Raucer Curdulino
Ricardo Takao Horikawa
Rodrigo Akira Sanomia Sabanai

# Ferramenta de Validação de Requisitos Utilizando Redes de Petri

Trabalho de Formatura apresentado
à Escola Politécnica da
Universidade de São Paulo.

Área de Concentração:
Engenharia Mecatrônica

São Paulo
2003

Raucer Curdulino
Ricardo Takao Horikawa
Rodrigo Akira Sanomia Sabanai

nota final
9,5
(nove e meio)

# Ferramenta de Validação de Requisitos Utilizando Redes de Petri

Trabalho de Formatura apresentado
à Escola Politécnica da
Universidade de São Paulo.

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. José Reinaldo Silva

São Paulo
2003

*Aos nossos Pais, pelo apoio e compreensão dedicados durante nossa trajetória.*
*Aos nossos amigos e colegas, pelo empenho e inestimável contribuição de conhecimento para a conclusão deste trabalho.*

# AGRADECIMENTOS

Ao nosso orientador Prof. Dr. José Reinaldo Silva pela orientação e apoio nos momentos de necessidade.

Ao Mestre Eston Almança pela importante contribuição intelectual, imprescindível à realização deste trabalho.

Ao Mestrando Pedro Luis Angel, que contribuiu intensa e solicitamente na implementação prática do trabalho, através do protótipo do Ghenesys.

A todos que, de alguma forma, contribuíram para a realização deste trabalho.

# RESUMO

A crescente demanda para a evolução do software, em função da disponibilidade de hardware mais avançado, gerou a necessidade da criação de técnicas e metodologias para suportarem esta investida. Desde então surgiram várias técnicas, da Análise Estruturada nos anos 70 às metodologias Orientadas a Objetos nos anos 90.

Para a Engenharia de Software tornou-se extremamente importante a fusão destas técnicas, primeiro pelo ganho das metodologias resultantes e segundo pela introdução da padronização dos modelos destes sistemas. Como resultado das fusões de diversos métodos surgiu o Processo Unificado de Desenvolvimento de Software, que utiliza a UML como sendo uma linguagem de modelagem para especificação, construção e documentação de seus modelos. Essa linguagem permitiu que a tarefa de análise e eliciação de requisitos se tornassem mais disciplinadas, além de uma aproximação maior com o usuário, que é quem melhor entende do processo de negócio do sistema em desenvolvimento.

Em função da informalidade desta fase, tornou-se mais difícil o processo de validação dos requisitos do sistema, uma vez que erros de modelagem nas fases preliminares identificados mais adiante têm um custo muito elevado, proporcionalmente ao tamanho e à complexidade do sistema. Esta dificuldade se agrava ainda mais quando consideramos os Sistemas de Informação para a Automação, que demandam uma integração vertical entre os processos gerenciais e as atividades de chão de fábrica.

Com o objetivo de tornar possível a minimização do número de erros na fase de eliciação de requisitos, neste trabalho será desenvolvida uma ferramenta que transforma os requisitos em especificação formal baseada na dissertação de mestrado de Eston Almança dos Santos defendida no ano de 2002. Este processo é apresentado como sendo um refinamento da descrição disciplinada dos Use Cases, utilizando Redes de Petri como linguagem formal e ferramenta de verificação.

Além disso, serão estudadas três metodologias consagradas de desenvolvimento de software: Rational Unified Process (RUP), Extreme Programming (XP) e Together

Soft (FDD) e posteriormente será realizada uma comparação para selecionar a mais adequada para ser aplicado ao projeto.

# ABSTRACT

The availability of hardware facilities for computer systems has increased the demand for large software and generated the need for new techniques and methodologies to support this offensive. Several paradigms have appeared, from the Structured Analysis in the 70's to Objects Orientation in the 90's.

The fusion of these techniques are extremely important for Software Engineering, first because of the increased potential of the resulting methodologies, and second by the introduction of standard models for the target systems. The Unified Software Development Process is the most accepted result of the fusion method concerning object-oriented approaches, which have defined the UML as a language of modeling for specification, construction and documentation of systems. The language also allows that Requirements Elicitation and Analysis could be disciplined always centered in the user who understands better the business process.

Due to the informality of the first step in the life cycle, the validation of the system requirements became more difficult, leading to very costly errors, which are proportional to the size and complexity of the project. Such difficulty becomes even worst when we consider Information Systems for Automation as a target. These systems demand a vertical integration between the management processes and shop floor activities.

The requirements phase's errors can be minimized by the conversion in formal specification of the systems requirements, which is the goal of this work based on the Eston Almança dos Santos master degree. This process is presented as a Use Cases specification refinement, by using Petri nets as formal language and verification tool.

Furthermore three well established software engineering process will be studied, they are: Rational Unified Process (RUP), Extreme Programming (XP) e Together Soft (FDD). Later a comparison will be carried out to select the one that best fits for this project.

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# 1 INTRODUÇÃO

O design em Engenharia é a base para o desenvolvimento de sistemas artificiais. Especialmente no caso de sistemas automatizados, com um intervalo de tempo fixo onde este atua de forma autônoma, as atividades de design, bem como as fases iniciais do processo herdam uma grande responsabilidade: a de garantir que em qualquer circunstância, o comportamento do artefato será, além de útil e funcional, seguro para o ambiente e para as pessoas próximas.

Além disso, como requisito da autonomia, estes sistemas deverão ter um módulo de controle computável, responsável por monitorar e ao mesmo tempo intervir no processo caso este não esteja reproduzindo um comportamento previsto e esperado.

Assim as fases iniciais do projeto, notadamente a eliciação e validação de requisitos passaram a ter uma importância muito grande no design de sistemas automatizados, justificando até que haja uma seção específica sobre requisitos em Mecatrônica no Congresso Internacional de Engineering Requirements.

A informalidade característica da fase de requisitos causa, no entanto, uma dificuldade para que processos computacionais sejam criados em seu auxílio. Por esta razão, levantamentos feitos por diversos autores mostram que há poucas ferramentas em uso para este fim, enquanto que abundam métodos e ferramentas para a fase de implementação. Esta informalidade é fundamental para não tolher a criatividade ou para não levar a uma escolha equivocada das soluções, porém é também um motivo de isolamento dos processos sistemáticos da engenharia.

A dissertação de Eston Almança, defendida em dezembro de 2002, apresentou uma proposta de transferência semântica de Use Cases (a linguagem mais indicada para expressar requisitos) e Redes de Petri. A representação esquemática resultante é formal e permite análise baseada na identificação das boas propriedades das redes, além de prover o processo da possibilidade de simulação.

A proposta apresentada na dissertação está sendo incorporada às perspectivas de customização do Rational Suite.

A ferramenta especificada na dissertação de Eston Almança sugere que um conjunto finito de requisitos expresso em Use Case seja transformado em uma "markup language" que posteriormente pode ser transformada em uma representação formal em Redes de Petri Elementares.

Neste trabalho de formatura pretendemos desenvolver um pequeno protótipo da ferramenta em plataforma Windows e avaliar a sua capacidade de análise em casos concretos de sistemas mais complexos. Esta ferramenta será a base para o desenvolvimento de uma futura versão baseada nas redes estendidas como o Ghenesys (uma rede estendida, hierárquica, orientada a objetos).

# 2 REVISÃO DA LITERATURA

Com o objetivo de contribuir para o aumento do índice de detecção de erros durante a fase de levantamento de Requisitos, uma vez que os custos de correção de erros detectados nas fases posteriores se tornam muito elevados, é apresentada neste capítulo uma breve discussão sobre algumas das principais técnicas e processos atualmente desenvolvidos, que figuram entre as melhores práticas utilizadas pelos profissionais que atuam na área de desenvolvimento de sistemas.

Inicialmente, como revisão da literatura, será apresentado o processo de Engenharia de Requisitos (*Requirements Engineering)*, que disciplina a tarefa de levantamento e análise de requisitos, gerando o documento de especificação do sistema. Em seguida, será discutido o Processo Unificado *(Unified Process)*, que é um método recente para desenvolvimento de Software. Será abordada a *Unified Modeling Language* (UML), que é uma linguagem de modelagem para, dentre outros aspectos, especificar sistemas.

## 2.1 Engenharia de Requisitos

Neste trabalho, a Engenharia de Requisitos será vista como um processo sistemático e disciplinado de levantamento de necessidades e funcionalidades de um sistema. Portanto, a Engenharia de Requisitos transcende o artefato ou sistema em questão, e pode estar presente em outros processos de desenvolvimento particulares, como a Engenharia de Software.

Requisitos (*Requirements*) são definidos nas fases iniciais do desenvolvimento de sistemas como sendo uma especificação do que deve ser implementado. São descrições, geralmente em linguagem natural, de como o sistema deve proceder, ou de como as pessoas devem desempenhar determinado papel em uma organização.

Engenharia de Requisitos (*Requirements Engineering*) é o processo que engloba "todas as atividades envolvidas na descoberta, documentação e manutenção do conjunto de requisitos de um sistema baseado em computador" (*Sommerville, 1997*). Segundo *Sommerville*, o termo "Engenharia" significa que técnicas sistemáticas e

repetitivas devem ser usadas de forma a garantir que os requisitos do sistema sejam completos, consistentes e relevantes.

A importância da aplicação deste processo pode ser observada na seguinte afirmação: "Diferentes pessoas geralmente executam processos de forma diferente em uma organização. Pessoas com mais experiência podem trocar a ordem das tarefas porque sabem das conseqüências do que estão fazendo. Por outro lado, os inexperientes necessitam seguir uma seqüência de execução das tarefas, pois não possuem este conhecimento" (*Sommerville, 1997*). Do ponto de vista do autor, essa afirmação faz sentido, pois durante sua experiência profissional presenciou situações em que, na falta de processos definidos para a execução de uma tarefa, pessoas com diferentes níveis de conhecimento executavam os mesmos processos de formas diferentes. Em alguns casos, a ausência de definições desses processos causava até situações de desconforto dentro do ambiente de trabalho da organização.

Um dado importante é que, dependendo da complexidade do sistema a ser desenvolvido, deve-se considerar que o custo com as atividades de requisitos do sistema situa-se entre 10% e 15% (*Kotonya, 1998*) do custo total.

Considerando que os erros detectados em fases posteriores à fase de requisitos podem chegar a custar muitas vezes mais e que os problemas desta fase podem causar sérios desvios na fase de modelagem e *design*, como atraso na entrega do sistema, insatisfação do cliente, solicitações de alterações no sistema assim que o mesmo entra em produção, dentre outros inconvenientes, torna-se extremamente importante que as atividades desenvolvidas nesta fase possuam um reduzido índice de erros.

A seguir são apresentados os pontos básicos da Engenharia de Requisitos e as considerações mais importantes para o presente trabalho.

### 2.1.1 Documento de Requisitos

É um documento oficial dos requisitos do sistema que servirá como base de comunicação tanto entre clientes e gerentes de desenvolvimento, como entre usuários e desenvolvedores do sistema. Este documento é conhecido por vários nomes, dentre eles são citados alguns:

- Especificação Funcional (*Functional Specification*),
- Definição de Requisitos (*Requirements Definition*),
- Especificação de Requisitos de Software (*Software Requirements Specification – SRS*), etc.

Este documento pode possuir uma estrutura tão detalhada quanto se desejar possuir um padrão que garanta a qualidade dos documentos de requisitos dentro da organização. Uma estrutura mínima deve conter:

- Visão geral do sistema,
- Glossário,
- Lista de requisitos funcionais, e
- Restrições operacionais do sistema.

Por ser consensual, é importante destacar aqui a relevância do Documento de Requisitos, não apenas como registro do processo de levantamento e elaboração dos requisitos, mas como resultado, tanto parcial como final deste processo.

### 2.1.2 Processo de Engenharia de Requisitos

Pode ser entendido como um processo estruturado, composto de um conjunto de atividades que são seguidas para desenvolver, validar e manter um documento de especificação de sistemas. Um processo idealmente completo poderia incluir quais atividades seriam desenvolvidas, o encadeamento ou cronograma dessas atividades, quem é responsável por cada atividade, as entradas e saídas das atividades e as ferramentas utilizadas para suportar a Engenharia de Requisitos.

As entradas do processo de Engenharia de Requisitos são informações sobre os sistemas existentes, necessidades dos usuários, padrões da organização, informações de domínio do sistema, e regras reguladoras externas à organização. A aplicação deste processo varia entre as organizações, porém a maioria envolve eliciação de requisitos, análise e negociação de requisitos e validação de requisitos. A saída deste processo é o Documento de Requisitos do sistema.

Para representação deste processo utilizam-se modelos esquemáticos que são descrições simplificadas vistas de uma determinada perspectiva. Exemplos destes

modelos são: modelos de atividades, modelos de funções (*role-action*) e modelos de entidade-relacionamento.

Fatores Humanos, Sociais e Organizacionais são influências importantes no processo de Engenharia de Requisitos. Eles geralmente estabelecem considerações técnicas.

A figura 2-1 ilustra o Processo de Engenharia de Requisitos, destacando as principais entradas e saídas do processo (*Kotonya, 1998*).

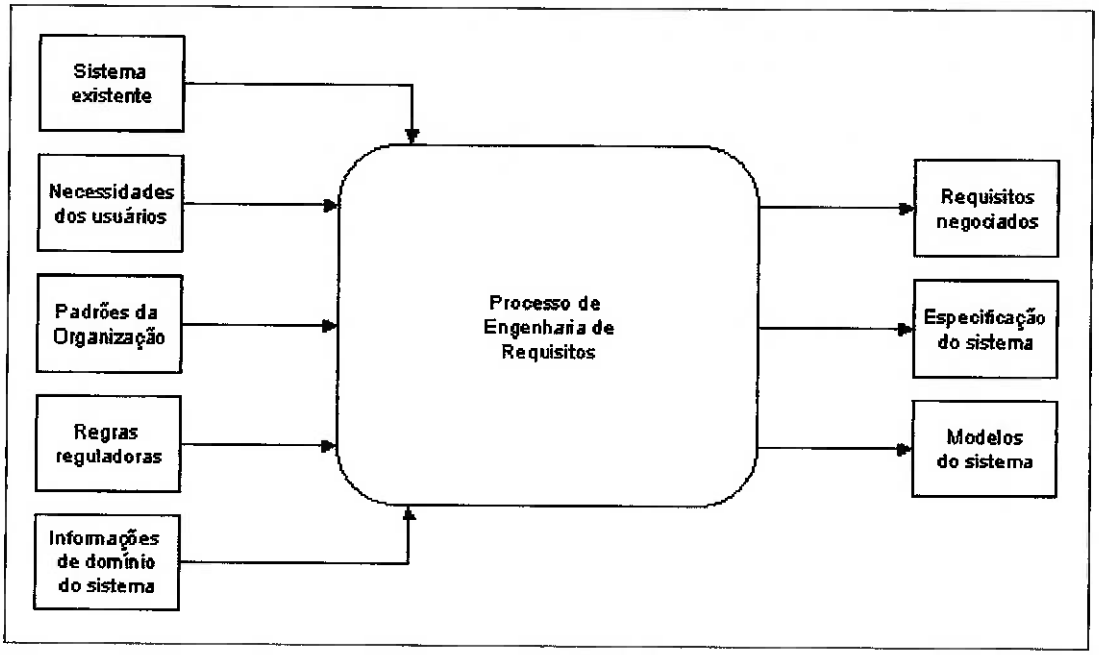



Figura 2-1 Entradas e saídas do processo de Engenharia de Requisitos

Da figura 2-1 pode-se destacar as entradas do processo de Engenharia de Requisitos que freqüentemente são desconsideradas quando não se executa esta tarefa: "Sistema existente" e "Regras reguladoras". Em muitos casos estas entradas são percebidas em fases adiantadas do processo de desenvolvimento e isto gera retrabalhos. As saídas deste processo são documentos formais que podem servir de base de negociação para contratação de equipes que farão a implementação ("especificação do sistema"), como também para acertos em casos de desentendimentos entre as partes ("requisitos negociados").

Uma abordagem clássica da Engenharia de Requisitos segundo *Sommervile* aborda quatro atividades e seus respectivos documentos:

- Estudo de Viabilidade (Relatório de Viabilidade)
- Levantamento e Análise de Requisitos (Modelos do Sistema)
- Validação de Requisitos (Documento de Requisitos)

A seguir uma explicação mais detalhada de cada uma destas atividades.

### 2.1.3 Estudo de Viabilidade

O Estudo de Viabilidade descreve sem muitos detalhes o sistema e como será empregado dentro de uma organização e se ele satisfaz seus objetivos de negócios, apontando uma conclusão sobre a validade do seu desenvolvimento e implantação.

### 2.1.4 Levantamento e Análise de Requisitos

As atividades de Levantamento e Análise de Requisitos envolvem o trabalho conjunto entre desenvolvedores, analistas e outros *stakeholders* a fim de especificar o sistema em termos de requisitos *funcionais* e *não-funcionais*, dando assim subsídios para que ele possa ser desenvolvido. Sugere-se também um modelo de processo genérico para o levantamento e a análise, e que compreende as seguintes atividades:

- *Entendimento do Domínio* da aplicação
- *Coleta de Requisitos*, que trata do levantamento dos requisitos dos *stakeholders*
- *Classificação* dos requisitos
- *Resolução de Conflitos* provenientes dos interesses de diversos *stakeholders*
- *Priorização* dos requisitos levantados
- *Verificação dos Requisitos* a fim de analisar sua consistência e validade com o que os *stakeholders* esperam do sistema

Afirma-se ainda que não existe uma técnica absoluta, recomendando-se a aplicação de várias delas, e são apresentadas três para levantamento e análise de requisitos [*Software Engineering – Sommerville, Ian - 2001*]:

- *Viewpoint-oriented Elicitation* (Levantamento orientado a ponto-de-vista)

- *Scenarios* (Cenários)
- *Ethnography* (Etnografia)

A técnica *Viewpoint-Oriented Elicitation* (ou Levantamento Orientado a Ponto-de-Vista) leva em consideração que para sistemas de médio e grande porte existem diversos tipos de usuário e cujos pontos-de-vista devem ser levados em consideração. Esses pontos-de-vista "enxergam" o problema de maneiras diferentes e não devem ser completamente independentes uns dos outros, oferecendo assim a possibilidade de se detectar conflitos entre os requisitos levantados. Dentro dessa abordagem, existem métodos que têm um modo de interpretar o que são *viewpoints*, cada um apresentando suas vantagens e desvantagens [*Software Engineering – Sommerville, Ian - 2001*]:

- *Viewpoints* como fonte ou sorvedouro de dados, sendo responsáveis pela produção ou consumo de dados.
- *Viewpoints* como uma estrutura de representação, sendo considerados como um tipo particular de modelo de sistema.
- *Viewpoints* como um receptor de serviços, sendo externos ao sistema e recebendo serviços deste.

O método VORD (Viewpoint-Oriented Requirements Definition) segue o segundo método apontado acima e seus principais estágios são:

- Identificação dos *Viewpoints*
- Estruturação dos *Viewpoints*
- Documentação dos *Viewpoints*
- Mapeamento dos *Viewpoints* de Sistema

A técnica *Scenarios* faz uso de sessões de interação, mostrando cenários de interação do usuário com o sistema, fazendo uso de diagramas e de *use cases* para representar essa interação.

Finalmente a técnica *Ethnography* faz uso da observação para entender os requisitos dentro de um contexto organizacional e social, existindo o envolvimento de analistas com as atividades diárias e reais dos futuros usuários do sistema.

### 2.1.5 Validação de Requisitos

A validação dos requisitos se preocupa principalmente em checar os requisitos quanto a omissão, conflitos e ambigüidades, e garantir que sigam os padrões de qualidade definidos na organização. Um exemplo de verificações durante este processo é:

- *Verificações de Validade*, que serve para avaliar se um conjunto de requisitos vale para um grupo diverso de usuários.
- *Verificações de Consistência*, para evitar conflitos entre requisitos.
- *Verificações de Completitude*, para avaliar se os requisitos levantados incluem todas as funções e restrições desejadas pelos usuários do sistema.
- *Verificações de Realismo*, para avaliar se o sistema é viável tecnicamente e se pode ser desenvolvido e implantado dentro das limitações impostas de custo e prazos.
- *Verificabilidade*, para avaliar se os requisitos levantados podem ser verificados.

De acordo com *Sommerville*, o uso da prototipação poderia ser útil nesta fase, já que usuários poderiam utilizar o protótipo como elemento para validação do processo. Desta forma a fase de análise do sistema, mesmo sendo uma análise essencial, deve ser antecipada, já que o protótipo deve conter funcionalidades do sistema, e não somente os requisitos, para a execução do processo de negócio do usuário.

Segundo *Kotonya*, a prototipagem pode ser dividida em duas categorias: "*Throw-away*" e "*Evolutionary*", ou seja, uma prototipagem descartável onde o protótipo apenas é aproveitado na validação dos requisitos e uma prototipagem evolutiva onde o protótipo evolui durante o projeto até tornar-se o sistema final.

Existem três principais tipos de prototipagem, são eles: "Prototipagem em Papel" onde é realizada uma simulação do sistema em forma de apresentação e os usuários "navegam" através das telas; "*Wizard of Oz*", ou Magico de Oz nesta prototipagem uma pessoa simula a resposta do sistema a uma determinada entrada do sistema e finalmente a "Prototipagem Automatizada" onde uma versão executável do sistema é gerada para que os usuários possam testá-la e verificar os requisitos impostos.

Segundo (*Sommerville, 1997*), as principais tarefas a serem executadas durante o processo de validação de requisitos são:

- Certifique-se que o Documento de Requisitos atenda seus padrões
- Formalize a inspeção de Requisitos
- Utilize equipes multidisciplinares para revisar Requisitos
- Defina *check lists* de Validação
- Use protótipos para visualizar Requisitos
- Escreva um rascunho do manual do usuário
- Planeje Casos de testes de Requisitos
- Traduza os requisitos em modelos de sistemas

As abordagens em validação de requisitos buscam garantir que os documentos de especificação reflitam a necessidade do sistema em questão. Os casos informais, que não requerem especialização do analista e estão mais próximas dos usuários do sistema, tendem a ser mais frágeis, uma vez que basicamente se apóia em revisão de documentos e comparações com necessidades da organização. Por outro lado, aqueles casos que buscam tentativas de formalização tendem a ser mais confiáveis; no entanto requerem maior especialização dos analistas no sentido da necessidade de expressar os requisitos do sistema em uma determinada linguagem formal. Isto faz com que esta abordagem seja utilizada em situações bem específicas, por exemplo em aplicações que exijam um nível de erro bem reduzido.

### 2.1.6 Pontos de Vista (Viewpoints)

Para entender os requisitos de um sistema, deve-se entender os serviços que o sistema fornece, o domínio de aplicação do sistema, restrições não funcionais, o processo de desenvolvimento do sistema, o ambiente onde o sistema será instalado e questões organizacionais que afetam a operação do sistema. Segundo (*Kotonya, 1998*), "o processo de Engenharia de Requisitos envolve a captura, análise e resolução de muitas idéias, perspectivas e relacionamentos de diferentes níveis de detalhes". No sentido de resolver este problema, ou de estruturar os diversos conhecimentos de requisitos, alguns métodos têm surgido baseados na noção de "pontos de vista" (*viewpoints*).

O *Viewpoint* é "uma coleção de informações sobre um sistema ou um problema, ambiente ou domínio relacionado, que são apresentados de uma perspectiva em particular" (*Kotonya, 1998*). Estas perspectivas podem ser de usuários do sistema, de outros sistemas, de profissionais envolvidos no processo de desenvolvimento, etc. Geralmente, a informação de cada *viewpoint* é incompleta, porém os requisitos gerais do sistema são derivados da integração das informações de cada *viewpoint*. Portanto, é de se esperar que haja um processo de resolução de conflitos para eliminar as inconsistências entre as diferentes especificações originadas pelos diferentes *viewpoints*.

A seguir, de acordo (*Kotonya, 1998*), são relacionadas as principais técnicas que foram desenvolvidas para tratar os requisitos utilizando *viewpoints*.

#### 2.1.6.1 Structured Analysis and Design Technique (SADT)

Esta técnica foi desenvolvida nos meados da década de 70 (*Marca, 1988*) e se baseia no modelo de fluxo de dados que visualiza o sistema como sendo um conjunto de atividades interativas. A SADT não possui um mecanismo explícito de *viewpoint*, entretanto eles podem ser vistos como extensões intuitivas desta técnica de modelagem já que cada módulo funcional é visto como um elemento que encapsula funções internas e interage por diferentes interfaces com o mundo externo (na maior parte dos casos composto por outros módulos, sistemas, etc.). Fontes de dados e repositórios constituem *viewpoints* nesta técnica. (*Ross, 1985*)

#### 2.1.6.2 Controlled Requirements Expression (CORE)

Criado nos anos 70 para atender à indústria aeroespacial, o CORE tornou-se padrão na Europa e é até hoje usado pela "*European Fighter Aircraft*". É portanto, um dos exemplos bem sucedidos de modelo de especificações utilizados tanto para sistemas (mecânicos, eletrônicos, etc.) quanto para software.

Fruto da linha de pensamento estruturada dos anos 70, o CORE também é baseado em decomposição funcional, como a SADT. Porém, ao contrário da SADT, nesta abordagem são observados, explicitamente, dois níveis de *viewpoints*: o primeiro nível é composto pelas entidades que interagem ou modificam o sistema (segundo o seu caráter ativo ou passivo estes *viewpoints* são denominados funcionais ou não-

funcionais por seu autor); o segundo nível é composto pelas entidades que definem ou impõem limites ao sistema. Assim, neste nível as entidades que suportam os *viewpoints* são todas passivas (os *viewpoints* de fronteira são, em uma visão *top down*, entidades que interagem indiretamente com o sistema, colocando pela primeira vez a propriedade de se representar *viewpoints* indiretos). (*Mullery, 1979*)

### 2.1.6.3 Viewpoint-oriented System Engineering (VOSE)

O VOSE foi desenvolvido nos anos 90 no Imperial College por *Finkelstein* e *Nuseibeh*, (*Finkelstein, et al., 1992*) com o intuito de permitir a integração de métodos de desenvolvimento. Embora se interprete por "métodos de desenvolvimento" como sendo métodos de desenvolvimento de software, o VOSE não deve ser entendido como sendo um método de desenvolvimento de software.

A filosofia básica do VOSE pode ser resumida no termo "*separation of concerns*", isto é, o fato de que no desenvolvimento de sistemas (mais até que no desenvolvimento de software) participam especialistas em vários aspectos do desenvolvimento de software e da área de aplicação. Portanto, é importante analisar como se modifica o ponto de vista de cada um destes agentes durante o processo de desenvolvimento.

No método VOSE, cada *viewpoint* padrão pode ter várias janelas, que seriam as sub-divisões dentro do próprio *viewpoint*. A figura 2-2 ilustra esta sub-divisão (*Kotonya, 1998*).

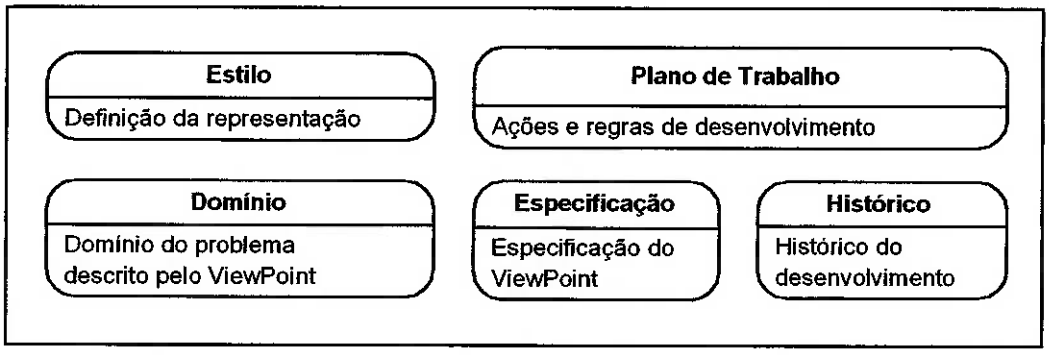



Figura 2-2 Janelas de um ViewPoint padrão

Associado ao *viewpoint* padrão, para cada agente pode ser associado um diagrama de fluxo de dados e um diagrama de estado (representando ações). A introdução de múltiplos diagramas na fase de requisitos é, portanto, apresentada explicitamente, pela primeira vez, na definição do VOSE.

A especificação de requisitos global do sistema é a combinação das visões de cada um dos agentes identificados, uma vez resolvidos os conflitos (*Finkelstein, et al., 1992*). A dificuldade para a utilização deste método é justamente a combinação das diversas visões (cujo número pode ser grande, principalmente em aplicações como sistemas de informação para manufatura). Como nos casos anteriores, não há nenhum método de validação previsto.

#### 2.1.6.4 Viewpoint-oriented Requirements Definition (VORD)

O VORD foi inicialmente feito para sistemas interativos, onde existe uma classe de usuários e *use cases* na qual estes usuários passam informação para o sistema e, em troca, recebem serviços e/ou mais informações. O VORD funciona de forma análoga a um sistema cliente-servidor, onde os *viewpoints* são os clientes (*Kotonya, 1996*).

Os requisitos são distribuídos em duas classes distintas:

- Diretos: que correspondem diretamente aos clientes, isto é, os que enviam informação e parâmetros e recebem de volta serviços;
- Indiretos: que correspondem aos *viewpoints* que não interagem e nem requerem serviços diretamente, mas têm "interesse" nos serviços. São de fato *viewpoints* que não estão diretamente ligados a usuários do sistema e nem à forma como este interage com o mundo exterior, mas são muito importantes para o seu funcionamento adequado. Os aspectos organizacionais de sistemas de informação são exemplos claros deste tipo de *viewpoint*.

#### 2.1.6.5 Viewpoint-oriented Requirements Validation (VORV)

O VORV (*Leite, 1991*) parte da constatação de que o processo de eliciação e, principalmente, de validação de requisitos em sistemas complexos tem se mostrado árduo, principalmente para sistemas de grande porte. Portanto é necessário que a

validação seja inerente ao processo de levantamento de requisitos ao invés de encerrar o processo, após o levantamento de todos os requisitos, diretos e indiretos, relativos a diversas classes de usuários e sistemas.

O método é baseado em três conceitos básicos: um *viewpoint*, uma perspectiva e uma visão. O *viewpoint* é um estado mental de um indivíduo que examina o universo de discurso (o contexto em que o sistema está inserido). Uma perspectiva é a descrição de fatos e ações decorrentes de um *viewpoint* e de aspectos de modelagem. Finalmente, uma visão é a integração dos dois aspectos anteriores.

A aplicação do método pressupõe a utilização de *viewpoints* de pelo menos dois analistas – muito embora estes não sejam particularmente caracterizados. A seguir esses *viewpoints* são sobrepostos de maneira a integrarem uma perspectiva do sistema. Os conflitos entre os *viewpoints* são resolvidos antes de se fechar a perspectiva final do sistema. (Leite, 1991)

#### 2.1.6.6 Uma discussão sobre os métodos de Requisitos de Sistemas

Os métodos de levantamento de Requisitos descritos anteriormente apontam para as características que as novas propostas devem conter.

As características a serem destacadas são:

1. os métodos apontam na direção da estruturação e hierarquização de agentes e *viewpoints*, como o CORE, VORD e VORV

2. a identificação dos agentes principais e seus respectivos *viewpoints* é de fundamental importância e é um elemento indispensável para integrar o levantamento de requisitos com o processo de geração de especificações

3. a classificação de *viewpoints*, seja conjuntamente com os agentes (VOSE) ou em separado (VORD, VORV), é fundamental para permitir a validação e para um bom entendimento dos próprios requisitos. Em geral, a classificação destaca os *viewpoints* "externos" e "diretos", envolvendo o núcleo do sistema e usuários, outros sistemas, etc. e os "internos", "indiretos", que não pressupõem interação ou troca de informação com o contexto, mas apenas o funcionamento interno do sistema

4. a validação, que não aparece explicitamente (ou apenas informalmente) nos demais métodos, é apresentada no VORV como resultante da estratégia de levantamento dos requisitos e integrada com esta. É opinião do autor que esta é a abordagem correta e que servirá de referência para futuras contribuições. A contextualização, que aparece no VORV, também é um aspecto de referência, permitindo a fundamentação dos *viewpoints* e, portanto contribuindo para o processo de validação.

Além destas observações, seria importante introduzir um questionamento de natureza conceitual e prático: seria conveniente caracterizar um conjunto mínimo de classes de *viewpoints* e agentes, cuja integração seria suficiente para justificar um novo projeto de sistema? Se existir tal classificação, esta deve estar de acordo com os métodos de desenvolvimento atualmente utilizados, como o Processo Unificado.

A seguir, apresenta-se uma possibilidade de especificar sistemas com um enfoque para a verificação dos Requisitos do mesmo.

Considerando três classes de atores hipotéticos, cujos *viewpoints* se deseja integrar como condição necessária para o sucesso de sistema de informação: o Usuário final, o Patrocinador do sistema, e o Analista (a figura 2-3 ilustra esta consideração). O Usuário final aparece em todos os métodos anteriormente descritos, embora não tenha sido destacado especialmente. O Patrocinador é o contratante e beneficiário do sistema, embora não seja necessariamente o seu usuário final. Este deseja que o sistema atenda aos seus interesses, satisfazendo o Usuário final.

O Analista é o que irá fazer o projeto e eventualmente a implementação do sistema e, portanto deve saber exatamente o que deve ser feito e quais as funcionalidades que devem ser atendidas.

Portanto a validação é uma integração harmônica destes *viewpoints*, como preconizado no VORV.

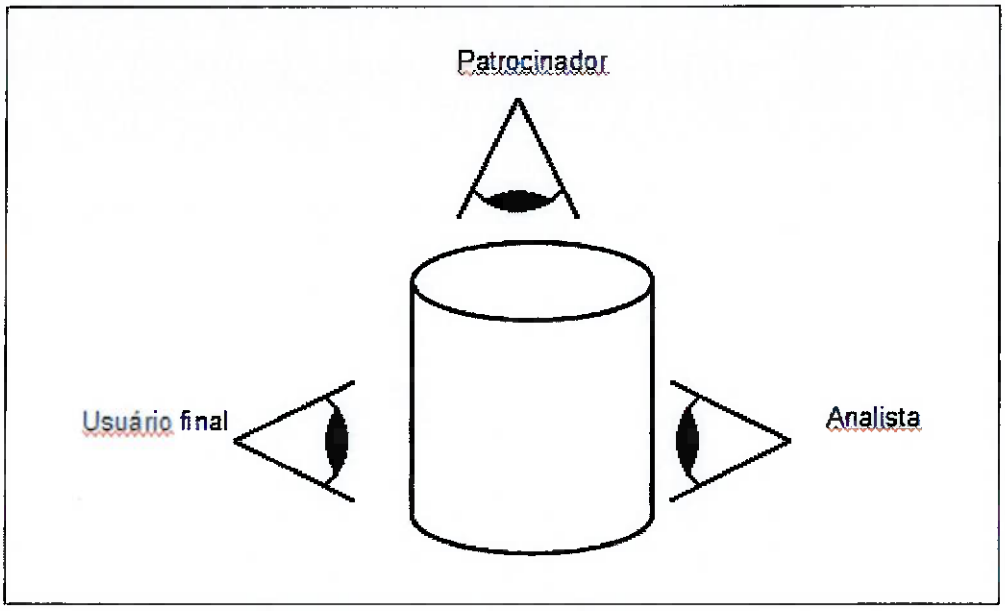



Figura 2-3 Pontos de vista de atores de um sistema

Estes são atores clássicos nos sistemas de informação, sejam estes destinados a serviço, negócios (*e-business*) ou para produção (*e-manufacturing*). Sob o ponto de vista do autor, um "bom sistema" seria, metaforicamente, um sistema cujos requisitos englobassem estes *viewpoints*.

Prosseguindo com a metáfora, destaca-se em primeiro lugar que, por ser a Engenharia de Requisitos baseada em eliciação e em processos intrinsecamente informais, esta admite uma dependência dos atores principais aqui citados. Segundo, que os requisitos eliciados, segundo o *viewpoint* de cada um destes atores, seria suficiente para determinar "fronteiras" do sistema (limites de separação entre o que é e o que não é o sistema), porém seriam insuficientes para encapsular todas as funcionalidades (seja por omissão de algumas, seja por falta de profundidade de visão sobre o sistema). Terceiro, o que poderia ser qualificado como um "bom sistema" (pensando, principalmente, nos sistemas de informação) seria representado por um conjunto de requisitos encapsulados pelas fronteiras de, pelo menos, estes três *viewpoints*.

Para que isto possa ser minimamente materializado seria necessário ter uma representação (preferencialmente formal) que integrasse todos os *Use Cases* levantados ou que permitisse enxergar as fronteiras metaforicamente referidas anteriormente.

São apresentados, na seqüência, os principais conceitos do Processo Unificado, da UML e da rede de Petri.

## 2.2 Processo Unificado (Unified Process)

Com a crescente demanda para Sistemas de Software cada vez mais complexos, a Engenharia de Software, assim como as demais áreas da Engenharia, precisou definir processos que garantissem a construção de sistemas que:

- Atendam a demanda dos usuários,
- Sejam economicamente viáveis (inclusive sob o ponto de vista do usuário),
- Respeitem o cronograma de entrega do projeto, e
- Possuam um alto índice de qualidade.

Comparando a Engenharia de Software com as demais áreas da Engenharia, verifica-se que esta disciplina é ainda muito jovem. Perante a necessidade de um projeto de construção de uma ponte, verifica-se que os engenheiros da construção civil possuem ampla experiência e, conseqüentemente, processos definidos que podem servir como base para a definição de um novo projeto. Já os engenheiros de software não possuem uma base de conhecimento tão extensa quanto os engenheiros civis e, além disso, de acordo com (*Pressman, 1993*), "o software não é um produto palpável", assim como uma ponte o é, o que dificulta ainda mais a tarefa de construção de software. Daí a necessidade de um processo para desenvolvimento de sistemas de software que possa tornar essa tarefa mais próxima daquela de se construir uma ponte.

Desde os anos 70 surgiram várias contribuições de autores preocupados com a tarefa de desenvolvimento de Software. Desde a Análise Estrutural (*Yourdon, 1989*) até as técnicas baseadas na Orientação a Objetos, pode-se perceber o empenho com que pesquisadores e praticantes de mercado se entregaram à criação de métodos para garantir a qualidade no desenvolvimento de Software (e conseqüentemente de sistemas de natureza semelhantes, como os sistemas automatizados). Na década de 90 surgiu a possibilidade de unificação dessas metodologias, que foi uma proposta (apoiada pelas principais empresas de Software e pelos principais pesquisadores desta área da engenharia) de fusão das várias técnicas existentes até então, para a

obtenção de um processo unificado, ou seja, algo que permitisse uma comunicação sem distorções entre diferentes equipes e, conseqüentemente, fosse adotado pela maioria dos profissionais envolvidos na tarefa de desenvolvimento de Software.

Assim surgiu o Processo de Desenvolvimento de Software Unificado, que é um guia que descreve "*Quem* está fazendo *O que*, *Como* e *Quando*" (*Jacobson, 1998*). Segundo Jacobson este processo:

- Proporciona orientação para ordenar as atividades de uma equipe
- Direciona as tarefas dos desenvolvedores individualmente e da equipe como um todo
- Especifica quais artefatos podem ser desenvolvidos
- Oferece critérios para monitorar e medir os produtos e atividades de um projeto

Descreve-se a seguir as principais características do Processo Unificado, segundo Jacobson:

### 2.2.1 Baseado em Componentes

O sistema de software construído a partir do Processo Unificado consistirá de componentes de software que podem ser interconectados a partir de interfaces bem definidas.

### 2.2.2 Dirigido a Use Cases

*Use Case* pode ser entendido como "o que" o sistema faz para cada usuário (que pode ser pessoas ou máquinas que interagem com o sistema). Baseado no modelo de *Use Cases* do sistema, os desenvolvedores criam modelos de projeto, implementação e testes que realizam as tarefas dos *Use Cases*. Por isso diz-se que o Processo Unificado é dirigido a *Use Cases*.

### 2.2.3 Centrado na Arquitetura

A arquitetura do software deve ser definida em paralelo com o entendimento do modelo de *Use Cases*. À medida que os principais *Use Cases* vão sendo especificados, a arquitetura do software vai se definindo. Para cada *Use Case*

analisado, o mesmo é especificado em detalhes e realizado em termos de sub-sistemas, classes e componentes.

### 2.2.4 Iterativo e Incremental

Com o objetivo de proporcionar um maior controle das atividades de um projeto, o Processo Unificado propõe que os projetos sejam divididos em "mini projetos", que são as iterações que resultam em um incremento no projeto. As iterações são estrategicamente definidas no planejamento de forma que a necessidade de repetição de uma delas não gere um impacto significativo no prazo ou custo do projeto como um todo.

### 2.2.5 Ciclo de Vida

O Processo Unificado repete uma série de ciclos durante a vida do sistema. Cada fase concluída corresponde a uma disponibilização do sistema para o cliente. Cada ciclo consiste de quatro fases: Concepção (*Inception*), Elaboração (*Elaboration*), Construção (*Construction*) e Transição (*Transition*). Cada fase compreende o desenvolvimento de cinco *workflows*: Requisitos (*Requirements*), Análise (*Analysis*), Projeto (*Design*), Implementação (*Implementation*) e Teste (*Test*). A figura 2-4 ilustra o ciclo de vida do Processo Unificado (*Rational Software Corporation, 2001*).

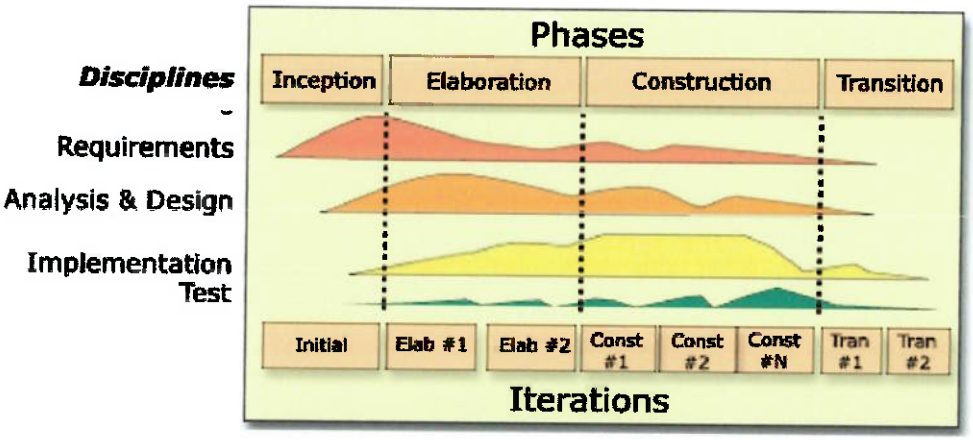



Figura 2-4 Ciclo de vida do Processo Unificado

## 2.3 Unified Modeling Language – UML

A UML é uma linguagem de modelagem para "especificação, construção e documentação de sistemas de software, bem como para a modelagem de negócios e

outros sistemas" (*Object Management Group, 2001*), e não uma metodologia para desenvolvimento de sistemas.

### 2.3.1 Histórico da UML

Em Junho de 1996 surgiu a UML 0.9, através de uma iniciativa dos principais pesquisadores sobre metodologias de desenvolvimento de sistemas e da empresas que atuavam na área de desenvolvimento de software. A "*UML Consortium*", que contava com parcerias importantes tais como HP, Oracle, Microsoft, dentre outras, lançou a UML 1.0 em Janeiro de 1997. Houve então a submissão de alguns trabalhos para padronização à OMG (*Object Management Group*), e em novembro de 1997 a versão 1.1 da UML foi aceita como padrão. A versão atual da UML é a 1.4 e já está em processo de homologação, pela OMG, a versão 2.0 (*Object Management Group, 2001*).

### 2.3.2 Objetivos da UML

A UML tem por objetivo padronizar uma notação para especificar, visualizar e documentar o sistema. Trata-se de uma fusão da forma gráfica de representação de modelos dos principais métodos desenvolvidos até o momento, onde se destacam os métodos de Booch (*Booch, 1991*), OMT (*Rumbaugh, 1991*) e OOSE (*Object Oriented Software Engineering*) (*Jacobson, 1992*). De acordo com (*Object Management Group, 2001*), os objetivos da UML são:

1. Fornecer aos usuários uma linguagem visual expressiva, pronta para o uso no desenvolvimento de modelos de negócios;
2. Fornecer mecanismos de extensibilidade e de especialização para apoiar os conceitos essenciais;
3. Ser independente da linguagem de implementação;
4. Prover uma base formal para entender a linguagem de modelagem;
5. Encorajar o crescimento do número de ferramentas orientadas a objetos;
6. Suportar conceitos de desenvolvimento de níveis mais elevados tais como colaborações, estrutura de trabalho, padrões e componentes;
7. Integrar as melhores práticas de desenvolvimento de software.

### 2.3.3 Vocabulário

A UML proporciona um vocabulário, que permite a comunicação entre os atores envolvidos no projeto, dividido em três categorias (*Jacobson, 1998*):

#### 2.3.3.1 Elementos

A primeira categoria possui quatro tipos de elementos, com seus respectivos subtipos:

1. Estrutural (*Structural*)
    a. *Use case*
    b. Classe
    c. Classe ativa
    d. Interface
    e. Componente
    f. Colaboração
    g. Nó
2. Comportamental (*Behavioral*)
    a. Interação
    b. Máquina de estado
3. Agrupamento (*Grouping*)
    a. Pacote
    b. Modelo
    c. Subsistema
    d. *Framework*
4. Anotações (*Annotational*)
    a. Nota

#### 2.3.3.2 Relacionamentos

Existem três tipos de relacionamentos na segunda categoria do vocabulário:

1. Dependência
2. Associação
3. Generalização

#### 2.3.3.3 Diagramas

Na terceira categoria, a UML dispõe de nove tipos de diagramas:

1. *Use case*
2. Classe
3. Objeto
4. Seqüência
5. Colaboração
6. *Statechart*
7. Atividade
8. Componente
9. *Deployment*

# 3 ESTUDO DE METODOLOGIAS DE PROJETO DE SOFTWARE

O mundo atual, seja nos negócios, saúde, governo ou entretenimento, está em grande parte fundamentado na utilização de sistemas automatizados que executam funções repetitivas, porém fundamentais. Por trás de todos esses processos estão os softwares.

Se por um lado isso é uma ótima perspectiva para os desenvolvedores de softwares por outro a expansão desses sistemas em tamanho, criticidade, complexidade e distribuição exigem cada vez mais conhecimentos técnicos e gerenciais para interfacear com sistemas legados e atender a crescentes demandas de produtividade e qualidade cumprindo prazos cada vez menores.

É neste contexto que metodologias de desenvolvimento de software se encaixam, auxiliando os desenvolvedores e gerentes de projetos a levantar requisitos, desenhar, desenvolver, coordenar e implantar sistemas que atendam a essas crescentes exigências com os recursos físicos e intelectuais disponíveis no mercado atual.

## 3.1 Metodologias Tradicionais de Software

Entende-se por metodologias por Ciclos de Vida de Desenvolvimento de Sistemas (*System Development Life Cycle -SDLC- methodologies*) os mecanismos criados para garantir que sistemas de software atinjam os requisitos estabelecidos. Estas metodologias aplicam diversos graus de instruções no processo de desenvolvimento com o objetivo de torná-lo mais eficiente e previsível.

Boas metodologias de Engenharia de Software sempre seguem quatro regras básicas:

- Fornecem uma ordem específica para as atividades das equipes envolvidas no projeto.
- Especificam quando e quais os artefatos que devem ser desenvolvidos durante o projeto.

- Direcionam as tarefas do desenvolvedor assim como de seu time como um todo.
- Fornecem critérios que sirvam para acompanhamento e como referência para os produtos e atividades do projeto.

Desta forma organizações maduras aplicam tais metodologias para desenvolver e implantar sistemas completos de uma forma previsível e utilizando uma forma iterativa. Assim é possível manter uma organização sustentável que possa se aperfeiçoar a cada projeto aumentando sua eficiência e produtividade.

