Resumo

Redes de Petri são uma poderosa ferramenta para o desenvolvimento de sistemas com características concorrentes e assíncronas, permitindo que sistemas complexos, como linhas de produção, redes de comunicação, sistemas de computação, dentre outros, sejam modelados e analisados, de forma gráfica e numérica. O EZPetri, projeto desenvolvido por alunos do Departamento de Sistemas Computacionais (DSC) da Universidade de Pernambuco, é um *framework* cujo objetivo é facilitar a criação e edição de redes de Petri, bem como facilitar a integração entre as diferentes ferramentas de modelagem existentes atualmente. Para isso, o *framework* tira proveito da portabilidade oferecida pela linguagem Java, do ambiente de desenvolvimento Eclipse, e da linguagem de intercâmbio de redes de Petri, a *Petri Net Markup Language* (PNML). Este trabalho apresenta uma extensão ao *framework* EZPetri, integrando ao ambiente a ferramenta de modelagem, simulação e análise de redes de Petri estocásticas, TimeNET. Esta ferramenta oferece diversas vantagens em relação à outras ferramentas existentes, sendo a principal os algoritmos de análise numérica de redes de Petri estocásticas implementados na mesma. A integração entre a ferramenta TimeNET e o EZPetri foi implementada em três módulos: um compilador capaz de traduzir entre os formatos de representação utilizados pela ferramenta TimeNET e a linguagem PNML, formato de representação interno do EZPetri; um ambiente para criação e edição de redes de Petri estocásticas; e, por fim, uma interface para a realização de análises transientes. Como resultado obtido é apresentado um exemplo de uso do ambiente desenvolvido, demonstrando a modelagem e análise transiente de um modelo de filas M/M/1.

Abstract

Petri Nets are a powerful tool for development of systems with concurrent and asynchronous characteristics, allowing that complex systems, like productions lines, communication networks, computer systems, amongst others, to be modeled and analyzed, in both graphical and numerical ways. The EZPetri is a project developed by students from the Department of Computational Systems (DSC), at the Pernambuco State University, and consists of a framework whose objective is to facilitate the creation and edition of Petri nets, as well as to make the integration between the different existing modeling tools easier. To achieve this objective, the framework takes advantage from the portability offered by the Java programming language, from the Eclipse development environment, and from the Petri net interchange language, Petri Net Markup Language (PNML). This work presents an extension to the EZPetri framework, integrating into the environment the TimeNET tool, a modeling, simulation and analysis tool for Petri nets. Many advantages are offered by it in comparison to other existing tools, being the most important of them the numerical analysis algorithms implemented in it. The integration between the TimeNET tool and the EZPetri framework was implemented in three modules: a compiler capable to translate between the representation formats used by the TimeNET tool and the language PNML, which is used as internal representation format of EZPetri; a creation and edition environment for stochastic Petri nets; and, finally, an interface for the execution of transient analysis from EZPetri. As result we present a case study, demonstrating the modeling and transient analysis of an M/M/1 queuing model in the developed environment.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)
INA – *Integrated Net Analyzer*
PNK – *Petri Net Kernel*
XML – *eXtensible Markup Language*
PNML – *Petri Net Markup Language*
IDE – *Integrated Development Environment*
SPN – *Stochastic Petri Net*
DTMC – *Discrete Time Markov Chain*
GSPN – *Generalized Stochastic Petri Net*
DTD – *Document Type Definition*
PNTD – *Petri Net Type Definition*
GUI – *Graphical User Interface*
EPIC – *EZPetri INA Compiler*
EPPC – *EZPetri PEP Compiler*
PEP – *Programming Environment based on Petri Nets*
PCAF – *Power Consumption Framework*
AGNES – *A Generic Net Editing System*
HCPN – *Hierachical Coloured Petri Net*
FSPN – *Fluid Stochastic Petri Net*
eDSPN – *Extended Deterministic and Stochastic Petri Nets*
PRD – *Preemptive Repeat Different*
PRS – *Preemptive Resume*
EPTC – *EZPetri TimeNET Compiler*
GEF – *Graphical Editing Framework*
MVC – *Model View Controller*
API – *Application Programming Interface*
JDOM – *Java Document Object Model*
GPL – *Gnu Public License*
λ – Tempo de intervalo entre chegadas
μ – Tempo de serviço
CTMC – *Continuous Time Markov Chain*

