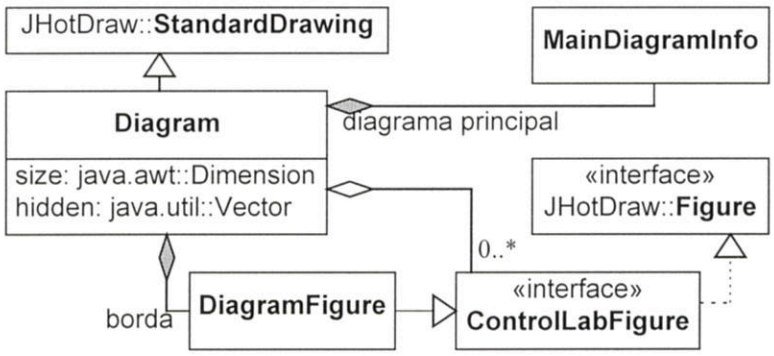


Figura 6.5 - Diagrama

Como afirmado no capítulo anterior, um diagrama apresenta elementos gráficos, as suas figuras e a borda, e elementos não gráficos, que definem a sua configuração. A borda é representa por uma figura especial, denominada *DiagramFigure*[5].

Dentre os elementos não gráficos, apenas o tamanho é uma característica particular de cada diagrama, enquanto que as demais são específicas do diagrama principal. Além do tamanho, cada diagrama realiza o gerenciamento das figuras ocultas. Tais figuras são retiradas do conjunto de figuras do diagrama e armazenadas

[5] As figuras (ControlLabFigures) serão descritas na seção seguinte.

em um conjunto auxiliar, denominado *hidden*. Ao serem mostradas novamente, as mesmas são reintegradas ao conjunto original.

Um diagrama executa o papel de subdiagrama ou diagrama principal. No entanto, são fornecidos métodos para a transformação de um diagrama principal em subdiagrama e vice-versa, utilizados em várias rotinas de exportação de diagramas.

6.2.1 Diagrama Principal

Um diagrama principal se diferencia de um subdiagrama por apresentar um elemento adicional, da classe *MainDiagramInfo*, que fornece ao mesmo as informações inerentes de um diagrama principal. A estrutura básica deste elemento pode ser observada na figura 6.6.

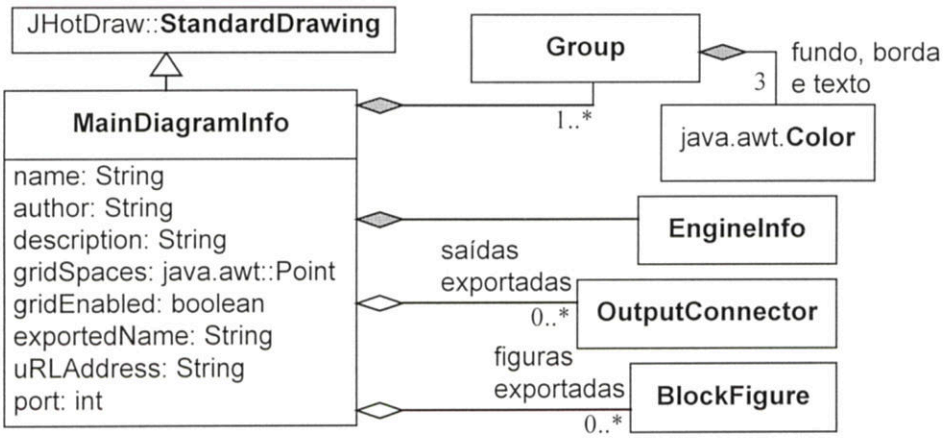


Figura 6.6 - Informações do Diagrama Principal - *MainDiagramInfo*

Compõem as informações do diagrama principal, os seguintes elementos:

- Informações gerais – Nome, autor e descrição.
- Informações da grade – Espaçamentos e indicação de se a mesma está sendo utilizada.
- Identificação Remota do Diagrama – Nome do diagrama, endereço e porta da máquina a serem utilizados na disponibilização do diagrama para acesso remoto.
- Elemento de execução do diagrama, representado por uma instância da classe *EngineInfo*.

- Grupos que formam o diagrama.
- Figuras de bloco exportadas.
- Saídas de figuras de bloco exportadas.

Além de manter informações válidas para si e seus subdiagramas e de ser a raiz de uma estrutura de diagramas, como mostrado na figura 5.5, o diagrama principal também é responsável por iniciar e parar a simulação, através de seu elemento de execução (ver seção 6.5 – Extensão de Controle). Nesta tarefa, quando da necessidade de dispor o diagrama para acesso remoto, deve ser verificada a existência de elementos exportados.

O diagrama principal também apresenta uma figura de bloco que representa o relógio do respectivo *BlockDiagram* e que não pode ser eliminada do diagrama, pois é essencial ao mesmo.

6.3 FIGURA – CONTROLLABFIGURE

No framework JHotDraw uma figura é uma classe que implementa a interface *Figure*. No ControlLab esta interface foi estendida pela interface *ControlLabFigure*, que também estende a interface *Explainable* e que é implementada por todas as figuras utilizadas no ControlLab, como pode ser visto no diagrama da figura 6.7.

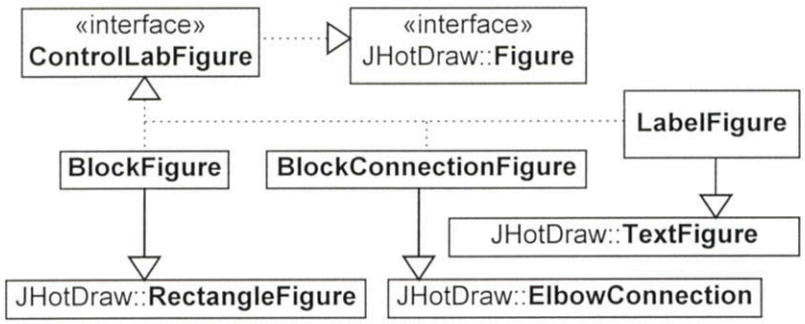


Figura 6.7 - ControlLabFigure

O principal objetivo desta interface é definir o comportamento requerido por uma figura para que a mesma seja utilizada no ControlLab. Uma solução alternativa seria a criação de uma hierarquia de classes cuja raiz fosse uma classe abstrata equivalente à interface ControlLabFigure. Neste caso, no entanto, todas as classes

deveriam pertencer a tal hierarquia, o que não ocorre na solução adotada, que permite a efetiva utilização do polimorfismo.

Dentre os métodos definidos por esta interface, pode-se destacar métodos:

- De identificação do tipo da figura, se é uma conexão, uma figura de bloco ou um rótulo.
- De definição do grupo da figura.
- Para limitação da área da figura de modo que ela não extrapole a área de trabalho de seu diagrama.
- Para definição de se determinada área de um diagrama contém a figura. É utilizado para a seleção de figuras.
- De indicação do início e final da sua edição (movimentação ou alteração do tamanho).
- De indicação e execução do ocultamento da figura, utilizado em conjunto com o ocultamento de figuras de um diagrama.

6.3.1 Rótulo

Uma figura de rótulo é uma instância da classe *LabelFigure* que estende a classe *TextFigure* do framework JHotDraw. Esta classe não adiciona nenhuma semântica a sua superclasse, apenas implementando o conhecimento requerido pela interface *ControlLabFigure*.

6.3.2 Figura de Bloco

Uma figura de bloco é uma instância da classe *BlockFigure* que estende a classe *RectangleFigure* do framework JHotDraw. Esta última, como o nome indica, fornece o comportamento para figuras com o formato retangular.

Uma figura de bloco é composta pelas informações do seu bloco (*BlockInfo*), por um gerenciador de conectores (*ConnectorsManager*) e pelo rosto *(Foreground)*, que são descritos nesta seção.

Além de implementar uma casca sobre estes elementos e de implementar o comportamento definido pela interface *ControlLabFigure*, a classe *BlockFigure* ainda gerencia, utilizando o mecanismo de notificação do JHotDraw para figuras de conexão, a conexão dos blocos do Framework de Controle de Processos. Estas conexões são realizadas em tempo de edição, permitindo que o diagrama de controle esteja pronto para ser simulado.

Os manipuladores de uma figura de bloco são adicionados em tempo de edição por uma classe denominada *BlockFigureHandleKit* que, em função das características da figura de bloco adiciona automaticamente manipuladores de movimentação (classe *MoveHandle*) e/ou de alteração do tamanho (classe *SizeHandle*).

6.3.2.1 Informações do Bloco – BlockInfo

Classe que representa um bloco de controle para o ControlLab, utilizando o modelo JavaBeans™ de componentes da linguagem Java™ para realizar o reconhecimento e a alteração de suas propriedades.

Com o objetivo de definir componentes de linguagem, o modelo JavaBeans™ permite indicar, para determinada classe, quais métodos, propriedades e eventos estão a disposição das outras classes. No caso dos blocos de controle, os métodos são definidos na classe *Block* do Framework de Controle de Processos e não é utilizado o conceito de eventos. Assim, o modelo torna-se especialmente útil na pesquisa das propriedades dos blocos de controle.