As primeiras metodologias consagradas, denominadas de tradicionais, conseguiam de forma coordenada alcançar o objetivo de disciplinar o processo, evitando erros decorrentes da fase de eliciação de requisitos.

Porém por focar muito na geração de documentação, com um gasto de tempo relativamente alto, e com a demanda por ciclos de vida dos projetos cada vez menores, elas passaram a ser empecilhos para o desenvolvimento adequado de sistemas.

Outro problema encontrado nas metodologias tradicionais está no fato de seguirem uma seqüência rígida de etapas de projeto:

- Definição de requisitos.
- Modelo de um sistema que englobe todos os requisitos.
- Codificação.
- Testes.

Como vemos, toda etapa necessita que a etapa anterior seja completamente finalizada para se iniciar, ou seja, os processos tradicionais tentam planejar detalhadamente uma grande parte do processo por um período de tempo razoável, sendo cada requisito ou especificação tomada como a melhor escolha e com mínimas possibilidades de mudanças, estas como sabemos corriqueiras. Como exemplo de metodologias tradicionais citamos: *code and fix*, *waterfall* e *staged and phased*.

Em contrapartida surgiram as denominadas metodologias rápidas ou leves, que enfatizam o desenvolvimento das funcionalidades dos sistemas, dispensando parte

da documentação e abordando pontos de vistas baseados em iterações, diferentes das metodologias tradicionais. Isso garantiu que o processo se tornasse mais rápido e flexível a eventuais mudanças no decorrer do seu ciclo de vida.

Como parte dessas metodologias podemos citar : *Extreme Programming (XP)*, *Rational Unified Process (RUP)*, *Feature-Driven Development (FDD)*, *Agile Software Process (ASP)* dentre outras.

A Figura 3-1 mostra um comparativo dessas metodologias.

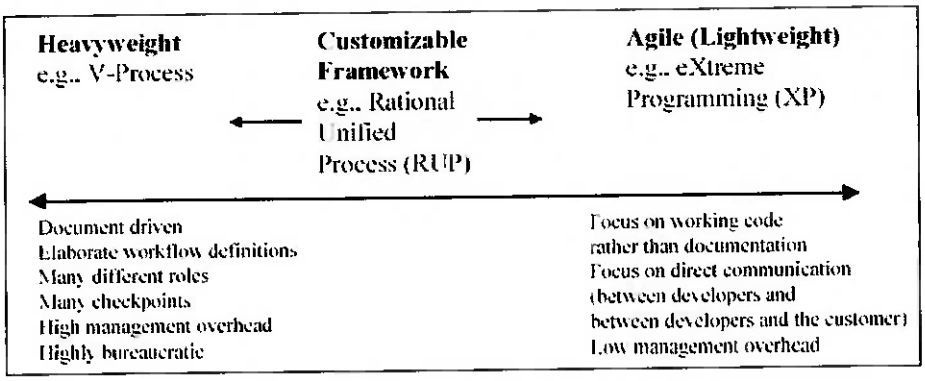



Figura 3-1 Metodologias leves x Metodologias Tradicionais

A seguir serão descritas as seguintes metodologias: Rational Unified Process , Extreme Programming e Feature Driven Development.

## 3.2 Rational Unified Process (RUP)

O RUP é uma metodologia de Engenharia de Software que permite a distribuição de tarefas e responsabilidades de uma forma sistemática dentro de uma organização. Seu objetivo é garantir a produção de softwares de alta qualidade que atendam às expectativas dos usuários finais dentro de cronograma e custos estimados.
Segundo *Lee Osterweil,* "Processos de software são softwares também", muitas organizações identificaram a necessidade de seguir processos de desenvolvimento bem definidos assim como a importância de documentá-los, entretanto na maioria das vezes esses documentos (livros ou artigos publicados, material de treinamento ou mesmo notas em um caderno) acabam acumulando poeira em prateleiras e em pouco tempo se tornam obsoletos. O RUP assemelha-se muito com um software devido aos seguintes fatores:

- A Rational Software constantemente libera atualizações.

- É uma ferramenta *online* utilizando tecnologia *web*.
- Pode ser customizado e configurado para atender às necessidades específicas de uma organização.
- Está integrado com muitas das ferramentas de desenvolvimento do Rational Suite permitindo ao desenvolvedor utilizar guias de processos da ferramenta que está utilizando.

Esta visão de tratar um processo de software como um software traz os seguintes benefícios:

- O processo nunca está obsoleto pois novas atualizações são constantemente liberadas.
- Todos os membros de um projeto têm acesso às últimas versões dos processos em uma intranet.
- Ferramentas permitem aos desenvolvedores encontrar facilmete orientação sobre processos assim como os *templates* que deve utilizar.
- Navegação via *hyperlinks* permite fácil e rápida navegação de uma parte para outra do processo assim como para outras ferramentas de desenvolvimento, referências e documentos.
- Processos especiais da empresa ou do projeto são facilmente inclusos no processo.
- Cada departamento ou projeto pode trabalhar com sua própria versão ou variante do processo.

Podemos dizer que todas as metodologias modernas de desenvolvimento permitem a aplicação das chamadas "*boas práticas de mercado*", o RUP é baseado em seis dessas práticas conforme pode ser observado na Figura 3-2.

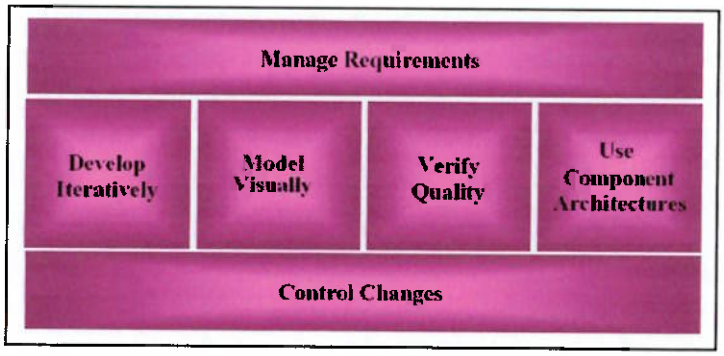



Figura 3-2 Seis *best practices* do mercado

Os items 3.2.1 a 3.2.6 irão descrever mais detalhadamente cada uma dessas práticas de mercado.

### 3.2.1 Desenvolver softwares iterativamente

O desenvolvimento iterativo geralmente é recomendado em detrimento de métodos lineares ou em cascata por uma série de razões:

- É possível levar em conta alterações dos requisitos, que de fato sempre sofrem alterações no decorrer do projeto. Requisitos alterados são comumente fontes de problemas com entrega do projeto, atrasos de cronograma e clientes insatisfeitos.
- A integração deixa de ser uma grande empreitada no final do projeto e passa a ser progressiva e contínua. Dessa forma algo que tomava grande parte do tempo ao final do projeto e que carregava consigo muita incerteza é quebrada em 6 ou menores integrações que são compostas de menos elementos e que podem ser tratadas mais facilmente.
- Devido ainda ao processo contínuo de integração, a identificação e tratamento de riscos é antecipada, pois geralmente é nesse processo que são identificados os riscos.
- Fornece um meio para mudanças táticas durante o projeto permitindo, por exemplo, liberar antes uma versão reduzida do sistema para obter vantagens competitivas.
- Facilita o reaproveitamento, pois fica fácil de identificar partes comuns visto que são parcialmente desenhadas ou implementadas a cada iteração.
- O resultado final é uma versão robusta, pois os erros são corrigidos em várias iterações, realizando testes em versões mais simples ao invés de uma etapa massiva de testes ao final do projeto. Dessa forma são evitadas surpresas funcionais e de performance na véspera da entrega final.
- Pessoal de desenvolvimento, teste e treinamento passam a trabalhar antes e de forma conjunta durante todo o projeto identificando falta de recursos antes durante avaliações mais constantes.
- A avaliação ao final de cada iteração promove discussões que podem gerar ações a serem tomadas que melhorem o andamento da próxima iteração.

- Ao fim de cada iteração uma versão executável e funcional é liberada para o cliente de forma que este possa usufruir mais rapidamente dos benefícios gerados pela ferramenta assim como se adaptar de forma progressiva e sem grandes impactos sobre sua operação normal.

A Figura 3-3 mostra a estrutura de um processo iterativo e incremental onde ao fim de cada iteração uma versão executável é liberada para o cliente.

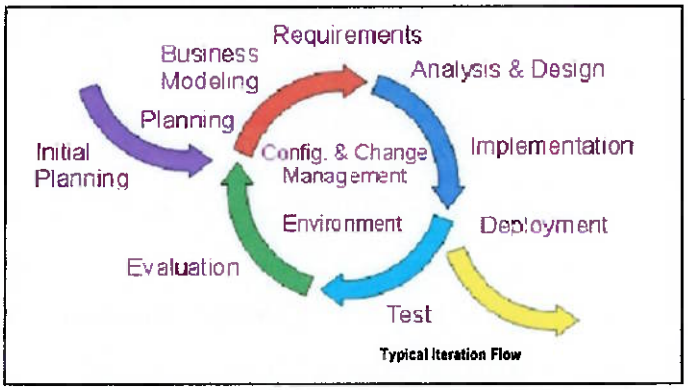



Figura 3-3 Processo iterativo e incremental

O RUP divide cada ciclo de desenvolvimento em 4 fases, a Figura 3-4 ilustra essas fases assim como seus principais *milestones*.

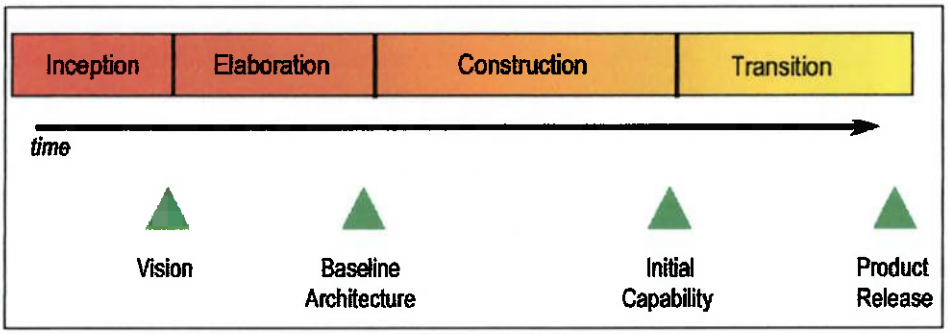



Figura 3-4 Principais Milestones de cada fase

- **Concepção**
- **Elaboração**
- **Construção**
- **Transição**

Abaixo se apresenta uma descrição mais detalhada de cada uma dessas fases:

### *3.2.1.1 Concepção*

Durante esta etapa se estabelece o *business case* do projeto e define-se seu escopo. Para tanto, devem-se identificar todas as entidades com as quais o sistema interage (atores) e definir a natureza de tais interações. Essa tarefa envolve a definição de todos os *use cases*.

O *business case* deve conter os critérios de sucesso, avaliação dos riscos, estimativa dos recursos necessários assim como um plano das fases da iteração definindo os principais *Milestones* e suas respectivas datas de entrega.

Espera-se como produtos gerados desta fase:

- *Vision*, ou seja, um documento que apresente uma visão geral dos principais requisitos, *features* chaves e principais restrições do projeto.
- Modelo inicial de *use cases* (10~20%).
- Glossário inicial do projeto.
- *Business Case* inicial do projeto contendo contexto do projeto, critérios de sucesso e orçamento esperado.
- Avaliação inicial dos riscos do projeto.
- Modelo de negócios se necessário.
- Protótipos do projeto.

Os critérios de avaliação desta fase são:

- Aceitação dos *stakeholders* com relação ao escopo definido assim como orçamento e cronograma estimados.
- Compreensão dos requisitos do projeto conforme deve ser demonstrado nos modelos iniciais de *use cases*.
- Credibilidade das estimativas de orçamento e cronograma, prioridades, riscos e processos de desenvolvimento.
- Comparação dos gastos atuais com os gastos planejados.

#### *3.2.1.2 Elaboração*

O objetivo desta fase é analisar o domínio e estabelecer a arquitetura do problema, desenvolver o plano e eliminar os elementos de maior risco do projeto. Para tanto é necessário ter uma visão bastante ampla do sistema compreendendo seu escopo, principais funcionalidades e requisitos não funcionais como performance, por exemplo.

Esta fase é bastante crítica no projeto e geralmente determina se é viável ou não prosseguir para as fases seguintes. Nesta fase deve-se ser capaz de assegurar que a

arquitetura, requisitos e planos são estáveis o suficiente e os principais riscos foram devidamente tratados a ponto de dar credibilidade ao orçamento e cronograma estipulados para o projeto.

Um protótipo executável da arquitetura deve ser construído incluindo pelo menos os *use cases* mais críticos e respectivos riscos identificados na fase anterior. Embora seja desejável que apenas um protótipo seja criado e evolua progressivamente até a solução final, protótipos "descartáveis" podem ser necessários para analisar riscos individualmente ou para realizar apresentações para investidores e *stakeholders*.

Espera-se como produtos gerados desta fase:

- Modelos de *use cases* (pelo menos 80% completo), todos atores identificados e descrições elaboradas.
- Definição dos requisitos não funcionais e de outros requisitos não associados aos *use cases*.
- Descrição da arquitetura do software.
- Protótipo executável da arquitetura.
- Lista revisada dos riscos e modelo de *business case* revisado.
- Plano de desenvolvimento detalhado do projeto contendo descrição das iterações e respectivos critérios de sucesso.
- *Development case* atualizado especificando o processo a ser utilizado.
- Pode-se começar a escrever o manual do usuário.

Os critérios de avaliação desta fase são:

- Estabilidade do escopo do projeto (*Vision*).
- Estabilidade da arquitetura do projeto.
- Tratamento e resolução dos principais riscos demonstrados no protótipo.
- Acurácia e nível de detalhe do plano da fase de construção e credibilidade das estimativas.
- Aceitação de todos *stakeholders* de que os pontos identificados no escopo (*Vision*) podem ser alcançados seguindo o plano de ação estabelecido.
- Os gastos com recursos atuais são aceitáveis perante os gastos estimados.

### *3.2.1.3 Construção*

Durante esta fase os componentes e *features* do sistema restantes são desenvolvidos e integrados ao produto e todas as *features* são testadas exaustivamente. Nesta fase ocorre uma transição do desenvolvimento intelectual para um desenvolvimento físico e palpável de produtos.

Espera-se como produtos gerados desta fase:

- O produto final (software) integrado com as plataformas do sistema.
- Manual do usuário.
- Descrição da versão liberada.

Os critérios de avaliação desta fase são:

- Estabilidade e maturidade do sistema desenvolvido para entrada definitiva em produção.
- Condição de preparo dos usuários para operar o sistema em ambiente de produção.
- Os gastos atuais comparados aos planejados ainda devem ser aceitáveis.

### *3.2.1.4 Transição*

O objetivo desta fase é realizar a transição do sistema para os usuários finais e através do seu *feedback* desenvolver novas *releases* que atendem às novas *issues* ou erros levantados nos primeiros dias de operação.

Para entrar nesta fase é necessário que um certo nível de maturidade do sistema seja alcançado e que haja documentação do sistema para realizar a transição para o usuário. Dessa forma nesta fase deve haver:

- Versão *beta* de teste para validar se o sistema atende às expectativas dos usuários.
- Operação em paralelo com o sistema legado que será substituído.
- Migração dos bancos de dados em operação.
- Treinamento dos usuários e do suporte do sistema.
- Liberar o produto para os setores de vendas, marketing e distribuição.

Espera-se como produtos gerados desta fase:

- Aceitação dos *stakeholders* de que todos os critérios de avaliação definidos no escopo (*Vision*) foram atendidos.
- Utilização auto-sustentável por parte dos usuários.

Os critérios de avaliação desta fase são:

- O usuário está satisfeito?
- Os gastos atuais comparados aos planejados ainda devem ser aceitáveis.

### 3.2.2 Gerenciar Requisitos

Gerenciamento de requisitos é uma forma sistemática de levantar, organizar e comunicar requisitos assim como lidar com os pedidos de mudanças de requisitos de uma aplicação. Os benefícios de um gerenciamento de requisitos eficaz são:

- **Melhor controle de projetos complexos**, pois a falha na compreensão do comportamento desejado do sistema é um fato corriqueiro em projetos fora de controle.
- **Melhor qualidade do software e maior satisfação do cliente**, pois somente com um entendimento entre todos *stakeholders* sobre o que deve ser construído e testado pode-se desenvolver um sistema que faça o que deve fazer.
- **Custos de projeto e atrasos reduzidos**, consertar erros de requisitos é geralmente bastante caro e consome muito tempo, reduzir estes erros no começo do ciclo de desenvolvimento reduz custos e a probabilidade de atrasos.
- **Melhor comunicação das equipes**, gerenciamento de requisitos facilita o envolvimento de usuários no princípio do processo ajudando a assegurar que o aplicativo atinja suas necessidades. Requisitos bem gerenciados permitem o entendimento comum das necessidades do projeto e maior comprometimento entre os *stakeholders*.

### 3.2.3 Utilizar arquiteturas baseadas em componentes

O foco das primeiras iterações é produzir e validar a arquitetura do software que no início do ciclo de desenvolvimento toma a forma de um protótipo executável que gradualmente evolui até se tornar o sistema final nas últimas iterações. O RUP fornece uma forma metódica e sistemática de desenhar, desenvolver e validar a arquitetura do software através de *templates*.

Um componente de software pode ser descrito como um "pedaço" não trivial de software, um módulo, pacote ou subsistema que cumpra uma função definida, tenha fronteiras bem definidas e possa ser integrado a uma arquitetura definida. Desenvolvimento baseado em componentes oferece as seguintes vantagens:

- Definindo uma arquitetura modular, é possível identificar, isolar, desenhar, desenvolver e testar cada módulo. Estes componentes podem dessa forma serem testados individualmente e integrados gradativamente até compor o sistema final.
- Esses componentes podem posteriormente serem reutilizados, especialmente os componentes que solucionem problemas comumente encontrados. Formam a base da reutilização em organizações aumentando a produtividade e qualidade.
- Atualmente existem infra-estruturas comerciais que suportam o conceito de componentes como CORBA, ActiveX e JavaBeans que possibilitem ao programador comprar e integrar componentes aos seus sistemas ao invés de desenvolver todas as linhas de código do sistema.

O RUP suporta arquiteturas baseadas em componentes da seguinte maneira:

- O processo iterativo permite que desenvolvedores identifiquem progressivamente componentes e decidam quais devem ser desenvolvidos, quais devem ser reutilizados e quais devem ser comprados.
- O foco na arquitetura de software permite articular a estrutura. A arquitetura enumera componentes e a forma como se integram e interagem entre si.
- Conceitos como pacotes, subsistemas e camadas são utilizados durante a análise e desenho para organizar componentes e especificar suas interfaces.

- Testes são organizados de forma a englobar desde componentes isolados expandindo até incluir um número maior de componentes integrados.

### 3.2.4 Modelar softwares visualmente

Modelos são abstrações da realidade que nos ajudam a lidar com um sistema complexo que não pode ser compreendido em sua totalidade. Grande parte do RUP está relacionado com o desenvolvimento e manutenção de modelos do sistema que auxiliam na compreensão e desenho tanto do problema como de sua solução. Exemplos de modelos utilizados pelo RUP são modelos de: *use cases, business models, design model, analysis model* e *test model*, conforme pode ser observado na Figura 3-5.

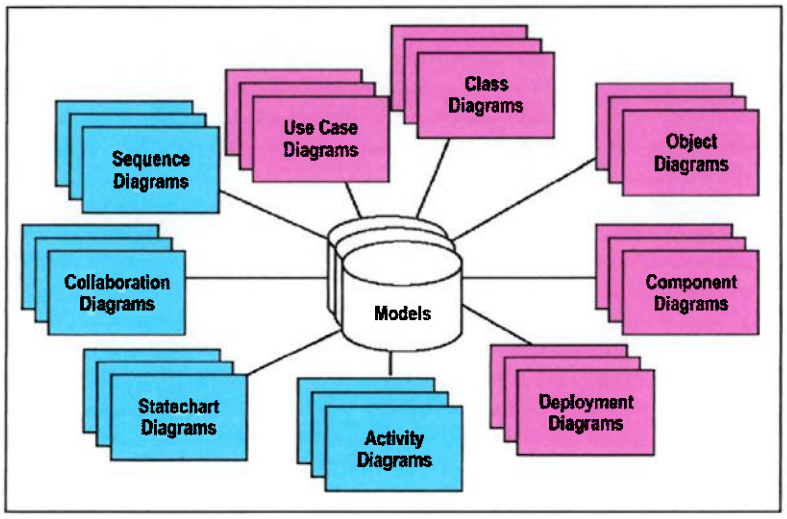



Figura 3-5 Base de Modelos unificada

A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos, fornecendo um padrão para descrever visualmente o sistema desde processos de negócios e funções do sistema até artefatos concretos como classes escritas em determinada linguagem de programação, *schema* de banco de dados e pedaços de software reutilizáveis.

A UML é a linguagem para descrever os modelos, porém não diz como desenvolver um software, é como se fornecesse o vocabulário porém não a gramática de uma língua.

Para isso a Rational desenvolveu o RUP que é um guia de utilização efetiva da UML para modelagem, descrevendo quais modelos são necessários, por que e como construí-los.

### 3.2.5 Verificar a qualidade do processo e produto

No RUP qualidade é uma responsabilidade de todos os membros da organização. No desenvolvimento de software há grande preocupação em duas áreas:

- **Qualidade do produto**, principal produto entregue (software) e seus elementos (componentes, subsistemas, arquitetura...).
- **Qualidade do processo**, o grau no qual um processo aceitável foi implementado e aderido durante o desenvolvimento do produto. A qualidade do processo está ainda preocupada com a qualidade dos artefatos (planos de iteração, planos de teste, *use case*, *design model*...) produzidos como suporte ao produto principal.

Um processo define quem está fazendo o que, quando e como no desenvolvimento de um software.

O foco do RUP, entretanto, está em objetivamente verificar e avaliar se o produto atende o nível esperado de qualidade através de fluxogramas de teste e outros processos e métricas.

### 3.2.6 Controlar alterações no software

O processo iterativo de desenvolvimento proporciona flexibilidade no planejamento e execução do desenvolvimento e permite que os requisitos evoluam com o tempo, a partir disso, o RUP atende aos principais pontos para o rastreamento de mudanças e assegura que tudo e todos estejam sincronizados.

Focado nas necessidades de uma organização de desenvolvimento, a gestão de mudanças é um processo sistemático para lidar com alterações nos requisitos, projeto e implementação. Também possibilita o controle dos defeitos, desentendimentos e compromissos endereçando-os para os corretos artefatos e *releases*.

É essencial manter o controle sobre as alterações e solicitações de mudança assegurando que as mudanças sejam aceitáveis. Também é possível estabelecer espaços "seguros" ou isolados para que desenvolvedores possam trabalhar sem serem afetados por alterações em outras partes do sistema e/ou ambiente.

### 3.2.7 Integração com outras ferramentas

Uma metodologia de desenvolvimento de software requer ferramentas para auxiliar no controle de todas as atividades do ciclo de vida de um sistema como desenvolvimento e manutenção.

O RUP pode ser utilizado com uma grande variedade de ferramentas sejam elas da Rational Software ou não. Abaixo serão listadas algumas ferramentas que compõem o Rational Suite e que podem ser facilmente integradas com o RUP. O RUP possui *tool mentors*, ou seja, um guia que explica passo a passo como utilizar a ferramenta, para quase todas essas ferramentas.

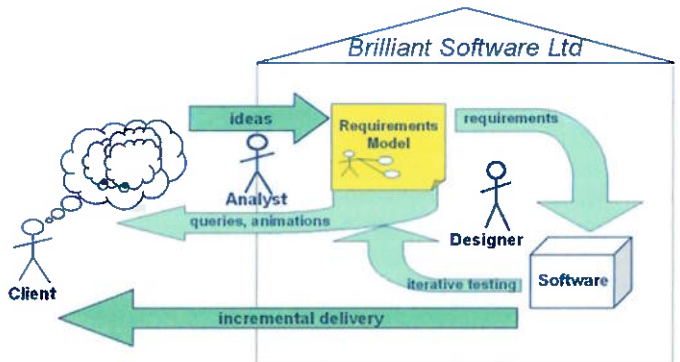
- **Rational Requisite®Pro** – Mantém toda a equipe de desenvolvimento atualizada e direcionada durante todo o processo desenvolvimento tornando os requisitos fáceis de se escrever, comunicar e alterar.
- **Rational ClearQuest™** – Produto *web* de gerenciamento de pedidos de alteração, que permite que as equipes do projeto encontrem e gerenciem todas as atividades de alteração que ocorrem durante o ciclo de desenvolvimento.
- **Rational Rose®** – Ferramenta líder de mercado em modelagem visual de processos de negócio, análise de requisitos e desenho de arquiteturas baseadas em componentes.
- **Rational SoDA®** - Automatiza a produção de documentação de todo o processo de desenvolvimento de software, reduzindo dramaticamente o tempo e custo desta tarefa.
- **Rational Purify®** - Ferramenta de verificação de erros de compilação (*run-time error*), para desenvolvedores programando em C/C++, ajuda a detectar erros de memória.

- **Rational Visual Quantify™** – Ferramenta de análise de performance para desenvolvedores programando em C++, Visual Basic e Java, auxilia na identificação de gargalos.
- **Rational Visual PureCoverage™** – Ferramenta que identifica automaticamente trechos do código que não foram englobados durante a rotina de testes, permitindo que os desenvolvedores possam testar suas aplicações exaustivamente.
- **Rational TeamTest** – Ferramenta que cria, mantém e executa testes funcionais, permitindo que desenvolvedores testem exaustivamente os aplicativos e analisem os resultados para ver se atendem aos requisitos impostos e se possuem uma performance aceitável.
- **Rational PerformanceStudio™** – Ferramenta escalonável e fácil de usar que mede e prevê a performance de um sistema *web* cliente/servidor.
- **Rational ClearCase®** - Ferramenta líder de mercado em gerenciamento de configuração de software, permitindo ao gerente de projeto acompanhar a evolução de todos projetos de desenvolvimento de software.

## 3.3 Extreme Programming

*Extreme Programming* ou XP é uma nova metodologia de desenvolvimento rápido de softwares. Ela conta com um *feedback* rápido e um eficiente canal de comunicação como formas de maximizar o valor agregado entregue ao cliente através da alocação no cliente, um planejamento particular e testes constantes.

A Figura 3-6 ilustra como funciona o XP e apresenta em um fluxograma da metodologia.



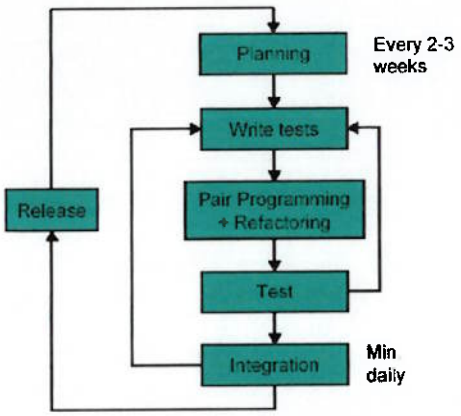



Figura 3-6 Funcionamento do XP

O XP enfatiza o envolvimento do cliente e um grande foco nos testes. Dessa forma o cliente procura uma equipe de desenvolvimento para iniciar um projeto. A equipe pede para que ele permaneça junto ao time durante o desenvolvimento. O projeto desta forma é composto de três fases:

- Uma fase de planejamento da *release*, onde o cliente descreve *stories* que são as funcionalidades de que necessita, a equipe avalia e elabora essas *stories* e o cliente seleciona a seqüência em que serão desenvolvidas.
- Uma fase de iteração, onde o cliente escreve scripts de testes e responde a questões levantadas, enquanto que os desenvolvedores programam.
- Uma fase de entrega do produto, onde os desenvolvedores instalam o sistema gerado e o cliente dá a aprovação do sistema.

Dessa forma, devido ao grande envolvimento do cliente com o desenvolvimento, ele possui grande liberdade de redirecionar a equipe de desenvolvimento se as circunstâncias mudarem. Isso é possível, pois a equipe desenvolvedora entrega pacotes do sistema a cada duas ou três semanas e como é dada muita ênfase nos testes e o envolvimento do cliente nestes é essencial ele está sempre alerta do real *status* do projeto.

Para o XP há uma divisão bem clara das funções do cliente e do desenvolvedor. Os dois estão no mesmo time, porém o primeiro está preocupado com "o que é entregue" e o segundo com "qual o custo disto". A Tabela 3-1 mostra quais as decisões a serem tomadas por cada um no projeto.

| Decisões do Cliente | Decisões do Desenvolvedor |
|---|---|
| **Escopo:** o que o sistema deve fazer. | **Estimativa do tempo** necessário para adicionar uma *feature* ao sistema. |
| **Prioridades:** o que é mais importante. | **Implicações Técnicas:** a equipe de desenvolvimento explica as consequências das opções, mas é o cliente quem faz a escolha. |
| **Conteúdo das *releases*:** o que cada *release* deve entregar para agregar valor. | **Processos:** como a equipe irá trabalhar. |
| **Datas de Entrega:** quando as *releases* devem ser entregues. | **Cronograma Detalhado** de cada iteração. |

Tabela 3-1 Decisões tomadas por Cliente e Desenvolvedor

As práticas do XP enfatizam essa distinção das funções de cliente de desenvolvedores. Esta divisão de trabalho ajuda a manter toda a equipe bem informada fazendo com que as conseqüências das decisões tomadas sejam visíveis e claras a todos.

### 3.3.1 XP por camadas

O XP pode ser entendido como uma cebola, ou seja, possui diversas camadas.

A camada mais interna é a de desenvolvimento, o XP utiliza um estilo particular de desenvolvimento possível mesmo para um desenvolvedor apenas. A camada central consiste de um conjunto de práticas orientadas à equipe. A camada mais externa define os processos através dos quais uma equipe de desenvolvimento interage com o cliente. A Figura 3-7 ilustra essa configuração.

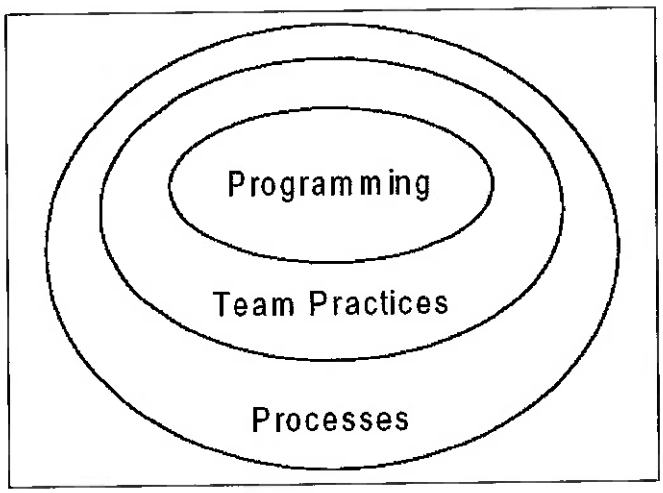



Figura 3-7 Camadas do XP

### 3.3.2 XP através das "12 práticas"

Uma outra forma de se entender o XP é como um conjunto de práticas. Em *Extreme Programming Explained, Kent Beck* enumera um conjunto de doze práticas principais que servem como um ponto de partida para uma equipe XP. A Tabela 3-2 enumera essas práticas relacionando-as com as camadas definidas anteriormente.

| | |
|---|---|
| Desenvolvimento | Desenho simples. |
| | Teste. |
| | *Refactoring*[*] |
| | Padronização do desenvolvimento. |
| Práticas de equipe | Autoria coletiva.[**] |
| | Integração contínua. |
| | *Metaphor.* |
| | Padronização de desenvolvimento. |
| | Semana de 40 horas. |
| | Programação em pares. |
| | *Releases* pequenas. |
| Processos | Alocação no cliente. |
| | Teste. |
| | *Releases* pequenas. |
| | Jogo de planejamento. |

Tabela 3-2 Práticas do XP e referência por camadas

[*] *Refactoring é melhorar o desenho de código já existente. É a melhor forma de limpar o código.*
[**] *Entenda-se por autoria coletiva que todo membro da equipe de desenvolvimento que identificar uma oportunidade de agregar valor a uma porção do código tem por dever fazê-lo.*

Notar que há sobreposição visto que algumas práticas atravessam as camadas.

Por trás destas práticas estão quatro valores: simplicidade, comunicação, *feedback* e coragem.

### 3.3.3 XP como Desenvolvimento

Desenvolvedores XP escrevem código de forma incremental e direcionados a cumprir scripts de teste pré-estabelecidos. Dessa forma são elaborados pequenos testes unitários e é codificado apenas o suficiente para atender a esses testes.

### 3.3.4 XP como Práticas Orientadas a Equipe

Vamos detalhar as práticas orientadas a equipe e algumas alternativas a elas: autoria de código, integração, horas extras, espaço de trabalho, cronograma e padrões de desenvolvimento.

Desenvolvimento em pares permite desenho e revisão de código durante a programação de cada linha de código.

O XP não enfatiza a arquitetura como um mecanismo chave, porém seus programas têm uma arquitetura. O *metaphor*, o desenho e o processo de desenvolvimento para a arquitetura de um programa XP. O *metaphor* fornece um *framework* conceitual para o sistema.

*Metaphor* entende-se por uma visão comum a todos no time de desenvolvimento de como o programa funciona.

### 3.3.5 XP como Processos

A alocação no cliente e os teste de aceitação automatizados são aspectos importantes do XP.

Um cliente de projetos XP permanece com a equipe de desenvolvimento em tempo integral para escrever scripts de testes, responder a dúvidas e indicar prioridades. A presença do cliente é essencial para auxiliar a equipe andar o mais rápido possível. É necessário tomar muitas decisões para desenvolver um software, se um time pode ter acesso a respostas ou decisões em minutos ao invés de horas ou dias a velocidade de desenvolvimento aumenta substancialmente.

XP utiliza dois tipos de teste: clientes elaboram testes de aceitação que determinam o comportamento desejado do sistema e programadores elaboram testes unitários enquanto programam.

O cliente cria testes para cada *story* requisitada. Equipes XP medem o progresso do desenvolvimento executando esses testes.

Finalmente a equipe deve se empenhar para ter seus testes automatizados. Isso pode significar que o cliente irá querer gastar parte do tempo de desenvolvimento

construindo uma infra-estrutura de testes. As telas não necessitam serem elaboradas, devem apenas conter as entradas (*inputs*) e o resultado esperado. Algumas equipes são capazes de resolver esse problema com planilhas comuns.

XP usa a noção de um jogo de planejamento para planejar a *release* e iterações. O jogo apresenta dois jogadores, cliente e desenvolvedor e define que jogador pode realizar que ação. Dessa forma o processo mantém uma divisão crítica: Clientes determinam valor e Desenvolvedores determinam custo.

No jogo de Planejamento de *Release* o objetivo é definir um conjunto de *features* para a próxima *release*. Ele está centrado nas *stories* definidas pelo usuário. O Cliente escreve *stories* e os Desenvolvedores analisam e o Cliente planeja a *release*.

O jogo de Planejamento de Iteração é similar. O Cliente seleciona as *stories* para a iteração e os desenvolvedores analisam e aceitam as tarefas definidas.

### 3.3.6 Conclusões sobre o XP

Extreme Programming é uma metodologia de desenvolvimento de software baseado em valores de simplicidade, comunicação, *feedback* e coragem. Funciona mantendo toda a equipe unida através de práticas simples, com um bom *feedback* para permitir que a equipe visualize onde está e sintonizar as práticas para sua situação particular.

## 3.4 Feature Driven Development

O Feature-Driven Development (FDD) é um processo de desenvolvimento de software composto de pequenas iterações dirigidas ao modelo, onde cada funcionalidade deste é listada, analisada e criada em diversas etapas. O método tenta enfatizar as funcionalidades mais importantes e visíveis do sistema do ponto de vista do cliente, sendo definidas pelo próprio em sentenças simples e diretas.

Podemos citar alguns benefícios desse tipo de abordagem:

- Uso de pequenos módulos para cada funcionalidade do cliente (*feature*).
- Organização em pequenos grupos de trabalho relacionados a cada aspecto.
- Desenvolvedores são direcionados a gerarem resultados em ciclos curtos (em torno de duas semanas).

- Facilidade de inspeções.
- Facilidade de demonstração de resultados aos clientes.
- Proporciona desenvolvimento em paralelo de cada funcionalidade.

### 3.4.1 Estrutura das equipes de desenvolvimento

Há seis grupos integrantes da fase de projeto, que trabalham juntos para alcançar os melhores resultados: o *Development Manager* (gerente de desenvolvimento), o *Chief Architect* (arquiteto chefe), o *Chief Programmer* (programador chefe), o cliente do sistema, o *Class Owner* (proprietário da classe) e as *Feature Teams* (equipes de desenvolvimento).

Cada integrante desempenha um papel fundamental na metodologia:

- **Gerente de desenvolvimento**, coordenada o andamento do projeto;
- **Arquiteto chefe**, deve ser escolhido dentro da equipe de projeto como o mais experiente na modelagem de sistemas, tendo como principal função a integração e orientação das equipes de projeto subordinadas à ele, portanto num mesmo projeto podem existir mais de um arquiteto chefe dependendo do grau de desenvolvimento do sistema.
- **Programador chefe**, deve ser escolhido entre os mais produtivos das equipes de trabalho, sendo definido como o chefe das equipes.
- **Cliente do sistema**, responsável por fornecer as impressões gerais do sistema a ser implementado, indicando suas funcionalidades gerais, importantes na fase inicial de modelagem do sistema.
- **Proprietário da classe**, responsável pelo desenvolvimento de determinada classe no projeto, garantindo consistência na classe local.
- **Equipes de desenvolvimento**, são formados temporariamente entre os proprietários das classes e um arquiteto chefe, dessa forma como o diálogo entre cada time é possível garantir a uniformidade no projeto e implementação.

### 3.4.2 Etapas de desenvolvimento

Para alcançar tais resultados o FDD é composto por cinco fases de desenvolvimento: definição de um modelo, listagem, planejamento, projeto e construção de cada funcionalidade.

A Figura 3-8 ilustra a seqüência dessas fases.

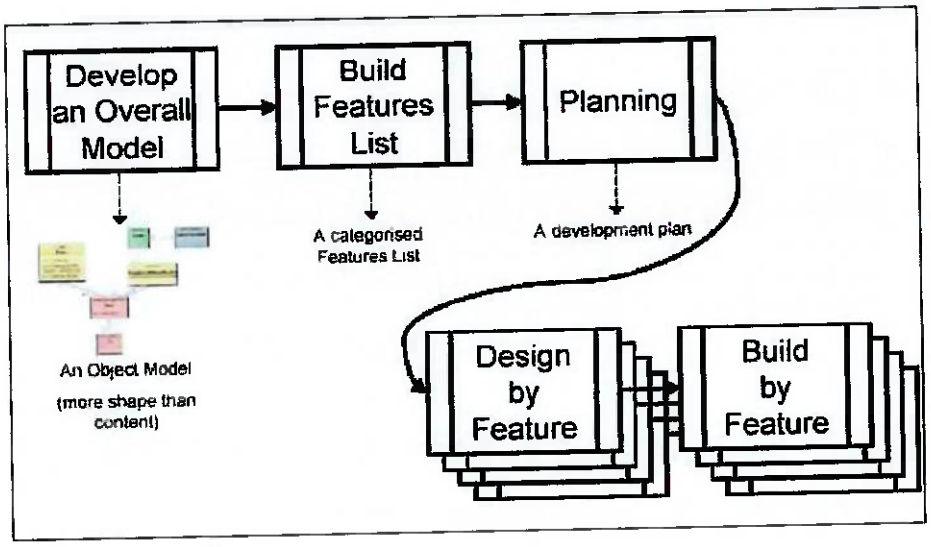



Figura 3-8 Processo FDD

A seguir apresenta-se uma breve introdução à cada fase:

#### *3.4.2.1 Definição do modelo*

Na primeira atividade, clientes e desenvolvedores trabalham juntos, sob a orientação de um experiente projetista de sistemas, o arquiteto chefe. O cliente apresenta uma descrição inicial do sistema, incluindo o escopo do projeto e o contexto envolvido, para que junto aos desenvolvedores criem esboço do sistema.

Parte-se então para uma descrição mais detalhada e geração de um modelo detalhado. Nessa fase procura-se dividir o projeto em pequenos grupos de trabalho, cada um modelando determinados aspectos do sistema, que serão ao final integrados (sob tutoria do arquiteto chefe) para formar o modelo geral.

Como critérios de aprovação dessa fase, a equipe deve entregar os seguintes resultados, sujeitos à revisão e aprovação do gerente de desenvolvimento e do arquiteto chefe:

- Diagramas de classes com respectivas classes, relacionamentos, métodos e atributos. Classes e relacionamentos estabelecem a forma do modelo, e métodos são a base para a construção de uma lista de funcionalidades;
- Lista informal de funcionalidades;
- Notas sobre modelos alternativos.

### *3.4.2.2 Definição da lista de funcionalidades*

Usando o conhecimento adquirido durante a fase anterior, a equipe gera uma lista de funcionalidades para o sistema (seguindo a definição apresentada anteriormente), sem restrições. Pode-se usar como dados de entrada documentos de requisitos, quando existentes, tais como especificações use case ou funcionais.

Essas funcionalidades são então agrupadas de acordo com suas semelhanças e, para sistemas grandes, essas listas também podem ser agrupadas em níveis mais altos. Junto aos clientes do sistema, essas listas sofrem sucessivas análises e classificações que definem funcionalidades principais e mínimas que o sistema deve possuir.

As funcionalidades devem ser especificadas tal que seja possível seu desenvolvimento em ciclos de duas semanas, não existindo assim ciclos de desenvolvimento que durem muito mais tempo.

Ao fim da fase, as equipes devem entregar a lista detalhada de funcionalidades, agrupadas de acordo com a classificação acima definida, sujeita novamente à revisão e aprovação do gerente e do arquiteto chefe.

### *3.4.2.3 Planejamento por funcionalidade*

A terceira fase consiste no planejamento mais detalhado de cada funcionalidade, incluindo a definição de classes e objetos a serem usados, sendo apresentados ao programador chefe, o qual é responsável pela divisão do desenvolvimento das classes entre os desenvolvedores.

Como finalização desse processo, a equipe deve produzir um planejamento de projeto (sujeito a revisão e aprovação do gerente e arquiteto chefe) contendo os seguintes dados:

- Prazos finais de desenvolvimento;
- Para cada funcionalidade, definição dos programadores chefes e seu prazo de término;
- Definição dos proprietários das classes.

### *3.4.2.4 Projeto / Construção por funcionalidade*

Essa fase consiste na criação propriamente dita do sistema. O desenvolvimento é dividido em pequenos ciclos (em torno de duas semanas) denominados iterações, responsáveis pela implementação das funcionalidades definidas pelo arquiteto chefe, cada iteração é composta por duas etapas: *Design by Feature* (DBF) e *Build by Feature* (BBF) conforme Figura 3-9.

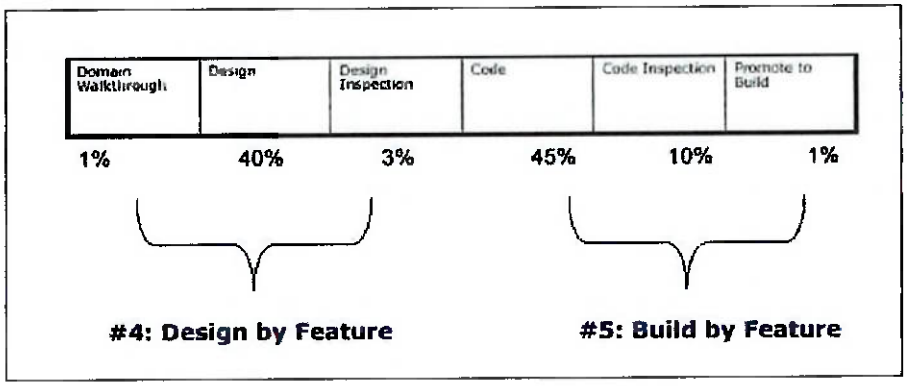



Figura 3-9 Iteração no FDD

Na primeira fase (DBF) são listadas as classes envolvidas em cada funcionalidade, formando-se sub-equipes de desenvolvimento entre os proprietários de cada classe. Essas sub-equipes então partem para a descrição das funcionalidades através da definição de métodos e diagramas de seqüência.

Ao final dessa etapa, a equipe deve apresentar como resultados, sujeitos a revisão a aprovação:

- Apresentação e documentação da funcionalidade;
- Diagramas de seqüência detalhados;

- Mudanças no diagrama de classes, classes e métodos;
- Notas sobre melhorias no projeto.

Depois de aprovadas as definições, em revisão feita por toda equipe, cada proprietário deve implementar as mudanças na sua classe, de forma a garantir o funcionamento local correto (fase 2-BBF) através de diversas inspeções e testes, restando ao programador chefe a última análise e tomada de decisão de integração do código ao sistema principal.

Para finalizar o processo, a equipe de desenvolvimento deve apresentar ao programador chefe:

- Métodos implementados, inspecionados e testados;
- Resultados de testes, para cada método e sobre a seqüência inteira;
- Checagem das classes e implantação no sistema principal.

Ao final da etapa o programador chefe entrega a funcionalidade para implementação.

As equipes de desenvolvimento podem ser definidas conforme a melhor escolha dada pelos programadores chefe, podendo este dividí-las da maneira mais produtiva conforme o decorrer do projeto. A Figura 3-10 dá um exemplo de possíveis divisões das equipes.

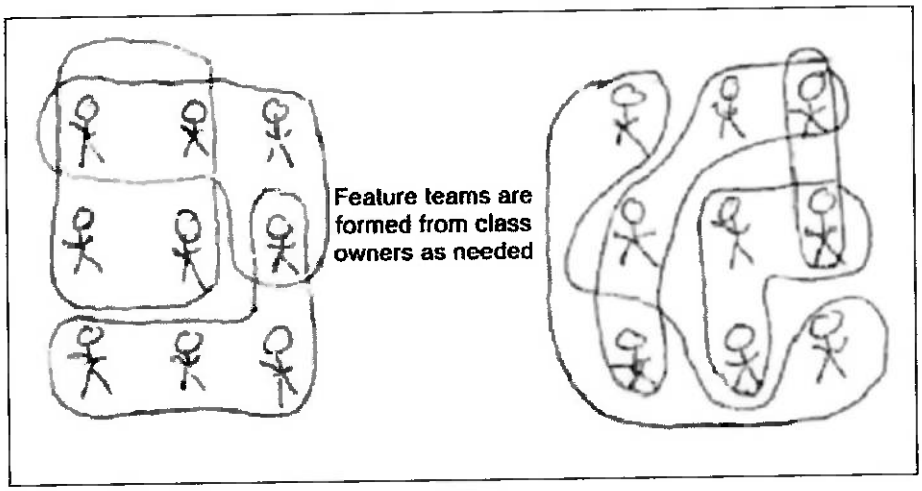



Figura 3-10 As equipes de trabalho

### 3.4.3 Considerações finais sobre o FDD

A principal vantagem de aplicar tais procedimentos e/ou método é a flexibilidade e robustez que o sistema consegue frente às eventuais mudanças, pois cada funcionalidade tem seu desenvolvimento independente. Com diversas verificações feitas ao longo do projeto é possível detectar os pontos de falhas ou pontos críticos do projeto, alterando-os de modo a satisfazer com maior eficiência os requisitos definidos.

O cronograma é definido pelo arquiteto chefe, sendo o andamento descrito através de tabelas e outros documentos, onde se exibe para cada funcionalidade a porcentagem de implementação concluída e os prazos a serem cumpridos. Nesse ponto o FDD apresenta diversos padrões para facilitar a inspeção até mesmo do ponto de vista do cliente do sistema, dentre eles citamos a divisão por cores do andamento da implementação das funcionalidades: cor verde fase completa, cor azul fase em andamento, cor vermelha fase que requer atenção.