Agradecimentos

- Primeiramente gostaria de agradecer a Deus, pela minha vida e por todas as incontáveis bênçãos que me tem concedido.
- Aos meus pais. Roosevelt Veloso e Maria Cândida, e a minha irmã Amélia, por todo amor e apoio que me tem dado durante toda a minha vida.
- À minha namorada Izaura, pelo companheirismo, amor e compreensão ao longo destes últimos 3 anos.
- Aos meus amigos, em especial a André Henrique, Arthur, Eduardo, Everes e Fred.
- À César Oliveira, pela colaboração e paciência no esclarecimento das dúvidas que surgiram durante o desenvolvimento do projeto.
- Aos colaboradores do projeto EZPetri.
- Ao meu orientador, Dr. Ricardo Massa Ferreira Lima, pelo apoio, incentivo e compreensão, que, com sua experiência, enriqueceu grandemente este trabalho.
- A todos os professores do Departamento de Sistemas Computacionais da Poli.

Capítulo 1

Introdução

O crescente avanço nas diversas áreas de desenvolvimento tecnológico e industrial tem ampliado cada vez mais a complexidade dos sistemas desenvolvidos. Este aumento de complexidade exige ferramentas e técnicas para descrever, modelar e analisar sistemas de forma concisa, objetiva e com o alto grau de precisão exigido por tais sistemas.

A abordagem utilizando métodos formais como técnica para modelagem e avaliação de sistemas vem sendo utilizada a muitos anos em pesquisas científicas e empresas. Dentre as diversas perspectivas de métodos formais existentes atualmente, a modelagem utilizando redes de Petri [1] é uma das que apresentam maior visibilidade no meio acadêmico e comercial [2]. Redes de Petri representam uma poderosa linguagem de especificação para modelagem de sistemas. Apresentam-se como uma ferramenta gráfica para descrição formal de sistemas que estão associados aos conceitos de concorrência, sincronização, exclusão mútua, conflitos, e disputa por recursos [2][3][4].

O formalismo proposto por C.A.Petri configura uma tentativa de descrever sistemas concorrentes em termos de relações de causa e efeito, sem considerar as características temporais, ao menos não explicitamente. A introdução dos conceitos de temporização se deu apenas mais tarde, com os trabalhos de C.Ramchandani [5] (1974), P.M. Merlin e D.J.Farber [6], J.Sifakis [7]. Estes trabalhos apresentavam noções determinísticas de temporização. [2]

Atualmente existem diversas ferramentas computacionais que auxiliam na modelagem e eventualmente análise, de redes de Petri, normalmente voltadas para tipos específicos de redes. Dentre as ferramentas que mais se destacam, podemos citar o JARP [8], CPNTools [9], o *Integrated NetAnalyser* (INA) [10], *Petri Net Kernel* (PNK) [11], bem como as ferramentas exploradas neste trabalho, o TimeNET [12][13] e o EZPetri [3].

O TimeNET surgiu a partir de uma iniciativa da *Technische Universität Berlin*, com a finalidade de realizar a modelagem e avaliação de desempenho utilizando redes de Petri estocásticas não markovianas [12]. A principal motivação de seu desenvolvimento foi a necessidade de fazer avaliações eficientes de redes de Petri temporizadas com tempos de disparo arbitrários.

A popularização do uso de ferramentas de modelagem levou a uma situação em que, na ausência de um padrão para representação de redes de Petri, cada ferramenta desenvolveu um formato próprio de representação dos seus modelos [3]. Isso dificulta a colaboração entre ferramentas, uma característica comum dos projetos que envolvem redes de Petri, pois muitas vezes o uso de apenas uma ferramenta não supre as necessidades exigidas pelo projeto. Esta

colaboração inter-ferramentas poderia trazer diversos benefícios [14], ampliando as possibilidades de simulação e validação dos modelos desenvolvidos.