O modelo JavaBeans™ utiliza o conceito de métodos *setter* e *getter* para, respectivamente, alterar e ler as propriedades de um objeto, definindo um conjunto de padrões que deve ser seguido para a nomeação destes métodos. Então, através de um mecanismo automático, denominado *Introspeção*, é gerado um objeto, denominado *BeanInfo*, que contém as informações sobre o objeto pesquisado. O objeto *BeanInfo* também pode ser definido explicitamente pelo projetista da classe, mas esta opção obrigaria a realização do projeto de uma classe adicional para cada classe de bloco de controle.

O mecanismo também suporta o conceito de propriedades indexadas, como por exemplo os sinais de entrada do bloco somador que são definidos por uma propriedade indexada denominada *Sign*.

A classe *BlockInfo*, através do uso do mecanismo de introspeção reconhece automaticamente as propriedades dos blocos de controle. Deste modo, quaisquer blocos de controle que sigam o modelo JavaBeans™ para a definição dos *getters* e *setters* de suas propriedades, terão as mesmas automaticamente reconhecidas pelo ControlLab.

A limitação imposta pelo ControlLab para a definição das propriedades dos blocos de controle deve-se aos mecanismos de edição disponíveis. A versão atual permite a edição de propriedades representadas por objetos das classes *String*, *Boolean*, *Float* e *Integer*, além dos tipos básicos *boolean*, *float* e *int*.

Uma informação de bloco é requerida na criação de uma instância de uma figura de bloco, que inicializa o gerenciador de conectores em função do bloco de controle respectivo.

6.3.2.2 Gerenciador de Conectores – ConnectorsManager

As entradas e saídas dos blocos do Framework de Controle de Processos, que podem ser chamadas de conectores, não apresentam classes correspondentes no framework, em função do seu compromisso com a simplicidade. No ControlLab, no entanto, até por influência do framework gráfico JHotDraw, que apresenta suporte para conectores, as entradas e saídas das figuras de bloco apresentam classes próprias, denominadas, respectivamente, de *InputConnector* e *OutputConnector*, subclasses da classe abstrata *BlockFigureConnector*.

Um conector implementa a interface *Explainable*, sendo responsável por mostrar-se na vista do diagrama. Também indica se já está conectado ou não, informação importante para conectores de entrada que não podem ser conectados a duas saídas. Além disso apresenta:

- Um manipulador, das classes *InputHandle* ou *OutputHandle*, respectivamente para entradas e saídas, que são subclasses de

ConnectorHandle e possibilitam a realização de conexões por manipulação direta.

- Um localizador, das classes *InputLocator* ou *OutputLocator*, respectivamente para entradas e saídas, que são subclasses de *ConnectorLocator*, que conhecem a localização (lado e posição) do conector na respectiva figura de bloco. Sua função é localizá-lo na vista, em função de sua figura de bloco e de seu posicionamento na mesma.

Uma figura de blocos pode conter vários conectores, apresentando um gerenciador de conectores, instância da classe *ConnectorsManager*, que os administra. Este gerenciador fornece métodos para a adição e remoção de conectores de entrada ou saída em qualquer um dos lados de uma figura de bloco. O esquema básico do gerenciador de conectores e dos conectores pode ser observado na figura 6.8.

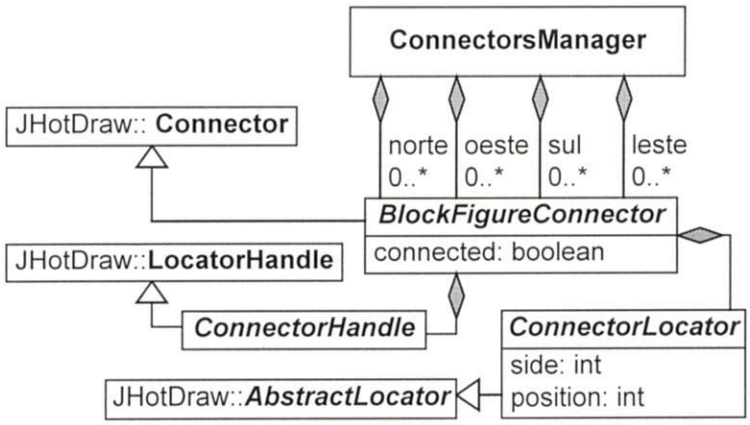


Figura 6.8 - Gerenciador de Conectores

A figura de bloco, por sua vez, define que lados estão disponíveis para as entradas e que lados estão disponíveis para as saídas, informações utilizadas para a intermediação da inserção e remoção de conectores no respectivo gerenciador.

6.3.2.3 Rosto - Foreground

O *Foreground* de uma figura de bloco representa a sua aparência na vista do diagrama para o usuário. Esta aparência é complementada pelo contorno da figura de bloco e pela aparência dos seus conectores e manipuladores, estes últimos quando a figura está selecionada.

Um *Foreground* é um objeto que implementa a interface de mesmo nome. Esta interface inclui métodos para a definição de tamanho e das cores de frente, fundo e do texto, além de métodos para o seu ocultamento. Em complemento à interface é implementada uma hierarquia de classes, que pode ser observada na figura 6.9, cuja raiz é a classe abstrata *AbstractForeground* que implementa a interface *Foreground*. Esta hierarquia inclui *Foregrounds* baseados em componentes gráficos Java™, em especial campos de texto (*TextFields*) e gráficos do Framework de Controle de Processos (*Plotter*), em textos normais e em imagens. Também é fornecido um *Foreground* nulo (*NullForeground*), ou seja, que não apresenta aparência.

O usuário de uma figura de blocos deve informar, no momento da criação da mesma, qual objeto irá representar a sua aparência. A classe *BlockFigure* repassa este objeto para a classe *AbstractForeground* que retorna o *Foreground* adequado para a figura de bloco. A versão atual do ControlLab não disponibiliza mecanismos ao usuário para a definição de imagens como objeto de aparência de uma figura de bloco, apesar de ser uma funcionalidade já incorporada às figuras de bloco.

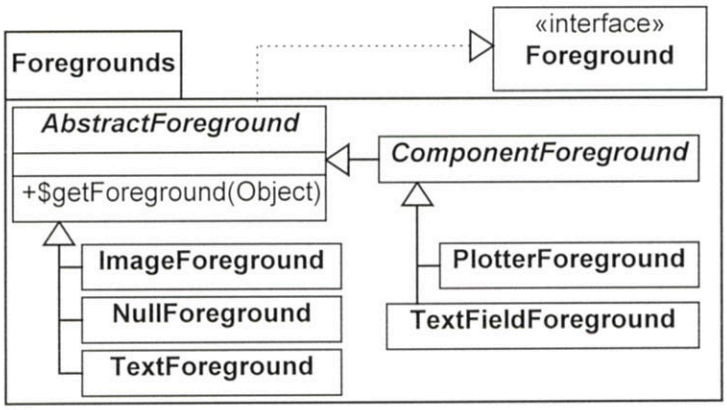


Figura 6.9 - Hierarquia de *Foregrounds*

6.3.2.4 Hierarquia de Figuras de Bloco

A maioria dos blocos de controle é representado no ControlLab diretamente por instâncias de *BlockFigure*. No entanto, alguns blocos especiais necessitam de um comportamento específico da respectiva figura de bloco, resultando na necessidade da implementação de uma hierarquia de classes sob *BlockFigure* que forneça tais comportamentos. Esta hierarquia pode ser observada na figura 6.10.

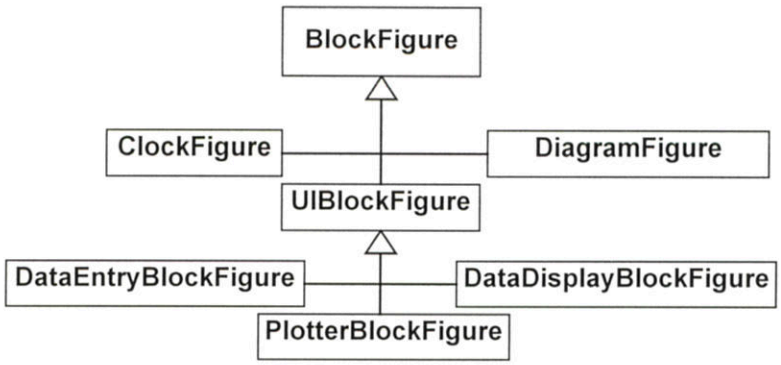


Figura 6.10 - Hierarquia de Figuras de Bloco

- *ClockFigure* – Para representar o relógio de um diagrama de blocos em um diagrama principal. Diferencia-se da classe *BlockFigure* por indicar que não pode ser eliminada de um diagrama, pois um diagrama de blocos sempre apresenta um relógio.

- *DiagramFigure* – Figura de bloco muito importante que representa a borda de um diagrama e está relacionado com um bloco de controle composto (*CompositeBlock*), que é um diagrama de blocos (*BlockDiagram*) quando seu diagrama é um diagrama principal. Quando seu diagrama é um subdiagrama e este está fechado, o representa como uma figura de blocos normal no diagrama que o contém. Apresenta dois gerenciadores de conectores, um herdado da classe *BlockFigure* que gerencia os conectores externos, como uma figura de bloco comum, e outro responsável por gerenciar os conectores internos que são visíveis quando a figura está na forma da borda de um diagrama. Também apresenta a definição dos nomes *Bean* das entradas e saídas que são utilizados para exportar um subdiagrama como um componente JavaBeans™. Apresenta uma referência para o diagrama que a contém (é nula para diagramas principais) que serve para realizar seu fechamento. Sua estrutura básica pode ser vista na figura 6.11.