Dessa forma, a metodologia busca integrar as equipes de desenvolvimento com os clientes facilitando a troca de informações entre eles, ressaltando a importância do arquiteto chefe no andamento do projeto.

Mais detalhes sobre a metodologia Feature-Driven Development podem ser encontrados na referência bibliográfica [1] onde existem informações que detalham os procedimentos que cada membro da equipe deve seguir, e os documentos que cada fase deve gerar.

## 3.5 Comparação entre as metodologias

Apresentadas as três metodologias selecionadas para estudo, segue-se uma comparação entre elas para escolha da metodologia adequada no contexto de desenvolvimento do projeto.

### 3.5.1 Comparação entre FDD e XP

A metodologia FDD foi desenvolvida para atender principalmente aos prazos de desenvolvimento de sistemas sendo direcionada para o desenvolvimento das suas funcionalidades, sem no entanto se preocupar com documentação em excesso.

A metodologia XP, da mesma forma preza pelo desenvolvimento funcional do sistema em prazos curtos, porém sua aplicabilidade é voltada adequadamente para o desenvolvimento de sistemas pequenos, exigindo para tanto equipes menores.

As duas metodologias buscam aumentar a integração entre clientes e desenvolvedores através da melhoria da comunicação entre ambas as partes.

O FDD suporta equipes maiores de desenvolvimento (de 10 a 20 integrantes) enquanto que o XP suporta equipes de 2 a 10 pessoas, dessa forma, uma das restrições do XP é a limitação à projetos relativamente pequenos. Além disso, o FDD apresenta uma maior formalidade dos relatórios de documentação, por exemplo, os de apresentação ao cliente, enquanto o XP não se preocupa com essa característica.

O FDD sugere a criação de um modelo que englobe uma visão mais ampla do sistema, enquanto o XP sugere um modelo mais simples, sendo a descrição mais detalhada feita conforme o andamento do projeto.

Enquanto o XP sugere a idéia de desenvolvimento coletivo (cada desenvolvedor consegue editar qualquer parte do código fonte), o FDD impõe que cada desenvolvedor construa sua própria classe.

O primeiro tipo de abordagem traz como vantagens evitar que o desenvolvimento de uma funcionalidade fique dependente das outras anteriores, cada integrante terá uma visão mais ampla do sistema evitando a existência de "pessoas chaves". O segundo tipo de abordagem em compensação diminui a possibilidade de existência de erros devido a alterações não relatadas no código.

Por ser o XP mais simplificado, possue maior facilidade de alteração no sistema, e um dos focos é também a integração e implementação no ambiente.

A principal vantagem do XP com relação ao FDD é a formalização de métodos para teste do sistema, enquanto o FDD apenas cita a necessidade de tais métodos.

### 3.5.2 Comparação entre RUP e XP

O RUP muito mais do que uma metodologia de desenvolvimento de software consiste de um *process framework*, ou seja, uma ferramenta que auxilia na aplicação efetiva da metodologia elaborada pela Rational Software e baseado no Unified Process. Ele já foi aplicado a uma grande variedade de projetos, de alguns meses e com equipes pequenas a projetos de anos com diversos times trabalhando juntos com sucesso.

A ferramenta possui uma infinidade de artefatos e *guidelines* que a tornaram conhecida como "pesada" e burocrática, entretanto o RUP pode ser customizado de forma a ser tão completo e "pesado" quanto for requerido ou simplificado de forma a se tornar tão "leve" e informal quanto for necessário. Os artefatos não precisam necessariamente serem utilizados em todos os projetos e o cliente pode também elaborar seus próprios artefatos conforme for adequado ao projeto em questão.

O Extreme Programming como descrito é uma nova metodologia que vem expandindo rapidamente sendo aplicado a uma grande porção de projetos da atualidade. Caracteriza-se por ser extremamente "leve", ou seja, não há perdas de tempo gerando extensa documentação que não será utilizada e as questões são resolvidas de forma mais eficiente e rápida de maneira informal.

O XP prega o desenvolvimento voltado para as necessidades do cliente, sendo que este trabalhando junto à equipe de desenvolvimento define o escopo do projeto de maneira informal (*stories*) e define os requisitos escrevendo scripts de testes aos quais o aplicativo deve ser submetido.

Desta forma as equipes começam a desenvolver rapidamente e o cliente usufrui dos benefícios gerados pela ferramenta em poucos meses. Conforme são entregues pacotes ao cliente ele passa a definir novo escopo e prazo para um novo pacote e dessa forma se mantém atualizado do andamento do projeto e tem a liberdade de comunicar alterações de requisitos conforme estes sofrem alterações já que os ciclos de desenvolvimento levam em média duas semanas.

Conforme dito acima não é verdade que o RUP seja uma metodologia burocrática e que não possa ser aplicada a projetos pequenos de forma eficiente como o XP. O

fato é que para se trabalhar em projetos maiores e mais longos ou com requisitos definidos de forma contratual alguma documentação é necessária e por esse fato o XP não é escalonável a projetos desse porte, ficando restrito a pequenos projetos com no máximo dez pessoas envolvidas.

Outro fato comum é dizer que o RUP possui muitos artefatos e atividades em comparação com o XP, pode-se dizer que isso não é completamente verdadeiro. O XP não possui artefatos definidos, mas na prática eles existem e podem inclusive serem relacionados com os existentes no RUP. A Tabela 3-3 apresenta um exemplo de mapeamento entre artefatos gerados pelo RUP e pelo XP.

| XP | RUP Small Project Roadmap |
|---|---|
| Stories<br>Additional documentation from conversations | Vision<br>Glossary<br>Use-Case Model |
| Constraints | Supplementary Specifications |
| Acceptance tests<br>Unit tests<br>Test data<br>Test results | Test Model |
| Software – code | Implementation Model |
| Releases | Product<br>Release Notes |
| Metaphor | Software Architecture Document |
| Design – CRC, UML sketch<br>Tasks<br>Technical Tasks<br>Design documents – produced at the end of the project<br>Supporting documentation | Design Model |

Tabela 3-3 Mapeamento entre artefatos do RUP e XP

Entretanto é certo que para projetos pequenos e de escopo bem definido o XP começa a liberar pacotes que agreguem valor ao cliente de forma muito mais eficiente e rápida.

Pode-se entender o XP como uma metodologia de escopo bem definido e que não prega uma utilização mais abrangente, de fato é excelente para as aplicações para as quais foi desenvolvido, porém não é escalonável a projetos maiores.

Muitas das práticas pregadas pelo XP são bastante eficientes e podem ser integradas ao RUP para melhorar sua aplicação. São exemplos dessas práticas: programação em duplas, carga horária de 40 horas semanais, definir o teste antes de desenvolver, etc.

Pode-se concluir dessa comparação que a maioria dos projetos se encontram em um meio termo entre as potencialidades das duas metodologias e a definição de um meio termo é o ideal para esses projetos. A maioria necessita de uma certa formalidade para definir escopo e requisitos, porém podem usufruir de certa informalidade para acelerar o processo de desenvolvimento e entrega do sistema.

### 3.5.3 Comparação entre FDD e RUP

O RUP é uma metodologia voltada para o gerenciamento do desenvolvimento de projetos, enquanto o FDD foca o desenvolvimento das funcionalidades. Elas são metodologias que diferente do XP, restringem a maior parte do tempo de planejamento na definição das funcionalidades do sistema, englobando o ponto de vista mais geral na modelagem do mesmo.

O FDD é mais eficiente no desenvolvimento de projetos menores, enquanto o RUP é extremamente vantajoso aplicado a sistemas grandes, como o FDD foca o desenvolvimento das funcionalidades, seus ciclos de desenvolvimento são mais rápidos.
A principal vantagem do FDD é a maior comunicação entre cliente e equipe de desenvolvimento, sendo as funcionalidades definidas com sentenças simples pelo cliente, com isso ele tem uma visão clara de cada funcionalidade do sistema, sendo o andamento do projeto descrito por documentos especificados pela metodologia que visam fácil compreensão do cliente.

O RUP por atingir sistemas maiores, possue uma documentação mais formal gerada ao longo do projeto, além de englobar conceitos relacionados à integração, gerenciamento, arquitetura e testes do sistema, características não presentes no FDD.

Outra vantagem é a especificação de critérios de avaliação e tomada de decisão formalizadas no método, enquanto o FDD depende exclusivamente da experiência de programadores e arquitetos chefes.

Para projetos de sistemas médios, os quais o FDD se adequa seria ideal aplicação de conceitos do RUP que cubram as deficiências do método.

# 4 TOGETHER CONTROL CENTER

Apresentadas todas as vantagens das metodologias rápidas, parte-se para a descrição do Together Software, mais especificamente utilizando o Together Control Center 6.0.

O Together Software foi criado baseado na preocupação de trazer ao usuário ferramentas que facilitassem o desenvolvimento do projeto sob o ponto de vista da metodologia FDD. A Figura 4-1 representa esquematicamente as funcionalidades do programa no contexto do FDD.

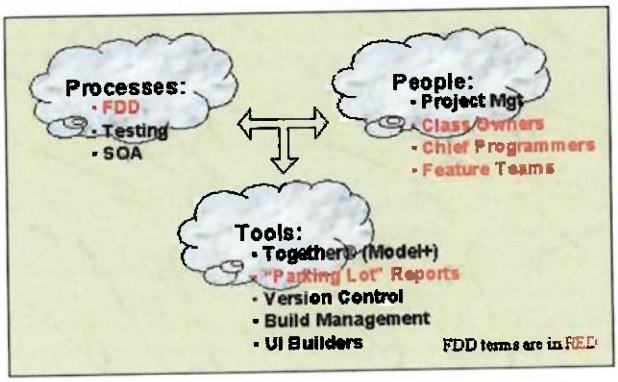



Figura 4-1 Together Software no contexto do FDD

Para isso ele traz funcionalidades como documentação UML, geração de inspeções, possibilidade de editar códigos fontes e classes, com atualização automática de todos os documentos relacionados.

## 4.1 Apresentação ao software

O principal propósito do Together Control Center é o de auxiliar no desenvolvimento de softwares, fornecendo ao usuário um ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*), onde estão inclusas, dentre outras, as funcionalidades:

- Geração rápida de documentos em diversos formatos e "templates";
- Geração de documentação UML;
- Capacidade de edição e atualização do código fonte e diagramas UML em tempo real;
- Integração com templates de linguagens orientadas a objetos como Java, C++ e Visual Basic;
- Compilação do código fonte;
- Inspeção do código fonte e coesão do sistema;
- Geração de métricas do sistema;
- Controle de versão.

Todas essas funcionalidades estão combinadas para incrementar a qualidade dos sistemas de software, tornando o trabalho do desenvolvedor mais fácil e garantindo o sucesso no desenvolvimento do sistema, já que a equipe tem a possibilidade de trabalhar com ferramentas afim de garantir padrão na implementação, evitando erros de formato decorrentes da junção das diversas partes do sistema, e a integridade do mesmo, através de testes e análises sucessivas. Verifica-se a relação entre as ferramentas que o Together Software oferece com as metodologias rápidas, em especial a FDD.

Abaixo apresentaremos um resumo das características de cada funcionalidade citada acima:

### 4.1.1 Geração de documentação

No Together Control Center (TCC) o usuário pode encontrar a possibilidade de gerar documentação completa sobre o sistema, incluindo diagramas, relações de classes e funções em diversos formatos e templates, podendo estes serem configurados pelo usuário à seu critério. Isso facilita o tempo gasto em documentação e facilita na uniformidade dos documentos, auxiliando nas inspeções do sistema ao longo e após o projeto.

### 4.1.2 Geração de documento UML

Além de possuir como funcionalidade a criação de todos os diagramas definidos na UML, o Together Software possui diagramas definidos como extensões da UML, diagramas de modelagem em tempo real e diagramas XML, com o propósito de auxiliar na modelagem e gerenciamento do projeto.

Os diagramas UML padrão e estendidos são ferramentas capazes de documentar os requisitos do sistema a ser criado. Os diagramas de modelagem em tempo real, além de possuírem tais funcionalidade servem como ferramentas de simulação dos use cases do sistema.

Essas funcionalidades extras (UML estendido e análise em tempo real), assim como geração de relatórios, provém da adaptação do programa às abordagens do Feature-Driven Development, onde sugere-se sucessivas análises do sistema feitas no projeto.

Todos estes diagramas servem como ferramentas importantes no desenvolvimento e integração do sistema projetado, porém não é escopo do relatório apresentar uma descrição detalhada da funcionalidade de cada diagrama.

A Tabela 4-1 lista os diagramas incluídos no programa:

| | |
|---|---|
| Modelagem UML | Diagramas *use case*, diagramas de classe, diagramas de mapa de estados, diagramas de componentes, diagramas de atividades, diagramas de implementação |
| Modelagem UML estendida | Diagramas de *entity relationship*, diagramas de *business process*, diagramas de análise de robustez |
| Modelagem em tempo real | Diagramas *use case*, diagramas de contexto do sistema, diagramas de arquitetura do sistema, diagramas de mapa de estados, diagramas de folha de eventos, diagramas de interação |
| Suporte à J2EE | Diagramas *EJB Assembler*, diagramas de *Enterprise application*, diagramas de *Taglib*, diagramas de adaptador de recursos |

Tabela 4-1 Diagramas fornecidos pelo Together Software

### 4.1.3 Edição de códigos fonte e diagramas UML

O TCC permite ao usuário editar o código fonte, atualizando automaticamente os diagramas UML gerados anteriormente, evitando eventuais conflitos entre o que é gerado no código e o que é documentado em UML.

### 4.1.4 Integração com linguagens de programação

O software também fornece a vantagem de possuir como template cabeçalhos para diversas linguagens de programação, como Java e C++. Dessa forma é possível criar projetos, incluindo documentação UML de códigos fonte pré-existentes, esse método é denominado como Engenharia Reversa. Ele também consegue integrar sistemas de software para aplicações em rede.

### 4.1.5 Compilação do código fonte

Conforme apresentado o Together Control Center é uma IDE, dessa forma como importante ferramenta ele fornece ao usuário a possibilidade de compilar o código fonte e executá-lo, disso surgem vantagens visíveis relacionadas à facilidade de integração e padronização citadas anteriormente.

### 4.1.6 Inspeção do código fonte e inspeção do sistema

Uma das principais ferramentas que o software oferece, e extremamente útil para garantir a integridade do projeto é a possibilidade de executar a verificação de todo o sistema com relação à formatação pré-definida pelo usuário.

As verificações buscam a conformidade dos padrões do código fonte com os parâmetros definidos, mantendo a sintaxe de programação, leitura, manutenção, robustez e performance do programa. Sem um processo automatizado é humanamente impossível manter a conformidade desses parâmetros em códigos muito grandes, o resultado disso são os visíveis ganhos na qualidade e produtividade no desenvolvimento do software.

### 4.1.7 Geração de métricas do sistema

Basicamente, métricas envolvem contagem ou medidas de determinada grandeza relacionada ao projeto, e baseado sobre o que está sendo medido, gera-se uma escala relativa (por exemplo a média dos parâmetros) que pode ser usada para analisar os resultados, indicando a atual condição de desenvolvimento do código fonte.

Enquanto auditorias focam principalmente em análise sintática, as métricas focam nos parâmetros do nível de desenvolvimento do projeto, o objetivo é coletar dados que possam indicar em futura análise, possíveis falhas no sistema.

Há duas categorias de métricas, as internas e as externas. As primeiras fornecem dados precisos com relação às características do código fonte, sendo mais claras do que as externas, estas fornecem dados relacionados à complexidade, manutenção, qualidade e entendimento do código, parâmetros não tão óbvios de serem estimados.

### 4.1.8 Controle de versão

Assim como as outras ferramentas, o controle de versão surgiu para auxiliar a aplicação de metodologias de desenvolvimento de software, permitindo o controle das mudanças feitas no programa, essencial para o desenvolvimento à múltiplos usuários.

## 4.2 Vantagens do Together Software como IDE

Comparando a utilização de um software integrado como o TCC com a utilização de diversos outros softwares, pode-se ver a brusca diferença entre comodidade e

tempos gastos em cada opção. A Figura 4-2 mostra a utilização de diversas ferramentas para desenvolvimento de software.

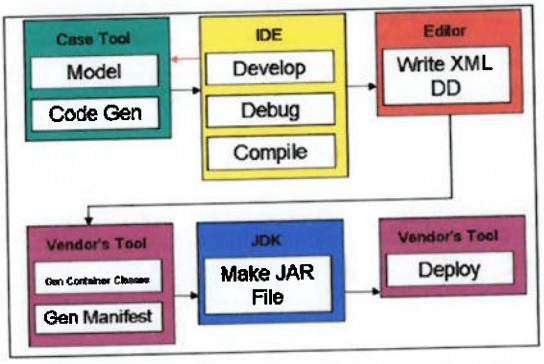



Figura 4-2 Processo de negócios tradicional

Nesse caso, devido a falta de interatividade das ferramentas, processo de negócios pode levar horas. Ao contrário do processo de negócios utilizando o Together Software, onde todas as ferramentas estão integradas, mantendo padrões e comunicação entre elas. A Figura 4-3 mostra a integração promovida pelo Together.

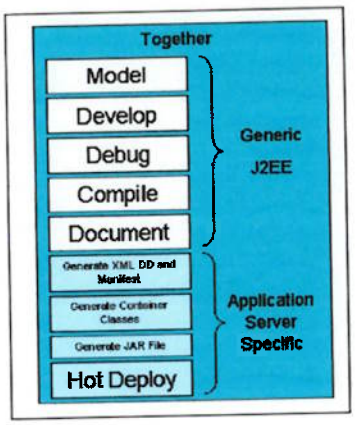



Figura 4-3 Processo de negócios pelo Together Software

## 4.3 Modelagem do sistema

O primeiro objetivo da modelagem é organizar e visualizar a estrutura e os componentes do sistema. Modelos visam representar requisitos, sub-sistemas, elementos físicos e lógicos, assim como modelos estruturais e comportamentais do sistema.

Enquanto práticas de criação de software contemporâneas se preocupam com a geração de modelos de desenvolvimento corretos, o Together Software busca os benefícios inerentes à modelagem, por sincronizar mudanças em diagramas a códigos fontes, gerando atualizações em tempo real e sob controle do usuário.

# 5 RATIONAL ROSE

O Rational Rose, é parte integrante do pacote Rational Suite, ferramenta fornecida pela Rational, e que está de acordo com a proposta dos princípios do RUP. O Rational Rose não apresenta todas as funcionalidades do TogetherSoft, já que elas são oferecidas por outras ferramentas da Rational. Dessa forma as funcionalidades de Controle de Versões, Documentação, Extração de Métricas e outras citadas para a ferramenta anterior são supridas por ferramentas complementares, usufruindo assim da forte integração entre elas.

A seguir listam-se as principais funcionalidades do software e os diagramas UML que ele contém.

1. Funcionalidades
    - Análise e Projeto: possui ferramentas para concepção e descrição do sistema, disponibilizando para tal, diagramas definidos pela UML. A proposta é fornecer uma visão geral do sistema, através de diferentes meios e pontos de vista. Apresenta recursos para a criação de diagramas de caso de uso, diagramas de atividades, diagramas de interação dentre vários outros;
    - Desenvolvimento em equipe: possui recursos que facilitam a integração das equipes de trabalho, permitindo criação de diferentes versões sobre um mesmo modelo, além de interagir com outras ferramentas oferecidas pela Rational (controle de versão, especificação de requisitos, testes, etc);
    - Unidades controladas: o Rational Rose estrutura os modelos em diferentes níveis, assim é possível mudar as bases de um projeto no programa, sem mudar o nível superior, através de controle de pastas e arquivos, seguindo o princípio da modularização;
    - Arquitetura baseada em componentes, permitindo integração de projetos relativamente grandes, diferentes equipes e focos de trabalho;
    - Geração de Códigos DDL: permite a criação de elementos possíveis de integração com SQL (Structured Query Language) e DDL (Data

Definition Language), provendo capacidade de interação com banco de dados;

- Criação de esteriótipos: esse recurso dá ao usuário a liberdade de expandir o pacote básioco de esteriótipos UML fornecidos com a ferramenta. Dessa forma o usuário pode criar esteriótipos que sejam aderentes às suas necessidades de modelagem, mantendo a integridade básica do modelo sobre o qual foi feito esteriótipo;

2. Pacotes de Modelos UML no Rational Rose

O Rational Rose divide a modelagem de um sistema segundo vários pontos de vista, permitindo assim que sejam abordados e modelados diversos aspectos do sistema em desenvolvimento. Apesar de existir essa divisão, os modelos mantêm entre si uma conexão coerente, de forma que alterações efetuadas em elementos presentes em diversos pacotes sejam disseminadas entre todos eles, mantendo assim a integridade do modelo.

| Pontos de Vista | Diagramas UML |
|---|---|
| *Use Case View* | - Diagramas de Caso de Uso<br>- Diagramas de Seqüência<br>- Diagramas de Atividades<br>- Diagramas de Classe<br>- Diagramas de Colaboração<br>- Diagramas de Estado |
| *Logical View* | - Diagramas de Caso de Uso<br>- Diagramas de Seqüência<br>- Diagramas de Atividades<br>- Diagramas de Classe<br>- Diagramas de Colaboração<br>- Diagramas de Estado |
| *Component View* | - Diagramas de Componentes |
| *Deployment View* | - Diagramas de *Deployment* |

Tabela 5-1 Diagramas UML do Rational Rose

Cada um desse diagramas permite a modelagem de diferentes aspectos do sistema, abordando desde a definição de casos de uso, passando pela definição das classes até o diagrama de implementação.É possível assim extrair do Rational Rose as informações necessárias para a criação de modelos de automação, proposta inicial do trabalho.

Através do Rational Rose também é possível gerar estruturas de código para diversas linguagens (C++, Visual Basic, Ada, Java, etc) a partir de diagramas de classe, bem como modelos de dados para bancos de dados. Através da característica de esteriotipação, o Rational Rose também traz um pacote de esteriótipos para modelagem de negócios (*Business Use Case*, *Business Actor*, etc), permitindo assim que a modelagem formal de um sistema se inicie já a partir da modelagem de negócios. Por ter uma forte coesão entre os diversos modelos e diagramas, o Rational Rose permite que esses modelos de negócios sejam conectados logicamente aos diagramas e modelos de sistema.

# 6 SELEÇÃO DA FERRAMENTA

A metodologia escolhida para utilização no projeto foi a proposta pelo RUP (Rational Unified Process), principalmente pela sua ampla aplicação e adequação a diversos tamanhos de projetos e abordagens de desenvolvimento.

Para implementação dos conceitos e princípios propostos pelo RUP, coube à equipe a escolha da ferramenta para gerar os diagramas e documentação UML, da qual provém os dados necessários para o projeto de um sistema de automação.

Foram analisadas essencialmente duas ferramentas de cases consagradas no mercado e apresentadas anteriormente, o TogetherSoft e o Rational Rose. A seguir apresentamos as principais características dos dois softwares:

TogetherSoft:

- Contém todas as ferramentas necessárias para desenvolvimento e projeto de softwares, tais como interface com compilador JDK, ferramentas de gerenciamento e geração de métricas de projetos;
- Diagramas UML para documentação;
- Integração em tempo real entre documentação UML e código fonte do programa.
- Devido ao número de funcionalidades impostas e a linguagem em que foi desenvolvido (Java), o software necessita de grande capacidade de memória.

Rational Rose:

- Contém as ferramentas para criação de diagramas UML para documentação;
- Integração entre os modelos e o código fonte, oferecendo a engenharia reversa, forte característica desta ferramenta;
- Integração com outras ferramentas da Rational para controle de versão, levantamento e gerenciamento de requisitos, gerenciamento de projetos, testes, etc;
- Software "leve", necessitando de baixa capacidade de memória;

- Recursos para criação de *Add-Ins* e *Scripts*, permitindo que sejam criadas funcionalidades de extras para a ferramenta e que diversas tarefas sejam automatizadas.

Analisando as características de cada software, podemos concluir que apesar de o TogetherSoft ser uma ferramenta que agrega várias funcionalidades em um único pacote (métricas, controle de versão, implementação e testes) o Rational Rose se mostrou mais adequado às necessidades deste trabalho, já que o escopo é a aplicação de conceitos do RUP no projeto de sistemas de automação e assim o Rose se mostrou a melhor solução. Além disso o Rational Rose conta que recursos para elaboração de scripts dentro do próprio ambiente da ferramenta, não exigindo a utilização de ferramentas externas para desenvolvimento de aplicações que possam trabalhar com os modelos e diagramas elaborados bem como com seus elementos.

# 7 MODELAGEM DO PROJETO

Este projeto está dividido em dois sistemas independentes. O primeiro parte de uma interface com o Rational Rose, da *Rational Software*, e partindo de um de seus diagramas (que descreve uma determinado caso de uso) gera um modelo BNF da especificação em questão. Este arquivo é o ponto de sinergia com o segundo sistema que, partindo do BNF, converte este modelo em uma Rede de Petri no formato do simulador Ghenesys.

Neste capítulo inicia-se com a descrição do estudo realizado para o primeiro sistema, finalizando com a descrição da metodologia aplicada no desenvolvimento do segundo. Chamaremos os dois sistemas de Sistema 1 e Sistema 2 respectivamente deste ponto em diante.

## *7.1 Descrição do Sistema 1*

### 7.1.1 *Estudo dos Diagramas e Elementos da UML*

#### 7.1.1.1 O que é a UML?

A UML, ou Unified Modeling Language, é uma linguagem de modelagem utilizada para comunicar idéias sobre o sistema que será desenvolvido ou negócio que está sendo modelado. Basicamente ela é uma linguagem utilizada para especificar, visualizar, construir e documentar tudo o que será produzido no desenvolvimento de um sistema de software. (Booch et al, 2001)

No final dos anos 80 havia vários metodologistas propondo sua próprias notações, fato que acabava trazendo uma série de problemas no momento de traduzir uma notação na outra, além de gerar uma "guerra de notações", pois desenvolvedores e analistas se viam constantemente no dilema da adoção da melhor notação ou da mais adequada aos seus propósitos. Em 1995, após serem contratados pela Rational Software, Grady Booch e Jim Rumbaugh produziram a primeira versão da UML e em 1997, após Ivar Jacobson ter ingressado no grupo, submeteram a UML à OMG para padronização.

Portanto a UML, como é hoje, é o resultado dos esforços realizados por diversos metodologistas (Grady Booch, Jim Rumbaugh, Ivar Jacobson e outros) para se produzir uma linguagem única para a modelagem de software e que acabou se tornando padrão do mercado.

Atualmente a UML é mantida pela OMG (Object Management Group), responsável pela sua evolução e padronização.

#### 7.1.1.2 Os Diagramas

A UML possui uma série de diagramas, cada um representando aspectos estáticos ou dinâmicos do sistema e oferecendo diversas visões da sua modelagem. Os diagramas são:

- Diagrama de Atividade (*Activity Diagrams*)
- Diagramas de Caso de Uso (*Use Case Diagrams*)
- Diagramas de Seqüência (*Sequence Diagrams*)
- Diagramas de Colaboração (*Collaboration Diagrams*)
- Diagramas de Classe (*Class Diagrams*)
- Diagramas de Transição de Estado (*Statechart Diagrams*)
- Diagramas de Componente (*Component Diagrams*)
- Diagramas de Implantação (*Deployment Diagrams*)

#### 7.1.1.3 Diagramas de Atividade (*Activity Diagrams*)

Os diagramas de atividade mostram o fluxo de controle, utilizando em sua notação elementos para representar vários aspectos da dinâmica das atividades modeladas. Na figura abaixo há um diagrama de atividade e na tabela seguinte, o significado de cada elemento desse diagrama. Usualmente é elaborado um diagrama de atividades para os casos de uso mais relevantes de um sistema. Consegue-se assim obter uma melhor compreensão de como se dá a seqüência de ações para um caso de uso. Podemos enriquecer um diagrama de atividades inserindo nele os responsáveis por uma ou mais atividades bem como atividades que acontecem em paralelo após a ocorrência de um determinado evento.

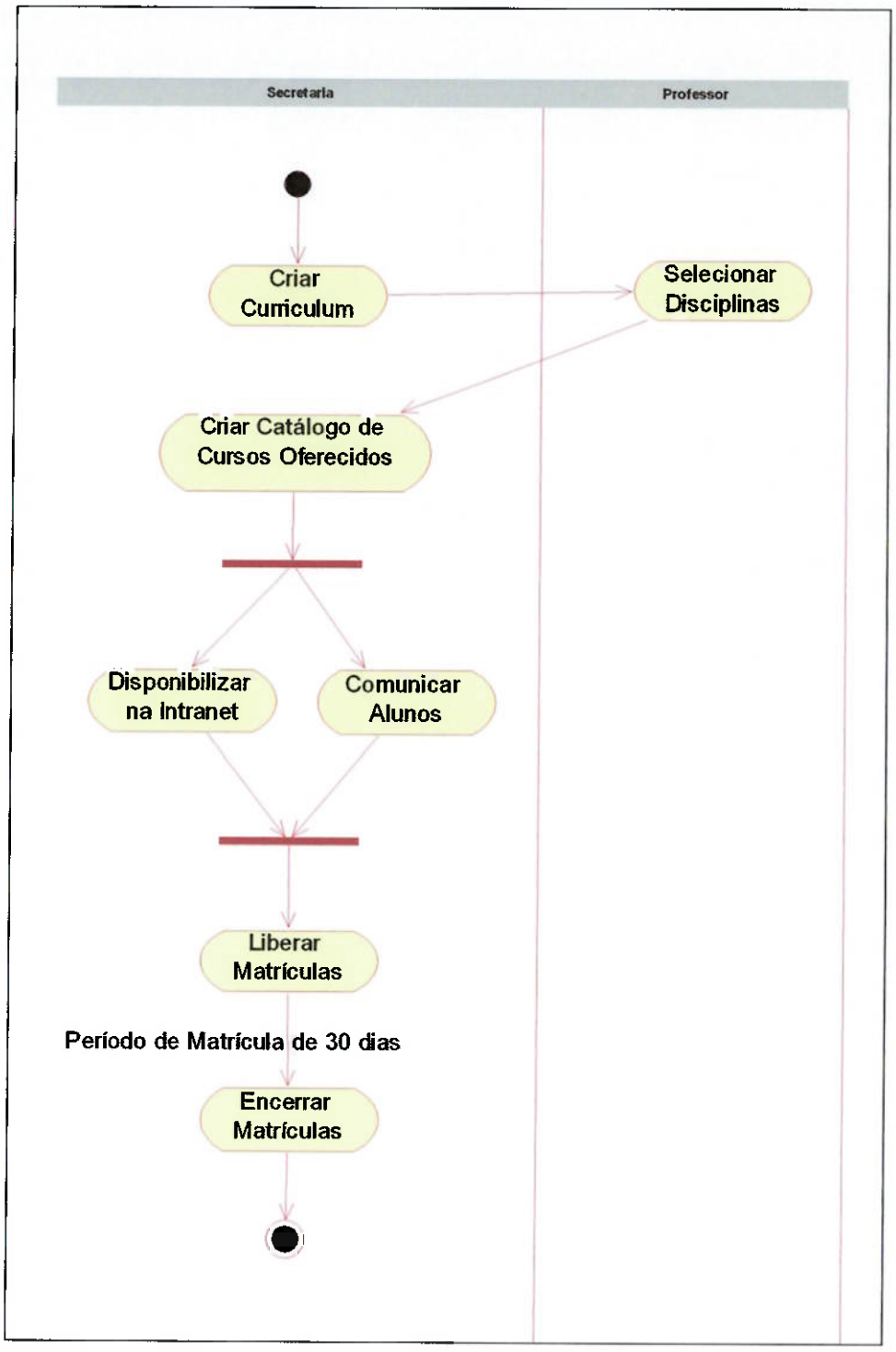



Figura 7-1 - Diagrama de Atividade – Matrículas

| | |
|---|---|
|  | Representa o início das atividades. Pode existir somente um para cada diagrama. |
|  | Representa o fim das Atividades. Podem existir vários estados de finalização em um diagrama de atividades. |
|  | Atividade: representa uma atividade do diagrama. Quando uma atividade é finalizada, a próxima poderá ser iniciada. |
|  | Barra de Sincronização: serve para sincronizar o início ou fim das atividades. Se estiver antes de um grupo de atividades, estas vão ocorrer em paralelo. Se estiver após um grupo de atividades, a próxima atividade após a barra só será iniciada após a finalização das atividades paralelas anteriores à barra. |

Tabela 7-1 - Notação Utilizada em um Diagrama de Atividades

No diagrama anterior, podemos notar a existência de duas áreas delimitadas e nomeadas (Secretaria e Professor). Essas áreas são denominadas *swimlanes* e agrupam as atividades que são realizadas por quem ou pelo quê. As setas determinam o relacionamento entre as atividades, denotando também a direção do fluxo.

#### 7.1.1.4 Diagramas de Caso de Uso (*Use Case Diagrams*)

Um caso de uso representa uma funcionalidade oferecida pelo sistema e o diagrama de caso de uso agrupa um conjunto de casos de uso, dando uma visão do que o sistema faz e não do como. Assim o diagrama de caso de uso auxilia no entendimento dos requisitos funcionais de um sistema, representando também quem ou o que interage com ele. Têm-se assim uma visão do escopo das funcionalidades do sistema a ser desenvolvido A figura abaixo apresenta um diagrama de caso de uso.

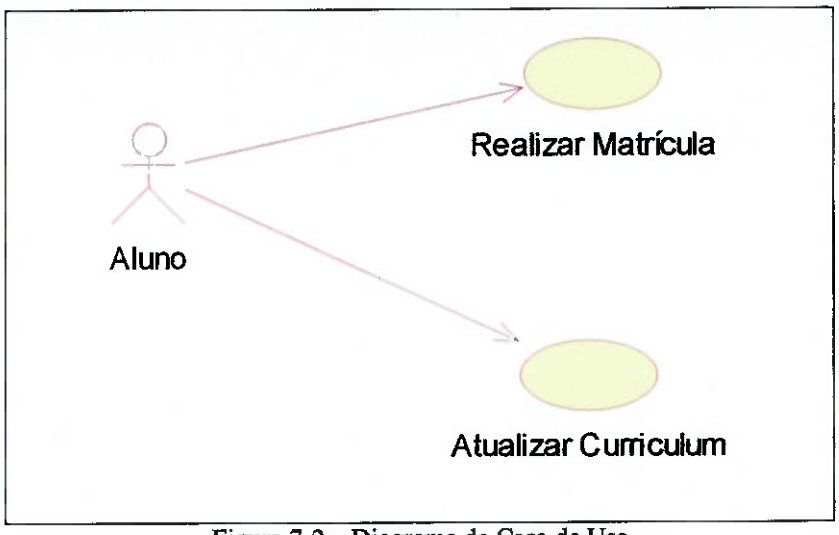



Figura 7-2 – Diagrama de Caso de Uso

| | |
|---|---|
|  | Ator: representa alguém ou algo que interage com o sistema. |
|  | Caso de Uso: representa uma funcionalidade oferecida pelo sistema e que apresenta valor para um ator. |

Tabela 7-2 - Notação Utilizada em um Diagrama de Caso de Uso

### 7.1.1.5 Diagramas de Seqüência (*Sequence Diagrams*)

Um diagrama de seqüência representa a interação entre objetos ao longo do tempo, descrevendo o fluxo de um determinado caso de uso ou de apenas parte dele. Sua estrutura está organizada da seguinte forma: na dimensão horizontal estão os objetos para os quais são enviadas as mensagens e na dimensão vertical está a seqüência das mensagens na ordem de ocorrência. A figura abaixo ilusta um diagrama de seqüência.

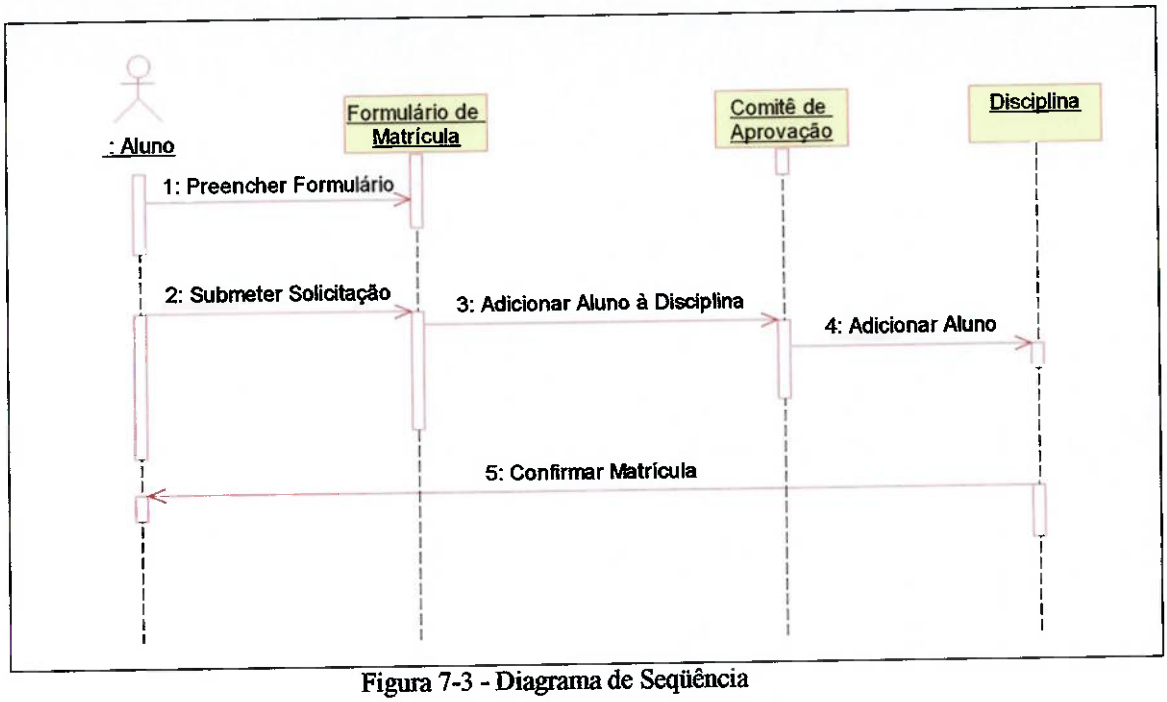



Figura 7-3 - Diagrama de Seqüência

Neste diagrama está documentado o aspecto dinâmico do caso de uso Realizar Matrícula. O aluno inicia a seqüência de mensagens ao preencher o formulário para fazer sua matrícula. O objeto Formulário de Matrícula provê ao usuário essa função (Preencher Formulário). Em seguida, o aluno submete a solicitação e uma outra série de objetos e mensagens dá continuidade ao processo de matrícula até que a ela seja realizada. As caixas representadas acima (Formulário de Matrícula, Comitê de Aprovação e Disciplina) representam instâncias de classes e as setas, as mensagens enviadas a cada objeto.

Um diagrama de seqüência enfatiza a ordem das mensagens ao longo do tempo.

### 7.1.1.6 Diagramas de Colaboração (*Collaboration Diagrams*)

Este diagrama é uma forma alternativa de mostrar o aspecto dinâmico de um caso de uso ou de um cenário qualquer. Nele a ênfase é dada para a interação entre os objetos e o relacionamento entre eles. A figura abaixo mostra um diagrama de colaboração.

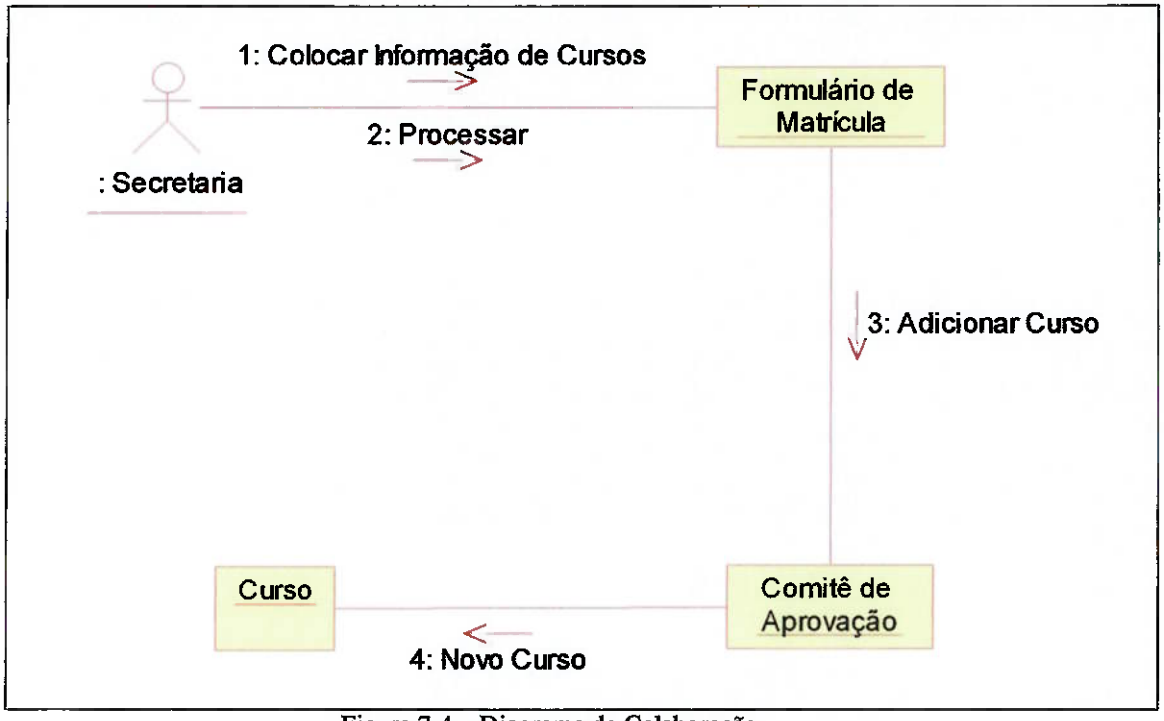



Figura 7-4 – Diagrama de Colaboração

Um diagrama de colaboração enfatiza a organização dos objetos que participam de uma determinada interação.

Os diagramas de seqüência e de colaboração são também conhecidos como Diagramas de Interação.

### 7.1.1.7 Diagramas de Classe (*Class Diagrams*)

Uma classe é uma descrição de um grupo de objetos com atributos em comun, operações em comum, semântica comum e relacionamento em comum com outros objetos. Elas são geralmente encotradas através da análise dos diagramas de interação. Assim os diagramas de classe agrupam as classes encontradas para o sistema, apresentando:

- o nome de uma classe, seus atributos e operações
- o relacionamento entre as classes
- o relacionamento de multiplicidade das classes
- os aspectos de herança de classes

Na figura abaixo está representado um diagrama de classes. Ele não é completo, mas oferece a noção de como ele é elaborado.


Figura 7-5 - Diagrama de Classe

### 7.1.1.8 Diagramas de Estado (*Statechart Diagrams*)

Os diagramas de estado nada mais são do que máquinas de estado e são utilizados para representar os estados nos quais um objeto pode estar em função da ocorrência de determinados eventos. São utilizados para objetos que apresentam um comportamento dinâmico intenso. A figura abaixo exibe um exemplo de diagrama de estado.

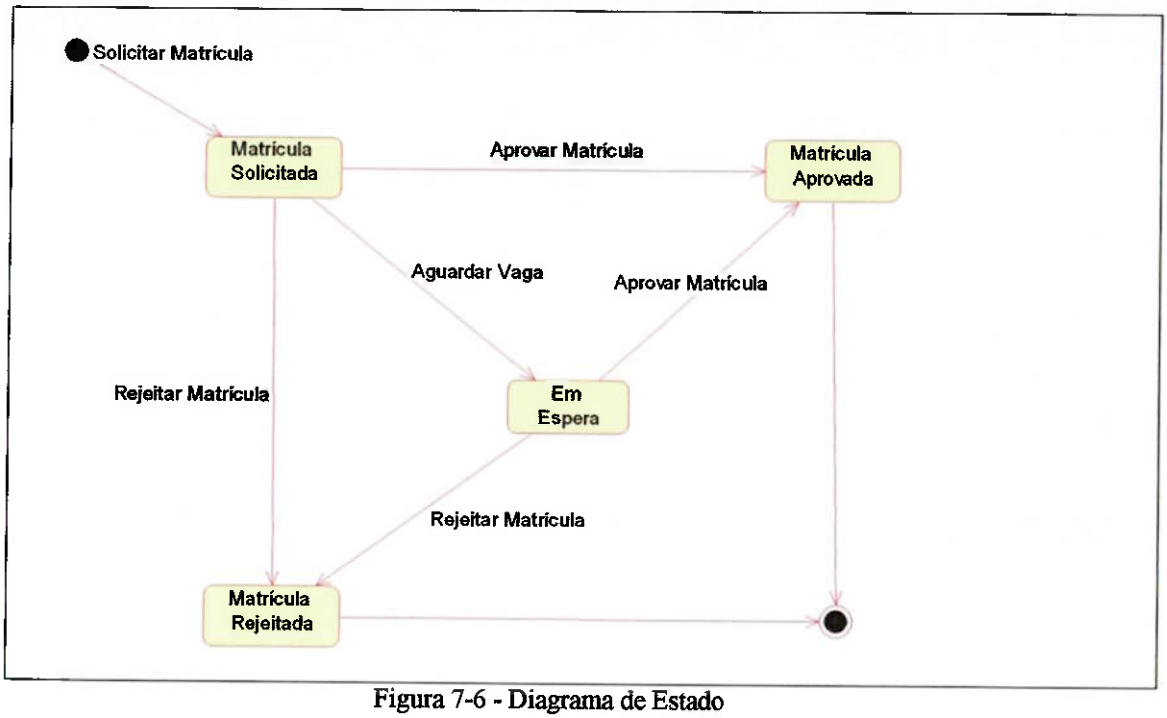



Figura 7-6 - Diagrama de Estado

### 7.1.1.9 Diagramas de Componentes (*Component Diagrams*)

O diagrama de componentes oferece uma visão física do sistema que está sendo implementado, representando as dependências entre os componentes de um sistema.

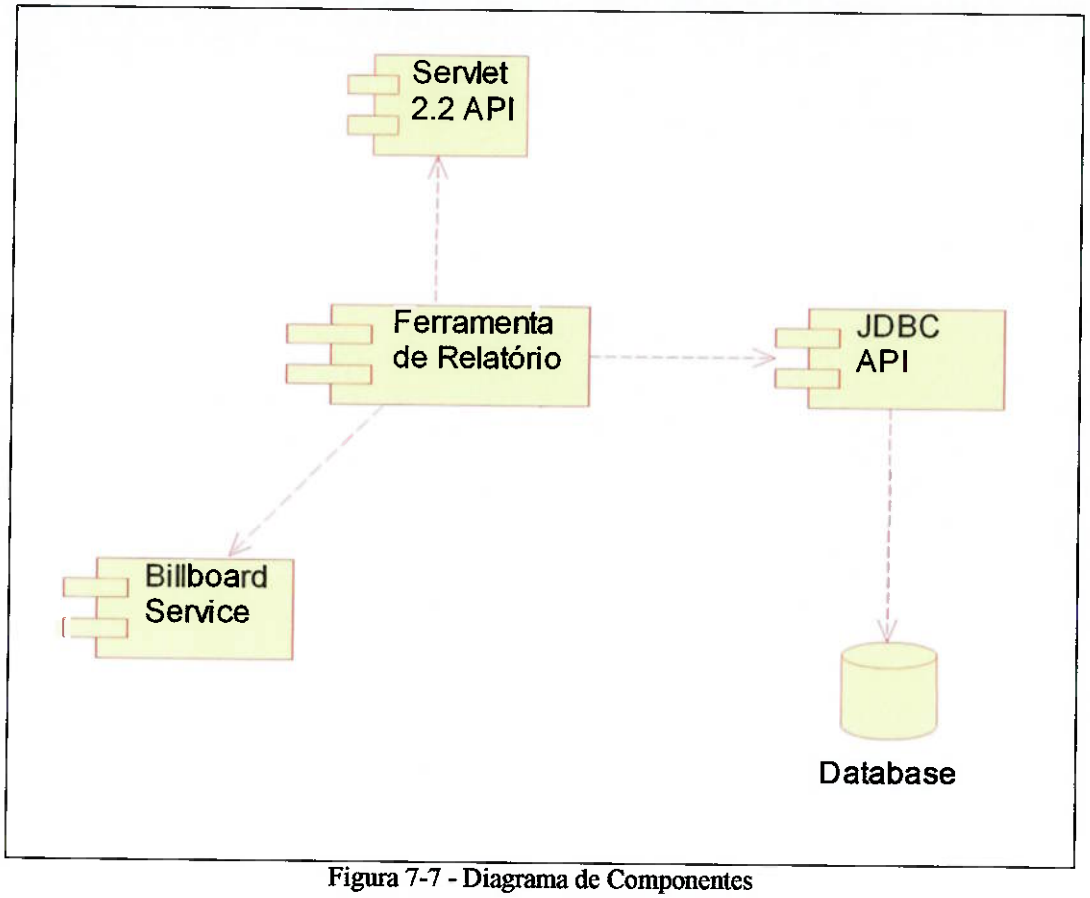



Figura 7-7 - Diagrama de Componentes

### 7.1.1.10 Diagramas de Distribuição (Deployment Diagrams)

Os diagramas de distribuição são utilizados para representar a distribuição física dos componentes de um sistema no ambiente de implantação, dando uma visão mais macro do cenário.