Devido à necessidade de um padrão comum às ferramentas, foi iniciado um trabalho de desenvolvimento de um formato de intercâmbio de arquivos. Diversas propostas foram realizadas no *International Conference on Application and Theory of Petri nets*, em 2000, muitas das quais baseadas no formato *Extensible Markup Language* (XML) [15]. Dentre as propostas apresentadas, a que mereceu maior destaque foi a *Petri Net Markup Language* (PNML) [14], cuja estrutura também é baseada na XML. Originalmente, o PNML tinha intenção de apenas servir como modelo de representação interna da versão Java [16] do projeto Petri Net Kernel. Entretanto, foi observada uma tendência entre outros grupos desenvolverem um formato também baseado em XML, incentivando a divulgação do PNML como proposta para o formato de intercâmbio.

Na PNML, características comuns a todos os tipos de redes de Petri são representadas por nós XML previamente estabelecidos, e características específicas das redes podem ser representadas, tornando possível a representação de qualquer tipo de rede de Petri neste mesmo formato. Desde a sua criação, o PNML tem como princípios fundamentais a flexibilidade e a compatibilidade, bem como a necessidade de que a mesma permaneça não-ambígua.

Apesar de todos os esforços, o PNML ainda não é amplamente aceito como formato de representação em todas as ferramentas de modelagem de redes de Petri. Diversas iniciativas têm incentivado o seu uso. Como exemplo podemos citar o JARP e o projeto EZPetri.

O projeto EZPetri foi desenvolvido por alunos do Departamento de Sistemas Computacionais da Universidade de Pernambuco, e consiste em um conjunto de *plugins*, que associadas à *Integrated Development Environment* (IDE) Eclipse [17], oferecem ambiente de modelagem e colaboração inter-ferramentas.

Este trabalho tem como objetivo o desenvolvimento de um ambiente de modelagem e análise de redes de Petri temporizadas, usando como suporte a plataforma EZPetri / Eclipse. O ambiente é composto por um compilador capaz de converter entre o formato de arquivo de representação da rede utilizado pelo TimeNET e o formato PNML. Também compõe este trabalho uma extensão ao editor de redes *Place / Transition* existente no EZPetri para redes estocásticas e determinísticas, capaz de criar e editar redes de Petri com os parâmetros utilizados pelo TimeNET em seus mecanismos de análise e avaliação. E, por fim, a disponibilização de um ambiente de comunicação entre a plataforma Eclipse e os poderosos algoritmos de análise de redes de Petri estocásticas existentes no TimeNET. Ainda como contribuição, é apresentada uma extensão ao formato PNML para a devida representação de redes de Petri estocásticas e uma descrição dos elementos e atributos existentes no formato NET, proprietário da ferramenta TimeNET.

Esta monografia é estruturada da seguinte forma: o Capítulo 2 apresenta alguns conceitos fundamentais relacionados a redes de Petri, bem como as extensões para redes de Petri temporizadas e estocásticas. No Capítulo 3 serão apresentadas as ferramentas com as quais o projeto aqui apresentado relaciona-se, bem como a descrição do formato de arquivo NET. O Capítulo 4 descreve as extensões desenvolvidas no projeto para a *Petri Net Markup Language* para a devida representação das redes de Petri estocásticas neste formato. No Capítulo 5 será apresentado o trabalho de implementação desenvolvido, descrevendo a arquitetura utilizada e interfaces com o usuário para o compilador, o editor gráfico de redes de Petri estocásticas e, por fim, o ambiente de análise transiente de redes de Petri. No Capítulo 6 será apresentado um estudo de caso, no qual um sistema de filas é modelado e analisado utilizando redes de Petri no ambiente desenvolvido. O Capítulo 7 relata as considerações finais, bem como sugere alguns trabalhos que podem ser desenvolvidos no futuro.

Capítulo 2

Redes de Petri

O formalismo redes de Petri foi introduzido na Alemanha, no ano de 1962, por Carl Adam Petri, cuja tese de doutorado, *Kommunikation mit Automaten* [18] ("Comunicação com Autômatos"), tratava da descrição de sistemas concorrentes em termos de relações de causa e efeito. Seu trabalho atraiu a atenção de outros pesquisadores, cujos trabalhos vieram a desenvolver muito da teoria proposta inicialmente.

As redes de Petri, por apresentarem noção de estados e de regras para mudanças dos mesmos, possibilitam a representação de sistemas reais de forma bastante satisfatória. Também podem ser vistas como uma técnica de especificação formal com métodos robustos para análise qualitativa e quantitativa [3] de tais sistemas. Desde o seu surgimento, têm se apresentado uma metodologia versátil o bastante para ser utilizada em aplicações que envolvem desde computação distribuída, como também na engenharia elétrica, eletrônica, protocolos de comunicação, sistemas de controle, ou qualquer outra aplicação que envolva um fluxo de trabalho.