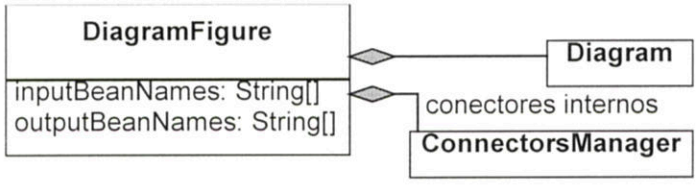


Figura 6.11 - Figura de Diagrama - *DiagramFigure*

- *UIBlockFigure* – Figura de bloco responsável por representar blocos que contenham componentes gráficos da linguagem Java™.
- *DataEntryBlockFigure* – Figura de bloco que representa um bloco da classe *DataEntryBlock*, denominado no ControlLab de bloco de entrada de dados.
- *DataDisplayBlockFigure* – Figura de bloco que representa um bloco da classe *DataDisplayBlock*, denominado no ControlLab de bloco de visualização de dados.
- *PlotterBlockFigure* – Figura de bloco que representa um bloco da classe *PlotterBlock*, denominado no ControlLab de gráfico.

6.3.3 Figura de Conexão

Uma figura de conexão é uma instância da classe *BlockConnectionFigure* que estende a classe *ElbowConnection* do framework JHotDraw. Esta última fornece o comportamento para conexões representadas por múltiplos segmentos.

Como pode ser observado no diagrama da figura 6.12, uma figura de conexão é formada basicamente por um conector de entrada e um conector de saída e apresenta um conjunto de pontos que define sua topologia no diagrama.

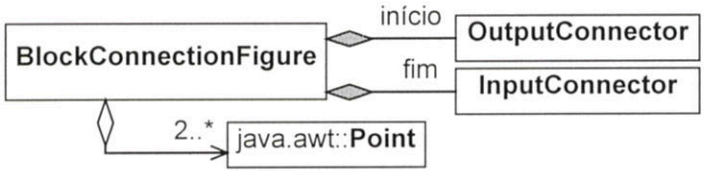


Figura 6.12 - Figura de Conexão

Além de definir o comportamento requerido pela interface *ControlLabFigure*, uma figura de conexão é responsável por evocar os métodos de conexão e desconexão das figuras de bloco, quando da sua inserção ou remoção do diagrama.

A classe *BlockConnectionFigure* também apresenta um algoritmo para a definição de seus pontos intermediários, que automaticamente procura a melhor topologia em função do posicionamento das respectivas figuras de bloco. Esta

características pode ser observada através da manipulação de diagramas do ControlLab.

Em complemento a este algoritmo estão os manipuladores de modificação dos segmentos intermediários, que podem ser observados na cor amarela na figura 5.16. Uma figura de conexão torna-se dependente das figuras de bloco de suas extremidades, de modo que é informada, através do mecanismo de notificação do JHotDraw, de qualquer modificação no tamanho ou posicionamento destas figuras. Assim, seus pontos extremos são atualizados automaticamente em função de tais figuras.

Os manipuladores de suas extremidades, em verde, como também pode ser observado na figura 5.16, permitem a alteração da respectiva extremidade. Isto é realizado através da evocação de uma instância da ferramenta de conexão de blocos, que, assim como cria novas conexões, também é responsável por reconectar conexões já existentes.

6.4 Interface Gráfica com o Usuário

Os componentes de interface gráfica com o usuário do ControlLab foram separados em módulos próprios. Assim, muitos dos elementos que compõem a aplicação apresentam mecanismos de edição independentes, melhorando a modularidade do projeto.

Estes componentes podem ser classificados nas seguintes categorias:

- Editor.
- Janelas auxiliares (*Dialogs*).
- Componentes auxiliares.

6.4.1 Editor

Como afirmado anteriormente, o editor de diagramas foi projetado para poder ser utilizado na forma de *Applet* ou de aplicação normal, sendo apenas esta última implementada na versão atual do ControlLab.

O editor implementado é representado pela classe *StandAloneEditor*, cuja esquema básico pode ser observado na figura 6.13. Esta classe além de implementar o comportamento requerido pela interface *DiagramEditor*, organiza os componentes gráficos da janela do editor: paleta de ferramentas, menu de comandos, linha de status e vista do diagrama.

A vista do diagrama é fornecida por seu intermediador, a respectiva instância da classe *ControlLab*, apresentando um relacionamento com o editor já definido pelo framework JHotDraw.

A linha de status é utilizada para a apresentação do texto explicativo do objeto do tipo *Explainable* indicado pela vista do diagrama.

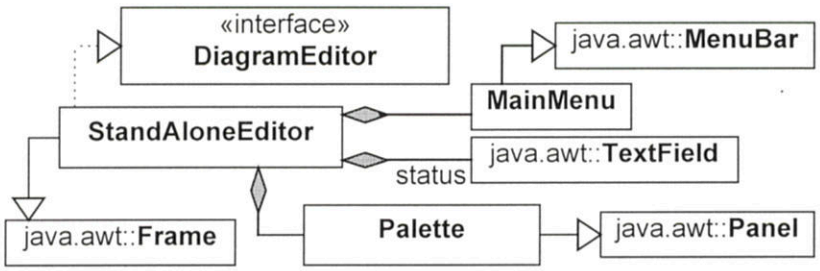


Figura 6.13 - *StandAloneEditor*

O menu de comandos é representado por uma classe próprias, denominada *MainMenu*, cuja responsabilidade é definir o menu de comandos do ControlLab, incluindo a inicialização dos respectivos comandos. Também são incluídos menus normais da linguagem Java™ em ações mais simples, como a abertura de um diagrama, que apresenta um método específico na classe *ControlLab*.

6.4.1.1 Paleta de Ferramentas

A paleta de ferramentas é um componente importante para o ControlLab, por fornecer um mecanismo dinâmico de seleção de ferramentas que complementa a manipulação direta da vista do diagrama. Foi projetada para ser utilizada tanto no editor na forma de *Applet* como no editor na forma de aplicação convencional.

Como afirmado anteriormente, o JHotDraw dispõe de uma paleta de ferramentas simples que não permite a adequada classificação de uma quantidade

maior de ferramentas, como pode ocorrer no ControlLab. Por esta razão, foi implementada uma paleta de ferramentas dupla, permitindo a definição de uma paleta para categorias de ferramentas e outra para as ferramentas da categorias selecionada. Esta paleta é representada pela classe *Palette*, cujo esquema básico pode ser observado no diagrama da figura 6.14.

A paleta é formada por duas paletas de botões, da classe *ButtonsPalette*, que realizam o gerenciamento do leiaute de seus botões, respondendo inclusive à necessidade da inclusão de uma barra de rolagem, quando a paleta for maior que o tamanho fornecido pelo editor.

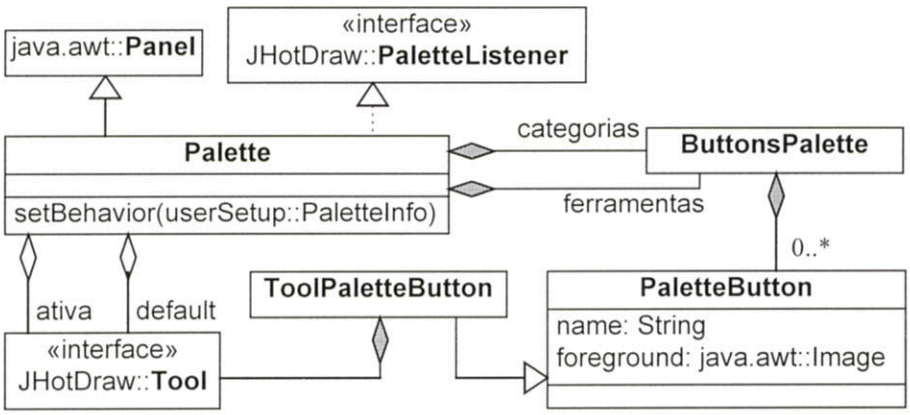


Figura 6.14 - Paleta de Ferramentas - *Palette*

Durante a inicialização da paleta de ferramentas, a mesma cria todas as suas paletas de botões, inclusive as de ferramentas de categorias não selecionadas, de modo que a troca de categorias é realizada rapidamente.

Os botões, do tipo *PaletteButton*, são criados com uma imagem que será o seu rosto (*foreground*), um nome que indica a sua finalidade e pela própria paleta de ferramentas. Esta recebe eventos dos botões, indicando a sua seleção e a movimentação do mouse sobre os mesmos. Estes eventos influenciam na mensagem mostrada na linha de status, pois a paleta de ferramentas também implementa a interface *Explainable*.

A classe *ToolPaletteButton*, subclasse de *PaletteButton*, é utilizada para a composição das paletas de ferramentas das categorias, sendo suas instâncias

compostas ainda por uma ferramenta que é automaticamente evocada quando da seleção do botão.