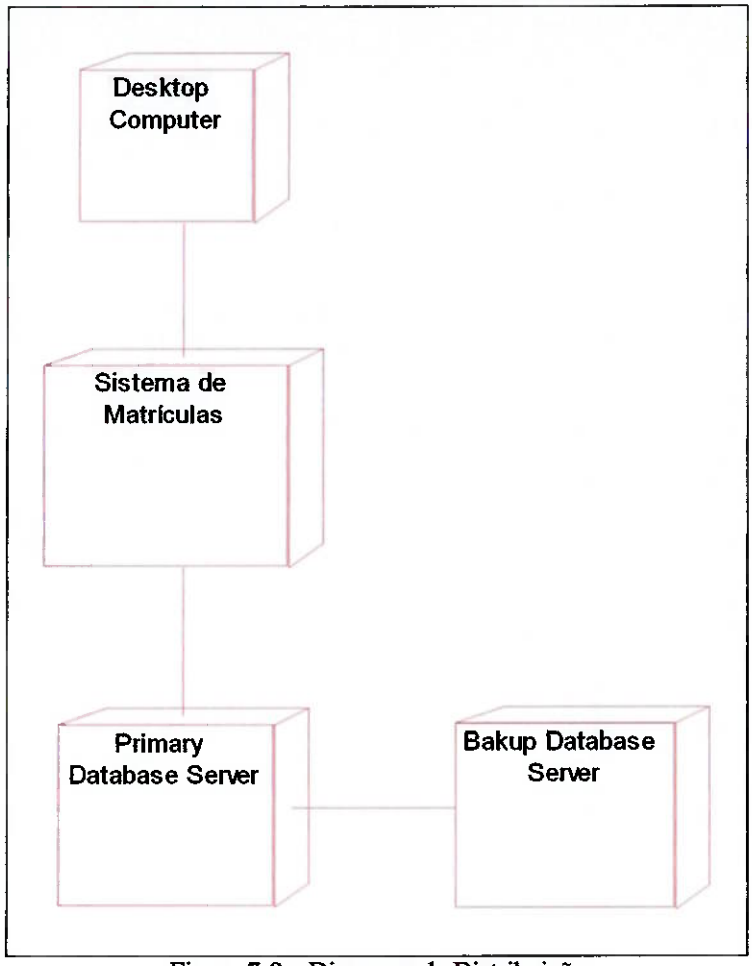



Figura 7-8 – Diagrama de Distribuição

### 7.1.2 Visão do sistema

O Sistema 1 gerará um documento de descrição BNF a partir de um Caso de Uso. Dessa forma, ao criar um determinado caso de uso, associaremos a ele um diagrama de atividades que por sua vez conterá informações como nome da atividade, descrição breve da finalidade de cada atividade para fins de documentação, os relacionamentos entre atividades e os seus estereótipos. Após a criação do diagrama de atividades, o usuário executará o script da aplicação sobre esse diagrama, que por sua vez produzirá o documento BNF necessário ao sistema 2. Se o usuário realizar alguma alteração no diagrama de atividades, ele deverá executar novamente esse script para que o um novo BNF seja gerado. Todas essas operações serão realizadas dentro da ferramenta Rational Rose.

O motivo da existência desse sistema é o fato de ele extrair as informações necessárias para a geração do BNF a partir de um diagrama da UML, já que é a linguagem padrão de mercado utilizada para a modelagem de sistemas e que possui um rico conjunto de elementos que permitem um rápido entendimento do sistema a ser desenvolvido ou estudado.

### 7.1.3 Stakeholders do sistema

Entende-se por *stakeholders* como pessoas ou organizações que serão afetadas pelo sistema e que tem influência direta ou indireta na definição dos requisitos. Dentro desse contexto, podemos definir três principais *stakeholders* :

- **Usuário :** é a pessoa que usufrui do sistema implementado. O intuito do projeto é atender às suas necessidades (supostas ou reais).
- **Desenvolvedor :** a pessoa ou equipe que deve desenvolver o sistema.
- **Cliente :** A pessoa ou corporação que se beneficiará (economicamente por exemplo) com a implantação do sistema. É também a patrocinadora e portadora desse sistema

### 7.1.4 Levantamento de requisitos

O sistema em questão deverá extrair do diagrama de atividades, modelado dentro da ferramenta Rational Rose, a informação necessária para a geração do documento BNF. Essa informação consta de:

- Atividades: as atividades representam os passos executados durante a realização de um determinado processo. Elas podem ser *principais*, quando pertencem ao fluxo principal de uma descrição de caso de uso, ou podem ser *alternativas*, quando representam variações relativas ao fluxo principal e contemplam caminhos alternativos seguidos para a realização do processo.
- Decisões: as decisões ocorrem durante o fluxo principal ou alternativo e representam pontos de bifurcação e direcionamento para o fluxo principal ou para fluxos alternativos do processo. Da mesma forma que as atividades, as decisões são classificadas em *principais* e *alternativas*. As principais ocorrem entre duas atividades principais e podem dar origem a um novo fluxo alternativo. As decisões

alternativas ocorrem entre duas atividades alternativas e podem dar origem a um novo fluxo principal.

- Transições: as transições são o elo entre as atividades, as decisões, e os pontos de início e de finalização. Ocorrem entre atividades, entre atividades e decisões e entre atividades e pontos de início e de finalização.
- Ponto de Início: representa o início das atividades de um processo. Após ela seguem as atividades e decisões que descrevem o processo em questão.
- Ponto de Finalização: representa o término de um processo. Para ele convergem atividades principais e alternativos que descrevem o processo em questão.

### 7.1.5 Atores do sistema e diagramas de caso de uso

A geração do BNF contempla uma série de atividades, várias das quais estão no escopo da ferramenta de modelagem utilizada neste trabalho, o Rational Rose. Assim no diagrama de caso de uso abaixo, apontamos os casos de uso que contemplam todos os aspectos de uso e geração do BNF do ponto de vista do usuário ou interessado na geração do BNF.

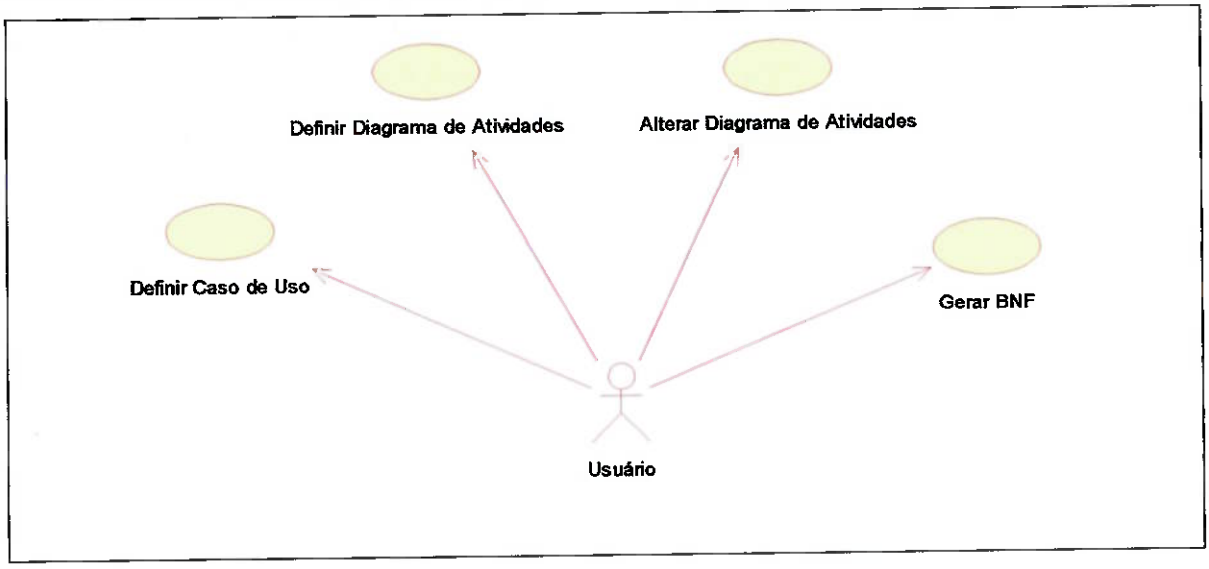



Figura 7-9 – Diagrama de Caso de Uso para o Sistema 1

1) Use case : Definir Caso de Uso.
   Ator : Usuário.
   Descrição :Trata-se da definição de caso de uso para o sistema para o qual se deseja gerar o BNF.
   Entrada :Levantamento de Requisitos realizado pelo Analista de Negócios ou de Sistemas.

Saída Principal : Caso de Uso.

2) Use case : Definir Diagrama de Atividades.
   Ator : Usuário.
   Descrição :Trata-se da descrição na forma de diagrama de atividades do processo para o qual se deseja gerar o BNF.
   Entrada :Levantamento de Requisitos realizado pelo Analista de Negócios ou de Sistemas e Caso de Uso definido anteriormente
   Saída Principal : Diagrama de Atividades.

3) Use case : Alterar Diagrama de Atividades.
   Ator : Usuário.
   Descrição : O usuário poderá realizar as alterações que julgar necessários no diagrama de atividades elaborado anteriormente. Após realizar as alterações, o usuário devera gerar novamente o BNF.
   Entrada :Diagrama de Atividades previamente criado e BNF alterado pelo Sistema 2.
   Saída Principal : Diagrama de Atividades alterado.

4) Use case : Gerar BNF.
   Ator : Usuário.
   Descrição : Geração do BNF a partir do diagrama de atividades definido anteriormente. O usuário inicia o processo de geração e o sistema 2 gera o BNF correspondente.
   Entrada : Diagrama de Atividades definido anteriormente.
   Saída Principal : Caso de Uso.
   Saída Alternativa : Podem ocorrer erros na geração do BNF se a modelagem do diagrama de atividades não for realizada corretamente.

## *7.2 Descrição do Sistema 2*

A modelagem do sistema apresenta documentos que detalham as funcionalidades do mesmo, relacionando-as entre si e com as pessoas envolvidas no desenvolvimento ou uso do sistema. Para geração do modelo é necessário seguir um caminho que garanta a especificação correta do sistema, dessa forma baseia-se na metodologia RUP para o processo de desenvolvimento e documentação.

### 7.2.1 Visão do sistema

Desenvolvimento de uma ferramenta de análise de requisitos através da análise de diagramas *use cases*. A ferramenta receberá como entrada um arquivo ASCII que descreve a seqüência de processos conforme a Figura 7-9.

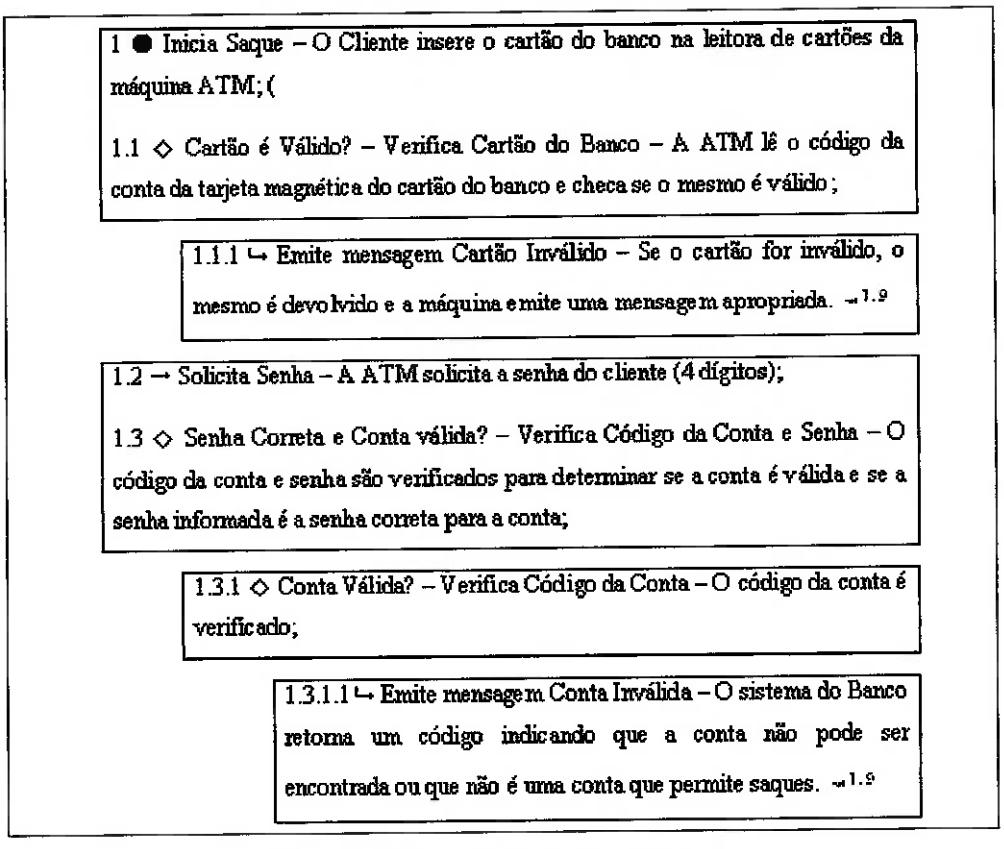



Figura 7-10 Exemplo de arquivo de entrada

A ferramenta permitirá que os requisitos sejam visualizados e analisados através de uma Rede de Petri, dessa forma será possível aplicar uma ferramenta consagrada na simulação de processos de automação para auxiliar no processo de desenvolvimento de softwares.

Será possível ainda realizar alterações na Rede de Petri e transmitir essas alterações para os diagrama *use case*. Finalmente a ferramenta disponibiliza a possibilidade de salvar os arquivos, tanto de Redes de Petri como de *Use Case* para posterior visualização e alteração.

### 7.2.2 Stakeholders do sistema

Entende-se por *stakeholders* como pessoas ou organizações que serão afetadas pelo sistema e que tem influência direta ou indireta na definição dos requisitos. Dentro desse contexto, podemos definir três principais *stakeholders* :

- **Usuário :** é a pessoa que usufrui do sistema implementado. O intuito do projeto é atender às suas necessidades (supostas ou reais).
- **Desenvolvedor :** a pessoa ou equipe que deve desenvolver o sistema.
- **Cliente :** A pessoa ou corporação que se beneficiará (economicamente por exemplo) com a implantação do sistema. É também a patrocinadora e portadora desse sistema.

### 7.2.3 Levantamento de requisitos

Parte-se então para o levantamento de requisitos do sistema, que devem englobar numa primeira listagem, uma descrição concisa e sem muito detalhamento das necessidades de cada *stakeholder*:

- Criar redes de Petri, use cases descritos conforme padrão UML (Usuário e Cliente).
- Criar use cases a partir de redes de Petri (Usuário e Cliente).
- Auxiliar na fase de integração de projetos (Cliente).
- A entrada do sistema deve ser em padrão ASCII e a descrição do use case em formato especificado (Desenvolvedor).

Definida a primeira descrição dos requisitos, listam-se as funcionalidades do sistema, dentre as funcionalidades principais e as funcionalidades secundárias:

- Gerar rede de Petri a partir de seqüência de use case.
    - Transferir para linguagem BNF.
    - Transferir para rede de Petri.

- Verificar input do sistema.

- Manter arquivo de projeto.
  - Salvar modelo de projeto.
  - Editar nome do arquivo.
  - Abrir modelo do projeto.
  - Escolher o formato (use case ou rede de Petri).
- Verificar consistência da entrada do sistema.
  - Verificar existência de erro.
  - Indicar local do erro.
- Gerar seqüência de use case a partir de rede de Petri.
  - Transferir para linguagem BNF.
  - Transferir para seqüência de use case.
  - Verificar input do sistema.

### 7.2.4 Atores do sistema e diagramas de caso de uso

Como parte da modelagem, deve-se partir para a descrição dos processos envolvidos. O modelo de um processo é uma descrição simplificada do mesmo de acordo com uma perspectiva em particular, dessa forma podem existir diferentes modelos de um mesmo processo.

Para o sistema em desenvolvimento, devido à sua atual e baixa complexidade, listaram-se dois processos que consistem de:

- Geração de por redes de Petri.
- Geração de seqüência de use case.

Abaixo na Figura 7-10 apresenta-se um fluxograma dos processos:

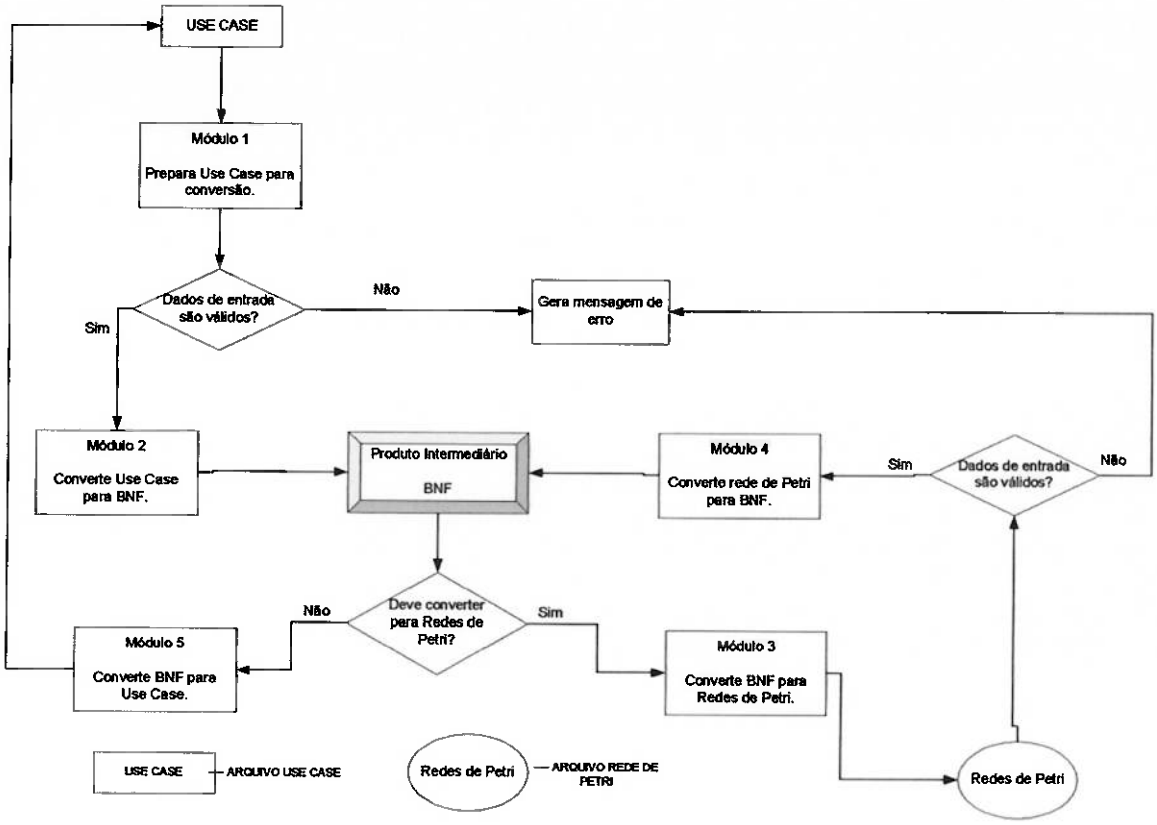



Figura 7-11 Fluxograma de processos do sistema

Define-se atores do processo como as pessoas encarregadas de conduzi-lo, normalmente são escalados conforme a sua ocupação dentro do sistema. Para os processos pertencentes ao sistema, definimos dois atores:

- **Usuário**.
- **Simulador de Redes de Petri :** sistema de simulação de redes de Petri.

Para modelar os processos, baseado na UML, parte-se para a apresentação dos *use cases* do sistema, com o domínio definido como o Sistema de simulação de use case.

Abaixo na Figura 7-11 apresenta-se o diagrama use cases, confeccionado na ferramenta Rose da Rational Software:

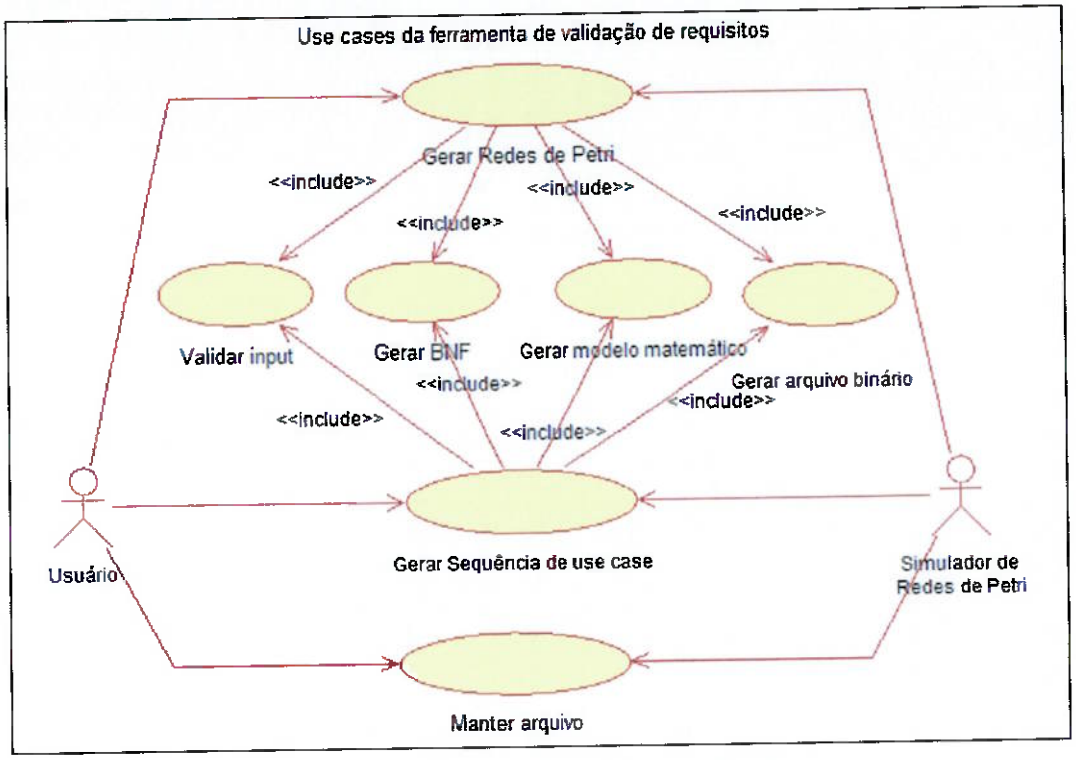



Figura 7-12 Diagrama de Use Case do Sistema

Abaixo apresenta-se a descrição de cada use case:

5) Use case : Gerar de Redes de Petri.
   Ator : Usuário e Simulador de Redes de Petri.
   Descrição : Analisa a entrada e gera arquivo em formato ASCII com a descrição matemática da rede de Petri. Caso não exista consistência nos dados de entrada, gera-se mensagem de erro.
   Entrada : Arquivo padrão ASCII com descrição do use case no formato seqüencial.
   Saída Principal : Geração de Rede de Petri.
   Saída Alternativa : Mensagem de erro.

6) Use case : Manter arquivo.
   Ator : Usuário e Simulador de Redes de Petri.
   Descrição : Mantém as modificações feitas no projeto, salvando, editando o nome e abrindo o arquivo em formato ASCII com a descrição do use case (seqüência ou rede

de Petri) em formato de linguagem BNF. A mensagem de erro é gerada caso algum erro ocorra.

Entrada : Arquivo de projeto.

Saída Principal : Arquivo salvo em linguagem BNF *.

Saída Alternativa : Mensagem de erro.

7) Use case : Validar input.

Descrição : Verifica se as entradas do sistema são consistentes, caso contrário gera mensagem de erro, indicando o local do erro.

Entrada : Arquivo em rede de Petri ou seqüência.

Saída Principal : Nenhum erro ocorreu, continua processo.

Saída Alternativa : Mensagem de erro.

8) Use case : Gerar seqüência de use case.

Ator : Usuário e Simulador de Redes de Petri.

Descrição : Analisa a entrada e gera arquivo em formato ASCII com a descrição da seqüência de use case especificada pela Rede de Petri. Caso não exista consistência nos dados de entrada gera mensagem de erro.

Entrada : Arquivo padrão ASCII com descrição da Rede de Petri.

Saída Principal : Arquivo em formato seqüência de use case.

Saída Alternativa : Mensagem de erro.

9) Use case : Gerar BNF *.

Descrição : Analisa a entrada e gera arquivo em formato ASCII com a descrição do use case em linguagem BNF.

Entrada : Arquivo em formato ASCII com descrição de Rede de Petri ou seqüência de use case.

Saída Principal : Arquivo criado.

Saída Alternativa : Mensagem de erro.

10) Use case : Gerar modelo matemático.

Descrição : Gera rede de Petri em modelo matemático.

Entrada : Descrição do use case em linguagem BNF ou arquivo binário do simulador.
Saída Principal : Descrição do use case em linguagem BNF ou arquivo binário do simulador.
Saída Alternativa : Mensagem de erro.

11) Use case : Gerar binário.
Descrição : Gera arquivo binário.
Entrada : Modelo matemático da rede de Petri.
Saída Principal : Arquivo de entrada do simulador.
Saída Alternativa : Mensagem de erro.

* Optou-se por utilizar o formato BNF como padrão de arquivo salvo em função da facilidade de modular o sistema, sendo o passo intermediário entre redes de Petri e seqüência use case. Com o mesmo propósito criou-se o use case "Gerar BNF" afim de permitir futuras mudanças ou implementações, tais como interface com software de simulação de Redes de Petri e software de engenharia de requisitos.

# 8 IMPLEMENTAÇÃO DOS SOFTWARES

Estando concluídas as fases de implementação e testes, descreve-se neste capítulo o software desenvolvido como produto final deste Trabalho de Graduação. Primeiro descrevemos os requisitos impostos para o correto funcionamento do sistema seguindo uma descrição sucinta do funcionamento do programa.

Para finalizar segue uma descrição mais técnica do software com diagramas descrevendo as classes utilizadas e finalmente uma seção para as restrições encontradas durante o desenvolvimento.

No que segue, descreve-se os dois sistemas abordados anteriormente.

## *8.1 Sistema 1*

### 8.1.1 Requisitos

Para satisfazer os requisitos descritos anteriormente para o sistema 1, foram criados esteriótipos para as atividades e as decisões, já que o Rational Rose não tinha os estereótipos necessários e a UML permite que essa extensão seja realizada. Esses esteriótipos e os símbolos são descritos abaixo:

#### 8.1.1.1 Atividades

Foram criados os seguintes estereótipos:
- Principal (P): para denotar atividades pertencentes ao fluxo principal
- Alternativa n (A1, A2 e A3): para denotar atividades pertencentes aos fluxos alternativos. Podemos ter até três níveis de atividades alternativas.

#### 8.1.1.2 Decisões

Foram criados os seguintes estereótipos:
- Principal (P): para denotar decisões pertencentes ao fluxo principal
- Alternativa n (A1 e A2): para denotar decisões pertencentes aos fluxos alternativos. Podemos ter até dois níveis de decisões alternativas.

### 8.1.2 Criação dos Estereótipos

Para que os estereótipos estejam disponíveis a partir do Rational Rose durante a criação dos diagramas de atividades e geração do BNF, foi alterado o arquivo (defaultstereotypes.ini) que contém a descrição dos elementos UML e seus esteriótipos. Basicamente foram acrescentadas as definições ActivityState (P, A1, A2 e A3) e Decision (P, A1 e A2). O Anexo A no final deste relatório apresenta o conteúdo desse arquivo com a extensão necessária à criação dos esteriótipos.

### 8.1.3 Funcionamento

A partir do Rational Rose, o usuário deve criar inicialmente um caso de uso e dentro desse caso de uso, inserir um diagrama de atividades conforme está mostrado na figura abaixo.

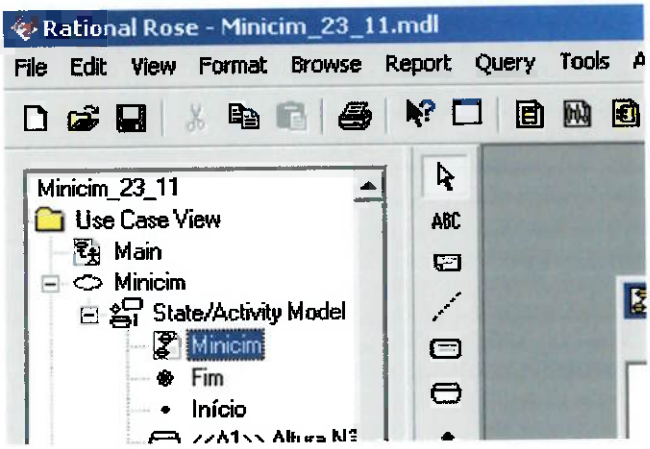



Figura 8-1 Criação do Diagrama de Atividades em um Caso de Uso

A partir desse diagrama de atividades, o usuário deverá então criar os estados incial e final, as atividades principal e alternativas e as decisões principal e alternativas. Para cada elemento criado, o usuário deverá atualizar suas propriedades, como nome, documentação e esteriótipo (das atividades e decisões apenas), já que a aplicação de geração do BNF utiliza essa informação. Na figura abaixo, segue um exemplo de parte de um diagrama de atividades criado dentro do Rational Rose. No final deste item, segue o diagrama de atividades para o caso de uso do Minicim.

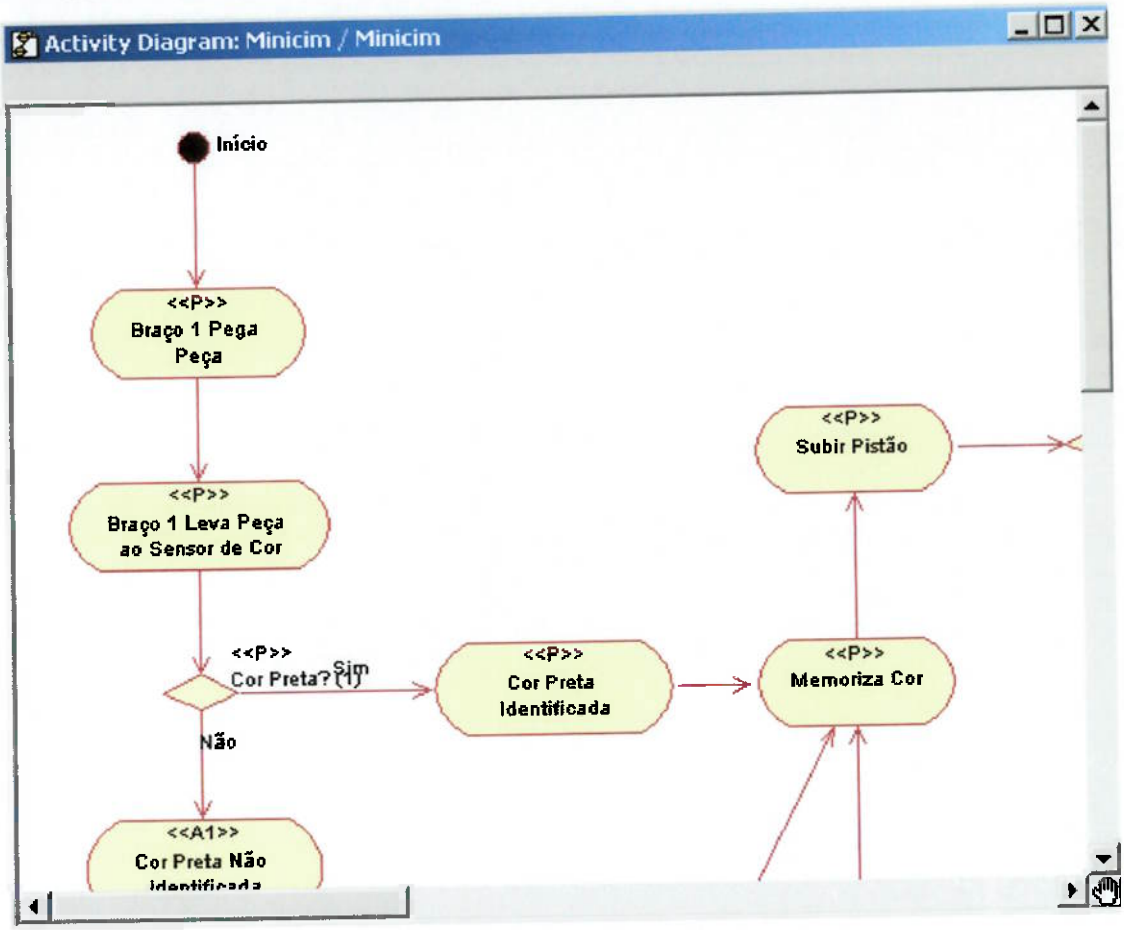



Figura 8-2 Criação do Diagrama de Atividades no Rose

Após criar o diagrama de atividades com todos os seus elementos, o usuário deverá então carregar o script da aplicação de conversão para BNF. No Rational Rose, ele deverá clicar em Tools -> Open Script..., conforme a figura abaixo.

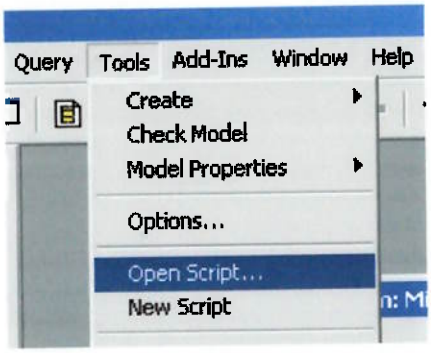



Figura 8-3 Menu de Acesso ao Script

Na janela que surgir, ele então deverá selecionar o *script* Gerador BNF.ebs. Após abrir o script, ele deverá executar o *script* de geração do BNF através do botão *Start*, conforme a figura abaixo:

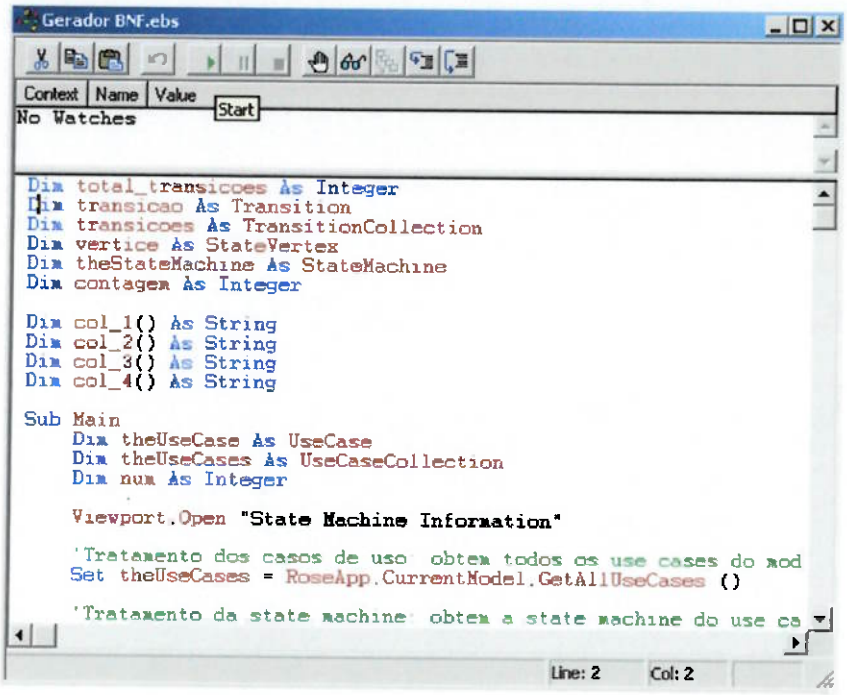



Figura 8-4 Inicialização do Script de Geração do BNF

O resultado da execução do script será uma descrição textual do diagrama de atividades no formato BNF, conforme a figura abaixo:

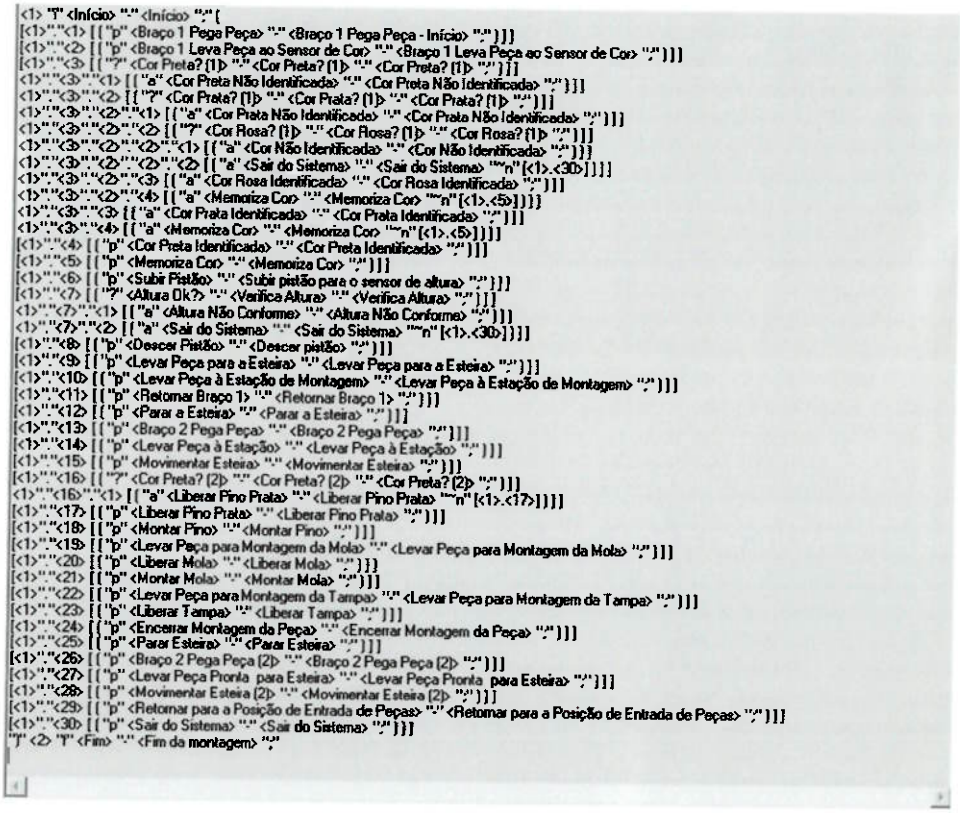



Figura 8-5 Saida em BNF a partir do Diagrama de Atividades

Essa será a saída para o Sistema 2.

Abaixo segue o diagrama de atividades para o Minicim, cuja saída BNF está apresentada acima.

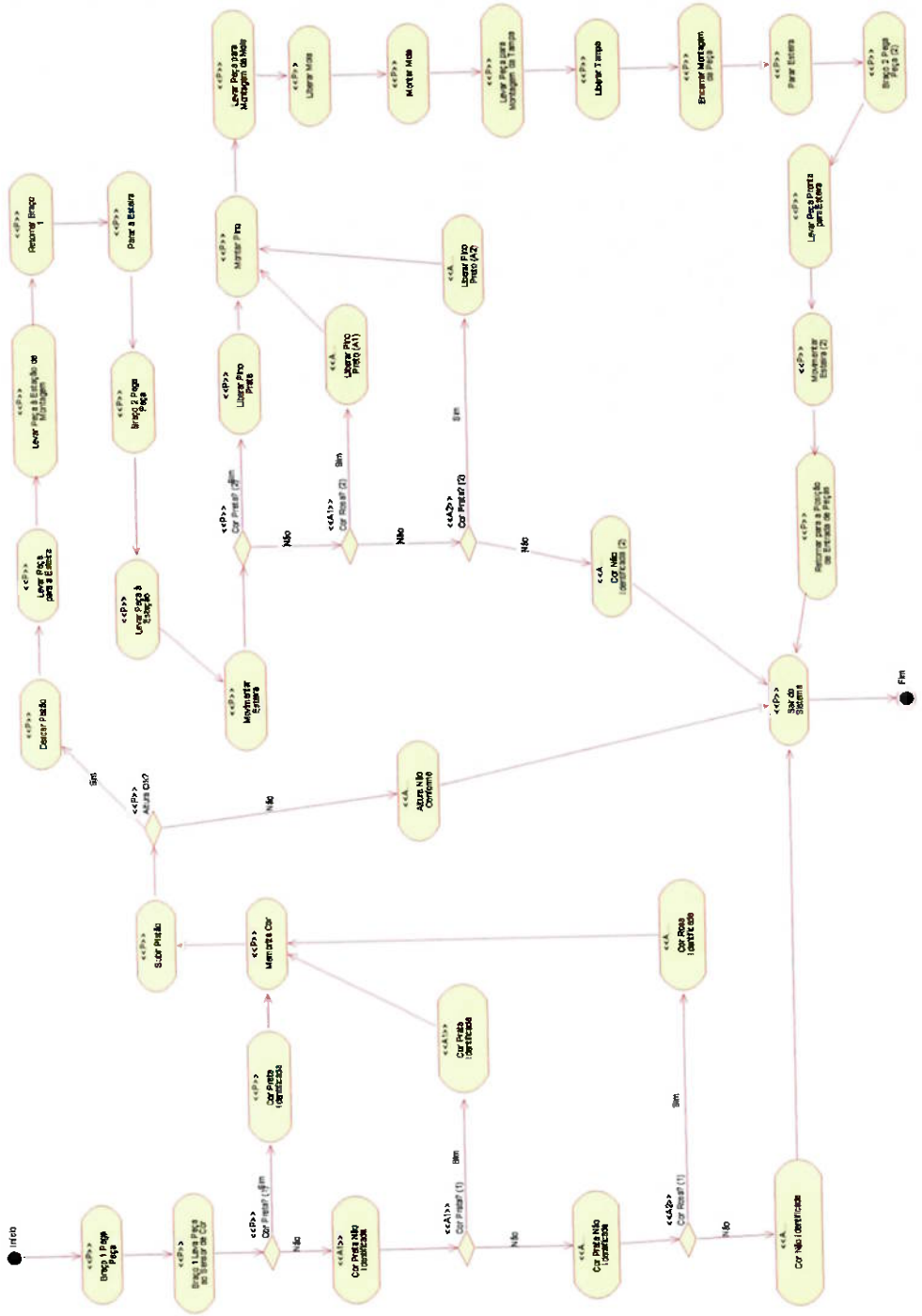



Figura 8-6 Diagrama de Atividades para o Minicim

### 8.1.4 Estrutura

O Rational Rose identifica cada elemento de forma única através de um *Unique ID*. No sistema 1, a busca e identificação de elementos se dá através de unique id. O sistema está dividido em várias funções e a descrição de cada uma segue abaixo:

- Módulo Principal (Sub Main): esse módulo inicia os objetos do Rose necessários à extração das informações do diagrama de atividades. O principal objeto, que dá acesso ao Rose é o RoseApp. Os objetos theUseCase e theStateMachine, extraem informações e dão acessos para o casos de uso e máquinas de estado respectivamente.

A partir dessa função, é inicializado o processo de geração do BNF através da chamada ao módulo ObtemTransições.

- Módulo ObtemTransicoes: esse módulo é responsável pela geração da saida BNF. Para isso ele executa chamadas a funções de busca do início do diagrama de atividades, dos fluxos principal e alternativos e do final do diagrama.

- Função Busca_ID: essa função é responsável pelo levantamento das transições que contém elementos (atividades e decisões) de esteriótipo do tipo principal. Aqui também são identificados os pontos de surgimento de fluxos alternativos.

- Função Obtem_Alternativos: essa função é responsável pelo levantamento das transições que contém elementos (atividades e decisões) de esteriótipo do tipo alternativo. A busca de fluxos alternativos é realizada de forma recursiva.

- Função Obtem_ID_Estado: esta função realizada a busca dos estados inicial e final do diagrama de atividades. Se não houver algum ou nenhum desses elementos, o sistema emite uma mensagem de erro avisando o fato.

- Função Transicao_final: esta função realiza a busca da atividade principal na qual uma atividade alternativa termina.

## *8.2 SISTEMA 2*

### 8.2.1 Requisitos

Para o correto funcionamento foram impostos alguns requisitos sobre o layout dos arquivos a serem tratados, dentre eles temos:

1. Deve ser acrescentada uma linha em branco ao final do arquivo BNF, este fato se deve a uma restrição de "NULL POINTER" encontrada e que a equipe adotou devido ao curto tempo de desenvolvimento solucionar de maneira tão simplória.
2. O arquivo BNF deve ser cuidadosamente formatado de forma a respeitar as tags "<" e ">" assim como os pontos que separam os diferentes níveis de fluxo (e.g. <1>"."<1> fluxo principal e <1>"."<1>"."<1> fluxo alternativo).
3. O arquivo BNF deve conter os caracteres 'i' no início e 'f' no final indicando início e fim do arquivo.