Segundo Heuser [4], as primeiras aplicações de redes de Petri aconteceram em 1968, no projeto norte-americano *Information System Theory*, da *Applied Data Research, Inc.*. Este trabalho ressaltou como redes de Petri poderiam ser aplicadas na análise e na modelagem de sistemas com componentes concorrentes.

Atualmente, as redes de Petri, ou simplesmente PN, constituem uma metodologia consolidada para descrição formal de sistemas que estão associados aos conceitos de concorrência, sincronização, exclusão mútua, conflitos, e disputa por recursos [2].

2.1 Introdução conceitual às redes de Petri

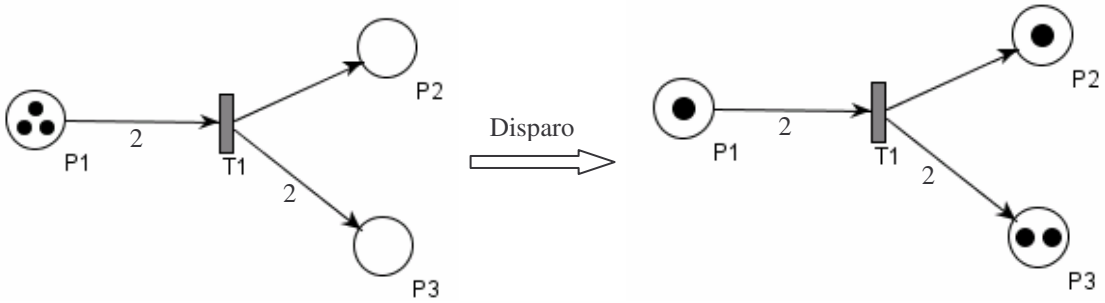
Uma rede de Petri é definida estruturalmente como uma composição de alguns elementos, como descritos a seguir:

- **Lugares**: são utilizados para modelar componentes passivos dos sistemas. O conceito de lugar está relacionado ao conceito de variáveis de estado. Podem conter ou não marcações, ou *tokens*, geralmente associadas a um determinado recurso do sistema. Lugares são representados graficamente como uma circunferência.

- **Transições ou ações**: são usadas para modelar os eventos que ocorrem em um sistema, componentes ativos dos sistemas. As transições são capazes de mudar o estado do sistema, e são representadas graficamente como uma barra ou retângulo.
- **Arcos ou relações de fluxo**: utilizados para especificar como se dá a transformação de um estado em outro pela ocorrência das ações no sistema. Podem ter um valor associado, chamado "peso", que, quando omitido, assume o valor 1. Possuem como representação gráfica uma seta.

O comportamento dinâmico de uma rede de Petri é dado pela "regra de disparo". Uma transição pode ser disparada, ou seja, permitir a ocorrência de um evento, se para todos os lugares de entrada (lugares que possuem arcos partindo deles em direção à transição) conectados à transição, a quantidade de *tokens* existentes nos lugares de entrada é maior ou igual ao peso dos arcos que ligam os lugares à transição. Quando esta condição é atendida, a transição é considerada como "habilitada".

O peso dos arcos associados determina o numero de *tokens* exigidos pela transição para ser habilitada, como também indicam a quantidade de *tokens* gerados nos lugares de saída após o disparo da transição. Esta atividade de transferência de *tokens* é considerada como uma ação atômica[1] e pode ser visualizada na Figura 1(a) e na Figura 1(b):



(a) (b)

Figura 1. Dinâmica da habilitação e disparo de uma transição

Podemos observar na Figura 1(a) que a transição T1 encontra-se habilitada, pois a quantidade de *tokens* no lugar de entrada P1 é maior do que o peso que liga P1 à T1. Ao disparar a transição T1, dois *tokens* (devido ao peso do arco ser 2) são retirados do lugar P1 e novos *tokens* são criados nos lugares P2 e P3. A quantidade de *tokens* gerados nos lugares de saída também é determinada pelo peso dos arcos de ligação entre estes e a transição disparada, como podemos visualizar na Figura 1(b).