A paleta de ferramentas apresenta uma ferramenta default, da classe *DefaultTool*, descrita na seção 5.3.1.1, e a ferramenta ativa, que corresponde à ferramenta do botão da classe *ToolPaletteButton* selecionado.

6.4.2 JANELAS AUXILIARES

Foi implementado no ControlLab uma classe abstrata, denominada *ControlLabDialog*, que serve de base para a implementação das janelas auxiliares do editor de diagramas do ControlLab. Esta classe apresenta uma referência para uma instância da classe que *ControlLab* que permite o acesso aos elementos da aplicação, em especial o suporte de linguagem.

A classe *ControlLabDialog* é formada por um painel de botões e um painel principal que são criados utilizando o padrão de projeto *FactoryMethod* ([Gamma et. al., 1994:110-116]). Este padrão de projeto também é utilizado na definição dos espaçamento dos painéis principal e de botões. Deste modo, tais elementos são definidos pelas suas subclasses em função de suas necessidades específicas, podendo, no entanto, serem utilizados na classe abstrata para a implementação do conhecimento comum a todas as subclasses.

Este conhecimento comum inclui as rotinas de abertura e fechamento das janelas e a definição de uma aparência consistente para todas as janelas de auxílio do ControlLab. Caso seja definido, por exemplo, um logotipo para o ControlLab, é possível inserir um painel com o mesmo em todas as suas janelas apenas modificando a classe do editor de diagramas e a classe abstrata *ControlLabDialog*, cuja hierarquia de janelas é apresentada na figura 6.15.

As janelas representadas por classes desta hierarquia apresentam as seguintes funções:

- *About* – Fornece algumas informações sobre o ControlLab: sua versão, entidades e pessoas envolvidas e forma de contato para a retirada de dúvidas e fornecimento de sugestões.

- *BlockFigurePropertyEditor* – Realiza a edição das propriedades de uma figura de bloco. Um exemplo pode ser observado na figura 5.18.
- *ChoiceDialog* – Permite ao usuário selecionar uma opção dentre um conjunto de opções.
- *ConfigureSimulationDialog* – Realiza a configuração da simulação de um diagrama. Pode ser observada na figura 5.21.
- *DiagramSizeDialog* – Altera o tamanho do diagrama sob edição do editor de diagramas.
- *GridDialog* – Altera as informações da grade do diagrama.
- *LabelFigurePropertyEditor* – Realiza a edição das propriedades de um rótulo. Um exemplo pode ser observado na figura 5.15.

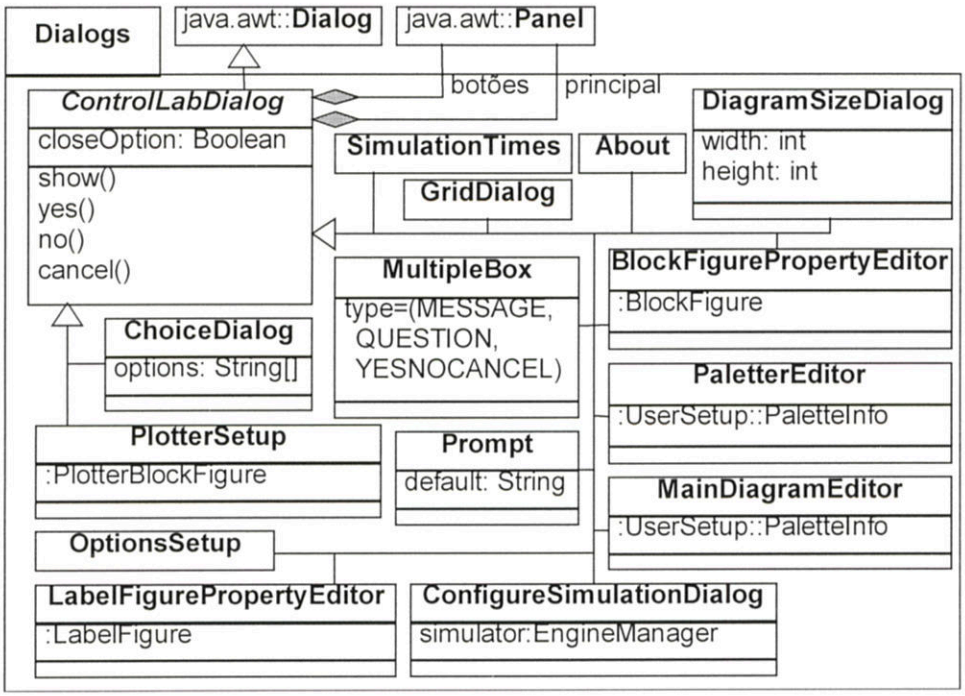


Figura 6.15 - Janelas Auxiliares

- *MainDiagramEditor* – Realiza a edição das informações relativas ao diagrama principal: informações gerais, definição de grupos e informações remotas, que podem ser observadas respectivamente nas figuras 5.4, 5.8 e 5.20.

- *MultipleBox* – Permite a passagem de informações ao usuário, através de três formas: como uma simples informação, como um questionamento na forma *Sim/Não* ou um questionamento na forma *Sim/Não/Cancelar*. É utilizado pela classe *ControlLab* que dispõe de métodos para a utilização destas janelas.
- *OptionsSetup* – Permite a alteração de opções da área de trabalho, podendo ser observada na figura 5.2.
- *PaletteEditor* – Realiza a edição das categorias e ferramentas da paleta de ferramentas, podendo ser observada nas figuras 5.12 e 5.13.
- *PlotterSetup* – Realiza a edição das propriedades de um gráfico. Um exemplo pode ser observado na figura 5.19.
- *Prompt* – Para a entrada de uma informação na forma de texto por parte do usuário.
- *SimulationTimes* – Mostra os tempos de simulação ao usuário como pode ser observado na figura 5.22.

6.4.3 Componentes Auxiliares

Em complemento às janelas projetadas para o ControlLab foram criados alguns componentes gráficos que fornecem serviços especiais às mesmas:

- *DecoratorPanel* – Painel especial que implementa uma forma simplificada do padrão de projeto *Decorator* ([Gamma et. al., 1994:175-184]). Fornece opções para utilização de diferentes bordas, leiautes e margens nos elementos decorados.
- *PagePanel* e *NotebookPanel* – Permitem a utilização de diferentes painéis principais em uma mesma janela auxiliar do ControlLab. Foram utilizados nas janelas de configuração do diagrama principal e no editor da paleta de ferramentas. Um *NotebookPanel* realiza o gerenciamento de diferentes páginas, da classe *PagePanel*. Seu relacionamento pode ser observado no diagrama da figura 6.16.

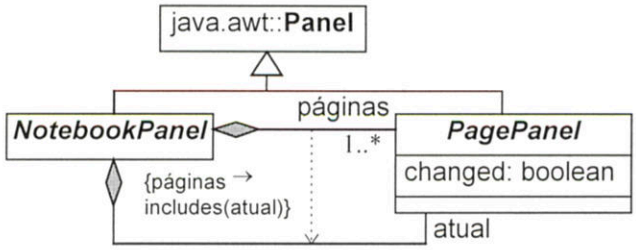


Figura 6.16 - *NotebookPanel* e *PagePanel*

- *ImageBrowser* – Permite a seleção de imagens baseados em uma tabela onde as chaves são os nomes e os valores são as imagens propriamente ditas. É utilizada na seleção dos ícones da paleta de ferramentas.

- *ColorBrowser* – Permite a seleção de cores baseados em uma tabela fornecida pela classe *ControlLab* onde as chaves são os nomes e os valores são as cores propriamente ditas.

- *NumericTextField* – Campo de entrada de textos especializado na entrada de números. Realiza a edição de números dos diferentes tipos da linguagem Java™, dependendo do modo como tenha sido inicializado. Permite, ainda, a definição dos limites inferior e superior.

- *PropertyEditorPanel* – Painel para a edição de uma propriedade de uma classe. É inicializado através do nome da propriedade e do seu valor atual. No caso de propriedades indexadas, este valor é representado por um *Array* da linguagem Java™. Sua estrutura básica pode ser observada no diagrama da figura 6.17. As janelas de edição de propriedades das figuras de bloco e dos rótulos são baseadas em objetos desta classe.

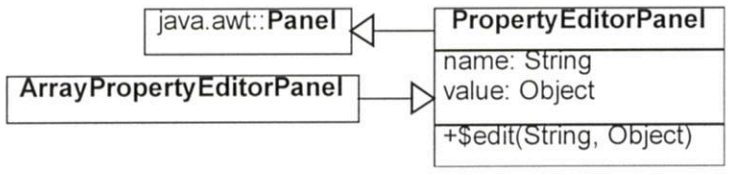


Figura 6.17 - *PropertyEditorPanel*

6.5 Extensão de Controle

O ControlLab utilizou o Framework de Controle de Processos como seu subsistema de controle. Como mecanismo de adaptação a este subsistema se encontram as figuras de bloco e de conexão que representam os blocos de controle e suas conexões, o que inclui, inclusive, um diagrama de blocos, que é representado por uma figura de diagrama (*DiagramFigure*).

Além destas figuras, existem as classe da configuração do usuário que foram projetadas para incluir, além de figuras de bloco, resolvedores de malha (*DataFlowSolvers*) e agentes de execução externa (*Engines*). Assim, todos os elementos que formam a estrutura básica do Framework de Controle de Processos possuem uma representação no ControlLab, embora estes dois últimos ainda não disponham de mecanismos de edição por parte do usuário.