### 8.2.2 Funcionamento

Todo o controle do sistema está disposto em um Menu Inicial conforme demonstrado na Figura 8.1. Existem cinco operações disponíveis conforme numerado, cada uma destas operações possui campos obrigatórios conforme descrito abaixo:

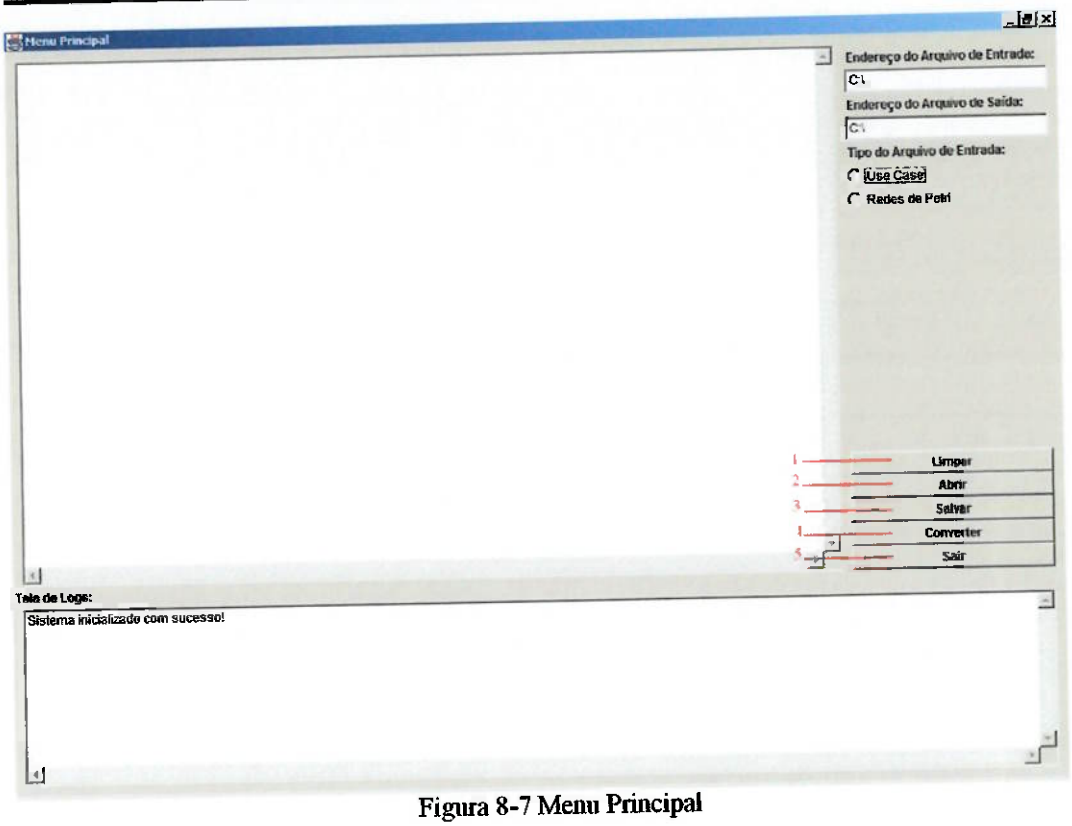



Figura 8-7 Menu Principal

1. Função Limpar: Esta opção apenas limpa todos os campos da tela, com exceção do log de mensagens, para iniciar o trabalho com um novo arquivo.
2. Função Abrir: Esta opção tem como requisito o preenchimento do campo "Endereço do Arquivo de Entrada" incluindo a terminação do arquivo (e.g. "C:\Use Case.txt"), ela abre o arquivo selecionado e apresenta seu conteúdo na tela da interface.
3. Função Salvar: Esta função tem como requisito o preenchimento do campo "Endereço do Arquivo de Saída" incluindo a terminação do arquivo (e.g. "C:\Use Case.txt"), ela salva as alterações realizadas no texto exposto na telano arquivo especificado.
4. Função Converter: Esta função tem como requisito o preenchimento dos campos "Endereço do Arquivo de Saída" e "Endereço do Arquivo de Saída" assim como a seleção de uma das opções de "Tipo de Arquivo de Entrada", ela realiza a conversão de formato do arquivo de entrada para o de saída (e.g. BNF para Rede de Petri).
5. Função Sair: Esta função encerra o sistema.

Para a visualização da Rede de Petri está sendo utilizado um software fruto da dissertação de mestrado de Pedro Luis Angel Restrepo chamado Ghenesys.

### 8.2.3 Estrutura

O conversor RP-BNF & BNF-RP foi estruturado em basicamente quatro blocos distintos, os quais interagem entre si, dessa forma, objetivou-se a maior modularização do programa, facilitando eventuais mudanças e futuras implementações. Esses blocos serão apresentados a seguir:

1. Bloco Menu principal: Inclui as classes responsáveis pela manipulação dos arquivos de entrada e saída, e chamada para execução dos blocos seguintes.
    - Application1.java: contém o método *main*;
    - bancodados.java: contém a estrutura para armazenamento das linhas a serem exibidas na caixa de texto e salvas no arquivo de saída;
    - Frame1.java: implementa a interface gráfica, e as ações dos botões, responsável pelas chamadas das principais funcionalidades do programa.

2. Bloco conversor Redes de Petri para BNF: Inclui as classes responsáveis pela conversão do arquivo em rede de Petri (RP) no formato da Ghenesys para o formato BNF, descrição de use case.
    - Conversor_RP_BNF.java: classe responsável pelo acesso ao arquivo de entrada (RP) e classe principal na conversão de RP para BNF, chamando as outras classes;
    - Trata_Elemento.java: classe responsável pela interpretação dos dados lidos do arquivo RP, interpretando e criando as listas ligadas de elementos formadores da rede de Petri;
    - Entrada_BNF.java: classe responsável pela criação da lista ligada de eventos a partir das outras listas ligadas;
    - Cria_BNF.java: responsável pelo acesso ao arquivo de saída BNF e impressão dos dados da lista de eventos criada anteriormente;

3. Bloco conversor BNF para Redes de Petri: Inclui as classes responsáveis pela conversão do arquivo em formato BNF para o formato em rede de Petri da Ghenesys (RP).

- Conversor_BNF_RP.java: classe responsável pelo acesso ao arquivo de entrada (BNF) e classe principal na conversão de BNF para RP, chamando as outras classes;
- Trata_Evento.java: classe responsável pela interpretação dos dados lidos do arquivo BNF, interpretando e criando a lista ligada de eventos;
- Entrada_Ghenesys.java: classe responsável pela interpretação da lista ligada de eventos e criação das listas ligadas de elementos formadores da rede de Petri;

4. Bloco Banco de Dados: Inclui as classes necessárias para criação das diversas listas ligadas utilizadas pelo programa, cada qual contendo estruturas pré-determinadas.
   - Arco.java: contém os elementos que descrevem um arco e um ponteiro para o próximo arco da lista ligada;
   - Evento.java: contém os elementos que descrevem um evento e um ponteiro para o próximo evento da lista ligada;
   - Lugar.java: contém os elementos que descrevem um lugar e um ponteiro para o próximo lugar da lista ligada;
   - PLugar.java: contém os elementos que descrevem um pseudolugar e um ponteiro para o próximo pseudolugar da lista ligada;
   - Transicao.java: contém os elementos que descrevem uma transição e um ponteiro para a próxima transição da lista ligada;

Após breve descrição das funcionalidades das classes, apresenta-se um diagrama de classes, mostrando a interação entre elas.

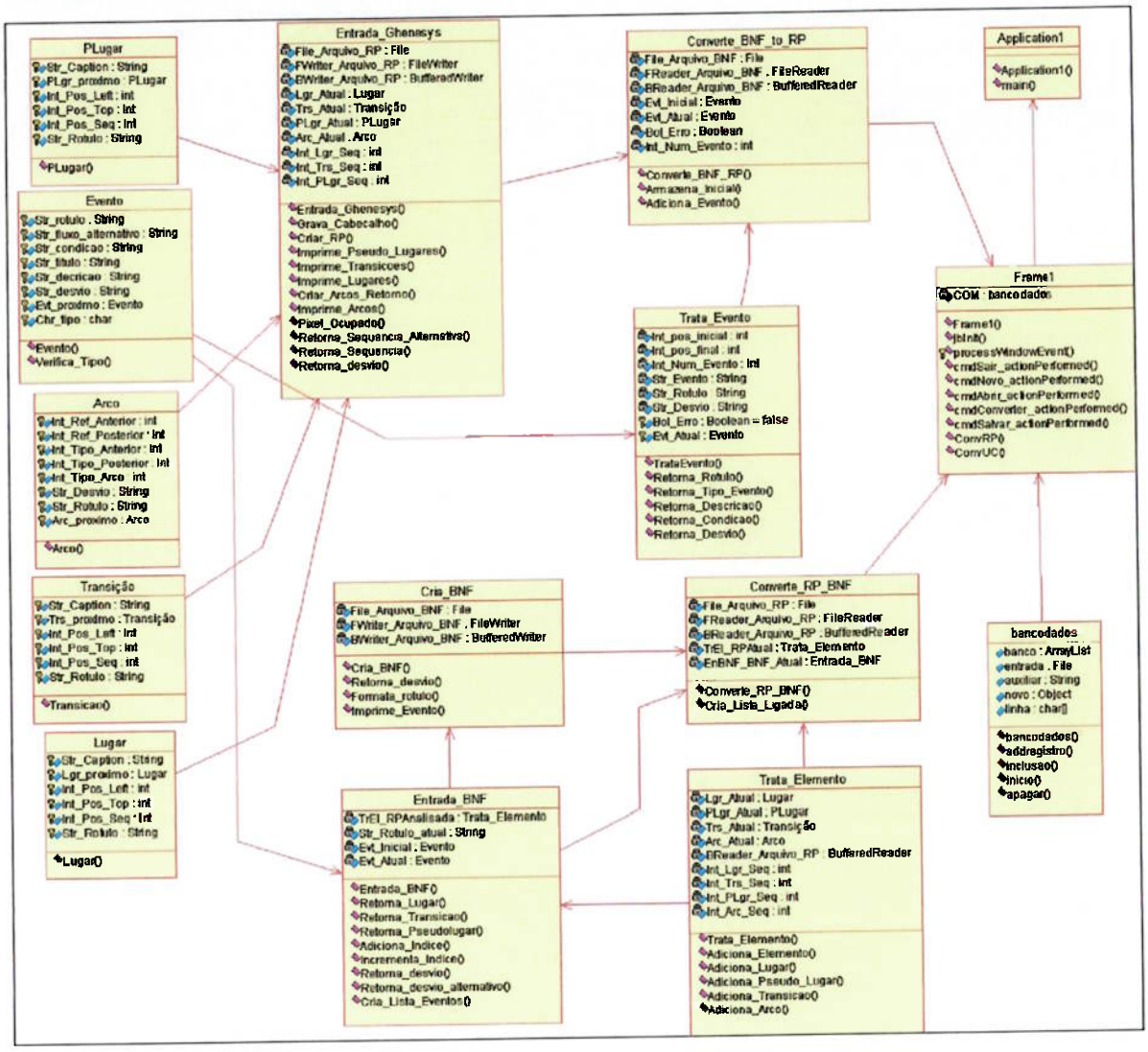



Figura 8-8 Diagrama de Classes

Para implementação das funcionalidades principais do programa (conversão de formato BNF para RP e RP para BNF), a equipe se baseou na seqüência de operações propostas, sendo base para execução do programa, este baseado na estrutura apresentada acima.

Há dessa forma dois processos principais, a conversão de BNF para Rede de Petri, e a conversão de Rede de Petri para BNF, apresentados abaixo.

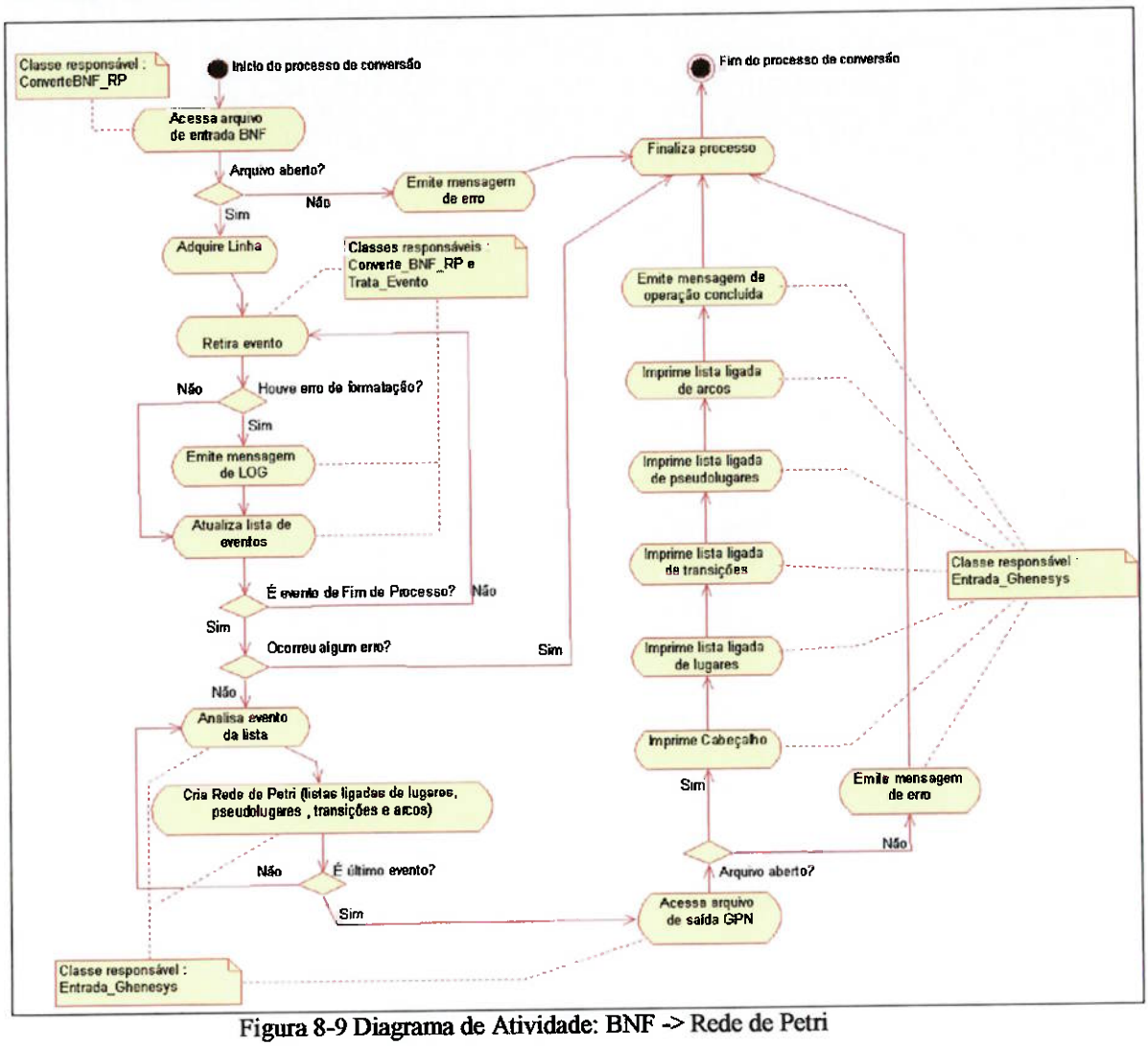



Figura 8-9 Diagrama de Atividade: BNF -> Rede de Petri

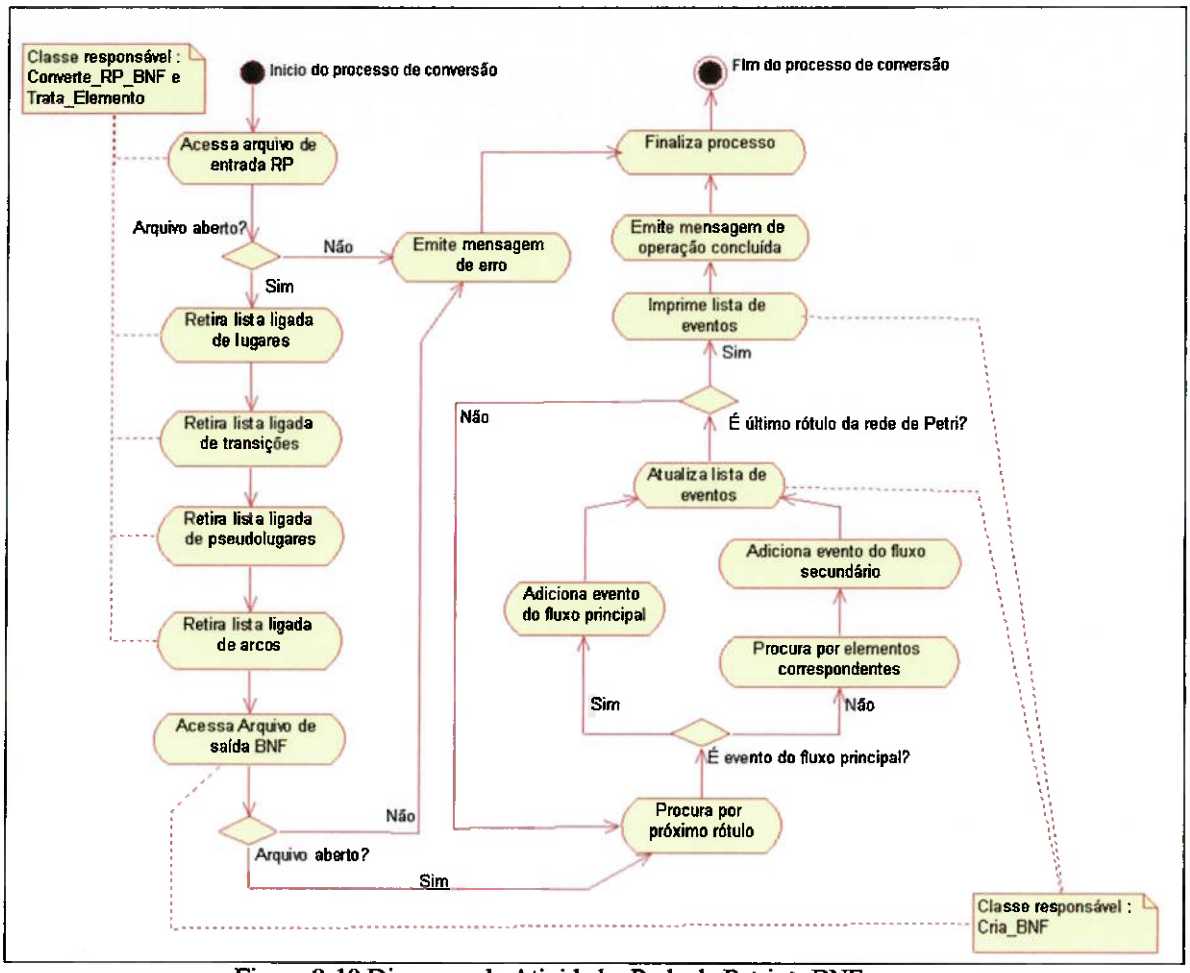



Figura 8-10 Diagrama de Atividade: Rede de Petri -> BNF

### 8.2.4 Restrições

Devido ao curto período de tempo disponível para o desenvolvimento do trabalho, especificamente da parte de implementação, alguns pontos se tornaram críticos ou restritos a algumas condições casuais de funcionamento. A seguir apresentam-se essas restrições detectadas pela equipe.

Na fase de tratamento dos eventos para conversão em redes de Petri, a linha de desenvolvimento seguiu por tratar a interpretação de eventos caso a caso, desse modo para os eventos chamados de condicionais e alternativos, a criação dos elementos da rede de Petri foi dependente do nível especificado pelo rótulo (1.5.1, 1.3, etc), sendo o terceiro nível o máximo permitido, assim é possível posicionar no máximo três eventos condicionais seguidos.

Ainda com relação a esses eventos, há o caso em que a seqüência sugere eventos condicionais no mesmo nível, seguido de eventos alternativos, este caso não funcionaria para nível maiores que dois (rótulos com o formato x.y.z.w).

Para análise dos rótulos e títulos dos eventos, o algoritmo procura pelas *tags* de início de campo ('<') e verifica se a *tag* de fim de campo ('>') está dentro dos padrões especificados para o mesmo (se for rótulo, se for título, etc) identificando o erro, caso ocorra. Porém surge uma restrição na identificação onde são especificados os nomes dos eventos, pois caso o algoritmo não consiga identificar a *tag* de início, ele automaticamente gera erro de formatação por uma *NullPointerException*, e fica difícil para o usuário reconhecer o local do erro.

Não foi incluso no algoritmo o armazenamento dos campos de descrição dos eventos, perdendo-se tais informações e nem o reconhecimento de eventos concorrentes.

# 9 CASOS PROPOSTOS

Nesta seção descrevemos os dois casos utilizados para aplicação dos cenários de teste ao sistema desenvolvido, conforme detalhado na seção 8.2.3. O primeiro exemplo retirado da dissertação de Eston Almança dos Santos foi utilizado como modelo para o desenvolvimento. O segundo exemplo foi elaborado baseado no MiniCim (Vide Figura 9.X) do laboratório de automação.

Nesta seção é ainda apresentada uma proposta alternativa à notação retirada da dissertação de Eston para suprir problemas de implementação.

## 9.1 Caso 1 : ATM

Este exemplo utilizado na dissertação de mestrado de Eston Almança dos Santos, na qual este Trabalho de Graduação está fundamentado, serviu de base para o desenvolvimento do software elaborado.

Trata-se da ilustração da operação de saque de dinheiro de uma ATM, máquina de serviços bancários automática. A modelagem foi desenvolvida pela Rational Software e o modelo abaixo é uma notação proposta por Eston.

### 9.1.1 Notação BNF proposta por Eston

1 ● Inicia Saque – O Cliente insere o cartão do banco na leitora de cartões da máquina ATM; (
1.1 ◇ Cartão é Válido? – Verifica Cartão do Banco – A ATM lê o código da conta da tarjeta magnética do cartão do banco e checa se o mesmo é válido;

- 1.1.1 ↳ Emite mensagem Cartão Inválido – Se o cartão for inválido, o mesmo é devolvido e a máquina emite uma mensagem apropriada. ~1.9

1.2 → Solicita Senha – A ATM solicita a senha do cliente (4 dígitos);
1.3 ◇ Senha Correta e Conta válida? – Verifica Código da Conta e Senha – O código da conta e senha são verificados para determinar se a conta é válida e se a senha informada é a senha correta para a conta;

- 1.3.1 ◇ Conta Válida? – Verifica Código da Conta – O código da conta é verificado;
  - 1.3.1.1 ↳ Emite mensagem Conta Inválida – O sistema do Banco retorna um código indicando que a conta não pode ser encontrada ou que não é uma conta que permite saques. ~1.9
- 1.3.2 ◇ Número de tentativas >= 3? – Verifica número de tentativas – O cliente tem três tentativas para informar a senha correta;

1.3.2.1↳ Emite mensagem Senha Incorreta – A ATM exibirá uma mensagem. ~1.2

1.3.3↳ Emite mensagem Excedeu número de tentativas – Ao final das tentativas o cartão será retido e a ATM retorna para o estado pronto. ~2

1.4 ◇ ATM possui Dinheiro? – Seleciona Saque – A ATM mostra as diferentes opções disponíveis. Nesse caso o cliente do banco sempre seleciona "Saque de Dinheiro";

1.4.1↳ Emite mensagem ATM sem Dinheiro – Se a ATM estiver ser dinheiro, a opção "Saque" não estará habilitada. ~1.4

1.5 ◇ Valor Suficiente e não Excede Limite? – Solicita Valor – A ATM solicita o valor do saque. Nesse caso o cliente seleciona os valores pré estabelecidos ($10, $20, $50, ou $100);

1.5.1 ◇ Valor suficiente? – Verifica valor suficiente – Verifica se existe dinheiro suficiente;

1.5.1.1↳ Emite mensagem Valor Insuficiente na ATM – A ATM não possui dinheiro suficiente para atender ao valor. ~1.5

1.5.2↳ Emite mensagem Valor Excede Limite de Saque Diário – O sistema do banco retorna um código indicando que, incluindo essa solicitação de saque, o cliente excedeu ou terá excedido o limite permitido no período de 24 horas. ~1.5

1.6 ◇ Saque Autorizado? – Autoriza saque – A ATM inicia o processo de verificação com o Sistema do Banco enviando o ID do cartão, a senha, valor e informações da conta como uma transação. Nesse ponto, o Sistema do Banco está on-line e retorna com a autorização para completar o saque com sucesso, atualizando a conta de acordo o saque;

1.6.1↳ Emite mensagem Saldo Insuficiente na Conta – O sistema do banco retorna um código informando que o saldo da conta é insuficiente. ~1.5

1.7 → Dispensa dinheiro – O Dinheiro é dispensado;
1.8 → Emite Recibo – O recibo é impresso e dispensado quando houve saque;
1.9 → Devolve Cartão – O cartão do banco é devolvido;
1.10 ◇ Operação Concluída? – Atualiza *log* – A ATM também atualiza o *log* interno de acordo a operação;

1.10.1↳ Suspende Operação – O *log* não pode ser atualizado, a ATM entrará em "Modo de Segurança". Um alarme apropriado é enviado para o sistema do Banco indicando que a operação da ATM foi suspensa. ~2

) 2 ◉ Finaliza Operação – O Use Case termina com a ATM no estado Pronto;

**Fluxo Alternativo de Eventos**

3↳ Cancela Operação – O cliente pode, a qualquer momento, decidir cancelar a transação (sair). A transação será suspensa e o cartão devolvido. ~1.9

4↳ Ativa sensor de "*Tilt*" – A ATM possui diversos sensores que monitoram diferentes funções, tais como energia elétrica, pressão exercida nos vários compartimentos que possam ser abertos e sensores de movimento. Se, a qualquer

momento, um sensor for ativado, um sinal de alarme é enviado para a Polícia e a ATM entrará em "Modo Seguro" onde todas as funções estarão suspensas até que seja executada uma ação de reinicialização. ~[2]

### 9.1.2 Notação BNF proposta por Eston adaptada a este Trabalho

A escolha para interface entre os programas, tanto os desenvolvidos pela equipe quanto ao Ghenesys, foi a de utilizar arquivos no padrão ASCII, para tal, seguindo a simbologia inicialmente proposta pela dissertação de dissertação de Eston Almança (Referência Bibliográfica ) alguns caracteres não seriam reconhecidos, para resolver tal problema, adotou-se uma nova simbologia para implementação, a qual é descrita abaixo.

| Símbolo | Descrição |
|---|---|
| i | Início de um processo (Use Case) |
| f | Fim de um processo (Use Case) |
| p | Início de um evento do fluxo principal do processo |
| a | Início de um evento do fluxo alternativo ao fluxo principal do processo |
| ? | Início de um evento condicional |
| ~n | Desvio da iteração corrente para a iteração [n] |
| \|\| | Seqüência de eventos concorrentes |

Tabela 9-1 Simbologia alternativa para descrição do BNF adequada ao padrão ASCII

Abaixo apresentamos o código BNF obtido com a nova notação proposta para o modelo da ATM.

#### 9.1.2.1 Modelo BNF final da ATM

```
<1> "i"  <Inicia Saque> "-" <O Cliente insere o cartao do banco na
leitora de cartoes da maquina ATM> ";" "("

      [<1>"."<1> [ ( "?" <Cartao e Valido?> "-" <Verifica Cartao do
      Banco> "-" <A ATM le o codigo da conta da tarjeta magnetica
      do cartao do banco e checa se o mesmo e valido> ";"

            <1>"."<1>"."<1> [ ( "a" <Emite mensagem Cartao
            Invalido> "-" <Se o cartao for invalido, o mesmo e
            devolvido e a maquina emite uma mensagem
            apropriada.>"~n" [<1>.<9>] ) ] ) ] ]

      [<1>"."<2> [ ( "p" <Solicita Senha> "-" <A ATM solicita a
      senha do cliente (4 digitos)>";" ) ] ]

      [<1>"."<3> [ ( "?" <Senha Correta e Conta valida?> "-"
      <Verifica Codigo da Conta e Senha> "-" <O codigo da conta e
      senha sao verificados para determinar se a conta e valida e
      se a senha informada e a senha correta para a conta>";"
```

```
        <1>"."<3>"."<1> [ ( "?" <Conta Valida?> "-" <Verifica
        Codigo da Conta> "-" <O codigo da conta e
        verificado>";"

              <1>"."<3>"."<1>"."<1> [ ( "a" <Emite mensagem
              Conta Invalida> "-" <O sistema do Banco retorna
              um codigo indicando que a conta nao pode ser
              encontrada ou que nao e uma conta que permite
              saques.> "~n" [<1>.<9>] ) ] ) ]
        <1>"."<3>"."<2> [ ( "?" <Numero de tentativas >= 3?> "-
        " <Verifica numero de tentativas> "-" <O cliente tem
        tres tentativas para informar a senha correta>";"

              <1>"."<3>"."<2>"."<1> [ ( "a" <Emite mensagem
              Senha Incorreta> "-" <A ATM exibira uma
              mensagem.> "~n" [<1>.<2>] ) ] ) ]

        <1>"."<3>"."<3> [ ( "a" <Emite mensagem Excedeu numero
        de tentativas> "-" <Ao final das tentativas o cartao
        sera retido e a ATM retorna para o estado pronto.> "~n"
        [<2>] ) ] ) ] ]

[<1>"."<4> [ ( "?" <ATM possui Dinheiro?> "-" <Seleciona
Saque> "-" <A ATM mostra as diferentes opcoes disponiveis.
Nesse caso o cliente do banco sempre seleciona "Saque de
Dinheiro"> ";"

        <1>"."<4>"."<1> [ ( "a" <Emite mensagem ATM sem
        Dinheiro> "-" <Se a ATM estiver ser dinheiro, a opcao
        "Saque" nao estara habilitada.> "~n" [<1>.<4>] ) ] ) ]
        ]

[<1>"."<5> [ ( "?" <Valor Suficiente e nao Excede Limite?> "-
" <Solicita Valor> "-" <A ATM solicita o valor do saque.
Nesse caso o cliente seleciona os valores pre estabelecidos
($10, $20, $50, ou $100)>";"

        <1>"."<5>"."<1> [ ( "?" <Valor suficiente?> "-"
        <Verifica valor suficiente> "-" <Verifica se existe
        dinheiro suficiente>";"

              <1>"."<5>"."<1>"."<1> [ ( "a" <Emite mensagem
              Valor Insuficiente na ATM> "-" <A ATM nao possui
              dinheiro suficiente para atender ao valor.> "~n"
              [<1>.<5>] ) ] ) ]

        <1>"."<5>"."<2> [ ( "a" <Emite mensagem Valor Excede
        Limite de Saque Diario> "-" <O sistema do banco retorna
        um codigo indicando que, incluindo essa solicitacao de
        saque, o cliente excedeu ou tera excedido o limite
        permitido no periodo de 24 horas.> "~n" [<1>.<5>] ) ] )
        ] ]

[<1>"."<6> [ ( "?" <Saque Autorizado?> "-" <Autoriza saque>
"-" <A ATM inicia o processo de verificacao com o Sistema do
Banco enviando o ID do cartao, a senha, valor e informacoes
da conta como uma transacao. Nesse ponto, o Sistema do Banco
esta on-line e retorna com a autorizacao para completar o
saque com sucesso, atualizando a conta de acordo o saque>";"
```

```
            <1>"."<6>"."<1> [ ( "a" <Emite mensagem Saldo
            Insuficiente na Conta> "-" <O sistema do banco retorna
            um codigo informando que o saldo da conta e
            insuficiente.> "~n" [<1>.<5>] ) ] ) ] ]

      [<1>"."<7> [ ( "p" <Dispensa dinheiro> "-" <O Dinheiro e
      dispensado>";" ) ] ]

      [<1>"."<8> [ ( "p" <Emite Recibo> "-" <O recibo e impresso e
      dispensado quando houve saque>";" ) ] ]

      [<1>"."<9> [ ( "p" <Devolve Cartao> "-" <O cartao do banco e
      devolvido>";" ) ] ]

      [<1>"."<10> [ ( "?" <Operacao Concluida?> "-" <Atualiza log>
      "-" <A ATM tambem atualiza o log interno de acordo a
      operacao>";"

            <1>"."<10>"."<1> [ ( "a" <Suspende Operacao> "-" <O log
            nao pode ser atualizado, a ATM entrara em "Modo de
            Seguranca". Um alarme apropriado e enviado para o
            sistema do Banco indicando que a operacao da ATM foi
            suspensa.> "~n" [<2>] ) ] ) ] ]

")" <2> "f" <Finaliza Operacao> "-" <O Use Case termina com a ATM
no estado Pronto> ";"

<3> "*" <Cancela Operacao> "-" <O cliente pode, a qualquer momento,
decidir cancelar a transacao (sair). A transacao sera suspensa e o
cartao devolvido.> "~n" [<1>.<9>]

<4> "*" <Ativa sensor de Tilt> "-" <A ATM possui diversos sensores
que monitoram diferentes funcoes, tais como energia eletrica,
pressao exercida nos varios compartimentos que possam ser abertos e
sensores de movimento. Se, a qualquer momento, um sensor for
ativado, um sinal de alarme e enviado para a Policia e a ATM
entrara em "Modo Seguro" onde todas as funcoes estarao suspensas
ate que seja executada uma acao de reinicializacao.> "~n" [<2>]
```

### 9.1.3 Resultado obtido com a conversão em Rede de Petri

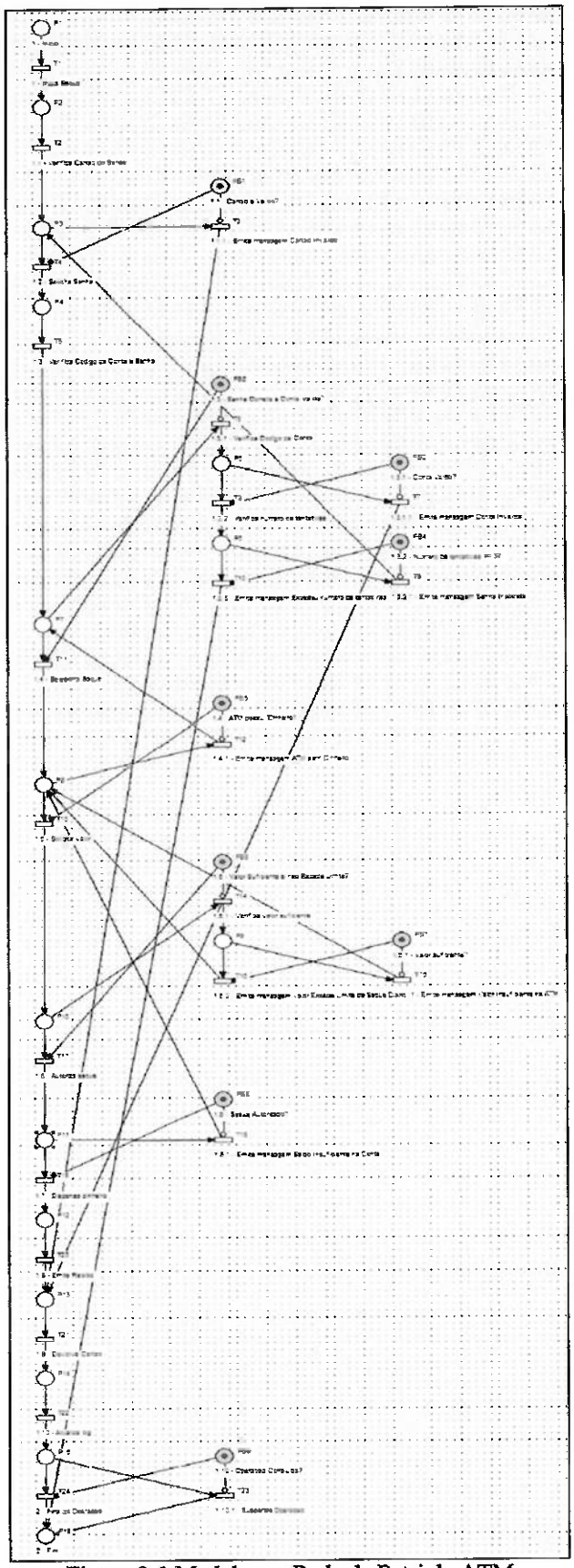


Figura 9-1 Modelo em Rede de Petri da ATM

## 9.2 Caso 2 : MiniCim

Este exemplo foi elaborado pelos membros da equipe para testar o software e ilustrar uma aplicação prática para a área da automação em engenharia.

O sistema em estudo trata-se do MiniCim do laboratório de automação. O MiniCim foi fornecido pela FESTO e simula uma linha de montagem de uma peça.

Esta peça é composta de:

- um cilindro que pode ser das cores preta, prata ou rosa e deve possuir uma altura inferior a um valor máximo especificado;
- um pino que pode ser das cores preta ou prata;
- uma mola e;
- uma tampa;

A linha de montagem identifica a presença de um cilindro na entrada do sistema, um braço mecânico pega o cilindro e leva para uma estação de medição onde a cor e altura são identificadas e caso uma delas não esteja dentro dos padrões esperados a peça é descartada.

Feitas as medições o cilindro é levado para uma esteira que o leva até uma segunda estação onde serão montados os outros componentes e a peça será finalizada.

Nesta estação um segundo braço pega o cilindro e o leva para o mecanismo que libera e monta o pino da cor referente à cor do cilindro identificada. Após esta montagem o braço leva a peça para o mecanismo que monta a mola e posteriormente para o mecanismo que libera a tampa e finaliza a montagem.

Figura 9-2 MiniCim vista 1

Figura 9-3 MiniCim vista 2

Figura 9-4 Sensores de Cor

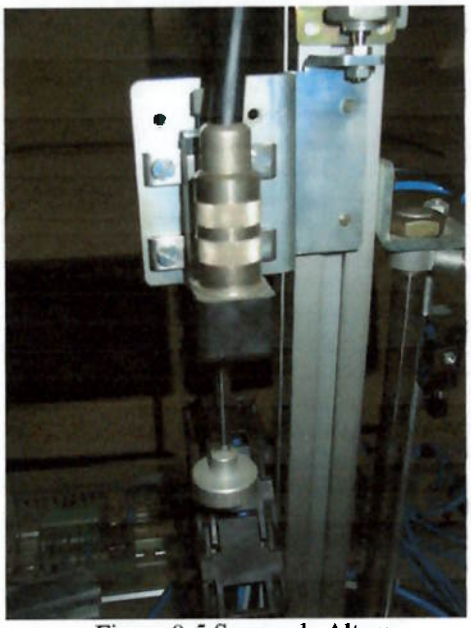


Figura 9-5 Sensor de Altura

### 9.2.1 Modelo BNF do MiniCim : Manual

O modelo abaixo foi escrito pela equipe para fins de teste.

```
<1> "i" <Sensor de Entrada Ativado> "-" <Peca presente na entrada
do sistema> ";" (

      [<1>"."<1> [ ( "p" <Braco 1 pega peca> "-" <Posicao 1> ";" )
      ] ]

      [<1>"."<2> [ ( "p" <Braco 1 leva peca ao sensor de cor> "-"
      <Posicao 2> ";" ) ] ]
```

```
[<1>"."<3> [ ( "?" <Cor Preta?> "-" <Verifica cor> "-"
<Sensor> ";" ) ] ]

      <1>"."<3>"."<1> [ ( "?" <Cor Prata?> "-" <Verifica cor>
      "-" <Segunda verificacao> ";" ) ] ]

            <1>"."<3>"."<1>"."<1> [ ( "?" <Cor Rosa?> "-"
            <Verifica cor> "-" <Terceira verificacao> ";" ) ]
            ]

                  <1>"."<3>"."<1>"."<1>"."<1> [ ( "a" <Cor
                  nao identificada> "-" <Saida do sistema>
                  "~n" [<1>.<32>] ) ] ]

            <1>"."<3>"."<1>"."<2> [ ( "a" <Cor rosa
            identificada> "-" <Proximo passo salvar a cor>
            "~n" [<1>.<5>] ) ] ]

      <1>"."<3>"."<2> [ ( "a" <Cor prata identificada> "-"
      <Proximo passo salvar a cor> "~n" [<1>.<5>] ) ] ]

[<1>"."<4> [ ( "p" <Cor preta identificada> "-" <Proximo
passo salvar a cor> ";" ) ] ]

[<1>"."<5> [ ( "p" <Memoriza cor> "-" <Guarda a cor para
selecao do pino> ";" ) ] ]

[<1>"."<6> [ ( "p" <Subir pistao para sensor de altura> "-"
<Medicao de altura> ";" ) ] ]

[<1>"."<7> [ ( "?" <Altura OK?> "-" <Verifica altura> "-"
<Verificacao da altura> ";" ) ] ]

      <1>"."<7>"."<1> [ ( "a" <Altura fora do padrao> "-"
      <Saida do sistema> "~n" [<1>.<32>] ) ] ]

[<1>"."<8> [ ( "p" <Altura OK> "-" <Verificacao da altura
positiva> ";" ) ] ]

[<1>"."<9> [ ( "p" <Descer pistão> "-" <Retornar pistao para
contiuar processo> ";" ) ] ]

[<1>"."<10> [ ( "p" <Braco 1 pega peca> "-" <Levar para
esteira> ";" ) ] ]

[<1>"."<11> [ ( "p" <Braco 1 leva peca a esteira> "-"
<Posicao 3> ";" ) ] ]

[<1>"."<12> [ ( "p" <Iniciar movimento da esteira> "-" <Levar
a peca para estacao de montagem> ";" ) ] ]

[<1>"."<13> [ ( "p" <Retornar Braco 1 para Posicao 1> "-"
<Verificacao da altura positiva> ";" ) ] ]

[<1>"."<14> [ ( "p" <Parar movimento da esteira> "-" <Estacao
de montagem> ";" ) ] ]

[<1>"."<15> [ ( "p" <Braco 2 pega peca na esteira> "-"
<Posicao 1> ";" ) ] ]
```

```
[<1>"."<16> [ ( "p" <Braco 2 leva peca para estacao de
montagem de pinos> "-" <Posicao 2> ";" ) ] ]

[<1>"."<17> [ ( "p" <Iniciar movimento da esteira> "-"
<Retornar carro para posicao original> ";" ) ] ]

[<1>"."<18> [ ( "?" <Cor Preta?> "-" <Verifica cor> "-"
<Memoria> ";" ) ] ]

      <1>"."<18>"."<1> [ ( "?" <Cor Prata?> "-" <Verifica
      cor> "-" <Memoria> ";" ) ] ]

            <1>"."<18>"."<1>"."<1> [ ( "?" <Cor Rosa?> "-"
            <Verifica cor> "-" <Memoria> ";" ) ] ]

                  <1>"."<18>"."<1>"."<1>"."<1> [ ( "a" <Cor
                  nao identificada> "-" <Saida do sistema>
                  "~n" [<1>.<32>] ) ] ]

            <1>"."<18>"."<1>"."<2> [ ( "a" <Libera pino
            prata> "-" <Proximo passo montar pino> "~n"
            [<1>.<20>] ) ] ]

      <1>"."<18>"."<2> [ ( "a" <Libera pino preto> "-"
      <Proximo passo montar pino> "~n" [<1>.<21>] ) ] ]

[<1>"."<19> [ ( "p" <Libera pino prata> "-" <Proximo passo
montar pino> ";" ) ] ]

[<1>"."<20> [ ( "p" <Montar pino> "-" <Primeira Etapa> ";" )
] ]

[<1>"."<21> [ ( "p" <Braco 2 pega peca> "-" <Posicao 2> ";" )
] ]

[<1>"."<22> [ ( "p" <Braco 2 leva peca para estacao de
montagem de molas> "-" <Posicao 3> ";" ) ] ]

[<1>"."<23> [ ( "p" <Liberar Mola> "-" <Prepara Montagem> ";"
) ] ]

[<1>"."<24> [ ( "p" <Montar mola> "-" <Segunda Etapa> ";" ) ]
]

[<1>"."<25> [ ( "p" <Braco 2 pega peca> "-" <Posicao 3> ";" )
] ]

[<1>"."<26> [ ( "p" <Braco 2 leva peca para estacao de
montagem final> "-" <Posicao 4> ";" ) ] ]

[<1>"."<27> [ ( "p" <Libera tampa> "-" <Prepara Montagem> ";"
) ] ]

[<1>"."<28> [ ( "p" <Finalizar montagem da peca> "-"
<Finalizacao da peca> ";" ) ] ]

[<1>"."<29> [ ( "p" <Braco 2 pega peca> "-" <Posicao 4> ";" )
] ]

[<1>"."<30> [ ( "p" <Parar movimento da esteira> "-" <Saida
final> ";" ) ] ]
```

```
      [<1>"."<31> [ ( "p" <Braco 2 leva peca de volta para esteira>
      "-" <Posicao 1> ";" ) ] ]

      [<1>"."<32> [ ( "p" <Peca sai do sistema> "-" <Fim> ";" ) ] ]

")" <2> "f" <Checar sensor de entrada> "-" <Sistema volta ao estado
inicial> ";"
```

### 9.2.2 Resultado obtido com a conversão em Rede de Petri

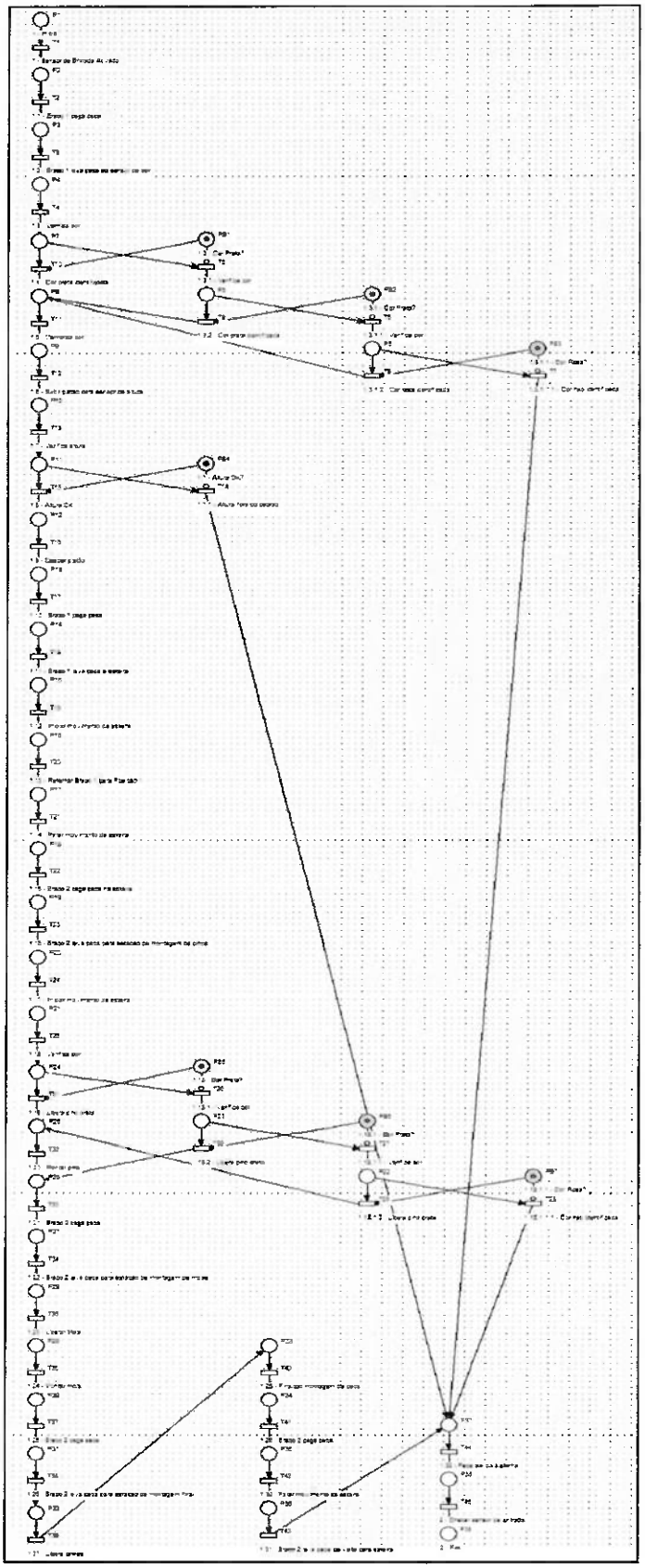


Figura 9-6 Modelo em Rede de Petri do MiniCim*

* - Esta Rede de Petri foi alterada graficamente devido a restrições sobre o tamanho que a Rede pode ocupar. As últimas transições do fluxo principal foram deslocadas lateralmente para visualização neste documento.

Outra observação sobre este modelo em Rede de Petri é que na segunda checagem de cores para seleção do pino não deveria haver verificação das três cores pois a cor já esta memorizada, após a verificação das duas primeiras em caso falso a terceira seria concluída por exclusão e não deve haver a possibilidade de peça descartada pois esse caso já foi abrangido no início do ciclo. Foi inserida esta possibilidade para que o fluxo não fosse interrompido.

## 9.3 Cenários de teste elaborados

Nesta seção descrevemos os cenários de testes elaborados para verificar o funcionamento do software. Os cenários foram divididos em três grupos: Interface Gráfica, Conversão de Use Case em Rede de Petri e Conversão de Rede de Petri em Use Case.

### 9.3.1 Interface Gráfica

Para o teste da interface gráfica foram feitas as seguintes verificações:

- Botão "Limpar" limpa todos os campos da tela com exceção do campo de Log.
- Botão "Abrir" exibe na tela o conteúdo do arquivo especificado no campo "Endereço do Arquivo de Entrada". Em caso de endereço e/ou nome de arquivo inválidos exibe uma mensagem de erro no campo de Log.
- Botão "Salvar" salva o conteúdo exibido na tela no arquivo especificado no campo "Endereço do Arquivo de Saída", em caso de erro exibe uma mensagem de erro no campo Log.
- Botão "Converter" checar se o tipo de arquivo de entrada foi selecionado para identificar qual a conversão a ser realizada. Em caso de tipo selecionado checar se a terminação do arquivo de entrada especificado está coerente com a operação indicada, em caso negativo para as duas verificações exibir uma mensagem de erro no campo de Log. Caso todos os requisitos estejam

corretos invocar a classe correta para realizar as conversões passando os parâmetros necessários.