Um exemplo do uso de redes de Petri para modelagem de sistemas paralelos é a linha de produção de canetas ilustrada na Figura 2. Nela, a transição t_0 está representando o início das atividades, enquanto o lugar p_i representa a disponibilidade para realização da montagem, e o lugar p_0 representa o depósito de matéria prima. A transição t_0, ao ser disparada, consome um *token* do lugar p_0, e dois novos são criados nos lugares de saída p_1 e p_2. Estes lugares indicam as condições necessárias para a fabricação da carga e do invólucro, respectivamente. Estas atividades podem ser executadas paralela e isoladamente. A transição t_3 indica a atividade de montagem da caneta, que só pode ser feita quando as atividades de fabricação da carga e do invólucro tenham sido concluídas. Quando disparada, um *token* é criado no lugar p_i, possibilitando que uma nova caneta seja produzida, caso haja disponibilidade de matéria prima.

[1] Uma ação atômica é uma ação indivisível em sub-etapas.

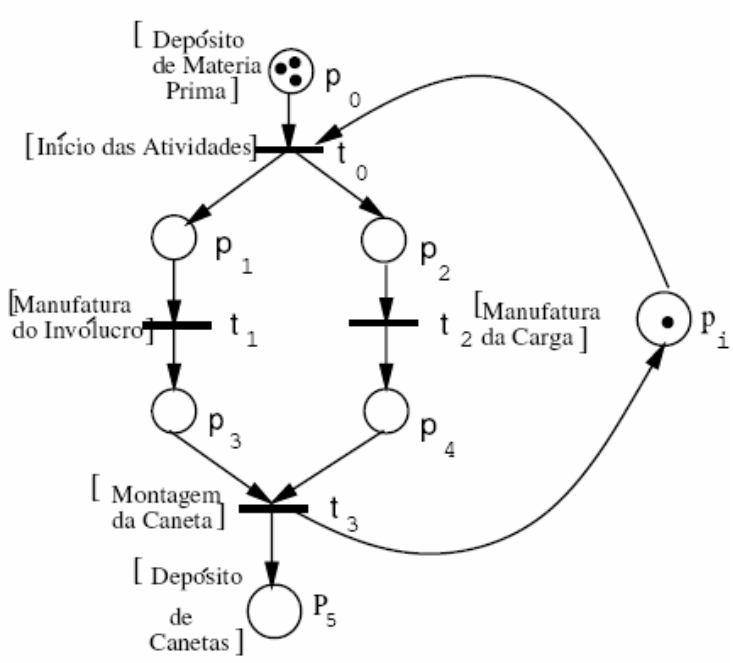


Figura 2. Atividades paralelas em processos de linha de produção

A representação gráfica de redes de Petri é bastante útil, pois permite a visualização dos processos e a comunicação entre eles. Uma outra forma de representar uma rede de Petri é através de uma 5-upla (P,T,I,O,K) [1], onde P representa o conjunto de lugares, T o conjunto de transições, I e O as matrizes de mapeamento de lugares de entrada para transições e de transições para lugares de saída, respectivamente. K representa o mapeamento entre os lugares e suas respectivas capacidades (número máximo de *tokens* que podem ser alocados em um lugar).

A Matriz de Incidência de uma rede de Petri pode ser definida como a matriz C = O – I. Esta matriz informa a incidência de arcos de entrada e saída em cada transição da rede, e a partir dela, normalmente é possível determinar toda a estrutura da rede. Isto se dá por ela representar de forma implícita a relação entre lugares e transições. A condição para que a matriz de incidência represente a rede por completo é que a rede em questão não possua *self-loops*, ou simplesmente, seja pura. Uma rede de Petri é considerada pura se nenhuma transição possui como origem e destino um mesmo lugar. Uma rede impura pode ser transformada em uma rede pura através da adição de pares *dummy*. Um par *dummy* é um par formado por um lugar e uma transição, e que são adicionados na rede a fim de refinar um *self-loop*. Um exemplo da transformação de uma rede impura numa rede pura é dado na Figura 3.