No caso do agente de execução externa, existe uma classe intermediária que adapta ao mesmo a relação entre o tempo simulado e o tempo real definida pelo usuário. Além disso, esta classe, denominada *EngineManager*, fornece os tempos real e simulado da simulação para a vista do diagrama, de modo que eles sejam eventualmente mostrados ao usuário.

O ControlLab apresenta um agente de simulação especial, denominado *DiagramEngine*, que inclui a possibilidade da execução de simulações com tempo finito. Uma instância desta classe é executada através de um processo independente do processo de execução do ControlLab. Este processo apresenta a prioridade máxima que o protege da ação dos outros processos. Seus tempos de execução são regulados através de uma temporização ajustada em função do último tempo de execução e do período desejado. O algoritmo que implementa este comportamento é representado no diagrama de estados da figura 6.18.

Quando um diagrama é exportado do ControlLab, passa a ser utilizado o agente de execução do Framework de Controle de Processos, representado pela classe *ThreadedEngine*, que não apresenta suporte para a execução finita, devendo ser parado por um mecanismo externo. No entanto, esta substituição impede que o

usuário do ControlLab seja obrigado a utilizar componentes do ControlLab fora do ambiente.

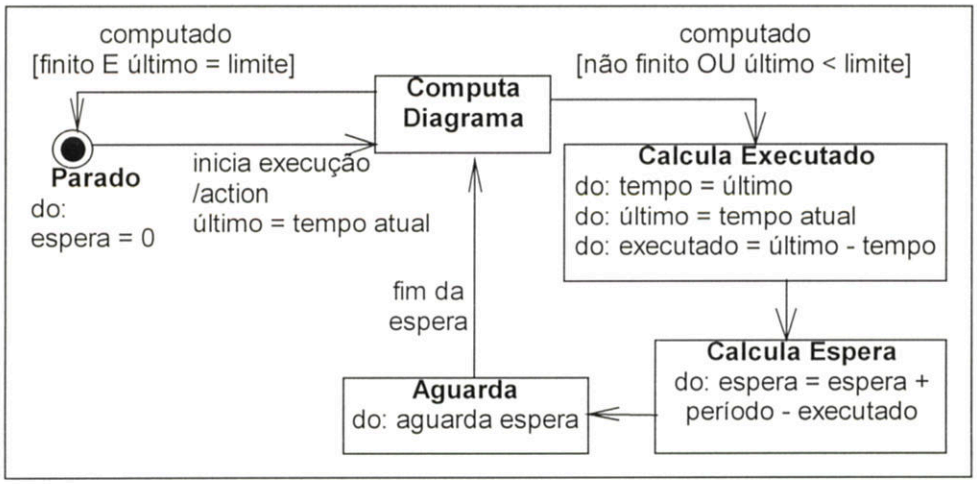


Figura 6.18 - Execução do *DiagramEngine*

6.6 FERRAMENTAS

Segundo o framework JHotDraw, após uma ferramenta ser ativada, passa a esperar os eventos do mouse da vista do diagrama, de modo que possa tratá-los. Quando sua execução não é mais necessária, a mesma é desativada.

O ControlLab estendeu a hierarquia de ferramentas fornecida pelo JHotDraw em função de suas necessidades específicas. São utilizadas as seguintes classes de ferramentas:

- *BlockConnectionTool* – Realiza a conexão de duas figuras de bloco. Representa a ferramenta *Conectar Blocos*. Também é utilizada pelos manipuladores das figuras de conexão para alterar suas extremidades e pela ferramenta *Default* para conectar dois blocos a partir de um conector de uma figura de bloco. Deste modo, em todos estes casos há um comportamento consistente.
- *BlockCreationTool* – Representa a ferramenta *Criar Figura de Bloco*, utilizando o padrão de projeto *Prototype* ([Gamma et. al., 1994:117-126]). Este padrão define a criação de objetos através de protótipos, ou seja, uma

instância desta classe apresenta uma figura de bloco que é o seu protótipo e que será clonada para a criação das respectivas figuras na vista do diagrama. Isto permite a configuração prévia das figuras de bloco no editor da paleta de ferramentas.

- *DefaultTool* – Representa a ferramenta *Default*, funcionando como um gerenciador de ferramentas auxiliares. Estas são evocadas em função da forma de ação do usuário sobre a vista do diagrama, conforme pode ser observado nas suas descrições.
- *DraggerTool* – Ativada pela ferramenta *Default* quando o usuário do ControlLab pressiona o botão do mouse sobre uma figura, realizando a sua movimentação ou, se a mesma já estiver selecionada, a movimentação da seleção.
- *HandleTracker* – Ferramenta do framework JHotDraw que é ativada pela ferramenta *Default* quando o mouse é acionado sobre um manipulador. Realiza a adaptação dos eventos do mouse para que o manipulador possa tratá-los.
- *InsertConnectorTool* – Classe abstrata que fornece a base para as ferramentas de inserção de conectores em figuras de bloco.
- *InsertInputConnectorTool* – Representa a ferramenta *Inserir Entrada*.
- *InsertOutputConnectorTool* – Representa a ferramenta *Inserir Saída*.
- *LabelTool* – Representa a ferramenta *Adicionar Rótulo*. Também é utilizada para a edição de rótulos já existentes através de manipulação direta, sendo evocada pela ferramenta *Default* quando o usuário pressiona a tecla *shift* enquanto executa um duplo clique sobre o rótulo. Quando o usuário pressiona o botão do mouse na vista do diagrama, a ferramenta abre um campo de texto onde o usuário define o texto do rótulo. A ferramenta é desativada quando o botão do mouse é pressionado fora deste campo.
- *RemoveConnectorTool* – Representa a ferramenta *Remover Entrada/Saída*.
- *RunTool* – Quando ativada tenta iniciar a execução do diagrama. Caso algum problema seja encontrado informa ao usuário, se desativando, com

exceção de quando alguma figura de bloco está inteiramente desconectada. Neste caso, ainda oferece a possibilidade da execução sem a figura desconectada. Ao ser desativada pára a execução do diagrama.

- *SelectionTool* – Ativada pela ferramenta *Default* quando o mouse é acionado em uma área livre do diagrama. Realiza a seleção de múltiplas figuras através de uma área de seleção indicada pelo usuário.

6.7 COMANDOS

O framework JHotDraw utiliza o padrão de projeto *Command* ([Gamma et. al., 1994:233-242]) para a definição de sua estrutura de comandos. Um comando é disponibilizado ao usuário através de um item de menu. Este item permanece desativado caso o comando não possa ser executado.

Os comandos do JHotDraw não apresentam o suporte para que sejam desfeitos ou refeitos. Por este motivo e para permitir o atendimento de necessidades específicas, foi estendida a hierarquia de comandos do JHotDraw através das seguintes classes:

- *ChangeSelectionGroup* – Altera o grupo das figuras selecionadas. Não pode ser executado quando não há figura selecionada, existe apenas um grupo no diagrama ou o diagrama presente na vista do diagrama é um subdiagrama.
- *ConfigureMainDiagram* – Abre a janela de configuração do diagrama principal e realiza as atualizações necessárias após seu fechamento.
- *ConfigureSimulation* – Abre a janela de configuração da simulação e realiza as atualizações necessárias após o seu fechamento.
- *Copy* – Copia as figuras selecionadas para a área de transferência. Não pode ser executado caso não exista figura selecionada ou a seleção apresente apenas figuras de conexão. Uma figura de conexão não é copiada para a área de transferência a menos que as figuras de bloco nas quais está conectada também sejam copiadas.
- *CreateSubDiagram* – Transforma as figuras selecionadas em um subdiagrama. Apresenta as mesmas condições de execução do comando

anterior, não permitindo a criação de subdiagramas a partir apenas de figuras de conexão.

- *Cut* – Apresenta o mesmo comportamento do classe *Copy*, com exceção de que remove as figuras de origem do diagrama.
- *Delete* – Elimina as figuras selecionadas do diagrama, não podendo, portanto, ser executado no caso de não haver figura selecionada. Pode ser substituído pela tecla *Del*.
- *Duplicate* – Duplica as figuras selecionadas no diagrama, promovendo uma movimentação nas figuras resultantes, de modo a não confundir com as figuras originais. Apresenta as mesmas condições de execução do comando *Copy*, somente duplicando figuras de conexão cujas respectivas figuras de bloco também sejam duplicadas.
- *ExportGroup* – Exporta um grupo nas formas definidas na seção 5.10. Não pode ser executado em diagramas com apenas um grupo.
- *ExportSubDiagram* – Exporta um subdiagrama nas formas definidas na seção 5.10. Somente pode ser executado quando a seleção da vista do diagrama conter apenas uma figura de bloco e esta figura seja um subdiagrama.
- *FigureTransfer* – Classe abstrata que apresenta o suporte para a cópia de figuras entre a vista do diagrama e a área de transferência, sendo superclasse das classes *Copy*, *Cut*, *Delete*, *Duplicate* e *Paste*.
- *GenerateMainDiagramCode* – Exporta um diagrama como arquivos de código Java™ nas formas definidas na seção 5.11.
- *HideSelection* – Oculta a seleção. Não pode ser executada quando não houver figura selecionada.
- *Paste* – Copia as figuras da área de transferência para a vista do diagrama. Não pode ser executado quando a área de transferência está vazia.