- Botão "Encerrar" fechar o Menu Principal do programa de conversão.

### 9.3.2 Conversão Use Case -> Rede de Petri

Os cenários para a conversão de Use Case em Rede de Petri abrangem os seguintes pontos:

- Recebe os parâmetros de entrada, saída e log corretamente da interface gráfica conseguindo acessar/criar arquivos e enviando as respectivas mensagens de sucesso/falha.
- Faz a validação das tags e símbolos existentes no arquivo BNF, conforme especificado no item 8.1, e em caso de erro identificado envia mensagem correspondente ao log.
- Gera arquivo gpn (Ghenesys) corretamente permitindo que o software abra uma rede de petri coerente.
- Conversão de casos com um, dois ou três eventos concorrentes (escopo adotado pelo grupo).
- Consegue salvar arquivo corretamente.
- Envia mensagem de término da operação em caso de sucesso.

### 9.3.3 Conversão Rede de Petri -> Use Case

Os cenários para a conversão de Rede de Petri em Use Case abrangem os seguintes pontos:

- Recebe os parâmetros de entrada, saída e log corretamente da interface gráfica conseguindo acessar/criar arquivos e enviando as respectivas mensagens de sucesso/falha.
- Gera arquivo txt (BNF) com formatação e tags de acordo com os requisitos impostos, conforme descrito no item 8.1.
- Consegue salvar arquivo corretamente.
- Envia mensagem de término da operação em caso de sucesso.

# 10 CONCLUSÃO

Conforme citado por Eston em sua dissertação "*A possibilidade que a ferramenta de Verificação de Requisitos terá em se tornar tendência para Validação de Requisitos será grande, uma vez que a incorporação da mesma ao "Rational Suite" deverá despertar o interesse de outros fabricantes de ferramentas de apoio a Engenharia de Requisitos*". Neste Trabalho dois pontos indicados como próximos passos para implementação de seu trabalho foram abordados, são eles:

1. Solidificação da fusão dos diversos pontos de vista do sistema, conforme proposto por (Leite, 1991), de forma a permitir que a verificação dos Requisitos do sistema seja mais intuitiva para os atores envolvidos no sistema. Esta seria a solução para os problemas hoje encontrados com a Especificação de Sistemas.

2. Estabelecimento de uma arquitetura de software para o desenvolvimento da Ferramenta de Verificação de Requisitos, onde toda a troca de informação, tanto com as ferramentas de especificação de requisitos, como com as ferramentas de edição e verificação de rede de Petri, serão baseadas em XML.

Não foi utilizado o padrão xml para troca de informações entre os sistemas, nesta implementação, foi utilizado o formato BNF que se aproxima muito das características do xml sendo estruturado por tags servindo perfeitamente para as comunicações necessárias entre os sistemas.

A principal conclusão obtida com este primeiro passo para a implementação de uma ferramenta de validação de requisitos através de redes de petri simples é que apesar de excelente para visualização do projeto modelado, identificando claramente os fluxos principal, secundário e subseqüentes a rede de petri simples não é a ferramente adequada para simular no caso de projetos complexos de automação. Este fato pode ser observado no segundo caso de teste realizado.

O MiniCim modelado apesar de ser uma representação bastante simplificada de um sistema de automação demonstra claramente esta restrição na utilização de uma rede

de petri simples. O modelo correto para este exemplo deveria contar de três fluxos paralelos, um para o caso de cada cor e somente integrar-se em um fluxo principal após a seleção da cor do pino, onde não mais é necessária a memória de qual cor foi selecionada no início do processo. Para esta modelagem foi utilizado um artifício de indicar como uma das tarefas a "Memorização da cor", entretanto as redes de petri não possuem memória.

Uma solução para este fato é a utilização de redes de petri coloridas onde as marcas também podem carregar informações, desta forma com a seleção da cor no início do sistema a marca pode "carregar" consigo esta informação funcionando como uma memória para a checagem de cor na seleção do pino. Um estudo mais aprofundado sobre redes de petri coloridas foge ao escopo deste trabalho.

Pudemos também constatar que a modelagem de um sistema ou processo (seja ele produtivo ou de negócios) na notação UML e sua posterior transição para a notação BNF constitui um indicador do poder dessa notação, já que podemos utilizar uma linguagem que está mais próxima do usuário final ou de stakeholders que não têm o conhecimento necessários à compreensão das Redes de Petri. Assim podemos descrever as necessidades do usuário em uma ferramenta e em seguida, através da integração com o validador de Redes de Petri, realizar a verificação necessária da coerência da modelagem e realizar as correções necessárias junto ao usuário final.

"*Finalmente, com o apoio da técnica proposta, a possibilidade de se especificar sistemas de grande porte aumenta, uma vez que a divisão do mesmo em pequenas partes e a validação das mesmas pode ser feita recursivamente*" (Eston, 2002).

# ANEXO A – DEFINIÇÃO DE ESTEREÓTIPOS E CÓDIGO FONTE – SISTEMA 1

## Arquivo DefaulStereotypes.ini

```
[Stereotyped Items]
ActivityState:P
ActivityState:A1
ActivityState:A2
ActivityState:A3
Decision:P
Decision:A1
Decision:A2
Class:Interface
Component:EXE
Component:DLL
Component:ActiveX
Component:Application
Component:Applet
Use Case:business use case
Use Case:business use-case realization
Use Case:use-case realization
Class:control
Class:boundary
Class:entity
Class:business actor
Class:business worker
Class:business entity
Class:View
Use Case Package:organization unit
Logical Package:organization unit
Use Case Package:subsystem
Logical Package:subsystem
Use Case Package:layer
Logical Package:layer
DependencyRel:refine
DependencyRel:extend
DependencyRel:include
DependencyRel:derive
Association:extend
Association:include
Association:communicate
Association:subscribe
Association:realize
Class:Table
Component:Database
Class:Domain
Logical Package:Domain Package

[ActivityState:P]
Item=ActivityState
Stereotype=P

[ActivityState:A1]
Item=ActivityState
Stereotype=A1

[ActivityState:A2]
Item=ActivityState
Stereotype=A2

[ActivityStato:A3]
Item=ActivityState
```

```
Stereotype=A3

[Decision:P]
Item=Decision
Stereotype=P

[Decision:A1]
Item=Decision
Stereotype=A1

[Decision:A2]
Item=Decision
Stereotype=A2

[Class:Interface]
Item=Class
Stereotype=Interface

[Component:EXE]
Item=Component
Stereotype=EXE

[Component:DLL]
Item=Component
Stereotype=DLL

[Component:ActiveX]
Item=Component
Stereotype=ActiveX

[Component:Application]
Item=Component
Stereotype=Application

[Component:Applet]
Item=Component
Stereotype=Applet

[Use Case:business use case]
Item=Use Case
Stereotype=business use case
Metafile=&\stereotypes\normal\rube_usecase.wmf
SmallPaletteImages=&\stereotypes\small\rube_usecase_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_usecase_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_usecase_l.bmp
ListIndex=1
ToolTip=Creates a business use case\nBusiness use case

[Use Case:business use-case realization]
Item=Use Case
Stereotype=business use-case realization
Metafile=&\stereotypes\normal\rube_usecase_real.wmf
SmallPaletteImages=&\stereotypes\small\rube_usecase_real_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_usecase_real_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_usecase_real_l.bmp
ListIndex=1
ToolTip=Creates a business use-case realization\nBusiness use-case realization

[Use Case:use-case realization]
Item=Use Case
Stereotype=use-case realization
Metafile=&\stereotypes\normal\usecase_real.wmf
SmallPaletteImages=&\stereotypes\small\usecase_real_s.bmp
SmallPaletteIndex=1
```

```
MediumPaletteImages=&\stereotypes\medium\usecase_real_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\usecase_real_l.bmp
ListIndex=1
ToolTip=Creates a use-case realization\nUse-case realization

[Class:control]
Item=Class
Stereotype=control
Metafile=&\stereotypes\normal\control.wmf
SmallPaletteImages=&\stereotypes\small\control_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\control_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\control_l.bmp
ListIndex=1
ToolTip=Creates a control\nControl

[Class:boundary]
Item=Class
Stereotype=boundary
Metafile=&\stereotypes\normal\boundary.wmf
SmallPaletteImages=&\stereotypes\small\boundary_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\boundary_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\boundary_l.bmp
ListIndex=1
ToolTip=Creates a boundary\nBoundary

[Class:entity]
Item=Class
Stereotype=entity
Metafile=&\stereotypes\normal\entity.wmf
SmallPaletteImages=&\stereotypes\small\entity_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\entity_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\entity_l.bmp
ListIndex=1
ToolTip=Creates an entity\nEntity

[Class:business actor]
Item=Class
Stereotype=business actor
Metafile=&\stereotypes\normal\rube_actor.wmf
SmallPaletteImages=&\stereotypes\small\rube_actor_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_actor_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_actor_l.bmp
ListIndex=1
ToolTip=Creates a business actor\nBusiness actor

[Class:business worker]
Item=Class
Stereotype=business worker
Metafile=&\stereotypes\normal\rube_worker.wmf
SmallPaletteImages=&\stereotypes\small\rube_worker_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_worker_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_worker_l.bmp
ListIndex=1
ToolTip=Creates a business worker\nBusiness worker

[Class:business entity]
Item=Class
```

```
Stereotype=business entity
Metafile=&\stereotypes\normal\rube_entity.wmf
SmallPaletteImages=&\stereotypes\small\rube_entity_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_entity_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_entity_l.bmp
ListIndex=1
ToolTip=Creates a business entity\nBusiness entity

[Class:View]
Item=Class
Stereotype=View
Metafile=&\stereotypes\normal\view.wmf
SmallPaletteImages=&\stereotypes\small\view_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\view_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\view_l.bmp
ListIndex=1
ToolTip=Creates a view\nView


[Use Case Package:organization unit]
Item=Use Case Package
Stereotype=organization unit
Metafile=&\stereotypes\normal\rube_orgunit.wmf
SmallPaletteImages=&\stereotypes\small\rube_orgunit_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_orgunit_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_orgunit_l.bmp
ListIndex=1
ToolTip=Creates an organization unit\nOrganization unit

[Logical Package:organization unit]
Item=Logical Package
Stereotype=organization unit
Metafile=&\stereotypes\normal\rube_orgunit.wmf
SmallPaletteImages=&\stereotypes\small\rube_orgunit_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\rube_orgunit_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\rube_orgunit_l.bmp
ListIndex=1
ToolTip=Creates an organization unit\nOrganization unit

[Use Case Package:subsystem]
Item=Use Case Package
Stereotype=subsystem

[Logical Package:subsystem]
Item=Logical Package
Stereotype=subsystem
SmallPaletteImages=&\stereotypes\small\subsystem_package_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\subsystem_package_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\subsystem_package_l.bmp
ListIndex=1
ToolTip=Creates a subsystem package\nSubsystem Package

[Use Case Package:layer]
Item=Use Case Package
Stereotype=layer

[Logical Package:layer]
Item=Logical Package
```

```
Stereotype=layer

[Association:extend]
Item=Association
Stereotype=extend

[Use Case:use-case collaboration]
Item=Use Case
Stereotype=use-case collaboration]
Metafile=&\stereotypes\normal\usecase_collab.wmf
SmallPaletteImages=&\stereotypes\small\usecase_collab_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\usecase_collab_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\usecase_collab_l.bmp
ListIndex=1
ToolTip=Creates a use-case collaboration\nUse-case collaboration


[DependencyRel:refine]
Item=DependencyRel
Stereotype=refine
SmallPaletteImages=&\stereotypes\small\dependency_refine_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\dependency_refine_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\dependency_refine_l.bmp
ListIndex=1
ToolTip=Refines a use case\nRefine use case

[DependencyRel:extend]
Item=DependencyRel
Stereotype=extend
SmallPaletteImages=&\stereotypes\small\dependency_extend_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\dependency_extend_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\dependency_extend_l.bmp
ListIndex=1
ToolTip=Extends a use case\nExtend use case

[DependencyRel:include]
Item=DependencyRel
Stereotype=include
SmallPaletteImages=&\stereotypes\small\dependency_include_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\dependency_include_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\dependency_include_l.bmp
ListIndex=1
ToolTip=Includes a use case\nInclude use case

[DependencyRel:derive]
Item=DependencyRel
Stereotype=derive
SmallPaletteImages=&\stereotypes\small\dependency_derive_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\dependency_derive_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\dependency_derive_l.bmp
ListIndex=1
ToolTip=Derives a use case\nDerive use case


[Association:include]
Item=Association
Stereotype=include
```

```
[Association:communicate]
Item=Association
Stereotype=communicate

[Association:subscribe]
Item=Association
Stereotype=subscribe

[Association:realize]
Item=Association
Stereotype=realize

[Class:Table]
Item=Class
Stereotype=Table
Metafile=&\stereotypes\normal\table.wmf
SmallPaletteImages=&\stereotypes\small\table_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\table_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\table_l.bmp
ListIndex=1
ToolTip=Creates a table\nTable

[Component:Database]
Item=Component
Stereotype=Database
Metafile=&\stereotypes\normal\database.wmf
SmallPaletteImages=&\stereotypes\small\database_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\database_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\database_l.bmp
ListIndex=1
ToolTip=Creates a database\nDatabase

[Class:Domain]
Item=Class
Stereotype=Domain
Metafile=&\stereotypes\normal\domain.wmf
SmallPaletteImages=&\stereotypes\small\domain_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\domain_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\domain_l.bmp
ListIndex=1
ToolTip=Creates a domain\nDomain

[Logical Package:Domain Package]
Item=Logical Package
Stereotype=Domain Package
Metafile=&\stereotypes\normal\domain_package.wmf
SmallPaletteImages=&\stereotypes\small\domain_package_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\domain_package_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\domain_package_l.bmp
ListIndex=1
ToolTip=Creates a domain package\nDomain Package
```

# Gerador BNF.ebs

```
Dim total_transicoes As Integer
Dim transicao As Transition
Dim transicoes As TransitionCollection
Dim vertice As StateVertex
Dim theStateMachine As StateMachine
Dim contagem As Integer

Dim col_1() As String
Dim col_2() As String
Dim col_3() As String
Dim col_4() As String

Sub Main
        Dim theUseCase As UseCase
        Dim theUseCases As UseCaseCollection
        Dim num As Integer

        Viewport.Open "State Machine Information"

        'Tratamento dos casos de uso: obtem todos os use cases do modelo
        Set theUseCases = RoseApp.CurrentModel.GetAllUseCases ()

        'Tratamento da state machine: obtem a state machine do use case de interesse
        Set theUseCase = theUseCases.GetAt(1)
        Set theStateMachine = theUseCase.StateMachine

        'Tratamento das transições
        Set transicoes = theStateMachine.Transitions

        'Deterima no número de transições
        num = transicoes.Count

        'Redimensiona o array para o tamanho da quantidade de transições do fluxo principal
        ReDim col_1$(num)
        ReDim col_2$(num)
        ReDim col_3$(num)
        ReDim col_4$(num)

        'Inicia a geração das transições
        Call ObtemTransicoes (transicoes)
End Sub

Sub ObtemTransicoes (transicoes As TransitionCollection)
        Dim numero As Integer
        Dim altern As String
        Dim id_state As String
        Dim nome As String
        Dim documentacao As String

        total_transicoes = transicoes.Count
        numero = Busca_ID ("P")

        'Obtem o id do estado inicial e o imprime em BNF
        id_state = Obtem_ID_Estado("Initial")
        nome = theStateMachine.States.GetWithUniqueId(id_state).Name
        documentacao = theStateMachine.States.GetWithUniqueId(id_state).Documentation
        Print "<1> " + """i""" + " <" & nome & "> " + """_""" + " <" & documentacao & "> " + """;""" + " ("

        'Imprime os fluxos principal e alternativos
        For numero = 1 To contagem
                Print "[" + col_1(numero) + col_2(numero)
                If col_4 (numero) = "1" Then
                        altern = Obtem_Alternativos (1, col_3(numero), col_1(numero))
                End If
        Next numero

        'Obtem o id do estado final e o imprime em BNF
        id_state = Obtem_ID_Estado("Final")
        nome = theStateMachine.States.GetWithUniqueId(id_state).Name
        documentacao = theStateMachine.States.GetWithUniqueId(id_state).Documentation
        Print """)""" + " <2> " + """f""" + " <" & nome & "> " + """_""" + " <" & documentacao & "> " + """;"""

End Sub

Function Busca_ID (marca As String) As Integer
        Dim contador As Integer
        Dim numero As Integer
        Dim vertice As StateVertex
        Dim id_vertice As String
        Dim id_cliente As String
        Dim esteriotipo As String
        Dim transicao As Transition
        Dim id_final As String
        Dim simbolo As String

        id_final = Transicao_final()
        id_vertice = Obtem_ID_Estado("Initial")

        If id_vertice = "Sem Estado Inicial/Final!" Then
                Print "Não existe estado inicial ou final no diagrama!"
        ElseIf id_vertice = "Sem Estados" Then
                Print "Não existem estados inicial e final no diagrama!"
        Else
                numero = 0
                Do
                        contador = 0
                        simbolo = "p"
                        Do
```

```
                                        contador = contador + 1
                                        Set transicao = transicoes.GetAt(contador)
                                        id_cliente = transicao.GetClient().GetUniqueId()
                                        esteriotipo = transicao.GetSupplier().Stereotype
                                Loop Until ((id_cliente = id_vertice) And (esteriotipo = marca))
                                numero = numero + 1
                                Set vertice = transicao.GetTargetStateVertex()
                                If vertice.Transitions.Count > 1 Then
                                        simbolo = "?"
                                        col_4(numero) = 1
                                End If
                                id_vertice = vertice.GetUniqueId()
                                col_1(numero) = "<1>" + """.""" + "<" & numero & ">"
                                If vertice.Transitions.Count > 1 Then
                                        col_2(numero) = " [ ( " + Chr(34) + simbolo & Chr(34) & " <" &
vertice.Name & "> " + """-""" + " <" & vertice.Documentation + "> " + """-""" + " <" & vertice.Documentation + ">
" + """;""" + " ) ] ]"
                                Else
                                        col_2(numero) = " [ ( " + Chr(34) + simbolo & Chr(34) & " <" &
vertice.Name & "> " + """-""" + " <" & vertice.Documentation + "> " + """;""" + " ) ] ]"
                                End If
                                col_3(numero) = id_vertice
                        Loop Until (id_vertice = id_final)
                        texto = Transicao_final()
                        contagem = numero
                End If
        End Function

        Function Obtem_Alternativos (marca As Integer, id As String, label_acima As String) As String
                Dim contador As Integer
                Dim numero As Integer
                Dim vertice As StateVertex
                Dim id_vertice As String
                Dim id_cliente As String
                Dim esteriotipo As String
                Dim transicao As Transition
                Dim i As Integer
                Dim simbolo As String
                Dim obtem As String
                Dim label As String
                Dim desvio As String
                Dim complemento As String
                Dim marca_string As String

                id_vertice = id
                numero = 0

                If marca = 1 Then
                        marca_string = "A1"
                ElseIf marca = 2 Then
                        marca_string = "A2"
                Else
                        marca_string = "A3"
                End If

                Do
                        contador = 0
                        Do
                                contador = contador + 1
                                Set transicao = transicoes.GetAt(contador)
                                id_cliente = transicao.GetClient().GetUniqueId()
                                esteriotipo = transicao.GetSupplier().Stereotype
                        Loop Until ((id_cliente = id_vertice) And (esteriotipo = marca_string) Or (id_cliente =
id_vertice) And (esteriotipo = "P"))
                        numero = numero + 1
                        Set vertice = transicao.GetTargetStateVertex()

                        label = label_acima + """.""" + "<" & numero & ">"
                        If vertice.Transitions.Count > 1 Then
                                simbolo = "?"
                                complemento = """-""" + " <" & vertice.Documentation & "> "
                        Else
                                simbolo = "a"
                                complemento = ""
                        End If
                        If esteriotipo = "P" Then
                                i = 0
                                Do
                                        i = i + 1
                                Loop Until (vertice.GetUniqueId() = col_3(i))
                                desvio = """~n""" + " [<1>.<" & i & ">]"
                        Else
                                desvio = """;"""
                        End If
                        label = label_acima + """.""" + "<" & numero & ">" '& " " & vertice.Name
                        Print label + " [ ( " + Chr(34) + simbolo & Chr(34) & " <" & vertice.Name & "> " + """-""" +
" <" & vertice.Documentation + "> " + complemento + desvio + " ) ] ]"
                        id_vertice = vertice.GetUniqueId()
                        If vertice.Transitions.Count > 1 Then
                                marca = marca + 1
                                obtem = Obtem_Alternativos (marca,id_vertice,label)
                        End If
                Loop Until (esteriotipo = "P")
        End Function

        Function Obtem_ID_Estado (tipo_estado As String) As String
                Dim texto As String
                Dim contador As Integer
                Dim total_estados As Integer
                contador = 0
                total_estados = theStateMachine.States.Count
                If total_estados > 0 Then
```

```
                    Do
                            contador = contador + 1
                            texto = theStateMachine.States.GetAt(contador).StateKind
                    Loop While (contador < total_estados And texto <> tipo_estado)
                    If texto <> tipo_estado Then
                            Obtem_ID_Estado = "Sem Estado Inicial/Final!"
                    Else
                            Obtem_ID_Estado = theStateMachine.States.GetAt(contador).GetUniqueId()
                    End If
          Else
                    Obtem_ID_Estado = "Sem Estados"
          End If
End Function

Function Transicao_final () As String
          Dim contador As Integer
          Dim id As String
          contador = 0
          id = Obtem_ID_Estado("Final")
          Do
                    contador = contador + 1
          Loop Until (transicoes.GetAt(contador).GetTargetStateVertex().GetUniqueId() = id)
          Transicao_final = transicoes.GetAt(contador).GetSourceStateVertex().GetUniqueId()
End Function
```

# ANEXO B – CÓDIGO FONTE – SISTEMA 2

## Classe Application1.java

```
package tf;

import javax.swing.UIManager;
import java.awt.*;

public class Application1 {
  boolean packFrame = false;

  //Construct the application
  public Application1() {
    Frame1 frame = new Frame1();
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);

  }
  //Main method
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new Application1();
  }
}
```

## Classe Arco.java

```
package tf;

public class Arco {

    /** notacao do arco 1(inicio) 2(fim)
     *  primeiro - tipo 1
     *  segundo  - numero posicao 1
     *  terceirao- tipo 2
     *  quarto   - numero posicao 2
     *  quinto   - tipo do arco
     **/

    /** tipos de referenciais :
     *  0 - lugar normal
     *  1 - transicao
     *  2 - macro boxes
     *  3 - macro activities
     *  4 - pseudo lugares
     **/

    /** tipos de arcos :
     *  0 - arco normal
     *  1 - arco habilitador
     *  2 - arco inibidor
     **/

    protected int Int_Ref_Anterior;   // Guarda o numero da referencia anterior
    protected int Int_Ref_Posterior;  // Guarda o numero da referencia posterior
    protected int Int_Tipo_Anterior;  // Guarda o tipo da referencia anterior
    protected int Int_Tipo_Posterior; // Guarda o tipo da referencia posterior
    protected int Int_Tipo_Arco;      // Guarda o tipo do arco
    protected String Str_Desvio;      // Guarda o desvio associado ao arco
    protected String Str_Rotulo;      // Guarda o rotulo associado ao arco
    protected Arco Arc_proximo;       // Guarda a referencia ao proximo arco da lista ligada

    /** Creates a new instance of Arco */
    public Arco() {
        Int_Ref_Anterior      = 500;
        Int_Ref_Posterior     = 500;
        Int_Tipo_Anterior     = 500;
        Int_Tipo_Posterior    = 500;
        Int_Tipo_Arco         = 500;
    }
```

```
}
```

## Classe bancodados.java

```
package tf;

import java.util.*;
import java.io.*;
import java.awt.*;

/* Classe que implementa a comunicacao com o arquivo txt */
public class bancodados
{
 public bancodados(String Path, TextArea txtLog){
   inicio(Path,txtLog);
  }

  /* Função que adiciona registro a array de comandos */
 public static void addregistro(String qualquer)
 {
  banco.add(qualquer);
 }

 /* Função que insere uma linha de comando do arquivo na array */
 public String inclusao(String qualquer,TextArea txtLog)
 {
  try{
        /* Objeto que permite a transmissao de dados */
        FileWriter dados2=new FileWriter(entrada);
          BufferedWriter out=new BufferedWriter(dados2);
        linha=qualquer.toCharArray();

        for(int i=0;i<(linha.length);i++){
          if(linha[i]=='$')
                i=i+5;

          out.write(linha[i]);
        }

        out.close();
        return("OK");
    }
    catch(IOException e){
        return("Erro");
     }
   }

  /* Função que inicializa o array de comandos com as linhas do arquivo txt */
 public String inicio(String Path,TextArea txtLog)
 {
    try{
        /* Objetos que permite a transmissao de dados */
        entrada =new File(Path);
        FileReader dados=new FileReader(entrada);
        BufferedReader in=new BufferedReader(dados);

        /* Loop que le linha por linha do arquivo */
        while((auxiliar=in.readLine())!=null)
            addregistro(auxiliar);
        return("OK");
    }
    catch(IOException e){
        return("Erro");
    }
 }

  /* Apaga todos registros de comandos */
 public static void apagar()
 {
   banco.clear();
 }

 /* Array que armazena as linhas de comando */
  public static ArrayList banco   =new ArrayList(50);

 /* Objeto que aponta para o arquivo externo */
  public static File      entrada;
  public static String    auxiliar=new String();
  public static Object    novo    =new Object();
  public static char[]    linha=new char[100];
}
```

## Classe Converte_BNF_RP.java

```
package tf;

import java.lang.*;
import java.io.*;
import java.awt.*;

public class Converte_BNF_RP {

    // variaveis utilizadas pela classe :
    private File                File_Arquivo_BNF;
    // Variável que representa o arquivo de entrada do BNF.
    private FileReader          FReader_Arquivo_BNF;
    // Variável que representa a Stream de entrada de dados do arquivo.
    private BufferedReader      BReader_Arquivo_BNF;
```

```
    // Variável para tratamento dos dados de entrada.
    private Evento              Evt_Inicial;
    // Variável que representa o evento inicial da descrição de use case.
    private Evento              Evt_Atual;
    // Variável que representa os eventos posteriores ao inicial da descrição de use case.
    private boolean             Bol_Erro;
    // Variavel que indica que houve erro
    private int                 Int_Num_Evento;
    // Variavel que guarda o numero do evento na sequencia para indicar erro

    /** Funcao que chama as outras funcoes afim de criar a lista ligada de eventos */
    public Converte_BNF_RP(String Str_Diretorio, String Str_Dir_saida,TextArea txtLog) {

        Bol_Erro                = false;
        Int_Num_Evento          = 2;

        try {
            File_Arquivo_BNF    =  new File(Str_Diretorio);
            FReader_Arquivo_BNF =  new FileReader(File_Arquivo_BNF);
            BReader_Arquivo_BNF =  new BufferedReader(FReader_Arquivo_BNF);

            // Armazena a sequencia de eventos do arquivo
            Armazena_Inicial(txtLog);
            Adiciona_Evento(txtLog);
            if (!Bol_Erro) {
                Entrada_Ghenesys EG_Modelo_RP  =  new Entrada_Ghenesys (Evt_Inicial, Str_Dir_saida,txtLog);
            }
            BReader_Arquivo_BNF.close();
        }

        catch (IOException e) {
            txtLog.append("ERRO : Nao pode acessar o arquivo"+"\n");
        }
    }

    /** Funcao que armazena o evento inicial */
    public void Armazena_Inicial(TextArea txtLog){

        String Str_Evento_Atual;
        int Int_pos_inicial  =  0;
        int Int_pos_final    =  0;

        Evt_Inicial = new Evento("");

        try {
            Str_Evento_Atual        =  new String(BReader_Arquivo_BNF.readLine());

            Trata_Evento TEvt_Atual =  new Trata_Evento (Str_Evento_Atual, 1,txtLog);
            Evt_Inicial             =  TEvt_Atual.Evt_Atual;
        }

        catch (IOException e) {
            txtLog.append("ERRO : Não pode acessar o arquivo!"+"\n");
        }
    }

    /** Funcao que armazena os eventos */
    public void Adiciona_Evento(TextArea txtLog){

        String Str_Evento_Atual;
        Evento Evt_Evento_auxiliar  =  new Evento ("");
        int Int_pos_inicial         =  0;
        int Int_pos_final           =  0;

        try {
            Str_Evento_Atual        =  new String(BReader_Arquivo_BNF.readLine());

            Trata_Evento TEvt_Atual =  new Trata_Evento (Str_Evento_Atual, Int_Num_Evento++ ,txtLog);
            Evt_Evento_auxiliar     =  TEvt_Atual.Evt_Atual;

            if (Evt_Inicial.Evt_proximo == null)
                Evt_Inicial.Evt_proximo  =  Evt_Evento_auxiliar;
            else
                Evt_Atual.Evt_proximo    =  Evt_Evento_auxiliar;

            Evt_Atual  =  Evt_Evento_auxiliar;

            if (TEvt_Atual.Bol_Erro)
                Bol_Erro  =  true;

            Adiciona_Evento(txtLog);
        }

        catch (NullPointerException e)
        {
            txtLog.append ("Foram encontrados erros com relacao a formatacao do arquivo"+"\n");
        }

        catch (IOException e) {
            txtLog.append("ERRO : Não pode acessar o arquivo!"+"\n");
        }

        catch (IndexOutOfBoundsException e) {
            txtLog.append("Chegou no fim de arquivo"+"\n");
        }
    }

    /** Funcao que imprime na tela os parametros do evento procurado pelo seu
     *  rotulo (UTILIZADO SOMENTE PARA TESTE) */
    void Retorna_evento(String Str_rotulo_procurado,TextArea txtLog) {

        Evento Evt_Buscado  =  new Evento("");
```

```
        Evt_Buscado         =  Evt_Inicial;

        try {

            while (Evt_Buscado.Evt_proximo != null) {
                System.out.println(Evt_Buscado.Str_rotulo);
                System.out.println(Evt_Buscado.Chr_tipo);
                System.out.println(Evt_Buscado.Str_condicao);
                System.out.println(Evt_Buscado.Str_titulo);
                System.out.println(Evt_Buscado.Str_descricao);
                Evt_Buscado  =  Evt_Buscado.Evt_proximo;
            }
        }

        catch (NullPointerException e) {
            txtLog.append("Evento nao encontrado"+"\n");
        }

    }

}
```

## Classe Converte_RP_BNF.java

```
package tf;

import java.lang.*;
import java.io.*;
import java.awt.*;

public class Converte_RP_BNF {

    static final String         LUGAR_NOTACAO           =       "[PLACES]";
    static final String         TRANS_NOTACAO           =       "[TRANSITIONS]";
    static final String         PSEUD_NOTACAO           =       "[PSEUDOBOXES]";
    static final String         MBOX_NOTACAO            =       "[MACROBOXES]";
    static final String         MATIV_NOTACAO           =       "[MACROACTIVITIES]";
    static final String         ARCOS_NOTACAO           =       "[ARCS]";
    static final String         FINAL_NOTACAO           =       "[END_GPTF]";
    static final String         INICIO_NOTACAO          =       "[GPNF]";

    private File                File_Arquivo_RP;
    // Variável que representa o arquivo de entrada do RP.
    private FileReader          FReader_Arquivo_RP;
    // Variável que representa a Stream de entrada de dados do arquivo.
    private BufferedReader      BReader_Arquivo_RP;
    // Variável para tratamento dos dados de entrada.
    private Trata_Elemento      TrEl_RPAtual;
    // Variavel que armazena as listas ligadas.
    private Entrada_BNF         EnBNF_BNF_Atual;
    // Variavel que armazena a lista ligada de eventos.

    /** Funcao que chama as outras afim de criar as listas ligadas de elementos */
    public Converte_RP_BNF(String Str_Diretorio, String Str_Dir_saida,TextArea txtLog) {

       try {
            // Inicializa arquivo de entrada
            File_Arquivo_RP     =  new File(Str_Diretorio);
            FReader_Arquivo_RP  =  new FileReader(File_Arquivo_RP);
            BReader_Arquivo_RP  =  new BufferedReader(FReader_Arquivo_RP);

            // Cria lista de lugares, pseudolugares, transicoes e arcos
            Cria_Lista_Ligada (txtLog);

            /* SOMENTE PARA TESTE
            Imprime_Lugares ();
            Imprime_Transicoes ();
            Imprime_Pseudolugares ();
            Imprime_Arcos ();
            SOMENTE PARA TESTE */

            // Fecha buffer de arquivo de entrada
            BReader_Arquivo_RP.close();

            // Cria lista ligada de eventos
            EnBNF_BNF_Atual  =  new Entrada_BNF (TrEl_RPAtual);
            EnBNF_BNF_Atual.Cria_Lista_Eventos ();

            /* SOMENTE PARA TESTE
            EnBNF_BNF_Atual.Imprime_Lista_Eventos ();
            SOMENTE PARA TESTE */

            // Gera arquivo de saida BNF a partir da lista de eventos
            Cria_BNF  Arquivo_BNF = new Cria_BNF(Str_Dir_saida , EnBNF_BNF_Atual,txtLog);

            txtLog.append("MSG: Arquivo BNF gerado com sucesso");
       }

       catch (IOException e) {
            txtLog.append("ERRO : Nao pode acessar o arquivo"+"\n");
       }

       catch (Exception e) {
            txtLog.append ("ERRO : Nao pode criar arquivo BNF"+"\n");
       }
    }

    /** Funcao que cria a lista ligada de lugares, transicoes, pseudolugares e arcos */
    void Cria_Lista_Ligada (TextArea txtLog) {

        TrEl_RPAtual                    =  new Trata_Elemento ( BReader_Arquivo_RP );
```

```
        // Inicializa Trata Elemento com endereco do arquivo de entrada
        String Str_Tipo_Elemento      =  new String ();
        // Variavel que armazena a string de evento
        int    Int_Tipo_Elemento      =  0;
        // Flag que indica qual o tipo do elemento atual a ser analisado

        /**
         *  0 - lugar
         *  1 - transicao
         *  2 - pseudolugar
         *  3 - macrobox
         *  4 - macroatividade
         *  5 - arcos */

        // Loop que varre as linhas do arquivo para montar as listas ligadas
        while (true) {

            try {
                // Retira a linha do arquivo de entrada
                Str_Tipo_Elemento  =  BReader_Arquivo_RP.readLine();

                // Verificou inicio do bloco de lugares
                if ( Str_Tipo_Elemento.equals(LUGAR_NOTACAO) ) {
                    Int_Tipo_Elemento  =  0;
                    //System.out.println ("Detectou lugar");
                }

                // Verificou inicio do bloco de transicoes
                else if ( Str_Tipo_Elemento.equals(TRANS_NOTACAO) ) {
                    Int_Tipo_Elemento  =  1;
                    //System.out.println ("Detectou transicao");
                }

                // Verificou inicio do bloco de pseudolugares
                else if ( Str_Tipo_Elemento.equals(PSEUD_NOTACAO) ) {
                    Int_Tipo_Elemento  =  2;
                    //System.out.println ("Detectou pseudolugar");
                }

                // Verificou inicio do bloco de macrolugares
                else if ( Str_Tipo_Elemento.equals(MBOX_NOTACAO) ) {
                    Int_Tipo_Elemento  =  3;
                    //System.out.println ("Detectou macrobox");
                }

                // Verificou inicio do bloco de macro atividades
                else if ( Str_Tipo_Elemento.equals(MATIV_NOTACAO) ) {
                    Int_Tipo_Elemento  =  4;
                    //System.out.println ("Detectou macroatividade");
                }

                // Verificou inicio do bloco de arcos
                else if ( Str_Tipo_Elemento.equals(ARCOS_NOTACAO) ) {
                    Int_Tipo_Elemento  =  5;
                    //System.out.println ("Detectou arco");
                }

                // Verificou inicio do cabecalho
                else if ( Str_Tipo_Elemento.equals(INICIO_NOTACAO) ) {
                    // Pula quatro linhas
                    for (int i=0 ; i<4 ; i++)
                        BReader_Arquivo_RP.readLine();
                    //System.out.println ("Detectou inicio da rede");
                }

                // Verificou final do arquivo
                else if ( Str_Tipo_Elemento.equals(FINAL_NOTACAO) ) {
                    //System.out.println ("Detectou fim de rede");
                }

                // Caso linha nula nao faca nada
                else if ( Str_Tipo_Elemento.equals("") ) {
                }

                // Caso dado qualquer, representa elemento da lista
                else
                    TrEl_RPAtual.Adiciona_Elemento ( Int_Tipo_Elemento , Str_Tipo_Elemento);
            }

            // Caso ocorra erro de I/O
            catch (IOException e) {
                txtLog.append ("ERRO : Ocorreu erro durante acesso ao arquivo de entrada"+"\n");
                break;
            }

            // Caso ocorra erro de variavel null
            catch (NullPointerException e) {
                break;
            }
        }
    }

    /** Funcao que imprime na tela a lista de lugares criada (SOMENTE PARA TESTE) */
    void Imprime_Lugares () {

        Lugar Lgr_auxiliar  =  new Lugar();
        Lgr_auxiliar  =  TrEl_RPAtual.Lgr_Inicial;

        // Loop que varre a lista ligada a partir do elemento inicial
        while (Lgr_auxiliar.Lgr_proximo  !=  null)
        {
            System.out.print ( Lgr_auxiliar.Str_Caption + Lgr_auxiliar.Int_Pos_Seq + ";");
```

```
            Lgr_auxiliar  =  Lgr_auxiliar.Lgr_proximo;
        }

        System.out.println ( Lgr_auxiliar.Str_Caption + Lgr_auxiliar.Int_Pos_Seq + ";");
    }

    /** Funcao que imprime na tela a lista de pseudolugares criada (SOMENTE PARA TESTE) */
    void Imprime_Pseudolugares () {

        PLugar PLgr_auxiliar  =  new PLugar();
        PLgr_auxiliar  =  TrEl_RPAtual.PLgr_Inicial;

        // Loop que varre a lista ligada a partir do elemento inicial
        while (PLgr_auxiliar.PLgr_proximo  !=  null)
        {
            System.out.print ( PLgr_auxiliar.Str_Caption + PLgr_auxiliar.Int_Pos_Seq + ";");
            PLgr_auxiliar  =  PLgr_auxiliar.PLgr_proximo;
        }

        System.out.println ( PLgr_auxiliar.Str_Caption + PLgr_auxiliar.Int_Pos_Seq + ";");
    }

    /** Funcao que imprime na tela a lista de transicoes criada (SOMENTE PARA TESTE) */
    void Imprime_Transicoes () {

        Transicao Trs_auxiliar  =  new Transicao();
        Trs_auxiliar  =  TrEl_RPAtual.Trs_Inicial;

        // Loop que varre a lista ligada a partir do primeiro elemento
        while (Trs_auxiliar.Trs_proximo  !=  null)
        {
            System.out.print ( Trs_auxiliar.Str_Caption + Trs_auxiliar.Int_Pos_Seq + ";");
            Trs_auxiliar  =  Trs_auxiliar.Trs_proximo;
        }

        System.out.println ( Trs_auxiliar.Str_Caption + Trs_auxiliar.Int_Pos_Seq + ";");
    }

    /** Funcao que imprime na tela a lista de arcos criada (SOMENTE PARA TESTE) */
    void Imprime_Arcos () {

        Arco Arc_auxiliar  =  new Arco();
        Arc_auxiliar  =  TrEl_RPAtual.Arc_Inicial;

        // Loop que varre a lista ligada a partir do primeiro elemento
        while (Arc_auxiliar.Arc_proximo  !=  null)
        {
            System.out.print (Arc_auxiliar.Int_Tipo_Anterior);
            System.out.print (Arc_auxiliar.Int_Ref_Anterior);
            System.out.print (Arc_auxiliar.Int_Tipo_Posterior);
            System.out.print (Arc_auxiliar.Int_Ref_Posterior);
            System.out.print (Arc_auxiliar.Int_Tipo_Arco + ";");
            Arc_auxiliar  =  Arc_auxiliar.Arc_proximo;
        }

        System.out.print (Arc_auxiliar.Int_Tipo_Anterior);
        System.out.print (Arc_auxiliar.Int_Ref_Anterior);
        System.out.print (Arc_auxiliar.Int_Tipo_Posterior);
        System.out.print (Arc_auxiliar.Int_Ref_Posterior);
        System.out.println (Arc_auxiliar.Int_Tipo_Arco + ";");
    }
}
```

## Classe Cria_BNF.java

```
package tf;

import java.lang.*;
import java.io.*;
import java.awt.*;

public class Cria_BNF {

    private File                File_Arquivo_BNF;       // Variável que representa o arquivo de saida BNF.
    private FileWriter          FWriter_Arquivo_BNF;    // Variável que representa a Stream de saida de dados do
arquivo.
    private BufferedWriter      BWriter_Arquivo_BNF;    // Variavel que representa o buffer de saida do arquivo
    private static final String TRACO  =  new String("" + '"' + '-' + '"' + "");
    // Constantes para escrita de caracteres no arquivo
    private static final String PONTO  =  new String("" + '"' + '.' + '"' + "");
    private static final String DESVIO =  new String("" + '"' + "\n" + '"' + "");
    private static final String PVIRGULA= new String("" + '"' + ";" + '"' + "");

    /** Construtor responsavel pela criacao do arquivo e chamada das funcoes para tal */
    public Cria_BNF( String diretorio , Entrada_BNF  entrada,TextArea txtLog) {

        File_Arquivo_BNF    =  new File (diretorio);
        Evento    auxiliar  =  entrada.Evt_Inicial;

        try {
            FWriter_Arquivo_BNF   =  new FileWriter(File_Arquivo_BNF);          // Cria buffer para arquivo de
saida
            BWriter_Arquivo_BNF   =  new BufferedWriter(FWriter_Arquivo_BNF);
            // Cria buffer de escrita para arquivo de saida

            // Varre lista ligada de eventos, imprimindo no arquivo
            while (auxiliar.Evt_proximo != null) {
                Imprime_Evento (auxiliar,txtLog);
                auxiliar  =  auxiliar.Evt_proximo;
            }
```

```
            Imprime_Evento(auxiliar,txtLog);

            // Fecha buffer de saida
            BWriter_Arquivo_BNF.close();
        }

        catch (IOException e) {
            txtLog.append ("ERRO : Arquivo BNF não pode ser criado!"+"\n");
            //txtLog.append ("ERRO : Arquivo BNF não pode ser criado!"+"\n");
            // Cancela toda operacao e emite mensagem de erro caso algum erro ocorra
        }
    }

    /** Retorna a quantidade de pontos do rotulo */
    public int Retorna_desvio(String Str_Rotulo){

        int Int_pontos  =  0;
        int Int_pos     =  0;

        // Loop que varre rotulo a procura de pontos e incrementando contador
        while (Str_Rotulo.indexOf('.',Int_pos) != -1) {
            Int_pos  =  Str_Rotulo.indexOf('.', Int_pos);
            Int_pos++;
            Int_pontos++;
        }
        return(Int_pontos);
    }

    /** Formata o rotulo com notacao BNF */
    public String Formata_rotulo (String rotulo , int Flag) {
        String auxiliar  = new String ("");      // Variavel que armazena o novo rotulo formatado
        int pos   =  0;                          // Variavel para manipulacao de substring
        int pos2  =  0;                          // Variavel para manipulacao de substring

        // Caso rotulo sem pontos, apenas acrescente colchetes
        if (Retorna_desvio(rotulo)  ==  0) {
            return("<" + rotulo + ">");
        }

        // Varre rotulo a procura dos indices
        while (rotulo.indexOf('.',pos+1) != -1)
        {
          if (pos!=0)
              pos2 = pos+1;
          pos = rotulo.indexOf('.',pos+1);

          // Caso flag=0, rotulo de numeracao, caso contrario rotulo de desvio
          if (Flag == 0)
              auxiliar  =  new String (auxiliar + '<' + rotulo.substring (pos2, pos) + '>' + PONTO);
          else
              auxiliar  =  new String (auxiliar + '<' + rotulo.substring (pos2, pos) + '>' + '.');
        }

        //Atualiza a string com o nova formatacao
        auxiliar  =  new String (auxiliar + '<' + rotulo.substring(pos+1));

        // Retorna do metodo
        return ("" + auxiliar +">");
    }

    /** funcao que imprime o arquivo BNF, aqui acrescenta-se a maior parte da notacao BNF  */
    public void Imprime_Evento (Evento Evt_atual,TextArea txtLog) {

        String Str_evento;        // Variavel utilizada para formatar a linha do evento

        // Caso condicional
        if (Evt_atual.Chr_tipo == '?')
            Str_evento = new String ( Formata_rotulo(Evt_atual.Str_rotulo , 0) + " [ ( " +'"' +
Evt_atual.Chr_tipo + '"' + " <" + Evt_atual.Str_condicao + "> " + TRACO + " <" + Evt_atual.Str_titulo + "> " +
TRACO + " <Adicione a descricao do evento> " + PVIRGULA);
        // Caso alternativo
        else if (Evt_atual.Chr_tipo == 'a')
            Str_evento = new String ( Formata_rotulo(Evt_atual.Str_rotulo , 0) + " [ ( " + '"' +
Evt_atual.Chr_tipo + '"' + " <" + Evt_atual.Str_titulo + "> " + TRACO + " <Adicione a descricao do evento> " +
DESVIO + " [" + Formata_rotulo(Evt_atual.Str_desvio , 1) + "]" );
        // Caso inicio
        else if (Evt_atual.Chr_tipo == 'i')
            Str_evento = new String ( Formata_rotulo(Evt_atual.Str_rotulo , 0) +  '"' + Evt_atual.Chr_tipo + '"'
+ "  <" + Evt_atual.Str_titulo + "> " + TRACO + " <Adicione a descricao do evento> " + PVIRGULA + " " + '"' + "("
+ '"');
        // Caso fim
        else if (Evt_atual.Chr_tipo == 'f')
            Str_evento = new String ('"' + ")" + '"' + " " + Formata_rotulo(Evt_atual.Str_rotulo , 0) + " [ ( " +
'"' + Evt_atual.Chr_tipo + '"' + " <" + Evt_atual.Str_titulo + "> " + TRACO + " <Adicione a descricao do evento>
" + PVIRGULA);
        // Caso principal
        else
            Str_evento = new String ( Formata_rotulo(Evt_atual.Str_rotulo , 0) + " [ ( " + '"' +
Evt_atual.Chr_tipo + '"' + " <" + Evt_atual.Str_titulo + "> " + TRACO + " <Adicione a descricao do evento> " +
PVIRGULA);

        try {
            // Imprime a string formatada
            BWriter_Arquivo_BNF.write(Str_evento);
            BWriter_Arquivo_BNF.newLine();

            // Caso final imprime mais uma linha
            if (Evt_atual.Chr_tipo == 'f') {
                BWriter_Arquivo_BNF.newLine();
//                BWriter_Arquivo_BNF.newLine();
            }
```

```
        }
        catch (IOException e)
        {
            txtLog.append ("ERRO : Ocorreu erro ao tentar gravar em arquivo");
        }
    }
}
```

## Classe Elemento.java

```
package tf;

import java.lang.*;
import java.io.*;

public class Elemento {

    protected Arco      Arc_atual;      // Variavel que armazena o arco ligado ao elemento.
    protected int       Int_tipo;       // Variavel que armazena o tipo do elemento.
    protected int       Int_indice;     // Variavel que armazena o indice do elemento.
    protected String    Str_direcao;    // Variavel que indica a direcao do arco com relacao ao elemento.

    /** Creates a new instance of Elemento */
    public Elemento(Arco auxiliar , int tipo , int indice , String direcao) {
        Arc_atual    =  auxiliar;
        Int_tipo     =  tipo;
        Int_indice   =  indice;
        Str_direcao  =  direcao;
    }

}
```