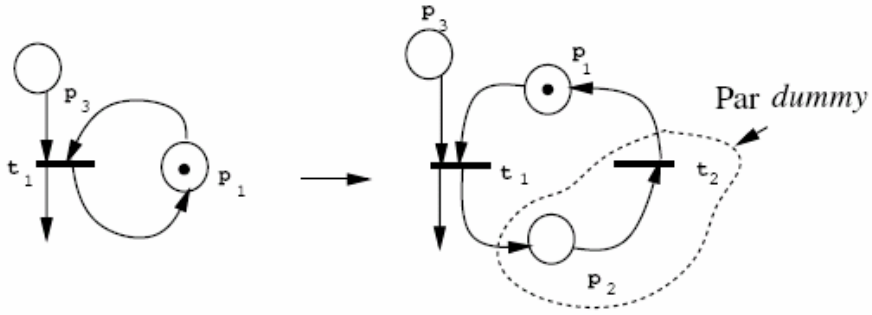


Figura 3. Transformação de uma rede impura em uma rede pura

2.1.1 Propriedades de Redes de Petri

A utilização de redes de Petri não se restringe apenas à modelagem de sistemas concorrentes, com problemas de sincronização e paralelismo. Uma série de métodos foram desenvolvidos com a finalidade de possibilitar a análise de uma grande quantidade de propriedades em sistemas [1]. Estas, por sua vez, dividem-se principalmente em: comportamentais e estruturais. As primeiras, são dependentes das marcações (*tokens*) da rede, enquanto, as segundas, não dependem das marcações. São consideradas propriedades comportamentais: Alcançabilidade, Limitação, Segurança, *Liveness*, Cobertura, Persistência, Reversibilidade e Justiça. As propriedades estruturais são: Limitação estrutural, Conservação, Repetitividade e Consistência. [19]

2.1.2 Classificação das Redes de Petri

As redes de Petri são aplicadas para modelar formalmente uma larga gama de problemas, e com a sua decorrente evolução, desenvolveu diversos níveis de abstração [1], que consistem basicamente de:

- **Redes de Petri Ordinárias;**
  - PN Binárias ou Condição-evento
  - PN *Place / Transition*
    - PN Ponderadas
    - PN Não-Ponderadas
- **Redes de Petri Não-Ordinárias ou de Alto Nível.**
  - PN Predicado-Evento
  - PN Coloridas
  - PN Hierárquicas

Nas redes de Petri Ordinárias, o tipo de seus *tokens* é um número inteiro não negativo, e estes tipos caracterizam a rede. Fazem parte da classe ordinária as redes de Petri Binárias (ou Condição-Evento) [19] e as redes de Petri *Place-Transition* [1]. As redes de Petri Binárias representam as redes mais simples de todas as classes, nas quais lugares só podem conter no máximo um *token* e todos os arcos tem peso unitário. As redes de Petri *Place-Transition* podem ser ponderadas (os pesos dos seus arcos podem assumir valores superiores a 1) ou não-ponderadas (os pesos de seus arcos possuem valor unitário).

Nas redes de Petri de Alto Nível, os *tokens* não são mais elementos do tipo inteiro positivo, e podem ser diferenciados com parâmetros de cor, o que permite a individualização dos mesmos. Há também a possibilidade de representar o *token* não apenas por um único objeto e sim por um conjunto de objetos.

Além desta classificação ainda existem outras extensões para o formalismo de redes de Petri, são elas: Redes de Petri com Arco Inibidor e as Redes de Petri Temporizadas. As redes de Petri com arco inibidor têm como principal característica possibilitar o teste a zero em lugares com capacidades ilimitadas. Esta extensão possui um arco diferenciado dos demais, o arco inibidor, que inibe o disparo de uma transição caso o lugar, que é pré-condição para a transição, possua algum *token*. Esta característica pode ser vista como inversa à condição de habilitação dada pelo arco comum, ou seja, a transição com um arco inibidor de entrada estará habilitada se o lugar de entrada desta transição não detiver nenhum *token*. É representado como um arco comum, com a diferença que possui uma circunferência em sua extremidade, em lugar de uma seta.

As redes de Petri Temporizadas são de especial interesse para este projeto, e por isso são explicadas com maior nível de detalhes na próxima seção.