- *Redo* – Refaz ações que tenham sido desfeitas pelo comando da classe *Undo* no diagrama sob edição. Não pode ser executado quando não houver ações para serem refeitas.
- *ResetMainDiagram* – Reinicializa o diagrama.
- *SelectAll* – Seleciona todas as figuras presentes na vista do diagrama.
- *SetDiagramGrid* – Abre a janela de configuração da grade e realiza as atualizações necessárias após o seu fechamento.
- *SetDiagramSize* – Abre a janela de configuração do tamanho do diagrama e realiza as atualizações necessárias após o seu fechamento.
- *ShowHidden* – Volta a mostrar as figuras ocultas do diagrama sob edição.
- *Undo* – Desfaz ações na diagrama sob edição. Não pode ser executado quando não houver ações para serem desfeitas.

6.7.1 Suporte para Desfazer/Refazer Ações

Um mecanismo para desfazer e refazer ações, denominado mecanismo de *Undo*, apresenta-se distribuído pelo conjunto de elementos que podem ser desfeitos. É importante, portanto, projetá-lo de modo que não gere um acoplamento indesejado entre os elementos envolvidos. Estes elementos podem ser tanto comandos como ferramentas, apesar de o mecanismo ser muito utilizado, como no caso do framework ET++, apenas para comandos, motivo de sua descrição dentro desta seção.

Para enfraquecer o acoplamento foi utilizada a vista do diagrama, que colabora com todos os comandos e ferramentas, para encapsular o mecanismo de *Undo*, cujo esquema básico pode ser observado no diagrama da figura 6.19.

Comandos e ferramentas que desejem ter a possibilidade de serem desfeitos e refeitos devem implementar a interface *Undoable*. Quando da execução de suas ações, devem registrar o contexto destas ações na vista do diagrama, que repassa o pedido a sua instância da classe *UndoKernel*, que realiza seu empilhamento. Este contexto, uma instância da classe *UndoContext*, inclui o elemento que executou a ação e um objeto que represente o estado de execução da ação.

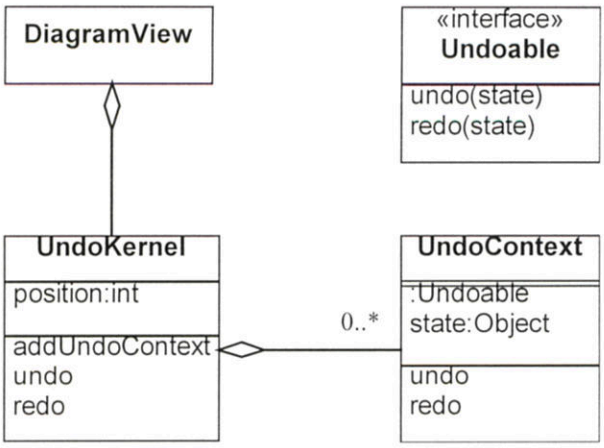


Figura 6.19 - Mecanismo de *Undo*

Este estado deve incluir todos os elementos necessários para que o agente da ação, comando ou ferramenta, seja capaz de desfazer e posteriormente refazer suas ações. Nesta tarefa, tais agentes devem atuar no diagrama de forma consistente, não acarretando efeitos colaterais no eventual comportamento de outros contextos.

Quando a vista do diagrama for requerida para refazer ou desfazer uma ação, repassa o pedido a sua instância da classe *UndoKernel*. Esta por sua vez, apresenta um valor que indica qual dentre o contextos de ação registrados, se houver algum, é o próximo a ser utilizado. Por exemplo, o registro de contextos da figura 6.20. O próximo contexto a ser desfeito é contexto de número 4. Nesta mesma situação, os contextos de número 5 e 6 são os próximos a serem refeitos. Caso algum novo contexto seja adicionado, ou seja, caso o usuário execute alguma nova ação no diagrama, os contextos 5 e 6 serão trocados pelo novo, não podendo mais ser refeitos.

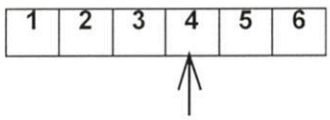


Figura 6.20 - Exemplo de Registro de Contextos

A classe *UndoKernel* apresenta um valor que indica a capacidade de contextos que podem ser registrados. Este valor pode ser alterado pelo usuário através da janela de opções da área de trabalho, mostrada na figura 5.2.

6.8 DOCUMENTAÇÃO

Além da documentação destinada ao usuário, citada na seção 5.12, o ControlLab dispõe de documentação a nível de desenvolvimento. Esta documentação é composta por este capítulo da monografia e por um conjunto de arquivos HTML™[6].

Estes arquivos HTML™ são resultantes da execução da ferramenta *JavaDoc*™[7], que se baseia em comentários especiais colocados junto ao código das classes e interfaces. As referências entre diferentes elementos, incluindo métodos e variáveis são transformadas automaticamente em *links* HTML™ para os elementos referenciados.

Para que os resultados deste mecanismo tenham validade, no entanto, é necessário que os comentários de código sejam constituídos de forma a descrevê-lo adequadamente.

No ControlLab, todas as classes foram documentadas segundo a especificação *JavaDoc*™, o que permite a disponibilização da documentação resultante para o acesso através de *browsers* Internet. Assim, é possível que as estruturas de implementação do ControlLab sejam analisadas sem a necessidade do entendimento do código propriamente dito.

6.9 CONSIDERAÇÕES FINAIS

Foram descritos neste capítulo o projeto dos principais componentes do ControlLab, cujas relações de dependência estão sintetizadas no esquema da figura 6.21.

Além destes componentes e das respectivas soluções de projeto envolvidas, ainda foram utilizados inúmeros outros elementos com o objetivo de melhorar a qualidade do projeto, dentre os quais pode-se destacar:

---

[6] HyperText Markup Language - http://www.w3.org/MarkUp/

[7] http://java.sun.com/products/jdk/javadoc/

- Tratamento de Exceções – Utilizado para o atendimento de casos anormais, tais como, por exemplo, a tentativa de abrir um arquivo com um formato desconhecido, o tratamento de exceções da linguagem Java™ permitiu a melhoria da robustez do ControlLab.
- Clone – O framework JHotDraw implementa o clone de figuras para complementar o padrão de projeto *Prototype*. No entanto, o modelo adotado, através da serialização e deserialização de objetos, é extremamente lento. Em seu lugar, o ControlLab implementa um mecanismo de cópia de objetos que não utiliza a serialização de objetos, mas a simples instanciação de novos objetos equivalentes aos objetos originais. Deste modo, a clonagem torna-se bem mais rápida.

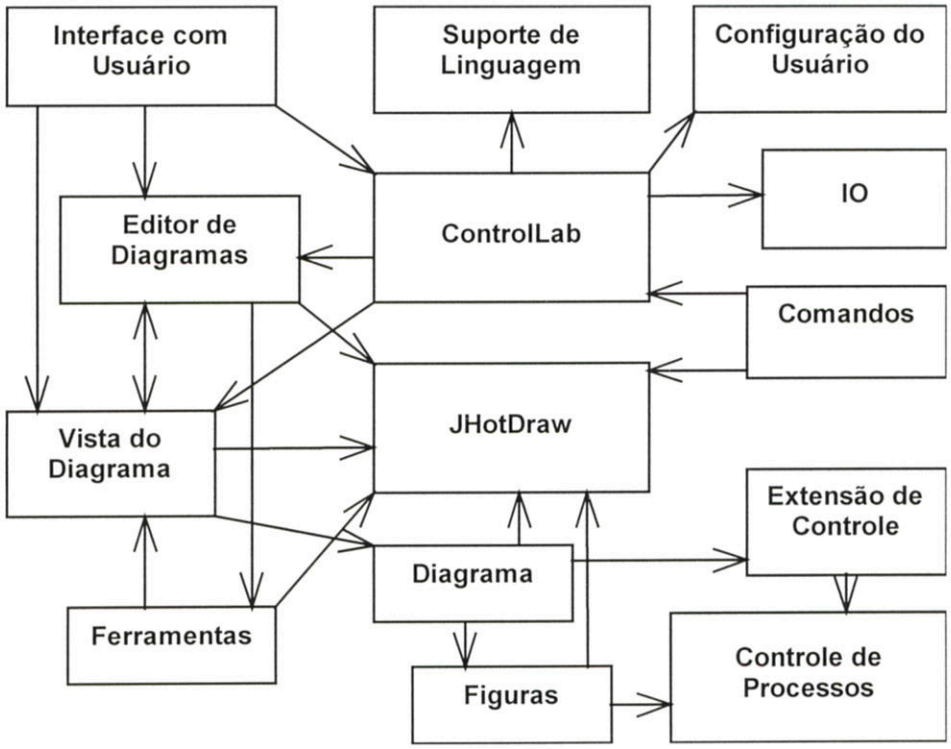


Figura 6.21 - Dependências

O resultado deste projeto, que contou com o auxílio dos frameworks JHotDraw e de Controle de Processos, de vários padrões de projeto e de diagramas UML foi a implementação do ControlLab como descrito no capítulo anterior.