## Classe Entrada_BNF.java

```
package tf;

import java.lang.*;
import java.io.*;

public class Entrada_BNF {

    private Trata_Elemento  TrEl_RPAnalisada;           // Variavel que armazena as listas ligadas criadas
anteriormente
    private String          Str_Rotulo_atual;           // Variavel que armazena o rotulo atual a ser
procurado
    public  Evento          Evt_Inicial;                // Variavel que armazena o evento inicial;
    private Evento          Evt_Atual;                  // Variavel que armazena o ultimo evento da lista;

    /** Creates a new instance of Entrada_BNF */
    public Entrada_BNF(Trata_Elemento TrEl_auxiliar) {
        TrEl_RPAnalisada  =  TrEl_auxiliar;
        Evt_Inicial       =  new Evento ("");
        Evt_Atual         =  new Evento ("");
        Str_Rotulo_atual  =  new String ("1");
    }

    /** Funcao que retorna o lugar correspondente ao rotulo fornecido */
    public Lugar Retorna_Lugar (String Str_rotulo) {
        Lugar Lgr_auxiliar  =  TrEl_RPAnalisada.Lgr_Inicial;

        try {
            // Varre lista ligada a procura do rotulo
            while (!Lgr_auxiliar.Str_Rotulo.equals (Str_rotulo))
            {
                Lgr_auxiliar  =  Lgr_auxiliar.Lgr_proximo;
            }
        }

        catch (NullPointerException e) {
            return (null);
        }

        return (Lgr_auxiliar);
    }

    /** Funcao que retorna a transicao correspondente ao rotulo fornecido */
    public Transicao Retorna_Transicao (String Str_rotulo) {
        Transicao Trs_auxiliar  =  TrEl_RPAnalisada.Trs_Inicial;

        try {
            // Varre a lista ligada a procura do rotulo
            while (!Trs_auxiliar.Str_Rotulo.equals(Str_rotulo))
            {
                Trs_auxiliar  =  Trs_auxiliar.Trs_proximo;
            }
        }

        catch (NullPointerException e) {
            return (null);
        }

        return (Trs_auxiliar);
    }

    /** Funcao que retorna o pseudolugar correspondente ao rotulo fornecido */
    public PLugar Retorna_Pseudolugar (String Str_rotulo) {
        PLugar PLgr_auxiliar  =  TrEl_RPAnalisada.PLgr_Inicial;

        try {
            // Varre a lista ligada a procura do rotulo
```

```
            while (!PLgr_auxiliar.Str_Rotulo.equals(Str_rotulo))
            {
                PLgr_auxiliar  =  PLgr_auxiliar.PLgr_proximo;
            }
        }

        catch (NullPointerException e) {
            return (null);
        }

        return (PLgr_auxiliar);
    }

    /** Funcao que adiciona mais um indice ao rotulo atual */
    public String Adiciona_Indice (String rotulo) {
      String Str_auxiliar  =  new String (rotulo + ".1");
      return (Str_auxiliar);
    }

    /** Funcao que incrementa o ultimo indice do rotulo atual */
    public String Incrementa_Indice (String rotulo) {
      String   Str_auxiliar;       // Variavel utilizada para manipular o rotulo
      Integer  INT_auxiliar;       // Variavel utilizada para converter string para inteiro
      int      pos;                // Variavel utilizada no posicao das substrings
      int      auxiliar;           // Variavel auxiliar na atualizacao do rotulo

      // Verifica caso em que nao existem pontos
      if (Retorna_desvio (rotulo) == 0) {
          INT_auxiliar  =  new Integer (rotulo);
          auxiliar      =  INT_auxiliar.intValue();
          auxiliar++;
          return ( new String ("" + auxiliar + "") );
      }

      // Retira o ultimo algarismo (indice)
      pos  =  rotulo.lastIndexOf('.');
      INT_auxiliar  =  new Integer ( rotulo.substring(pos + 1) );
      auxiliar = INT_auxiliar.intValue();
      auxiliar++;

      // Reinicializa string com o restante do rotulo mais o incremento do ultimo indice
      Str_auxiliar  =  new String (rotulo.substring(0,pos+1) + auxiliar);

      // Verifica se rotulo existe, caso nao exista incrementa o penultimo algarismo
      if ( Retorna_Transicao (Str_auxiliar) == null)
          return (Incrementa_Indice (rotulo.substring (0,pos)));
      else
          return (Str_auxiliar);
    }

    /** Funcao que imprime a lista ligada de eventos (SOMENTE PARA TESTE) */
    public void Imprime_Lista_Eventos () {
        Evento auxiliar = Evt_Inicial;

        // Varre a lista ligada de eventos
        while (auxiliar.Evt_proximo != null) {
            System.out.println (auxiliar.Str_rotulo);
            System.out.println (auxiliar.Chr_tipo);
            System.out.println (auxiliar.Str_titulo);
            System.out.println (auxiliar.Str_condicao);
            System.out.println ("----------------------");
            auxiliar  =  auxiliar.Evt_proximo;
        }
        System.out.println (auxiliar.Chr_tipo);
        System.out.println (auxiliar.Str_titulo);
        System.out.println (auxiliar.Str_condicao);
    }

    /** Retorna a quantidade de pontos (desvios) de determinado rotulo) */
    public int Retorna_desvio(String Str_Rotulo){

        int Int_pontos  =  0;
        int Int_pos     =  0;

        // Procura por pontos, incrementando contadores conforme os acha
        while (Str_Rotulo.indexOf('.',Int_pos) != -1) {
           Int_pos  =  Str_Rotulo.indexOf('.', Int_pos);
           Int_pos++;
           Int_pontos++;
        }
        return(Int_pontos);
    }

    /** Retorna o rotulo de desvio do fluxo alternativo */
    public String Retorna_desvio_alternativo (int numero)
    {
        Arco Arc_auxiliar  =  TrEl_RPAnalisada.Arc_Inicial;       // Variavel que compara os arcos da lista
ligada
        int  numero2       =  0;                                  // Variavel que auxilia na manipulacao das
strings

        // Inicieliza com o primeiro arco e varre a lista
        while (Arc_auxiliar.Arc_proximo != null)
        {
            // Se o referencial anterior do arco for igual ao indice e o tipo for normal
            if (Arc_auxiliar.Int_Ref_Anterior==numero  &&  Arc_auxiliar.Int_Tipo_Anterior==1)
                break;
            Arc_auxiliar  =  Arc_auxiliar.Arc_proximo;
        }

        // Armazena referencial posterior do arco (lugar)
        numero2  =  Arc_auxiliar.Int_Ref_Posterior;
```

```
        // Caso particular para ultimo lugar (nao existe arco posterior)
        Arc_auxiliar  =  TrEl_RPAnalisada.Arc_Inicial;
        while (Arc_auxiliar.Arc_proximo != null)
            Arc_auxiliar=Arc_auxiliar.Arc_proximo;
        numero  =  Arc_auxiliar.Int_Ref_Anterior;

        // Procura pela transicao ligada ao lugar de desvio
        Arc_auxiliar  =  TrEl_RPAnalisada.Arc_Inicial;
        while (Arc_auxiliar.Arc_proximo != null)
        {
            /* Caso encontre arco cuja ref_anterior seja igual ao numero2, verifique se a transicao e
             * do fluxo alternativo, caso seja descarte e continue procurando */
            if (Arc_auxiliar.Int_Ref_Anterior==numero2  &&  Arc_auxiliar.Int_Tipo_Anterior==0)
            {
                boolean flag   =  true;
                Arco auxiliar  =  TrEl_RPAnalisada.Arc_Inicial;

                // Varre novamente a lista a procura de arcos tipo inibidor e ref_posterior=indice encontrado
                while (auxiliar.Arc_proximo != null)
                {
                    // Se achou condicao indesejavel
                    if (auxiliar.Int_Ref_Posterior==Arc_auxiliar.Int_Ref_Posterior &&  auxiliar.Int_Tipo_Arco==2)
                    {
                      flag  =  false;
                      break;
                    }
                    // Caso contrario
                    else
                      auxiliar  =  auxiliar.Arc_proximo;
                }

                // Caso nao ache condicao indesejavel, a transicao encontrada e valida
                if (flag)
                {
                    numero  =  Arc_auxiliar.Int_Ref_Posterior;
                    break;
                }
            }

            // Varre lista ligada no while maior
            Arc_auxiliar  =  Arc_auxiliar.Arc_proximo;
        }

        // Varre lista ligada de transicao
        Transicao Trs_auxiliar  =  TrEl_RPAnalisada.Trs_Inicial;

        while (Trs_auxiliar.Trs_proximo != null)
        {
            // Se transicao tem o mesmo indice de numero
            if (Trs_auxiliar.Int_Pos_Seq==numero)
                break;
            Trs_auxiliar  =  Trs_auxiliar.Trs_proximo;
        }

        // Retorna o rotulo da transicao
        return (Trs_auxiliar.Str_Rotulo);
    }

    /** Funcao que cria a lista ligada de eventos */
    public void Cria_Lista_Eventos () {

      Evento    Evt_auxiliar  =  new Evento ("");       // Variavel auxiliar para manipulacao de eventos
      Lugar     Lgr_auxiliar  =  new Lugar ();          // Variavel auxiliar para manipulacao de lugares
      PLugar    PLgr_auxiliar =  new PLugar ();         // Variavel auxiliar para manipulacao de pseudolugares
      Transicao Trs_auxiliar  =  new Transicao ();      // Variavel auxiliar para manipulacao de transicoes

      // Loop que verifica o tipo de evento, enquanto nao chegar no fim da rede
      while (true)
      {
          Evt_auxiliar  =  new Evento ("");

      /****  PRIMEIRO PASSO - VERIFICA TIPO DE EVENTO  ****/

          // Caso seja primeiro elemento (evento inicial)
          if (Str_Rotulo_atual.equals ("1") ){
              Trs_auxiliar  =  Retorna_Transicao (Str_Rotulo_atual);
              Evt_auxiliar.Str_titulo  =  Trs_auxiliar.Str_Caption;
              Evt_auxiliar.Str_rotulo  =  Trs_auxiliar.Str_Rotulo;
              Evt_auxiliar.Chr_tipo    =  'i';
          }

          // Caso seja ultimo elemento (evento final)
          else if (Str_Rotulo_atual.equals ("2") ){
              Trs_auxiliar  =  Retorna_Transicao (Str_Rotulo_atual);
              Evt_auxiliar.Str_titulo  =  Trs_auxiliar.Str_Caption;
              Evt_auxiliar.Str_rotulo  =  Trs_auxiliar.Str_Rotulo;
              Evt_auxiliar.Chr_tipo    =  'f';
          }

          else {
              // Caso seja condicional, verifica se existe pseudolugar com rotulo
              if ( Retorna_Pseudolugar (Str_Rotulo_atual) != null) {
                  PLgr_auxiliar  =  Retorna_Pseudolugar (Str_Rotulo_atual);
                  Trs_auxiliar   =  Retorna_Transicao (Str_Rotulo_atual);
                  Evt_auxiliar.Str_titulo  =  Trs_auxiliar.Str_Caption;
                  Evt_auxiliar.Str_condicao=  PLgr_auxiliar.Str_Caption;
                  Evt_auxiliar.Str_rotulo  =  Trs_auxiliar.Str_Rotulo;
                  Evt_auxiliar.Chr_tipo    =  '?';
              }

              // Caso negativo, verifica se desvios = 1, evento principal
```

```
            else if ( Retorna_desvio (Str_Rotulo_atual)  ==  1) {
                Trs_auxiliar  =  Retorna_Transicao (Str_Rotulo_atual);
                Evt_auxiliar.Str_titulo  =  Trs_auxiliar.Str_Caption;
                Evt_auxiliar.Str_rotulo  =  Trs_auxiliar.Str_Rotulo;
                Evt_auxiliar.Chr_tipo    =  'p';
            }

            // Caso negativo, evento alternativo
            else {
                Trs_auxiliar  =  Retorna_Transicao (Str_Rotulo_atual);
                Evt_auxiliar.Str_titulo  =  Trs_auxiliar.Str_Caption;
                Evt_auxiliar.Str_rotulo  =  Trs_auxiliar.Str_Rotulo;
                Evt_auxiliar.Chr_tipo    =  'a';
                Evt_auxiliar.Str_desvio  =  Retorna_desvio_alternativo(Trs_auxiliar.Int_Pos_Seq);
            }
        }

    /****  SEGUNDO PASSO - ATUALIZA LISTA LIGADA DE EVENTOS  ****/

        // Verifica se e evento inicial
        if ( Evt_Inicial.Str_rotulo.equals ("") )
        {
            Evt_Inicial  =  Evt_auxiliar;
        }

        // Verifica se e segundo evento
        else if ( Evt_Inicial.Evt_proximo == null )
        {
            Evt_Atual  =  Evt_auxiliar;
            Evt_Inicial.Evt_proximo  =  Evt_Atual;
        }

        // Verifica caso geral
        else {
            Evt_Atual.Evt_proximo  =  Evt_auxiliar;
            Evt_Atual  =  Evt_auxiliar;

            // Se ultimo sai do while
            if (Str_Rotulo_atual.equals("2"))
                break;
        }

    /****  TERCEIRO PASSO - ATUALIZA INDICE A PROCURAR  ****/

        // Verifica se existe proximo indice adicionado
        if (Retorna_Transicao (Adiciona_Indice (Str_Rotulo_atual)) == null)
            // Nao, entao incremente o ultimo indice
            Str_Rotulo_atual  =  Incrementa_Indice (Str_Rotulo_atual);
        else
            // Sim, entao procure por este rotulo
            Str_Rotulo_atual  =  Adiciona_Indice (Str_Rotulo_atual);
    }
  }
}
```

## Classe Entrada_Ghenesys.java

```
package tf;

import java.lang.*;
import java.io.*;
import java.awt.*;

public class Entrada_Ghenesys {

    // definicoes de caracteres :
    static final char INICIO_PROCESSO       = 'i';
    static final char FIM_PROCESSO          = 'f';
    static final char EVENTO_FP             = 'p';
    static final char EVENTO_FA             = 'a';
    static final char EVENTO_CONDICIONAL    = '?';
    static final char SIMBOLO_DESVIO        = '~';
    static final char EVENTO_CONCORRENTE    = '|';
    static final int  DELTA_TOP             =  60;
    static final int  DELTA_LEFT            =  240;
    static final int  DELTA_HALF_LEFT       =  120;
    static final int  TOP_INICIAL           =  50;
    static final int  LEFT_INICIAL          =  30;

    // variaveis utilizadas pela classe :
    private File                File_Arquivo_RP;                    // Variável que representa o arquivo de saida
Redes de Petri.
    private FileWriter          FWriter_Arquivo_RP;                 // Variável que representa a Stream de saida
de dados do arquivo.
    private BufferedWriter      BWriter_Arquivo_RP;
    private int                 Int_Top     =       50;             // Variavel global de posicao Top
    private int                 Int_Top_aux =       0;              // Variavel auxiliar de posicao Top
    private int                 Int_Left    =       30;             // Variavel global da posicao Left
    private int                 Int_Lgr_Seq =       0;              // Numero sequencial dos lugares
    private int                 Int_Trs_Seq =       0;              // Numero sequencial das transicoes
    private int                 Int_PLgr_Seq=       0;              // Numero sequencial dos pseudolugares
    private Lugar               Lgr_Inicial;                        // Guarda lugar inicial
    private Transicao           Trs_Inicial;                        // Guarda transicao inicial
    private PLugar              PLgr_Inicial;                       // Guarda pseudolugar inicial
    private Arco                Arc_Inicial;                        // Guarda arco inicial
    private Lugar               Lgr_Atual;                          // Variavel auxiliar para manipulacao dos
lugares
    private Transicao           Trs_Atual;                          // Variavel auxiliar para manipulacao das
transicoes
    private PLugar              PLgr_Atual;                         // Variavel auxiliar para manipulacao dos
pseudolugares
```

```
    private Arco                Arc_Atual;                          // Variavel auxiliar para manipulacao dos
arcos


    /** Esta funcao chama todas as outras funcoes para criacao do arquivo de entrada do ghenesys,
     *  funciona como funcao main da classe*/
    public Entrada_Ghenesys(Evento Evt_Inicial, String Str_diretorio,TextArea txtLog) {

        File_Arquivo_RP    =  new File (Str_diretorio);

        try {
            FWriter_Arquivo_RP   =  new FileWriter(File_Arquivo_RP);          // Cria buffer para arquivo de
saida
            BWriter_Arquivo_RP   =  new BufferedWriter(FWriter_Arquivo_RP);   // Cria buffer de escrita para
arquivo de saida
            Grava_Cabecalho(txtLog);                                          // Grava o cabecalho no
arquivo de saida
            Criar_RP(Evt_Inicial,txtLog);                                      // Analisa eventos e cria
lista ligada da rede de petri
            Imprime_Lugares(txtLog);                                          // Imprime lista ligada de
lugares no arquivo de saida
            Imprime_Transicoes(txtLog);                                       // Imprime lista ligada de
transicoes no arquivo de saida
            Imprime_Pseudo_Lugares(txtLog);                                   // Imprime lista ligada de
pseudolugares no arquivo de saida
            Imprime_Arcos(Evt_Inicial,txtLog);                                 // Imprime lista ligada de
arcos no arquivo de saida
            BWriter_Arquivo_RP.close();                                       // Fecha arquivo
        }

        catch (IOException e) {
            txtLog.append ("ERRO : Arquivo não pode ser criado!"+"\n");
            // Cancela toda operacao e emite mensagem de erro caso algum erro ocorra
        }
    }

    /** Esta funcao grava o cabecalho do arquivo do ghenesys, apenas com o formato
     *  da pagina (Tamanho A4/A3/A2/A1)&(Orientacao Landscape/Portrait) e os dados
     *  iniciais necessarios para o programa */
    private void Grava_Cabecalho (TextArea txtLog) {

        try {
            BWriter_Arquivo_RP.write("[GPNF]");
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.write("PaperType= A2");
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.write("Orientation= poPortrait");
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.newLine();
        }

        catch (IOException e) {
            txtLog.append ("ERRO : Nao foi possivel gravar o cabecalho!"+"\n");
        }
    }

    /** Esta funcao e a responsavel por analisar a lista ligada de eventos e criar
     *  todas as outras listas ligadas */
    private void Criar_RP (Evento Evt_Atual,TextArea txtLog) {

        /* Variaveis auxiliares para manipulacao dos dados */
        Lugar     Lgr_Auxiliar          =  new Lugar();
        PLugar    PLgr_Auxiliar         =  new PLugar();
        Transicao Trs_Auxiliar          =  new Transicao();
        Arco      Arc_Auxiliar          =  new Arco();

        /* Variavel que retorna o numero de pontos no label, usado para manipular a posicao
         * dos elementos na rede*/
        int       Int_multiplicador     =       1;

        Int_multiplicador  =  Retorna_desvio (Evt_Atual.Str_rotulo);

        switch (Evt_Atual.Chr_tipo) {

            case  INICIO_PROCESSO :

               // Cria lugar
               Lgr_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - Inicio";
               //Lgr_Auxiliar.Str_Caption   =  "Inicio";
               Lgr_Auxiliar.Int_Pos_Top   =  TOP_INICIAL;
               Lgr_Auxiliar.Int_Pos_Left  =  LEFT_INICIAL;
               Lgr_Auxiliar.Int_Pos_Seq   =  0;
               Lgr_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;
               Lgr_Inicial       =  Lgr_Auxiliar;

               // Cria transicao
               Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - " + Evt_Atual.Str_titulo;
               //Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_titulo;
               Trs_Auxiliar.Int_Pos_Left  =  LEFT_INICIAL;
               Trs_Auxiliar.Int_Pos_Top   =  TOP_INICIAL + DELTA_TOP;
               Int_Top+=DELTA_TOP;
               Trs_Auxiliar.Int_Pos_Seq   =  0;
                 Trs_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;
               Trs_Inicial  =  Trs_Auxiliar;

               // Cria arco lugar - transicao
               Arc_Auxiliar.Int_Tipo_Arco      =  0;
               Arc_Auxiliar.Int_Ref_Anterior   =  0;
               Arc_Auxiliar.Int_Ref_Posterior  =  0;
               Arc_Auxiliar.Int_Tipo_Anterior  =  0;
               Arc_Auxiliar.Int_Tipo_Posterior =  1;
```

```
        Arc_Inicial  =  Arc_Auxiliar;

        // Cria arco transicao - lugar (proximo)
        Arc_Auxiliar  =  new Arco ();
        Arc_Auxiliar.Int_Tipo_Arco       =  0;
        Arc_Auxiliar.Int_Ref_Anterior    =  0;
        Arc_Auxiliar.Int_Ref_Posterior   =  1;
        Arc_Auxiliar.Int_Tipo_Anterior   =  1;
        Arc_Auxiliar.Int_Tipo_Posterior  =  0;
        Arc_Inicial.Arc_proximo  =  Arc_Auxiliar;

        // Atualiza lista ligada
        Arc_Atual    =  Arc_Auxiliar;
        Trs_Atual    =  Trs_Inicial;
        Lgr_Atual    =  Lgr_Inicial;
    break;

    case FIM_PROCESSO :

        // Cria lugar
        Int_Top+=DELTA_TOP;
        Lgr_Auxiliar.Int_Pos_Top   =  Int_Top;
        Lgr_Auxiliar.Int_Pos_Left  =  Int_Left;
        Lgr_Auxiliar.Int_Pos_Seq   =  ++Int_Lgr_Seq;
        Lgr_Auxiliar.Str_Rotulo    =  "";
        Lgr_Atual.Lgr_proximo      =  Lgr_Auxiliar;
        Lgr_Atual                  =  Lgr_Auxiliar;

        // Cria transicao
        Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - " + Evt_Atual.Str_titulo;
        //Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_titulo;
        Int_Top+=DELTA_TOP;
        Trs_Auxiliar.Int_Pos_Top   =  Int_Top;
        Trs_Auxiliar.Int_Pos_Left  =  Int_Left;
        Trs_Auxiliar.Int_Pos_Seq   =  ++Int_Trs_Seq;
          Trs_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;
        Trs_Atual.Trs_proximo      =  Trs_Auxiliar;

        // Cria arco lugar - transicao
        Arc_Auxiliar.Int_Tipo_Arco       =  0;
        Arc_Auxiliar.Int_Ref_Anterior    =  Int_Lgr_Seq;
        Arc_Auxiliar.Int_Ref_Posterior   =  Int_Trs_Seq;
        Arc_Auxiliar.Int_Tipo_Anterior   =  0;
        Arc_Auxiliar.Int_Tipo_Posterior  =  1;
        Arc_Atual.Arc_proximo            =  Arc_Auxiliar;
        Arc_Atual                        =  Arc_Auxiliar;

        // Cria lugar
        Lgr_Auxiliar               =  new Lugar();
        Lgr_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - " + "Fim";
        //Lgr_Auxiliar.Str_Caption   =  "Fim";
        Int_Top+=DELTA_TOP;
        Lgr_Auxiliar.Int_Pos_Top   =  Int_Top;
        Lgr_Auxiliar.Int_Pos_Left  =  Int_Left;
        Lgr_Auxiliar.Int_Pos_Seq   =  ++Int_Lgr_Seq;
        Lgr_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;
        Lgr_Atual.Lgr_proximo      =  Lgr_Auxiliar;

        // Cria arco transicao - lugar (segundo)
        Arc_Auxiliar                     =  new Arco();
        Arc_Auxiliar.Int_Tipo_Arco       =  0;
        Arc_Auxiliar.Int_Ref_Anterior    =  Int_Trs_Seq;
        Arc_Auxiliar.Int_Ref_Posterior   =  Int_Lgr_Seq;
        Arc_Auxiliar.Int_Tipo_Anterior   =  1;
        Arc_Auxiliar.Int_Tipo_Posterior  =  0;
        Arc_Atual.Arc_proximo  =  Arc_Auxiliar;

        // Atualiza lista ligada
        Arc_Atual  =  Arc_Auxiliar;
        Trs_Atual  =  Trs_Auxiliar;
        Lgr_Atual  =  Lgr_Auxiliar;
break;

case EVENTO_FP :

    // Verifica se posicao atual e zero
    if (Int_Top_aux != 0)
        Int_Top  =  Int_Top_aux;
    else
        Int_Top+=DELTA_TOP;
    Int_Top_aux  =  0;

    // Cria lugar
    Lgr_Auxiliar.Int_Pos_Top   =  Int_Top;
    Lgr_Auxiliar.Int_Pos_Left  =  Int_Left;
    Lgr_Auxiliar.Int_Pos_Seq   =  ++Int_Lgr_Seq;
    Lgr_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;
    Lgr_Atual.Lgr_proximo      =  Lgr_Auxiliar;

    // Cria transicao
    Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - " + Evt_Atual.Str_titulo;
    //Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_titulo;
    Trs_Auxiliar.Int_Pos_Left  =  Int_Left;
    Int_Top+=DELTA_TOP;
    Trs_Auxiliar.Int_Pos_Top   =  Int_Top;
    Trs_Auxiliar.Int_Pos_Seq   =  ++Int_Trs_Seq;
      Trs_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;
    Trs_Atual.Trs_proximo      =  Trs_Auxiliar;

    // Cria arco lugar - transicao
    Arc_Auxiliar.Int_Tipo_Arco       =  0;
    Arc_Auxiliar.Int_Ref_Anterior    =  Int_Lgr_Seq;
```

```
                Arc_Auxiliar.Int_Ref_Posterior  =  Int_Trs_Seq;
                Arc_Auxiliar.Int_Tipo_Anterior  =  0;
                Arc_Auxiliar.Int_Tipo_Posterior =  1;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                Arc_Atual                       =  Arc_Auxiliar;

                // Cria arco transicao - lugar (proximo)
                Arc_Auxiliar  =  new Arco ();
                Arc_Auxiliar.Int_Tipo_Arco      =  0;
                Arc_Auxiliar.Int_Ref_Anterior   =  Int_Trs_Seq;
                Arc_Auxiliar.Int_Ref_Posterior  =  Int_Lgr_Seq+1;
                Arc_Auxiliar.Int_Tipo_Anterior  =  1;
                Arc_Auxiliar.Int_Tipo_Posterior =  0;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;

                // Atualiza lista ligada
                Arc_Atual  =  Arc_Auxiliar;
                Trs_Atual  =  Trs_Auxiliar;
                Lgr_Atual  =  Lgr_Auxiliar;
            break;

            case EVENTO_FA :

                // Cria lugar
                Lgr_Auxiliar.Int_Pos_Top      =  Int_Top  + DELTA_TOP*((2*Int_multiplicador - 3) +
(Retorna_Sequencia(Evt_Atual.Str_rotulo)-1)*2) ;
                Lgr_Auxiliar.Int_Pos_Left     =  Int_Left + DELTA_LEFT*(Int_multiplicador-1);
                Lgr_Auxiliar.Str_Rotulo       =  Evt_Atual.Str_rotulo;

                /* Verifica se ja existe elemento na posicao, isso implica no caso especial
                 * de ser um pseudolugar, se estiver desocupado, coloque na lista o novo
                 * lugar */
                if (!Pixel_Ocupado (Lgr_Auxiliar.Int_Pos_Top,Lgr_Auxiliar.Int_Pos_Left)) {
                    Lgr_Auxiliar.Int_Pos_Seq      =  ++Int_Lgr_Seq;
                    Lgr_Atual.Lgr_proximo         =  Lgr_Auxiliar;
                    Lgr_Atual                     =  Lgr_Auxiliar;
                }
                else {}

                // Cria transicao
                Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - " + Evt_Atual.Str_titulo;
                //Trs_Auxiliar.Str_Caption   =  Evt_Atual.Str_titulo;
                Trs_Auxiliar.Int_Pos_Left  =  Int_Left + DELTA_LEFT*(Int_multiplicador-1);
                  Trs_Auxiliar.Str_Rotulo    =  Evt_Atual.Str_rotulo;

                // Posiciona transicao
                if (Int_multiplicador == 1)
                    Trs_Auxiliar.Int_Pos_Top   =  Int_Top + DELTA_TOP*(2*Int_multiplicador-2);
                else if (Int_multiplicador == 3 && Retorna_Sequencia(Evt_Atual.Str_rotulo)!=1)
                    Trs_Auxiliar.Int_Pos_Top   =  Int_Top + DELTA_TOP*((2*Int_multiplicador-2) +
(Retorna_Sequencia_Alternativa(Evt_Atual.Str_rotulo)-1)*2 + Retorna_Sequencia(Evt_Atual.Str_rotulo));
                else
                    Trs_Auxiliar.Int_Pos_Top   =  Int_Top + DELTA_TOP*((2*Int_multiplicador-2) +
(Retorna_Sequencia_Alternativa(Evt_Atual.Str_rotulo)-1)*2);

                // Acrescenta na lista ligada
                Trs_Auxiliar.Int_Pos_Seq =  ++Int_Trs_Seq;
                Trs_Atual.Trs_proximo    =  Trs_Auxiliar;
                Int_Top_aux              =  Trs_Auxiliar.Int_Pos_Top;

                // Cria arco transicao - lugar
                Arc_Auxiliar.Int_Tipo_Arco      =  0;
                Arc_Auxiliar.Int_Ref_Anterior   =  Int_Trs_Seq;
                Arc_Auxiliar.Int_Tipo_Anterior  =  1;
                Arc_Auxiliar.Int_Tipo_Posterior =  0;
                Arc_Auxiliar.Str_Desvio         =  Evt_Atual.Str_desvio;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;

                Arc_Atual  =  Arc_Auxiliar;
                Trs_Atual  =  Trs_Auxiliar;

                /* Verifica se o proximo evento e do fluxo alternativo tambem, se positivo deve-se
                 * acrescentar um arco entre o lugar e a transicao */
                if (Evt_Atual.Evt_proximo.Chr_tipo  ==  EVENTO_FA) {
                        Arc_Auxiliar  =  new Arco();
                        Arc_Auxiliar.Int_Tipo_Arco      =  0;
                        Arc_Auxiliar.Int_Ref_Anterior   =  Int_Lgr_Seq + 1;
                        Arc_Auxiliar.Int_Ref_Posterior  =  Int_Trs_Seq + 1;
                        Arc_Auxiliar.Int_Tipo_Anterior  =  0;
                        Arc_Auxiliar.Int_Tipo_Posterior =  1;
                        Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                        Arc_Atual                       =  Arc_Auxiliar;
                }
            break;

            case EVENTO_CONDICIONAL :

                // Verifica a posicao a ser colocada
                if (Int_multiplicador  ==  1  &&  (Int_Top_aux  !=  0)) {
                    Int_Top  =  Int_Top_aux;
                    Int_Top_aux  =  0;
                }

                if(Int_multiplicador  == 1)
                     PLgr_Auxiliar.Int_Pos_Top   =  Int_Top  + (DELTA_TOP)*3;
                else if (Int_multiplicador  ==  2)
                     PLgr_Auxiliar.Int_Pos_Top   =  Int_Top  + (DELTA_TOP)*(3 +
(Retorna_Sequencia(Evt_Atual.Str_rotulo)-1)*2);
                else
                     PLgr_Auxiliar.Int_Pos_Top  =  Int_Top  +  DELTA_TOP*(Int_multiplicador+2);
                     //PLgr_Auxiliar.Int_Pos_Top   =  Int_Top  + (DELTA_TOP*((2*Int_multiplicador-3)) +
(Retorna_Sequencia (Evt_Atual.Str_rotulo)-1)*2);
```

```
                // Atualiza o pseudo lugar
                PLgr_Auxiliar.Str_Caption   =  Evt_Atual.Str_rotulo + " - " + Evt_Atual.Str_condicao;
                //PLgr_Auxiliar.Str_Caption   =  Evt_Atual.Str_condicao;
                PLgr_Auxiliar.Int_Pos_Left  =  Int_Left + DELTA_LEFT*(Int_multiplicador);
                PLgr_Auxiliar.Int_Pos_Seq   = Int_PLgr_Seq++;

                // verifica se tem algo no lugar ja, caso exista ele desloca o objeto para esquerda
                if (Pixel_Ocupado (PLgr_Auxiliar.Int_Pos_Top,PLgr_Auxiliar.Int_Pos_Left))
                     PLgr_Auxiliar.Int_Pos_Left  =  PLgr_Auxiliar.Int_Pos_Left  -  DELTA_HALF_LEFT;
                else{}

                /* verifica se lista ligada de pseudolugares e nula, caso positivo inicializa o pseudo
                 * lugar inicial */
                if (PLgr_Inicial == null) {
                    PLgr_Inicial  =  PLgr_Auxiliar;
                    PLgr_Atual    =  PLgr_Inicial;
                }
                else {
                    PLgr_Atual.PLgr_proximo  =  PLgr_Auxiliar;
                    PLgr_Atual               =  PLgr_Auxiliar;
                }

                // Cria arco inibidor pseudolugar - transicao
                Arc_Auxiliar.Int_Tipo_Arco      =  2;
                Arc_Auxiliar.Int_Ref_Anterior   =  Int_PLgr_Seq - 1;
                Arc_Auxiliar.Int_Ref_Posterior  =  Int_Trs_Seq + 2;
                Arc_Auxiliar.Int_Tipo_Anterior  =  4;
                Arc_Auxiliar.Int_Tipo_Posterior =  1;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                Arc_Atual                       =  Arc_Auxiliar;

                // Cria arco habilitador pseudolugar - transicao
                Arc_Auxiliar  =  new Arco();
                Arc_Auxiliar.Int_Tipo_Arco      =  1;
                  Arc_Auxiliar.Int_Ref_Anterior   =  Int_PLgr_Seq - 1;
                  Arc_Auxiliar.Int_Tipo_Anterior  =  4;
                  Arc_Auxiliar.Int_Tipo_Posterior =  1;
                  Arc_Auxiliar.Str_Rotulo         =  Evt_Atual.Str_rotulo;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                Arc_Atual                       =  Arc_Auxiliar;

                // Reposiciona um lugar
                if (Int_multiplicador  ==  1)
                     Lgr_Auxiliar.Int_Pos_Top      =  Int_Top  + DELTA_TOP*(Int_multiplicador);
                else if (Int_multiplicador  ==  2)
                     Lgr_Auxiliar.Int_Pos_Top      =  Int_Top  + DELTA_TOP*((2*Int_multiplicador - 3) +
(Retorna_Sequencia(Evt_Atual.Str_rotulo)-1)*2) ;
                else
                     Lgr_Auxiliar.Int_Pos_Top      =  Int_Top  + DELTA_TOP*(Int_multiplicador);

                Lgr_Auxiliar.Int_Pos_Left     =  Int_Left + DELTA_LEFT*(Int_multiplicador-1);
                Lgr_Auxiliar.Str_Rotulo       =  Evt_Atual.Str_rotulo;

                /* Caso o espaco ja esteja ocupado, entao o elemento que esta na posicao passa a
                 * ser o lugar atual, caso contrario cria-se novo lugar
                 * Fluxo alternativo condicional */
                if (!Pixel_Ocupado (Lgr_Auxiliar.Int_Pos_Top,Lgr_Auxiliar.Int_Pos_Left)) {
                    //Acrescenta o novo lugar na rede
                    Lgr_Auxiliar.Int_Pos_Seq      =  ++Int_Lgr_Seq;
                    Lgr_Atual.Lgr_proximo         =  Lgr_Auxiliar;
                    Lgr_Atual                     =  Lgr_Auxiliar;

                    // Cria arco lugar - transicao
                    Arc_Auxiliar  =  new Arco();
                    Arc_Auxiliar.Int_Tipo_Arco      =  0;
                    Arc_Auxiliar.Int_Ref_Anterior   =  Int_Lgr_Seq;
                    Arc_Auxiliar.Int_Ref_Posterior  =  Int_Trs_Seq + 1;
                    Arc_Auxiliar.Int_Tipo_Anterior  =  0;
                    Arc_Auxiliar.Int_Tipo_Posterior =  1;
                    Arc_Atual.Arc_proximo    =  Arc_Auxiliar;
                    Arc_Atual  =  Arc_Auxiliar;

                    // Cria arco transicao - lugar (proximo)
                        Arc_Auxiliar  =  new Arco();
                      Arc_Auxiliar.Int_Tipo_Arco      =  0;
                        Arc_Auxiliar.Int_Ref_Anterior   =  Int_Trs_Seq + 1;
                      Arc_Auxiliar.Int_Tipo_Anterior  =  1;
                      Arc_Auxiliar.Int_Tipo_Posterior =  0;
                    Arc_Auxiliar.Str_Rotulo  =  Evt_Atual.Str_rotulo;
                    Arc_Atual.Arc_proximo    =  Arc_Auxiliar;
                    Arc_Atual  =  Arc_Auxiliar;

                    // Cria arco lugar (anterior) - transicao
                    Arc_Auxiliar  =  new Arco();
                    Arc_Auxiliar.Int_Tipo_Arco      =  0;
                    Arc_Auxiliar.Int_Ref_Posterior  =  Int_Trs_Seq + 2;
                    Arc_Auxiliar.Int_Tipo_Anterior  =  0;
                    Arc_Auxiliar.Int_Tipo_Posterior =  1;
                    Arc_Auxiliar.Str_Rotulo         =  Evt_Atual.Str_rotulo;
                    Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                    Arc_Atual                       =  Arc_Auxiliar;
                }

                // Caso o lugar esteja ocupado, apenas cria os arcos
                else {

                    // Cria arco transicao (anterior) - lugar
                        Arc_Auxiliar  =  new Arco();
                      Arc_Auxiliar.Int_Tipo_Arco      =  0;
                    Arc_Auxiliar.Int_Ref_Anterior   =  Int_Trs_Seq + 1;
                    // Verifica caso especifico para tres condicionais seguidos
```

```
                if (Evt_Atual.Evt_proximo.Chr_tipo  == EVENTO_CONDICIONAL)
                    Arc_Auxiliar.Int_Ref_Posterior  =  Int_Lgr_Seq + 2;
                else
                            Arc_Auxiliar.Int_Ref_Posterior  =  Int_Lgr_Seq + 1;
                  Arc_Auxiliar.Int_Tipo_Anterior  =  1;
                  Arc_Auxiliar.Int_Tipo_Posterior =  0;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                Arc_Atual                       =  Arc_Auxiliar;

                // Cria arco lugar - transicao (proximo)
                Arc_Auxiliar  =  new Arco();
                Arc_Auxiliar.Int_Tipo_Arco      =  0;
                //Verifica caso específico para tres condicionais seguidos
                if (Evt_Atual.Evt_proximo.Chr_tipo  == EVENTO_CONDICIONAL)
                     Arc_Auxiliar.Int_Ref_Anterior   =  Int_Lgr_Seq + 2;
                else
                     Arc_Auxiliar.Int_Ref_Anterior   =  Int_Lgr_Seq + 1;
                Arc_Auxiliar.Int_Ref_Posterior  =  Int_Trs_Seq + 2;
                Arc_Auxiliar.Int_Tipo_Anterior  =  0;
                Arc_Auxiliar.Int_Tipo_Posterior =  1;
                Arc_Atual.Arc_proximo           =  Arc_Auxiliar;
                Arc_Atual                       =  Arc_Auxiliar;
            }

            // Posiciona transicao do Fluxo principal
            if (Int_multiplicador  ==  1)
                Trs_Auxiliar.Int_Pos_Top      =  Int_Top  + DELTA_TOP*(Int_multiplicador+1);
            else
                Trs_Auxiliar.Int_Pos_Top      =  Int_Top  + DELTA_TOP*((2*Int_multiplicador-2) +
(Retorna_Sequencia(Evt_Atual.Str_rotulo)-1)*2);

            // Acrescenta transicao a lista ligada
            Trs_Auxiliar.Str_Caption    =  Evt_Atual.Str_rotulo + " - " + Evt_Atual.Str_titulo;
            //Trs_Auxiliar.Str_Caption    =  Evt_Atual.Str_titulo;
            Trs_Auxiliar.Int_Pos_Left   =  Int_Left + DELTA_LEFT*(Int_multiplicador-1);
            Trs_Auxiliar.Int_Pos_Seq    = ++Int_Trs_Seq;
              Trs_Auxiliar.Str_Rotulo     =  Evt_Atual.Str_rotulo;
            Trs_Atual.Trs_proximo       =  Trs_Auxiliar;
            Trs_Atual                   =  Trs_Auxiliar;

            // Atualiza posicao
            if(Int_multiplicador == 1)
                Int_Top = Int_Top + 2*DELTA_TOP;
            else{
            }
        break;

        case EVENTO_CONCORRENTE:
        break;
    }

    // Atualiza o evento atual a ser analisado, chama por recursao Criar_RP
    if (Evt_Atual.Evt_proximo  !=  null)
        Criar_RP (Evt_Atual.Evt_proximo,txtLog);
    else
        txtLog.append ("MSG : Chegou no fim da rede"+"\n");

}

/** Esta funcao le a lista ligada de lugares e imprime no arquivo */
private void Imprime_Lugares (TextArea txtLog) {

    Lugar       Lgr_Auxiliar  =  Lgr_Inicial;

    try {
        BWriter_Arquivo_RP.write("{PLACES}");
        BWriter_Arquivo_RP.newLine();
        BWriter_Arquivo_RP.newLine();

        // rotina para imprimir os lugares no arquivo
        while (true) {
                BWriter_Arquivo_RP.write("Left= " + Lgr_Auxiliar.Int_Pos_Left);
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Top= " + Lgr_Auxiliar.Int_Pos_Top);
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Show Name= True");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Caption= " + Lgr_Auxiliar.Str_Caption);
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Show Caption= True");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Capacity= 1");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("ShowCapacity= False");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Initial Tokens= 0");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Current Tokens= 0");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.newLine();

                if (Lgr_Auxiliar.Lgr_proximo == null)
                    break;
                else
                    Lgr_Auxiliar  =  Lgr_Auxiliar.Lgr_proximo;
        }
    }

    catch (IOException e) {
        txtLog.append ("ERRO : Nao foi possivel gravar a Rede de Petri!"+"\n");
    }
    catch (NullPointerException e) {
```

```
            txtLog.append ("Arquivo para entrada no Ghenesys criado com sucesso"+"\n");
        }
    }

    /** Esta funcao le a lista ligada de pseudolugares e imprime no arquivo */
    private void Imprime_Pseudo_Lugares (TextArea txtLog) {

        PLugar       PLgr_Auxiliar  =  PLgr_Inicial;

        try {
            BWriter_Arquivo_RP.write("[PSEUDOBOXES]");
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.newLine();

            // rotina para imprimir os lugares no arquivo
            while (true) {
                    BWriter_Arquivo_RP.write("Left= " + PLgr_Auxiliar.Int_Pos_Left);
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Top= " + PLgr_Auxiliar.Int_Pos_Top);
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Show Name= True");
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Caption= " + PLgr_Auxiliar.Str_Caption);
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Show Caption= True");
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Initialy Marked= True");
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Token Present= True");
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("Persistent Mark= True");
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.newLine();
                    if (PLgr_Auxiliar.PLgr_proximo == null) {
                        BWriter_Arquivo_RP.write("[MACROBOXES]");
                        BWriter_Arquivo_RP.newLine();
                        BWriter_Arquivo_RP.write("[MACROACTIVITIES]");
                        break;
                    }
                    else
                        PLgr_Auxiliar  =  PLgr_Auxiliar.PLgr_proximo;
            }
        }
        catch (IOException e) {
            txtLog.append ("ERRO : Nao foi possivel gravar a Rede de Petri!"+"\n");
        }
        catch (NullPointerException e) {
            txtLog.append ("Arquivo para entrada no Ghenesys criado com sucesso"+"\n");
        }
    }

    /** Esta funcao le a lista ligada de transicoes e imprime no arquivo */
    private void Imprime_Transicoes (TextArea txtLog) {

        Transicao   Trs_Auxiliar  =  Trs_Inicial;

        try {
            BWriter_Arquivo_RP.write("[TRANSITIONS]");
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.newLine();

            // rotina para imprimir as transicoes no arquivo
            while (true) {
                BWriter_Arquivo_RP.write("Left= " + Trs_Auxiliar.Int_Pos_Left);
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Top= " + Trs_Auxiliar.Int_Pos_Top);
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Show Name= True");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Caption= " + Trs_Auxiliar.Str_Caption);
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Show Caption= True");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Posicion= Horizontal");
                BWriter_Arquivo_RP.newLine();
                BWriter_Arquivo_RP.write("Time= 0");
                BWriter_Arquivo_RP.newLine();

                BWriter_Arquivo_RP.newLine();
                if (Trs_Auxiliar.Trs_proximo == null)
                    break;
                else
                    Trs_Auxiliar  =  Trs_Auxiliar.Trs_proximo;
            }
        }
        catch (IOException e) {
            txtLog.append ("ERRO : Nao foi possivel gravar a Rede de Petri!"+"\n");
        }
        catch (NullPointerException e) {
            txtLog.append ("Arquivo para entrada no Ghenesys criado com sucesso"+"\n");
        }
    }

    /** Esta funcao retorna o numero do lugar correspondente ao rotulo de retorno de
        um evento do fluxo alternativo */
    private int Criar_Arcos_Desvios (Arco Arc_Variavel) {
        Lugar Lgr_Variavel  =  Lgr_Inicial;
        while (true) {
            if (!Lgr_Variavel.Str_Rotulo.equalsIgnoreCase(Arc_Variavel.Str_Desvio))
                Lgr_Variavel = Lgr_Variavel.Lgr_proximo;
            else
                break;
```

```
        }
        return (Lgr_Variavel.Int_Pos_Seq);
    }

    /** Esta funcao retorna o numero do lugar correspondente a determinado evento
        condicional passado */
    private int Criar_Arcos_Retorno (Arco Arc_Variavel , Evento Evt_Inicial ){//, boolean Bol_Flag) {
          int       Int_retorno  =  500;
          int     Int_pontos    =  0;

        // Para o caso de retorno de uma transicao
        if (Arc_Variavel.Int_Tipo_Anterior  ==  1) {
                    Lugar     Lgr_Variavel =  new Lugar();

                    try {

                // Encontra o rotulo equivalente ao do arco
                    while (!Evt_Inicial.Str_rotulo.equals(Arc_Variavel.Str_Rotulo))
                            Evt_Inicial  =  Evt_Inicial.Evt_proximo;

                // Verifica o numero de pontos do rotulo
                    Int_pontos    =  Retorna_desvio (Evt_Inicial.Str_rotulo);
                    Evt_Inicial  =  Evt_Inicial.Evt_proximo;

                /* Compara com os rotulos posteriores, para verificar o primeiro que
                   contenha mesmo numero de pontos, sendo este o ponto de retorno */
                    while (Int_pontos  !=  Retorna_desvio (Evt_Inicial.Str_rotulo) )
                            Evt_Inicial  =  Evt_Inicial.Evt_proximo;

                /* Achado o evento de retorno, procure o lugar associado a ele com o
                 * mesmo rotulo */
                Lgr_Variavel  =  Lgr_Inicial;
                    while (!Lgr_Variavel.Str_Rotulo.equals(Evt_Inicial.Str_rotulo))
                            Lgr_Variavel  =  Lgr_Variavel.Lgr_proximo;

                  return(Lgr_Variavel.Int_Pos_Seq);
                    }

                    catch(NullPointerException e){
                            Lgr_Variavel  =  Lgr_Inicial;
                            while (Lgr_Variavel.Lgr_proximo != null)
                                    Lgr_Variavel  =  Lgr_Variavel.Lgr_proximo;
                            return (Lgr_Variavel.Int_Pos_Seq - 1);
                    }
         }

       // Para o caso de retorno de um pseudolugar
         else if(Arc_Variavel.Int_Tipo_Anterior  ==  4) {
                    Transicao     Trs_Variavel =  new Transicao();

                    try {

                    while (!Evt_Inicial.Str_rotulo.equals(Arc_Variavel.Str_Rotulo))
                            Evt_Inicial  =  Evt_Inicial.Evt_proximo;

                    Int_pontos    =  Retorna_desvio (Evt_Inicial.Str_rotulo);
                    Evt_Inicial  =  Evt_Inicial.Evt_proximo;

                    while (Int_pontos  !=  Retorna_desvio (Evt_Inicial.Str_rotulo) )
                            Evt_Inicial  =  Evt_Inicial.Evt_proximo;

                    Trs_Variavel  =  Trs_Inicial;
                    while (!Evt_Inicial.Str_rotulo.equals(Trs_Variavel.Str_Rotulo))
                            Trs_Variavel  =  Trs_Variavel.Trs_proximo;

                return(Trs_Variavel.Int_Pos_Seq);
                    }

                    catch(NullPointerException e){
                            Trs_Variavel  =  Trs_Inicial;
                            while (Trs_Variavel.Trs_proximo != null)
                                    Trs_Variavel  =  Trs_Variavel.Trs_proximo;
                            return (Trs_Variavel.Int_Pos_Seq);
                    }
         }

         else
                return(500);

    }

    /** Imprime a lista ligada de arcos no arquivo */
    private void Imprime_Arcos (Evento Evt_Inicial,TextArea txtLog) {

        Arco    Arc_Auxiliar  =  Arc_Inicial;
        int     Int_Retorno   =  0;                    // Essa variavel armazena o valor anterior do arco

        try {
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.write("[ARCS]");
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.newLine();
            BWriter_Arquivo_RP.newLine();

            while (true) {

                // Verifica o caso para arco de desvio do fluxo principal
                if ( Arc_Auxiliar.Int_Ref_Posterior == 500 && Arc_Auxiliar.Str_Desvio != null) {
                    Arc_Auxiliar.Int_Ref_Posterior = Criar_Arcos_Desvios(Arc_Auxiliar);
                }

                // Verifica o caso para arco de retorno para o fluxo principal
```

```
                else if ( Arc_Auxiliar.Int_Ref_Posterior == 500 && Arc_Auxiliar.Str_Desvio == null) {
                    Arc_Auxiliar.Int_Ref_Posterior = Criar_Arcos_Retorno(Arc_Auxiliar , Evt_Inicial );

                    if (Arc_Auxiliar.Int_Tipo_Posterior  ==  0)
                        Int_Retorno  =  Arc_Auxiliar.Int_Ref_Posterior;
                    else
                        Int_Retorno  =  1000;
                }

                // Verifica o caso de arcos onde o elemento anterior nao e definido
                else if ( Arc_Auxiliar.Int_Ref_Anterior == 500 && Arc_Auxiliar.Str_Desvio == null) {
                    if (Int_Retorno  !=  1000)
                        Arc_Auxiliar.Int_Ref_Anterior  =  Int_Retorno;
                    else;
                    //   Arc_Auxiliar.Int_Ref_Anterior  =  Criar_Arcos_Retorno(Arc_Auxiliar , Evt_Inicial );
                }

                else{
                }

                // Coloca os dados do arco no formato do arquivo Ghenesys
                BWriter_Arquivo_RP.write(Arc_Auxiliar.Int_Tipo_Anterior + " " + Arc_Auxiliar.Int_Ref_Anterior + "
" + Arc_Auxiliar.Int_Tipo_Posterior + " " + Arc_Auxiliar.Int_Ref_Posterior + " " +  Arc_Auxiliar.Int_Tipo_Arco);
                BWriter_Arquivo_RP.newLine();

                // Verifica fim da lista, senao atualiza
                if (Arc_Auxiliar.Arc_proximo == null) {
                    BWriter_Arquivo_RP.newLine();
                    BWriter_Arquivo_RP.write("[END_GPTF]");
                    break;
                }
                else
                    Arc_Auxiliar  =  Arc_Auxiliar.Arc_proximo;
            }
        }

        // Caso algum erro de acesso ocorra, cancela operacao e imprime mensagem de erro
        catch (IOException e) {
            txtLog.append ("ERRO : Nao pode gravar arcos"+"\n");
        }
    }

    /** Verifica se dado uma posicao, ela esta ocupada por um elemento ou nao */
    private boolean Pixel_Ocupado (int Int_TOP, int Int_LEFT) {

        Transicao Trs_variavel  =  new Transicao();
        Lugar     Lgr_variavel  =  new Lugar();
        PLugar    PLgr_variavel =  new PLugar();

        if (Trs_Inicial != null)
            Trs_variavel  =  Trs_Inicial;
        if (Lgr_Inicial != null)
            Lgr_variavel  =  Lgr_Inicial;
        if (PLgr_Inicial != null)
            PLgr_variavel =  PLgr_Inicial;

        // Varre lista ligada de transicoes para verificar se coordenadas sao iguais
        while (true){
            if (Int_TOP == Trs_variavel.Int_Pos_Top  &&  Int_LEFT == Trs_variavel.Int_Pos_Left)
                return (true);
            else if (Trs_variavel.Trs_proximo != null)
                Trs_variavel  =  Trs_variavel.Trs_proximo;
            else
                break;
        }

        // Varre lista ligada de lugares para verificar se coordenadas sao iguais
        while (true) {
            if (Int_TOP == Lgr_variavel.Int_Pos_Top  &&  Int_LEFT == Lgr_variavel.Int_Pos_Left)
                return (true);
            else if (Lgr_variavel.Lgr_proximo != null)
                Lgr_variavel  =  Lgr_variavel.Lgr_proximo;
            else
                break;
        }

        // Varre lista ligada do pseudolugares para verificar se coordenadas sao iguais
        while (true){
            if (Int_TOP == PLgr_variavel.Int_Pos_Top  &&  Int_LEFT == PLgr_variavel.Int_Pos_Left)
                return (true);
            else if (PLgr_variavel.PLgr_proximo != null)
                PLgr_variavel  =  PLgr_variavel.PLgr_proximo;
            else
                break;
        }

        // Retorna falso caso posicao nao esta ocupada
        return (false);
    }

    /** Retorna */
    private int Retorna_Sequencia_Alternativa (String Str_Rotulo) {

        int     Int_pos_final  =  Str_Rotulo.lastIndexOf('.');
        int     Int_pos        =  Str_Rotulo.indexOf('.');

        while (true) {
            if (Int_pos  ==  Int_pos_final)
                break;
            else if (Str_Rotulo.indexOf('.',Int_pos+1) == Int_pos_final)
                break;
            else
```

```
                Int_pos = Str_Rotulo.indexOf('.', Int_pos+1);
            }
            String  Str_Seq;
            if (Retorna_desvio(Str_Rotulo) <= 2)
                Str_Seq  =  new String(Str_Rotulo.substring(Int_pos_final+1, Str_Rotulo.length()));
            else
                Str_Seq  =  new String(Str_Rotulo.substring(Int_pos+1, Int_pos_final));
            Integer INT_Seq  =  new Integer(2);
            return (INT_Seq.parseInt(Str_Seq));
        }

        /** Retorna */
        private int Retorna_Sequencia (String Str_Rotulo) {

            int     Int_pos  =  Str_Rotulo.lastIndexOf('.');
            String  Str_Seq  =  new String(Str_Rotulo.substring(Int_pos+1, Str_Rotulo.length()));
            Integer INT_Seq  =  new Integer(2);
            return (INT_Seq.parseInt(Str_Seq));
        }

        /** Retorna a quantidade de pontos (desvios) de determinado rotulo) */
        private int Retorna_desvio(String Str_Rotulo){

            int Int_pontos  =  0;
            int Int_pos     =  0;

            while (Str_Rotulo.indexOf('.',Int_pos) != -1) {
               Int_pos  =  Str_Rotulo.indexOf('.', Int_pos);
               Int_pos++;
               Int_pontos++;
            }
            return(Int_pontos);
        }
}
```