2.2 Redes de Petri Temporizadas

O modelo clássico das PN não inclui a noção de tempo. O conceito de tempo em redes de Petri tornou-se fundamental quando cresceu o interesse na modelagem de sistemas reais, cuja principal preocupação está relacionada à eficiência. A fim de que o formalismo das redes de Petri viesse ser utilizado para a análise quantitativa, de desempenho e confiabilidade de sistemas, o conceito de tempo foi introduzido, em especial, pelos trabalhos de C.Ramchandani [5], P.M. Merlin e D.J.Farber [6], J.Sifakis [7]. Estes trabalhos apresentavam noções determinísticas de temporização, mas outros surgiram mais tarde introduzindo conceitos de variáveis randômicas (levando a classe das redes de Petri estocásticas – SPN) [2].

O tempo em uma rede de Petri pode ser especificado basicamente de três maneiras: tempo associado aos lugares, tempo associado às marcas ou tempo associado às transições. Em [1] há uma descrição detalhada das três abordagens. Este trabalho mantém um maior enfoque nas redes de Petri com tempos associados a transições.

O disparo de uma transição em um modelo de rede de Petri corresponde ao evento cujo resultado muda o estado do sistema real. Esta mudança de estado pode estar relacionada a duas causas: pode resultar da verificação de alguma condição lógica no sistema, ou pode ser resultado da conclusão de alguma atividade. O segundo caso indica uma abordagem na qual uma transição pode ser utilizada para modelar atividades, pois os tempos de atraso das transições estariam associados à execução de atividades, enquanto o disparo da mesma estaria associado à conclusão da atividade. Transições com tempos associados (representando atividades) são chamadas transições temporizadas no formalismo de redes de Petri temporizadas.

Entretanto, nas redes de Petri temporizadas, existem situações em que transições não precisam ter um tempo de atraso associado, e descrevem a lógica ou o algoritmo de evolução de estado do modelo causados por eventos. Sendo assim, tornou-se necessária a definição de um outro tipo de transição, chamada transição imediata. Estas transições não possuem um atraso associado, e disparam assim que se tornam habilitadas, e, por isso, têm precedência ante o disparo de transições temporizadas.

A introdução de conceitos de temporização nas redes de Petri não deve reduzir as capacidades de modelagem com relação ao modelo não temporizado. Para isso, algumas considerações devem ser feitas, especialmente no que se refere aos aspectos de paralelismo e conflito no disparo de transições. Considere a redes de Petri mostrada na Figura 4.

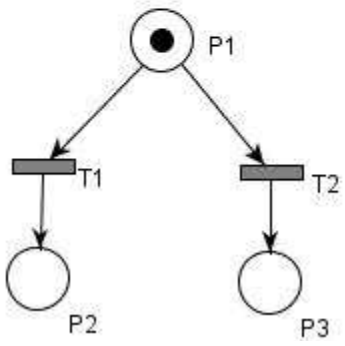


Figura 4. Conflito entre transições

Considerando as transições T1 e T2 como imediatas, em uma rede temporizada, estando ambas habilitadas no mesmo instante, duas situações podem ocorrer: a primeira é fazer a escolha da transição a ser disparada de forma determinística, introduzindo o conceito de prioridade, que em termos gerais, associa valores correspondentes à prioridade de disparo das transições, disparando a que apresentar o maior valor. A segunda abordagem consiste na associação de uma função de distribuição discreta de probabilidade a um conjunto de transições conflitantes. Neste caso, os conflitos entre transições são resolvidos randomicamente.

Já no caso das transições T1 e T2 serem temporizadas, estando ambas habilitadas em um determinado instante, a transição que tiver um menor tempo associado vence a "disputa" pelo *token*. O problema, porém, consiste em como gerenciar o tempo restante na transição que foi desabilitada, bem como nas demais transições temporizadas da rede que não estavam envolvidas no conflito. Este problema está diretamente relacionado à "política de memória" das transições, e apresenta basicamente dois mecanismos:

- Continuar (*Continue*): O temporizador associado à transição mantém o valor atual e continuará sua contagem a partir do ponto onde parou.
- Reiniciar (*Restart*): o temporizador é reiniciado, assumindo um novo valor.