7. Conclusões e Perspectivas

Este projeto, através do desenvolvimento do ControlLab, fornece um mecanismo alternativo para a área de pesquisa e ensino na área de sistemas monovariáveis. Esta condição é conseqüência da realização dos seus objetivos, definidos na introdução desta monografia.

O Framework de Controle de Processos, que é baseado em conceitos simples e diretamente relacionados com o domínio de controle, fornece um suporte aberto de controle. Este suporte também é extensível, através das seguintes opções:

- Utilização dos próprios mecanismos do ControlLab na criação de uma biblioteca de subdiagramas.
- Utilização de recursos da linguagem Java™ na definição de novos blocos do Framework de Controle de Processos. Através da utilização do modelo JavaBeans™, as propriedades de tais blocos são automaticamente reconhecidas pelo ControlLab.
- Transformação de subdiagramas em subclasses da classe *CompositeBlock* do Framework de Controle de Processos, que podem ser adicionados na paleta de ferramentas do mesmo modo que a opção anterior.

O desenvolvimento de um ambiente de modelagem baseado em manipulação direta simplifica a utilização do ControlLab e proporciona a rápida prototipação de sistemas. Sua amigabilidade também á aprimorada com a utilização de alguns dispositivos normalmente encontrados em outros softwares, tais como a área de transferência e o mecanismo de *Undo*.

A possível utilização dos sistemas modelados no ControlLab em outros contextos deve-se ao encapsulamento do Framework de Controle de Processos, que não é violado no ControlLab. Assim, através da geração de código a partir dos diagramas e subdiagramas, esses sistemas podem ser aplicados diretamente em outros softwares que utilizem o Framework de Controle de Processos.

Este framework ainda aprimora o alcance dos objetivos do ControlLab através da possibilidade da execução de sistemas monovariáveis distribuídos. Assim, o projeto também aplica-se ao estudo do comportamento de tais sistemas.

Além da aplicação no estudo de sistemas monovariáveis, o ControlLab também pode ser utilizado como um estudo de caso da Engenharia de Software, o que inclui as diferentes técnicas relacionadas com o seu desenvolvimento.

Este projeto terá continuidade através do desenvolvimento de um mestrado na área de controle de processos da Universidade Federal de Santa Catarina. Neste processo, tentar-se-á validar o ControlLab como um laboratório virtual para sistemas monovariáveis de controle, com a sua aplicação no estudo e ensino de diferentes áreas, tais como, por exemplo, o controle adaptativo.

Como uma característica inerente aos softwares, espera-se que, durante este mesmo período, sejam efetuadas modificações no ControlLab. Uma dessas modificações é o desenvolvimento de um ambiente de prototipação na forma de *Applet*, ampliando o espectro de utilização do ControlLab, que atualmente depende da instalação de um interpretador Java™.

Espera-se, deste modo, aprimorar ainda mais a sua capacidade de aplicação na pesquisa e ensino de sistemas monovariáveis de controle. Além disso, eventuais alterações no software servem para a verificação da qualidade de seu projeto e da utilidade das respectivas técnicas de Engenharia de Software envolvidas.

APÊNDICE A - UNIFIED MODELING LANGUAGE

O linguagem de modelagem UML foi desenvolvida por Grady Booch, Ivar Jacobson e Jim Rumbaugh com base métodos de modelagem que os mesmos desenvolveram em estudos independentes ([Booch, 1991], [Jacobson, 1992] e [Rumbaugh et. al., 1991]), procurando definir um padrão para a representação de projetos de software orientados a objetos.

Este apêndice descreve a simbologia básica que representa as duas formas de diagramas UML utilizadas nesta monografia.

A.1 DIAGRAMA DE CLASSES

Diagramas de classe indicam as classes de um projeto enfatizando o relacionamento entre as mesmas.

Nome/*Nome*
propriedades
métodos

Indicação de classe. Permite a inclusão de seus métodos e propriedades. O nome da classe em itálico indica ser uma classe abstrata.

nome: tipo

Definição do atributo de uma classe ou instância, através do seu nome, do seu tipo, e, opcionalmente, do módulo onde este tipo pode ser encontrado.

nome(parâmetros):retorno

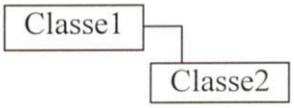
Definição de um método, através do seu nome, de uma lista de parâmetros e do tipo de objeto a ser retornado.

módulo::tipo

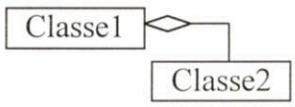
Opção para a inclusão do nome do módulo onde a definição de um determinado tipo de objeto pode ser encontrada.

«interface»

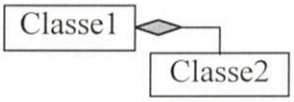
Indicação de uma *interface*, um conjunto de operações que define um tipo de objeto.



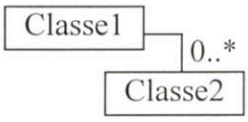
Indicação de uma relação de associação entre duas classes.



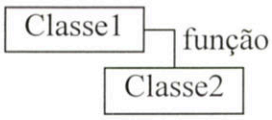
Indicação de uma relação de agregação entre duas classes.



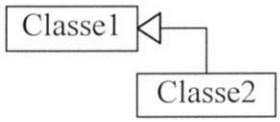
Indicação de uma relação de composição entre duas classes (um objeto da classe 1 é composto por um objeto da classe2).



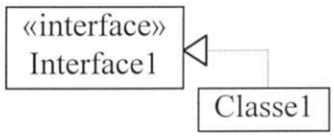
Especificação da multiplicidade de uma relação. Neste caso, um objeto da classe 1 está associado a um conjunto de zero a infinitos objetos da classe 2).



Indicação da função da classe na relação



Indicação de herança (a classe 2 é subclasse da classe 1).



Indicação de implementação (a classe 1 implementa as operações definidas na interface 1).

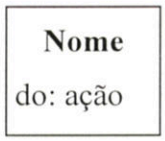
Para o melhor entendimento dos diagramas é interessante diferenciar os relacionamentos entre classes que são definidos através de propriedades:

- Associação – Representa uma conexão semântica entre duas classes, ou seja todas as formas de relacionamentos na forma de propriedades são associações.
- Agregação – Espécie de associação em que um objeto é agregado a outro auxiliando na criação de sua representação. Trabalhadores de uma empresa, por exemplo, estão agregados à empresa.
- Composição – Espécie de agregação mais forte, com o mesmo tempo de duração dos objetos de ambas as classes. Uma vez estabelecida não pode ser alterada. O objeto que está agregado não pode ser compartilhado em outras agregações.

UML ainda apresenta uma linguagem para auxiliar na especificação dos relacionamentos entre classes. Esta linguagem, denominada OCL (*Object Constraint Language*), foi utilizada no diagrama da figura 6.16, através do comandos *includes*, que indica que um conjunto de objetos inclui uma determinada instância.

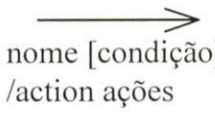
A.2 Diagrama de Estados

Representa os estados das instâncias de uma classe, indicando as transações entre os mesmos. Os elementos de um diagrama de estados utilizados na monografia são os seguintes:



Indicação de um estado. A seção *do* indica que ações o estado executa

Estado Final. Nos diagramas utilizados o estado inicial coincide com o estado final.



Transação entre estados. A condição entre colchetes indica os requisitos para a execução da transação. Os comandos que seguem a indicação */action* são executados durante a transição.

APÊNDICE B - BIBLIOTECA DEFAULT

Este apêndice apresenta a biblioteca default de figuras de blocos que é fornecida com o ControlLab. Cada tabela representa as figuras disponíveis em uma categoria específica. A primeira coluna indica o nome da figura, enquanto que a segunda indica a respectiva classe do bloco do Framework de Controle de Processos.

Cada uma destas figuras já apresenta-se configurada, inclusive com as respectivas explicações, além das explicações das saídas e das entradas.