## Classe Evento.java

```
package tf;

public class Evento {

    /* Notacao para classificacao dos eventos
     * i - Inicio de processo;
     * f - Fim de processo;
     * p - Inicio de um evento do fluxo principal;
     * a - Inicio de um evento do fluxo alternativo;
     * ? - Inicio de um evento condicional;
     * | - Eventos concorrentes;
     */

     // Convencoes da linguagem BNF adotada
    static final char INICIO_CAMPO          = '<';
    static final char FIM_CAMPO             = '>';
    static final char SEPARADOR_ROTULO      = '.';
    static final char SEPARADOR_STRING      = '-';
    static final char MARCA_SIMBOLO         = '"';
    static final char INICIO_PROCESSO       = 'i';
    static final char FIM_PROCESSO          = 'f';
    static final char EVENTO_FP             = 'p';
    static final char EVENTO_FA             = 'a';
    static final char EVENTO_CONDICIONAL    = '?';
    static final char SIMBOLO_DESVIO        = '~';
    static final char EVENTO_CONCORRENTE    = '|';

    // Variaveis do programa
    protected String Str_rotulo             = new String();
    protected String Str_fluxo_alternativo  = new String();
    protected String Str_condicao           = new String();
    protected String Str_titulo             = new String();
    protected String Str_descricao          = new String();
    protected String Str_desvio             = new String();
    protected Evento Evt_proximo;
    protected char   Chr_tipo;

    /** Creates a new instance of Evento */
    public Evento(String valor_rotulo) {
        Str_rotulo          =  new String(valor_rotulo);
    }

    // Retorna o tipo do evento
    public boolean Verifica_Tipo (char simbolo) {
        boolean Boo_Teste   =  false;

        switch (simbolo) {
            case INICIO_PROCESSO :
                this.Chr_tipo  =  simbolo;
                Boo_Teste      =  true;
            break;

            case FIM_PROCESSO :
                this.Chr_tipo  =  simbolo;
                Boo_Teste      =  true;
            break;

            case EVENTO_FP :
                this.Chr_tipo  =  simbolo;
                Boo_Teste      =  true;
            break;

            case EVENTO_FA :
                this.Chr_tipo  =  simbolo;
```

```
                Boo_Teste       =  true;
            break;

            case EVENTO_CONDICIONAL :
                this.Chr_tipo   =  simbolo;
                Boo_Teste       =  true;
            break;

            case SIMBOLO_DESVIO :
                this.Chr_tipo   =  simbolo;
                Boo_Teste       =  true;
            break;

            case EVENTO_CONCORRENTE :
                this.Chr_tipo   =  simbolo;
                Boo_Teste       =  true;
            break;
        }

        if (Boo_Teste)
            return (true);
        else
            return (false);
    }
}
```

## Classe Frame1.java

```
package tf;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Frame1 extends JFrame {
//Variaveis da Interface
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();
  Panel panel1 = new Panel();
  Panel panel2 = new Panel();
  BorderLayout borderLayout2 = new BorderLayout();
  TextArea txtArea = new TextArea();
  Panel panel9 = new Panel();
  CheckboxGroup checkboxGroup1 = new CheckboxGroup();
  Panel panel14 = new Panel();
  FlowLayout flowLayout2 = new FlowLayout();
  Panel panel15 = new Panel();
  Panel panel16 = new Panel();
  Button cmdSair = new Button();
  Button cmdConverter = new Button();
  Button cmdSalvar = new Button();
  Button cmdAbrir = new Button();
  Button cmdNovo = new Button();
  Label label3 = new Label();
  Label label2 = new Label();
  Label label1 = new Label();
  Panel panel8 = new Panel();
  TextField txtEndereco_Saida = new TextField();
  Panel panel7 = new Panel();
  TextField txtEndereco_Entrada = new TextField();
  Panel panel6 = new Panel();
  Panel panel5 = new Panel();
  Panel panel4 = new Panel();
  Panel panel3 = new Panel();
  BorderLayout borderLayout12 = new BorderLayout();
  BorderLayout borderLayout11 = new BorderLayout();
  BorderLayout borderLayout9 = new BorderLayout();
  BorderLayout borderLayout8 = new BorderLayout();
  BorderLayout borderLayout7 = new BorderLayout();
  BorderLayout borderLayout6 = new BorderLayout();
  Panel panel13 = new Panel();
  BorderLayout borderLayout5 = new BorderLayout();
  Panel panel12 = new Panel();
  BorderLayout borderLayout4 = new BorderLayout();
  Checkbox chkRede_Petri = new Checkbox();
  Panel panel11 = new Panel();
  BorderLayout borderLayout3 = new BorderLayout();
  Checkbox chkUse_Case = new Checkbox();
  Panel panel10 = new Panel();
  BorderLayout borderLayout10 = new BorderLayout();
  Panel panel17 = new Panel();

//Variaveis do Programa
 /* Objeto para comunicacao com o arquivo txt */
  private bancodados COM;
  BorderLayout borderLayout13 = new BorderLayout();
  Panel panel18 = new Panel();
  Panel panel19 = new Panel();
  Panel panel20 = new Panel();
  Panel panel21 = new Panel();
  Panel panel22 = new Panel();
  TextArea txtLog = new TextArea();
  BorderLayout borderLayout14 = new BorderLayout();
  Label label4 = new Label();
  BorderLayout borderLayout15 = new BorderLayout();

  //Construct the frame
  public Frame1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
```

```
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
    //setIconImage(Toolkit.getDefaultToolkit().createImage(Frame1.class.getResource("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(1006, 738));
    this.setTitle("Menu Principal");
    panel2.setLayout(borderLayout2);
    panel9.setLayout(flowLayout2);
    panel14.setLayout(borderLayout13);
    cmdSair.setFont(new java.awt.Font("Dialog", 1, 12));
    cmdSair.setLabel("Sair");
    cmdSair.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cmdSair_actionPerformed(e);
      }
    });
    cmdConverter.setFont(new java.awt.Font("Dialog", 1, 12));
    cmdConverter.setLabel("Converter");
    cmdConverter.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cmdConverter_actionPerformed(e);
      }
    });
    cmdSalvar.setFont(new java.awt.Font("Dialog", 1, 12));
    cmdSalvar.setLabel("Salvar");
    cmdSalvar.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cmdSalvar_actionPerformed(e);
      }
    });
    cmdAbrir.setFont(new java.awt.Font("Dialog", 1, 12));
    cmdAbrir.setLabel("Abrir");
    cmdAbrir.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cmdAbrir_actionPerformed(e);
      }
    });
    cmdNovo.setFont(new java.awt.Font("Dialog", 1, 12));
    cmdNovo.setLabel("Limpar");
    cmdNovo.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cmdNovo_actionPerformed(e);
      }
    });
    label3.setFont(new java.awt.Font("Dialog", 1, 12));
    label3.setText("Tipo do Arquivo de Entrada:");
    label2.setFont(new java.awt.Font("Dialog", 1, 12));
    label2.setText("Endereço do Arquivo de Saída:");
    label1.setFont(new java.awt.Font("Dialog", 1, 12));
    label1.setText("Endereço do Arquivo de Entrada:");
    panel8.setLayout(borderLayout4);
    txtEndereco_Saida.setText("C:\\");
    panel7.setLayout(borderLayout5);
    txtEndereco_Entrada.setText("C:\\");
    panel6.setLayout(borderLayout6);
    panel5.setLayout(borderLayout7);
    panel4.setLayout(borderLayout8);
    panel3.setLayout(borderLayout3);
    panel12.setLayout(borderLayout12);
    chkRede_Petri.setCheckboxGroup(checkboxGroup1);
    chkRede_Petri.setLabel("Redes de Petri");
    panel11.setLayout(borderLayout11);
    chkUse_Case.setCheckboxGroup(checkboxGroup1);
    chkUse_Case.setLabel("Use Case");
    panel10.setLayout(borderLayout9);
    panel15.setLayout(borderLayout10);
    panel19.setLayout(borderLayout14);
    label4.setFont(new java.awt.Font("Dialog", 1, 12));
    label4.setText("Tela de Logs:");
    panel22.setLayout(borderLayout15);
    contentPane.add(panel2,  BorderLayout.EAST);
    panel2.add(panel15,  BorderLayout.CENTER);
    panel5.add(label1, BorderLayout.WEST);
    panel5.add(txtEndereco_Entrada, BorderLayout.SOUTH);
    panel15.add(panel4, BorderLayout.CENTER);
    panel10.add(txtEndereco_Saida, BorderLayout.CENTER);
    panel10.add(label2, BorderLayout.NORTH);
    panel4.add(panel11, BorderLayout.CENTER);
    panel4.add(panel10, BorderLayout.NORTH);
    panel12.add(chkUse_Case, BorderLayout.CENTER);
    panel12.add(chkRede_Petri, BorderLayout.SOUTH);
    panel12.add(label3, BorderLayout.NORTH);
    panel11.add(panel13, BorderLayout.CENTER);
    panel15.add(panel3,  BorderLayout.SOUTH);
    panel11.add(panel12, BorderLayout.NORTH);
    panel6.add(cmdNovo, BorderLayout.NORTH);
    panel6.add(cmdAbrir, BorderLayout.CENTER);
    panel3.add(panel7, BorderLayout.CENTER);
    panel3.add(panel6, BorderLayout.NORTH);
    panel7.add(cmdSalvar, BorderLayout.NORTH);
    panel7.add(cmdConverter, BorderLayout.CENTER);
    panel7.add(cmdSair, BorderLayout.SOUTH);
    panel3.add(panel8, BorderLayout.SOUTH);
    panel15.add(panel5,  BorderLayout.NORTH);
    panel2.add(panel16,  BorderLayout.WEST);
    panel2.add(panel17,  BorderLayout.EAST);
```

```
    contentPane.add(panel1,  BorderLayout.WEST);
    contentPane.add(txtArea, BorderLayout.CENTER);
    contentPane.add(panel9, BorderLayout.NORTH);
    contentPane.add(panel14,  BorderLayout.SOUTH);
    panel14.add(panel18, BorderLayout.WEST);
    panel14.add(panel19, BorderLayout.CENTER);
    panel19.add(txtLog,  BorderLayout.CENTER);
    panel14.add(panel20, BorderLayout.SOUTH);
    panel14.add(panel21, BorderLayout.EAST);
    panel14.add(panel22, BorderLayout.NORTH);
    panel22.add(label4,  BorderLayout.WEST);

    //Envia primeira mensagem para o Log
    txtLog.append("Sistema inicializado com sucesso!"+"\n");
  }
  //Overridden so we can exit when window is closed
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }

//Este comando encerra o programa fechando todas janelas abertas
  void cmdSair_actionPerformed(ActionEvent e) {
    System.exit(0);
  }

//Esse comando limpa a textArea e permite que um novo arquivo seja digitado
  void cmdNovo_actionPerformed(ActionEvent e) {
    txtArea.setText("");
    txtEndereco_Entrada.setText("C:\\");
    txtEndereco_Saida.setText("C:\\");
    txtLog.append("\n" +
"_____________________"+"\n");
    //txtLog.append("Novo arquivo criado com sucesso!"+"\n");
  }

  //Esse comando abre o arquivo especificado no endereco indicado
  void cmdAbrir_actionPerformed(ActionEvent e) {
    try{
        String Path = new String();
        String Flag_Erro = new String();

        Path=txtEndereco_Entrada.getText();
        COM=new bancodados(Path,txtLog);

        txtArea.setText("");
        COM.apagar();

        Flag_Erro = COM.inicio(Path,txtLog);

        if (Flag_Erro=="OK"){
          for(int i=1;i<=COM.banco.size();i++){
            if(i<10)
              txtArea.append("§ " + i + " : ");
            else
              txtArea.append("§" + i + " : ");

            txtArea.append(COM.banco.get(i-1)+"\n");
          }
          txtLog.append("\n" +
"_____________________"+"\n");
          txtLog.append("Arquivo aberto com sucesso!"+"\n");
        }
        else;
    }
    catch(Exception erro){
      this.txtLog.append("Arquivo nao pode ser aberto");
    }
  }

  //Esse comando fecha qualquer janela aberta e limpa todos os campos do Menu Principal

  //Esse comando salva o arquivo no formato e endereco especificados no Menu Principal
  void cmdSalvar_actionPerformed(ActionEvent e) {
    String dados =new String();
    String Path = new String();
    String Flag_Erro = new String();

    Path=txtEndereco_Saida.getText();

    COM=new bancodados(Path,txtLog);

    dados=txtArea.getText();

    Flag_Erro = COM.inclusao(dados,txtLog);

    if (Flag_Erro=="OK"){
      txtLog.append("\n" +
"_____________________"+"\n");
      txtLog.append("Arquivo salvo com sucesso!"+"\n");
    }
    else
      txtLog.append("Erro ao tentar salvar arquivo! Arquivo não salvo!"+"\n");
  }

  void Conv_RP(){
    String Path_Entrada = new String();
```

```
    Path_Entrada=txtEndereco_Entrada.getText();

    String Path_Saida = new String();
    Path_Saida=txtEndereco_Saida.getText();

    Converte_BNF_RP Conversor = new Converte_BNF_RP(Path_Entrada,Path_Saida,txtLog);
  }

  void Conv_UC(){
    String Path_Entrada = new String();
    Path_Entrada=txtEndereco_Entrada.getText();

    String Path_Saida = new String();
    Path_Saida=txtEndereco_Saida.getText();

    Converte_RP_BNF Conversor1 = new Converte_RP_BNF(Path_Entrada,Path_Saida,txtLog);
  }

  //Este comando chama a classe correspondente a conversao conforme o formato do arquivo especificado no Menu
Principal
  void cmdConverter_actionPerformed(ActionEvent e) {
    String Path_Entrada = new String();
    Path_Entrada=txtEndereco_Entrada.getText();

    String Path_Saida = new String();
    Path_Saida=txtEndereco_Saida.getText();

    char[] Extensao = new char[5];
    char[] teste = new char[100];

    if(chkUse_Case.getState()){
      teste=Path_Entrada.toCharArray();
      for(int i=teste.length, j=1;i>teste.length-4;i--,j++){
        Extensao[j]=teste[i-1];
      }
      if(Extensao[1]=='t' & Extensao[2]=='x' & Extensao[3]=='t')
        Conv_RP();
      else
        txtLog.append("Arquivo de entrada incompatível com Conversão selecionada!"+"\n");
    }
    else{
      if(chkRede_Petri.getState()){
        teste=Path_Entrada.toCharArray();
        for(int i=teste.length, j=1;i>teste.length-4;i--,j++){
          Extensao[j]=teste[i-1];
        }
        if(Extensao[1]=='n' & Extensao[2]=='p' & Extensao[3]=='g')
          Conv_UC();
        else
          txtLog.append("Arquivo de entrada incompatível com Conversão selecionada!"+"\n");
      }
      else
        txtLog.append("ERRO: Operação de Conversão não identificada!"+"\n");
    }

    txtLog.append("\n" +
"
_____
_____"+"\n");
  }
}
```

## Classe Lugar.java

```
package tf;

import java.lang.*;
import java.io.*;

public class Lugar {

    protected String          Str_Caption;          // Armazena a legenda do lugar
    protected Lugar           Lgr_proximo;          // Referencia ao proximo lugar na lista ligada
    protected int             Int_Pos_Left;         // Posicao LEFT do lugar no Ghenesys
    protected int             Int_Pos_Top;          // Posicao TOP do lugar no Ghenesys
    protected int             Int_Pos_Seq;          // Posicao na sequencia da lista ligada
    protected String          Str_Rotulo;           // Guarda o rotulo do evento associado

    /** Creates a new instance of Lugar */
    public Lugar() {
        Str_Caption  =  "";
    }
}
```

## Classe PLugar.java

```
package tf;

public class PLugar {

    protected String          Str_Caption;          // Guarda a legenda da condicao do pseudolugar
    protected PLugar          PLgr_proximo;         // Guarda a referencia ao proximo pseudo lugar da lista
ligada
    protected int             Int_Pos_Left;         // Guarda a posicao LEFT no Ghenesys
    protected int             Int_Pos_Top;          // Guarda a posicao TOP no Ghenesys
    protected int             Int_Pos_Seq;          // Guarda o numero da sequencia na lista ligada
    protected String          Str_Rotulo;           // Guarda o rotulo associado ao pseudolugar

    /** Creates a new instance of PLugar */
    public PLugar() {
        Str_Caption  =  "Caption";
```

```
    }

}
```

## Classe Transicao.java

```
package tf;

import java.lang.*;
import java.io.*;

public class Transicao {

    protected String          Str_Caption;          // Guarda a legenda da transicao
    protected Transicao       Trs_proximo;          // Guarda a referencia a proxima transicao na lista
ligada
    protected int             Int_Pos_Left;         // Guarda a posicao LEFT no Ghenesys
    protected int             Int_Pos_Top;          // Guarda a posicao TOP no Ghenesys
    protected int             Int_Pos_Seq;          // Guarda o numero da sequencia na lista ligada
    protected String          Str_Rotulo;           // Guarda o rotulo associado a transicao

    /** Creates a new instance of Transicao */
    public Transicao() {
        Str_Caption = "Caption";
    }
}
```

## Classe Trata_Elemento.java

```
package tf;

import java.lang.*;
import java.io.*;

public class Trata_Elemento {

    public  Lugar             Lgr_Inicial   = new Lugar();
    // Variavel auxiliar que guarda o lugar inicial da lista ligada
    public  PLugar            PLgr_Inicial  = new PLugar();
    // Variavel auxiliar que guarda o pseudo lugar inicial da lista ligada
    public  Transicao         Trs_Inicial   = new Transicao();
    // Variavel auxiliar que guarda a transicao inicial da lista ligada
    public  Arco              Arc_Inicial   = new Arco();
    // Variavel auxiliar que guarda o arco inicial da lista ligada
    private Lugar             Lgr_Atual     = new Lugar();
    // Variavel auxiliar que guarda o ultimo lugar da lista ligada
    private PLugar            PLgr_Atual    = new PLugar();
    // Variavel auxiliar que guarda o ultimo pseudo lugar da lista ligada
    private Transicao         Trs_Atual     = new Transicao();
    // Variavel auxiliar que guarda a ultima transicao da lista ligada
    private Arco              Arc_Atual     = new Arco();
    // Variavel auxiliar que guarda o ultimo arco da lista ligada
    public  BufferedReader    BReader_Arquivo_RP;
    // Variavel que representa o buffer de leitura do arquivo de entrada
    private int               Int_Lgr_Seq;
    // Variavel que armazena o numero do lugar na lista ligada
    private int               Int_Trs_Seq;
    // Variavel que armazena o numero da transicao na lista ligada
    private int               Int_PLgr_Seq;
    // Variavel que armazena o numero do pseudo lugar na lista ligada
    private int               Int_Arc_Seq;
    // Variavel que armazena o numero do arco na lista ligada

    /** Funcao que inicializa as variaveis da classe */
    public Trata_Elemento (BufferedReader BReader_Arquivo) {
        BReader_Arquivo_RP  =  BReader_Arquivo;
        Int_Lgr_Seq  =  -1;
        Int_Trs_Seq  =  -1;
        Int_PLgr_Seq =  -1;
        Int_Arc_Seq  =  -1;
    }

    /** Funcao que adiciona um elemento do tipo Int_Tipo_Elemento na sua lista ligada
     *  criada anteriormente */
    public void Adiciona_Elemento (int Int_Tipo_Elemento , String Str_linha_atual) {

        switch ( Int_Tipo_Elemento ) {

            // Caso elemento lugar
            case 0 :
                Int_Lgr_Seq++;
                Adiciona_Lugar ();
            break;

            // Caso elemento transicao
            case 1 :
                Int_Trs_Seq++;
                Adiciona_Transicao ();
            break;

            // Caso elemento pseudo lugar
            case 2 :
                Int_PLgr_Seq++;
                Adiciona_Pseudo_Lugar ();
            break;

            // Caso elemento macrobox
            case 3 :
                System.out.println ("e chegou macrobox");
            break;
```

```
            // Caso elemento macroatividade
            case 4 :
                System.out.println ("e chegou macroatividade");
            break;

            // Caso elemento arco
            case 5 :
                Int_Arc_Seq++;
                Adiciona_Arco (Str_linha_atual);
            break;
        }
    }

    /** Funcao que adiciona um novo lugar a lista ligada de lugares */
    public void Adiciona_Lugar () {

        String Str_linha     =  new String();          // Variavel que armazena o valor retirado da linha lida
do arquivo
        String Str_auxiliar  =  new String();          // Variavel auxiliar para manipulacao dos dados
        Lugar  Lgr_auxiliar  =  new Lugar();           // Variavel auxiliar para manipulacao da lista ligada de
lugares
        int    Int_pos;                                // Variavel para manipulacao da string
        int    Int_pos2;                               // Variavel para manipulacao da string

        try {
            // loop que pula duas linhas e armazena a terceira em Str_auxiliar
            for (int i=0 ; i<3 ; i++)
                Str_auxiliar  =  BReader_Arquivo_RP.readLine();

            // Procura pelo caracter = e retira a string correspondente ao titulo
            Int_pos    =  Str_auxiliar.indexOf('=');
            //Int_pos2   =  Str_auxiliar.indexOf('-');
            //Str_linha  =  Str_auxiliar.substring(Int_pos2 + 2);
            Str_linha  =  Str_auxiliar.substring(Int_pos + 2);

            // pula cinco linhas
            for (int i=0 ; i<5 ; i++)
                BReader_Arquivo_RP.readLine();

            // atualiza lista ligada , caso primeiro elemento
            if (Int_Lgr_Seq  ==  0) {
                Lgr_Inicial.Str_Caption  =  Str_linha;
                //Lgr_Inicial.Str_Rotulo   =  Str_auxiliar.substring(Int_pos+2,Int_pos2-1);
                Lgr_Inicial.Int_Pos_Seq  =  Int_Lgr_Seq;
            }

            else {
                Lgr_auxiliar.Str_Caption  =  Str_linha;
                //Lgr_auxiliar.Str_Rotulo   =  Str_auxiliar.substring(Int_pos+2,Int_pos2-1);
                Lgr_auxiliar.Int_Pos_Seq  =  Int_Lgr_Seq;

                // caso segundo elemento
                if (Int_Lgr_Seq  ==  1) {
                    Lgr_Inicial.Lgr_proximo  =  Lgr_auxiliar;
                    Lgr_Atual                =  Lgr_auxiliar;
                }

                // caso elemento N
                else {
                    Lgr_Atual.Lgr_proximo    =  Lgr_auxiliar;
                    Lgr_Atual                =  Lgr_auxiliar;
                }
            }
        }

        catch (IOException e) {
            System.out.println ("ERRO : Acesso ao arquivo de entrada nao concluido");
        }
    }

    /** Funcao que adiciona um novo pseudo lugar a lista ligada de pseudo lugares */
    public void Adiciona_Pseudo_Lugar () {

        String    Str_linha      =  new String();       // Variavel que armazena o valor retirado da linha lida
do arquivo
        String    Str_auxiliar   =  new String();       // Variavel auxiliar para manipulacao dos dados
        PLugar    PLgr_auxiliar  =  new PLugar();       // Variavel auxiliar para manipulacao da lista ligada de
pseudo lugares
        int       Int_pos;                              // Variavel para manipulacao da string
        int       Int_pos2;                             // Variavel para manipulacao da string

        try {
            // loop que pula duas linhas e armazena a terceira em Str_auxiliar
            for (int i=0 ; i<3 ; i++)
                Str_auxiliar  =  BReader_Arquivo_RP.readLine();

            // procura por titulo valido
            Int_pos    =  Str_auxiliar.indexOf('=');
            Int_pos2   =  Str_auxiliar.indexOf('-');
            Str_linha  =  Str_auxiliar.substring(Int_pos2 + 2);

            // pula quatro linhas
            for (int i=0 ; i<4 ; i++)
                BReader_Arquivo_RP.readLine();

            // inicializa lista ligada de pseudolugares, caso primeiro elemento
            if (Int_PLgr_Seq  ==  0) {
                PLgr_Inicial.Str_Caption  =  Str_linha;
                PLgr_Inicial.Str_Rotulo   =  Str_auxiliar.substring(Int_pos+2, Int_pos2-1);
                PLgr_Inicial.Int_Pos_Seq  =  Int_PLgr_Seq;
            }
```

```
            else {
                PLgr_auxiliar.Str_Caption  =  Str_linha;
                PLgr_auxiliar.Str_Rotulo   =  Str_auxiliar.substring(Int_pos+2, Int_pos2-1);
                PLgr_auxiliar.Int_Pos_Seq  =  Int_PLgr_Seq;

                // caso segundo elemento
                if (Int_PLgr_Seq  ==  1) {
                    PLgr_Inicial.PLgr_proximo  =  PLgr_auxiliar;
                    PLgr_Atual                 =  PLgr_auxiliar;
                }

                // caso elemento N
                else {
                    PLgr_Atual.PLgr_proximo    =  PLgr_auxiliar;
                    PLgr_Atual                 =  PLgr_auxiliar;
                }
            }
        }

        catch (IOException e) {
            System.out.println ("ERRO : Acesso ao arquivo de entrada nao concluido");
        }
    }

    /** Funcao que adiciona uma nova transicao a lista ligada de transicoes */
    public void Adiciona_Transicao () {

        String    Str_linha     =  new String();          // Variavel que armazena o dado valido da linha
        String    Str_auxiliar  =  new String();          // Variavel auxiliar para manipular string de dados
da linha
        Transicao Trs_auxiliar  =  new Transicao();       // Variavel para manipular lista ligada de transicoes
        int       Int_pos;                                // Variavel auxiliar para manipular substring de
dados
        int       Int_pos2;                               // Variavel para manipulacao da string

        try {
            // loop que pula duas linhas e armazena a terceira em Str_auxiliar
            for (int i=0 ; i<3 ; i++)
                Str_auxiliar  =  BReader_Arquivo_RP.readLine();

            // procura por dado valido de string
            Int_pos    =  Str_auxiliar.indexOf('=');
            Int_pos2   =  Str_auxiliar.indexOf('-');
            Str_linha  =  Str_auxiliar.substring(Int_pos2 + 2);

            // pula tres linhas
            for (int i=0 ; i<3 ; i++)
                BReader_Arquivo_RP.readLine();

            // atualiza lista ligada, caso primeiro elemento
            if (Int_Trs_Seq  ==  0) {
                Trs_Inicial.Str_Caption  =  Str_linha;
                Trs_Inicial.Str_Rotulo   =  Str_auxiliar.substring(Int_pos+2, Int_pos2-1);
                Trs_Inicial.Int_Pos_Seq  =  Int_Trs_Seq;
            }

            else {
                Trs_auxiliar.Str_Caption  =  Str_linha;
                Trs_auxiliar.Str_Rotulo   =  Str_auxiliar.substring(Int_pos+2, Int_pos2-1);
                Trs_auxiliar.Int_Pos_Seq  =  Int_Trs_Seq;

                // caso segundo elemento
                if (Int_Trs_Seq  ==  1) {
                    Trs_Inicial.Trs_proximo  =  Trs_auxiliar;
                    Trs_Atual                =  Trs_auxiliar;
                }

                // caso elemento N
                else {
                    Trs_Atual.Trs_proximo    =  Trs_auxiliar;
                    Trs_Atual                =  Trs_auxiliar;
                }
            }
        }

        catch (IOException e) {
            System.out.println ("ERRO : Acesso ao arquivo de entrada nao concluido");
        }

    }

    /** Funcao que adiciona novo arco na lista ligada de arcos */
    public void Adiciona_Arco (String Str_auxiliar) {

        String    Str_linha     =  new String();         // Variavel que armazena dado valido da linha
        Arco      Arc_auxiliar  =  new Arco();           // Variavel utilizada na manipulacao da lista ligada de
arcos
        int       pos_final;                             // Variavel utilizada na manipulacao de substring
        int       pos_inicial   =  0;                    // Variavel utilizada na manipulacao de substring

        // Loop que varre pelos cinco dados contidos na linha lida
        for (int i=0 ; i<5 ; i++)
        {
            // procura posicao final
            pos_final = Str_auxiliar.indexOf(" " , pos_inicial);

            // caso final de string
            if (pos_final  ==  -1)
                Str_linha = Str_auxiliar.substring(pos_inicial);
            else
                Str_linha = Str_auxiliar.substring(pos_inicial , pos_final);
```

```
            // transforma dado lido em inteiro
            Integer INT_auxiliar = new Integer (Str_linha);

            switch (i) {

                // caso tipo elemento anterior
                case 0:
                    Arc_auxiliar.Int_Tipo_Anterior = INT_auxiliar.intValue();
                break;

                // caso referencia elemento anterior
                case 1:
                    Arc_auxiliar.Int_Ref_Anterior = INT_auxiliar.intValue();
                break;

                // caso tipo elemento posterior
                case 2:
                    Arc_auxiliar.Int_Tipo_Posterior = INT_auxiliar.intValue();
                break;

                // caso referencia elemento posterior
                case 3:
                    Arc_auxiliar.Int_Ref_Posterior = INT_auxiliar.intValue();
                break;

                // caso tipo de arco
                case 4:
                    Arc_auxiliar.Int_Tipo_Arco = INT_auxiliar.intValue();
                break;
            }

            // atualiza posicao a ser analisada
            pos_inicial = pos_final + 1;
        }

        if (Int_Arc_Seq == 0)
        {
            Arc_Inicial = Arc_auxiliar;
        }

        else if (Int_Arc_Seq == 1) {
            Arc_Inicial.Arc_proximo = Arc_auxiliar;
            Arc_Atual               = Arc_auxiliar;
        }

        else {
            Arc_Atual.Arc_proximo   = Arc_auxiliar;
            Arc_Atual               = Arc_auxiliar;
        }
    }
}
```

## Classe Trata_Evento.java

```
package tf;

import java.lang.*;
import java.io.*;
import java.awt.*;

public class Trata_Evento {

    // Convencoes da linguagem BNF adotada
    static final char FIM_LINHA             = ';';
    static final char INICIO_CAMPO          = '<';
    static final char FIM_CAMPO             = '>';
    static final char SEPARADOR_ROTULO      = '.';
    static final char SEPARADOR_STRING      = '-';
    static final char MARCA_SIMBOLO         = '"';
    static final char INICIO_PROCESSO       = 'i';
    static final char FIM_PROCESSO          = 'f';
    static final char EVENTO_FP             = 'p';
    static final char EVENTO_FA             = 'a';
    static final char EVENTO_CONDICIONAL    = '?';
    static final char SIMBOLO_DESVIO        = '~';
    static final char EVENTO_CONCORRENTE    = '|';

    // Variaveis utilizadas pela classe
    private     int       Int_pos_inicial;
    private     int       Int_pos_final;
    private     int       Int_Num_Evento;
    private     String    Str_Evento;
    private     String    Str_Rotulo        = new String ();
    private     String    Str_Desvio        = new String ();
    protected   boolean   Bol_Erro          = false;
    protected   Evento    Evt_Atual;

    /** Essa funcao recebe uma string com a descricao do evento, e passa a analisa-la
     *  de acordo com os parametros de um evento (rotulo, nome, descricao, etc) */
    public Trata_Evento(String Str_Evento_Analisado, int Int_Evento,TextArea txtLog) {

        Str_Evento      =  Str_Evento_Analisado;
        Int_Num_Evento  =  Int_Evento;

        Int_pos_inicial = 0;
        Int_pos_final   = 0;

        for (int i=0 ; i<5 && Bol_Erro== false ; i++) {
            switch (i) {
                case 0:
```

```
                Retorna_Rotulo(txtLog);
                break;
            case 1:
                Retorna_Tipo_Evento(txtLog);
                break;
            case 2:
                Retorna_Condicao();
                break;
            case 3:
                Retorna_Descricao();
                break;
            case 4:
                Retorna_Desvio();
                break;
        }
    }
}

/** Funcao que retorna o rotulo associado ao evento */
void Retorna_Rotulo(TextArea txtLog) {

    // Localiza o ponto onde se inicia o rotulo
    Int_pos_inicial     =  Str_Evento.indexOf(INICIO_CAMPO, Int_pos_final);
    Int_pos_final       =  Str_Evento.indexOf(FIM_CAMPO, Int_pos_inicial);

    // Verifica se o caracter '>' encontrado corresponde mesmo ao rotulo
    if (Int_pos_final - Int_pos_inicial < 5) {
        Str_Rotulo      =  Str_Rotulo + Str_Evento.substring(Int_pos_inicial + 1, Int_pos_final);

        Int_pos_inicial =  Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_final);

        // Verifica se o rotulo contem mais de um numero sequencial
        if (Str_Evento.charAt(Int_pos_inicial + 1) == SEPARADOR_ROTULO) {
            Str_Rotulo  =  Str_Rotulo + SEPARADOR_ROTULO;
            Retorna_Rotulo(txtLog);
        }

        // Caso o rotulo nao contem mais extensao
        else
            Evt_Atual   =  new Evento(Str_Rotulo);
    }

    // Caso o rotulo nao esteja formatado com o caracter '>'
    else {
        txtLog.append ("ERRO : Formato incorreto no rotulo do evento na linha " + (Int_Num_Evento) + "\n");
        Evt_Atual       =  new Evento(Str_Rotulo);
        Bol_Erro        =  true;
    }

}

/** Retornao tipo do evento */
void Retorna_Tipo_Evento(TextArea txtLog) {

    char CH_simbolo;

    // Localiza o ponto onde se inicia o tipo de evento
    Int_pos_inicial     =  Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_final);
    Int_pos_final       =  Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_inicial);

    // Verifica se o caracter '"' encontrado corresponde mesmo ao tipo de evento
    if (Int_pos_final - Int_pos_inicial < 5) {
        CH_simbolo      =  Str_Evento.charAt(Int_pos_inicial + 1);
        if(!Evt_Atual.Verifica_Tipo(CH_simbolo)) {
            txtLog.append ("ERRO : Simbolo nao especificado na linha " + (Int_Num_Evento) + "\n");
            Bol_Erro    =  true;
        }
    }

    // Caso o tipo nao esteja formatado com o caracter '>'
    else {
        txtLog.append ("ERRO : Formato incorreto no tipo do evento na linha " + (Int_Num_Evento) + "\n");
        Bol_Erro        =  true;
    }

}

/** Retorna o nome do evento */
void Retorna_Descricao () {

    int Int_verifica_pos    =  0;
    int Int_pos_auxiliar    =  0;

    // Localiza o ponto onde se inicia a descricao do evento
    Int_pos_inicial         =  Str_Evento.indexOf(INICIO_CAMPO, Int_pos_final);
    Int_pos_auxiliar        =  Int_pos_inicial;

    // Loop que procura pela notacao de fim de descricao
    while (true) {
        Int_pos_final       =  Str_Evento.indexOf(FIM_CAMPO, Int_pos_inicial);
        Int_verifica_pos    =  Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_final);

        // Verifica se notacao e consistente (pode ser parte da string e nao fim dela)
        if ( (Int_verifica_pos - Int_pos_final) < 4 ) {
            Evt_Atual.Str_descricao  =  Str_Evento.substring(Int_pos_auxiliar + 1, Int_pos_final);
            break;
        }
        // Incrementa posicao de fim de descricao
        else
            Int_pos_inicial  =  Int_pos_final + 1;
    }
}
```

# BIBLIOGRAFIA

[1] Coad, Peter, Lefebvre, Eric, De Luca, Jeff, *Java Modeling In Color With UML: Enterprise Components and Process*, Prentice Hall, 1999.

[2] Eriksson, Hans-Erik, Penker, Magnus, UML Toolkit, Estados Unidos da América: John Wiley & Sons, Inc., 1998.

[3] Kotonya, Gerald, Sommerville, I., Requirements engineering: processes and techniques, New York: John Wiley & Sons, 1998.

[4] Philippe Kruchten, *The Rational Unified Process, an Introduction – Second Edition.*

[5] Process Manager's Guide, *Rational Unified Process Builder.*

[6] Rational Software White Paper, *A Comparison of RUP and XP*

[7] Rational Software White Paper, *Rational Unified Process, Best Practices for Software Development Teams.*

[8] Rational Software White Paper TP 165A, 5/02, *Rational Unified Process for System Engineering.*

[9] Santos, Eston Almança dos, Verificação de Requisitos de Sistemas utilizando Redes de Petri, 2002. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2002.

[10] Sommerville, I., Sawyer, P., Requirements engineering: a good practice guide, New York: John Wiley & Sons, 1997.

[11] Wolak, Ronald G., *DISS 725 – System Development : Research Paper 1 SDLC on a Diet,* School of Computer and Information Sciences, Nova Southeastern University, Estados Unidos da América,1 2001.

[12] Borland Software Corporation, *A Practical Guide to Getting Started with Together Control Center 6.0,* disponível em <http://www.togethersoft.com/services/practical_guides/getting_started/index.html>, Acesso em 14 de Junho de 2003.

[13] Borland Software Corporation, *Together Control Center 6.0 User Guide,* disponível em <http://cis.bentley.edu/lwaguespack/DownLoad/Together6_User_Guide.pdf> , Acesso em 14 de Junho de 2003.

```
    /** Retorna condicao associada a evento */
    void Retorna_Condicao()
    {
        int Int_verifica_pos    = 0;
        int Int_pos_auxiliar    = 0;

        // Localiza o ponto onde se inicia a condicao ou titulo do evento
        Int_pos_inicial         = Str_Evento.indexOf(INICIO_CAMPO, Int_pos_final);
        Int_pos_auxiliar        = Int_pos_inicial;

        // Loop que procura pelo inicio da string de condicao dada a notacao definida
        while (true) {
            Int_pos_final       = Str_Evento.indexOf(FIM_CAMPO, Int_pos_inicial);
            Int_verifica_pos    = Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_final);

            // Verifica se posicao de notacao encontrada e consistente
            if ( (Evt_Atual.Chr_tipo == EVENTO_CONDICIONAL) && (Str_Evento.charAt(Int_verifica_pos + 1) ==
SEPARADOR_STRING) && ((Int_verifica_pos - Int_pos_final) < 4) ) {
                Evt_Atual.Str_condicao  = Str_Evento.substring(Int_pos_auxiliar + 1, Int_pos_final);
                Int_pos_inicial         = Str_Evento.indexOf(INICIO_CAMPO, Int_pos_final);
                Int_pos_auxiliar        = Int_pos_inicial;

                // Loop que procura pelo fim da string de condicao dada a notacao definida
                while (true) {
                    Int_pos_final       = Str_Evento.indexOf(FIM_CAMPO, Int_pos_inicial);
                    Int_verifica_pos    = Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_final);

                    // Verifica se posicao de notacao encontrada e consistente
                    if (Int_verifica_pos - Int_pos_final < 4) {
                        Evt_Atual.Str_titulo  = Str_Evento.substring(Int_pos_auxiliar + 1, Int_pos_final);
                        break;
                    }
                    // Incrementa posicao
                    else
                        Int_pos_inicial  =  Int_pos_final + 1;
                }
                break;
            }
            // Verifica se chegou no fim da string
            else if ( (Int_verifica_pos - Int_pos_final) < 4 ) {
                Evt_Atual.Str_titulo    = Str_Evento.substring(Int_pos_auxiliar + 1, Int_pos_final);
                break;
            }

            // Incrementa posicao
            else
                Int_pos_inicial  =  Int_pos_final + 1;
        }
    }

    /** Funcao que retorna o desvio de um evento do fluxo alternativo */
    void Retorna_Desvio(){
        if (Evt_Atual.Chr_tipo  ==  EVENTO_FA) {
            Int_pos_inicial         = Str_Evento.indexOf(MARCA_SIMBOLO, Int_pos_final);
            Int_pos_inicial         = Str_Evento.indexOf(INICIO_CAMPO, Int_pos_inicial + 1) + 1;
            Int_pos_final           = Str_Evento.indexOf(']', Int_pos_inicial);

            while (true) {
                while ( Str_Evento.charAt(Int_pos_inicial) != FIM_CAMPO) {
                    Str_Desvio  =  new String(Str_Desvio + Str_Evento.charAt(Int_pos_inicial));
                    Int_pos_inicial++;
                }
                Int_pos_inicial++;

                if (Str_Evento.charAt(Int_pos_inicial) != ']') {
                    Str_Desvio  =  Str_Desvio + SEPARADOR_ROTULO;
                    Int_pos_inicial  =  Str_Evento.indexOf(INICIO_CAMPO, Int_pos_inicial) + 1;
                }
                else
                    break;
            }
            Evt_Atual.Str_desvio  =  Str_Desvio;
        }
    }
}
```

[14] Borland Software Corporation, Apresentações e tutoriais confeccionados pela equipe do TogetherSoft, disponível em <http://www.togethersoft.com/services/>, Acesso em 14 de Junho de 2003.

[15] Site da Java, Ferramentas e documentos básicos para desenvolvimento na linguagem java, fornecido pela Sun, disponível em <http://java.sun.com>, Acesso em 21 de Novembro de 2003.