Ferramentas Básicas	
Figura	Bloco
Saída Remota	RemoteOutputBlock
Valor Remoto	RemoteValueBlock
Entrada de Dados	EntryBlock
Monitoramento de Dados	DisplayBlock
Gráfico	PlotterBlock

Sinais	
Figura	Bloco
Constante	Constant
Degrau	Step
Rampa	Ramp
Parábola	Parabola
Senóide	Sinusoid

Controle	
Figura	Bloco
Equação a Diferenças	DifferenceEquationBlock
Processo de 1ª Ordem	FirstOrderProcess
Ganho	Gain
Saturação	Saturation
Conversor de Escala	ScaleConverter
Controlador PID	PIDController
Controlador PID Ajustável	AdjustablePIDController
Arquivo	FileOutputBlock

Numérica	
Figura	Bloco
Somador	Sum
Multiplicador	Multiply
Divisor	Divide
Potência	Power
Raiz Quadrada	Square Root
Exponencial	Exponential
Inversor	Invert
Valor Absoluto	Absolute
Arredondamento	Round
Sinal	Signal
Máximo	Maximum
Mínimo	Minimum

Trigonométrica	
Figura	Bloco
Seno	Sine
Coseno	Cosine
Tangente	Tangent
Arco Seno	ArcSine
Arco Coseno	ArcCosine
Arco Tangente	ArcTangent

Comparação	
Figura	Bloco
Igual	Equal
Não Igual	NotEqual
Maior Que	GreaterThan
Maior Que	LessThan
Maior ou Igual Que	GreaterThanOrEqual
Menor ou Igual Que	LessThanOrEqual

Booleana	
Figura	Bloco
NÃO	Not
E	And
OU	Or
XOR	Xor
Multiplexador	Multiplexer
Demultiplexador	Demultiplexer

APÊNDICE C - ESTRUTURA DE DIRETÓRIOS

Os arquivos de instalação do ControlLab incluem um documento que descreve os procedimentos de instalação para duas máquinas virtuais Java™: Sun JDK™[1] e IBM VisualAge™[2]. Após instalado, o ControlLab apresenta a seguinte estrutura de diretórios:

- <ControlLab> – Diretório onde o ControlLab foi instalado, e que deve estar presente na variável de ambiente *CLASSPATH* da linguagem Java. Inclui o arquivo *CLSetup.obj* que guarda as informações da paleta de ferramentas e da configuração do ambiente.
- <ControlLab>/examples – Apresenta alguns exemplos de diagramas e subdiagramas.
- <ControlLab>/Icons – Contêm os ícones que são utilizados na paleta de ferramentas.
- <ControlLab>/ls – Contém o suporte de linguagem do ControlLab.
- <ControlLab>/Tutorial – Contém a documentação relacionada ao tutorial do ControlLab.

Além destes diretórios e dos respectivos arquivos, o ControlLab ainda é formado pelos arquivos no formato *.class* que realizam a sua execução, e cuja localização também deve estar na variável de ambiente *CLASSPATH* da linguagem Java™.

---

[1] http://java.sun.com/products/jdk/

[2] http://www.software.ibm.com/ad/vajava/

BIBLIOGRAFIA

[Ambler, 1997]

Scott W. Ambler, "The Unified Modeling Language v1.1 and Beyond: The Techniques of Object-Oriented Modeling - A White Paper", AmbySoft Inc., Setembro, 1997.

[Alexander et. al., 1977]

Christopher Alexander, Shlomo Angel, Ingrid Fiksdahl-King, Max Jacobson, Sara Ishikawa e Murray Silverstein, "A Pattern Language", Oxford University Press, Nova Iorque, NY, Estados Unidos, 1977.

[Barry, 1991]

Brian M. Berry, "Real-Time Object Oriented Programming Systems", American Programmer, Outubro, 1991.

[Booch, 1991]

Grady Booch, "Object-Oriented Design with Applications", Benjamin/Cummings, Redwood City, CA, Estados Unidos, 1991.

[Booch et. al., 1995]

Grady Booch e Jim Rumbaugh, "Introduction to the Unified Method: Unifiyng the Booch & OMT Methods", OOPSLA'95 Tutorial, Outubro, 1995.

[Brant, 1995]

John M. Brant, "HotDraw"; Thesis, University of Illinois, Urbana-Champaign, IL, Estados Unidos, 1995.

[Bray et. al., 1997]

Michael Bray, Kimberly Brune, David A. Fisher, John Foreman et. al., "Software Technology - Reference Guide", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Estados Unidos, Janeiro, 1997.

[Constantine et. al., 1979]

Larry L. Constantine e Edward Yourdon, "Structured Design: Fundamentals of a Discipline of Computer Program and System Design", Prentice-Hall, Englewood Cliffs, NJ, Estados Unidos, 1979.

[DeSoto, 1997]

Alden DeSoto, "Using the Beans Development Kit 1.0", JavaSoft, Mountain View, C.A., Estados Unidos, Abril, 1997.

[Dod, 1995]

U. S. Department of Defense - Software Reuse Initiative, "Technology Roadmap", Versão 2.2, Washington D.C., Estados Unidos, 1995.

[Dony, 1990]

Christophe Dony, "Improving Exception Handling with Object-Oriented Programming", IEEE, N° 0730-3157/90/0000/0036, pp. 36-42, 1990.

[Flanagan, 1996]

David Flanagan, "Java in a Nutshell", O'Reilly & Associates, Sebastopol, CA, Estados Unidos, 1996.

[Foote et. al., 1991]

Brian Foote e Ralph E. Johnson, "Designing Reusable Classes", Department of Computer Science, University of Illinois, Urbana, IL, Estados Unidos, Agosto, 1991.

[Gamma, 1988]

Erich Gamma, André Weinand, Rudolf Marty, "ET++ - An Object-Oriented Application Framework in C++", OOPSLA'88 Proceedings, 1988.

[Gamma, 1994]

Erich Gamma, André Weinand, "ET++ - a Portable, Homogenous Class Library and Application Framework", UBILAB'94 Proceedings, Zurique, pp. 66-92, Setembro, 1994.

[Gamma et. al., 1995]

Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, Estados Unidos, 1995.

[Gosling et. al., 1996]

James Gosling, Bill Joy e Guy Steele, "The Java Language Specification", Addison-Wesley, Reading, MA, Estados Unidos, 1996.

[Goldberg, 1984]

Adele Goldberg, "Smalltalk-80: The interactive Programming Environment", Addison-Wesley, Reading, MA, Estados Unidos, 1984.

[Green, 1995]

Thomas Green, "Noddy's Guide to Visual Programming", The British Computer Society - Human-Computer Interaction Group, Interfaces, 1995.

[Howard, 1988]

Geoffry S. Howard, "Object Oriented Programming Explained", Journal of Systems Management, pp. 13-19, Julho, 1988.

[IEC9126,1991]

International Organization for Standardization, "ISO/IEC 9126 - Information technology -- Software product evaluation – Quality characteristics and guidelines for their use", Genebra, Suíça, 1991.

[Jacobson, 1992]

Ivar Jacobson, "Object-Oriented Software Engineering", Addison-Wesley, Reading, MA, Estados Unidos, 1992.

[Johnson et. al., 1990]

Ralph E. Johnson e Rebecca J. Wirfs-Brock, "Surveying Current Research in Object-Oriented Design", Communications of the ACM, Vol. 33, N° 9, pp. 104-124, Setembro, 1990.

[Johnson et. al., 1991]

Ralph E. Johnson e Vicente F. Russo, "Reusing Object-Oriented Designs", Department of Computer Science, University of Illinois, Urbana, IL, Estados Unidos, Maio, 1991.

[LaLonde et. al., 1990]

Wilf R. LaLonde e John R. Pugh, "Inside Smalltalk", Prentice Hall, Englewood Cliffs, NJ, Estados Unidos, 1990.

[Lamb, 1988]

David A. Lamb, "Reducing Workload in a Software Engineering Project Course", External Technical Report, Department of Computing and Information Science, Queen's University, Kingston, ON, Canadá, 1988.

[Loomis, 1987]

Mary E. S. Loomis, Ashwin V. Shah e James E. Rumbaugh, "An Object Modelling Technique for Conceptual Design", ECCOP'87, pp. 192-202, 1987.

[McIntyre, 1997]

Emily V. Veer, "Java Beans for Dummies", IDG Books, Foster City, C.A., Estados Unidos, 1997.

[Meyer, 1986]

Bertrand Meyer, "Genericity versus Inheritance", OOPSLA '86 Proceedings, Setembro, 1986.

[Parnas, 1984]

David L. Parnas, "Software Engineering Principles", Infor, Vol. 22, N° 4, pp. 303-306, Novembro, 1984.

[Parnas, 1994]

David L. Parnas, "Software Aging", 16th International Conference on Software Engineering Proceedings, Sorento Itália, pp. 279-287, Maio, 1994.

[Pree, 1994]

Wolfgand Pree, "Design Patterns for Object-Oriented Software Development", Addison-Wesley, Reading, MA, Estados Unidos, 1994.

[Pressman, 1987]

Roger S. Pressman, "Software Engineering - A Practitioner's Approach", McGraw-Hill, 1987.

[Puttick et. al., 1994]

Richard Puttick e Daniel Tkach, "Object Technology in Application Development", Benjamim/Cummings, Redwood City, CA, Estados Unidos, 1994.

[Ross, 1977]

Douglass T. Ross e K. E. Schoman, "Structured Analysis for Requirements Definition" IEEE Transactions on Software Engineering, Vol. 3, N° 1, pp. 6-15, Janeiro, 1977.

[Rumbaugh et. al., 1991]

James Rumbaugh, Michael Blaha, William Lorensen, Frederick Eddy e William Premerlan, "Object-Oriented Modeling and Design" , Prentice Hall, Englewood Cliffs, NJ, Estados Unidos, 1991.

[Schneiderman, 1982]

Ben Schneiderman, "The future of interactive systems and the emergence of direct manipulation", Behavior Information Technology, N° 1, pp. 237-256, 1982.

[Snyder, 1986]

Alan Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Languages", OOPSLA '86 Proceedings, Setembro, 1986.

[Quarti, 1997]

Silênio Sullivan Quarti, "Um Framework em Java para Aplicações de Controle e Monitoramento de Processos", Projeto Final, Engenharia de Controle e Automação Industrial - UFSC, Florianópolis, 1997.

ÍNDICE REMISSIVO

V