Subdivisões destas políticas existem, com diferentes abordagens para manter a memória do sistema, entre as principais podemos destacar:

- *Resampling*: A cada disparo de toda e qualquer transição do modelo, todos os temporizadores existentes são reiniciados (*Restart*), e, sendo assim, não há memória. O temporizador de cada transição será reiniciado sempre que a transição tornar-se habilitada;
- *Enabling memory*: A cada disparo de transição, os temporizadores das transições que estavam desabilitadas são reiniciados, enquanto que os temporizadores das transições que estavam habilitadas mantêm o valor atual (*Continue*). Assim que estas transições tornarem-se habilitadas novamente, seus temporizadores continuam do ponto onde foram parados. Uma variável (*enabling memory variable*) mede o tempo que a transição passou habilitada desde o último instante de tempo em que ela tornou-se habilitada;
- *Age memory*: Após cada disparo, os temporizadores de todas as transições mantêm seus valores atuais (*Continue*). Uma memória do passado é mantida por uma variável (*age memory variable*) associada a cada transição temporizada. Esta variável contabiliza o tempo gasto na atividade modelada pela transição, medindo o tempo cumulativo de habilitação, desde o instante do seu último disparo.

Estas diferentes políticas de memória podem ser utilizadas para modelar os mais diversos sistemas, dependendo do nível de realismo desejado pelo projetista. O grau de habilitação determina a quantidade de vezes em que uma transição temporizada pode ser disparada novamente após o seu temporizador ter sido zerado. Quando o grau de habilitação for maior do que um e vários *tokens* estão presentes em todos os lugares de entrada, três diferentes semânticas podem ser adotadas:

- Semântica *Single-Server:* Nesta abordagem, o temporizador é iniciado na primeira vez em que a transição torna-se habilitada, e é reiniciado em todos os instantes enquanto a transição estiver habilitada, ou seja, os *tokens* são processados de forma serial.

- Semântica *Infinite-Server:* Assim que a transição torna-se habilitada, todos os conjuntos de *tokens* de habilitação presentes nos lugares de entrada são consumidos instantaneamente, e seus tempos de processamento são computados em paralelo.
- Semântica *Multiple-Server*: Nessa abordagem os *tokens* de habilitação também são consumidos assim que são formados nos lugares de entrada, porém são processados com um grau K de paralelismo.

Um exemplo das três diferentes semânticas aplicadas a um sistema de computação *timesharing* com um número fixo de usuários (Figura 5). Um sistema *timesharing* é um sistema no qual uma única CPU atende às requisições de diversos terminais. Tais sistemas podem ser modelados como um sistema de fila única, onde as requisições dos terminais são atendidas por ordem de chegada. O intervalo de tempo entre o instante em que um terminal tem sua requisição atendida e o tempo em que é feita uma nova requisição é chamado de *think time*.

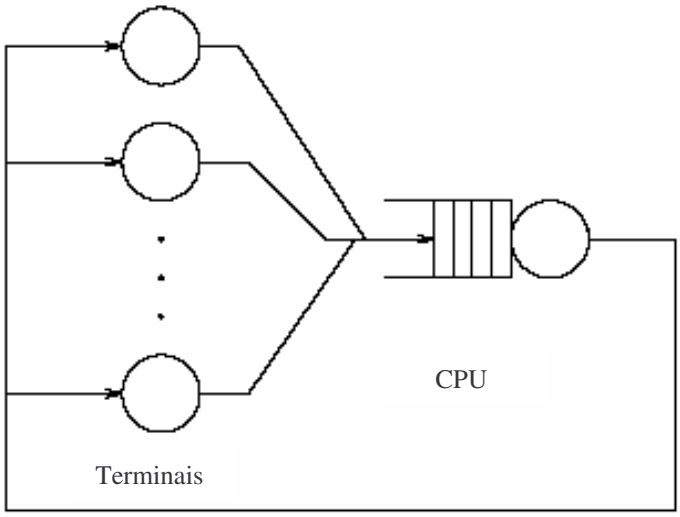


Figura 5. Modelo de filas de um sistema *timesharing*

O sistema de filas demonstrado na Figura 5 pode ser modelado utilizando redes de Petri temporizadas. Esta abordagem é apresentada na Figura 6. Nela, a transição temporizada T1, com semântica *infinite-server,* representa o tempo de processamento (*think time*) dos terminais, uma vez que tal processamento ocorre de forma paralela. A transição T2, com semântica *single-server,* representa o tempo de processamento da CPU, pois num modelo de fila única, as requisições são atendidas de forma serial, por ordem de chegada.

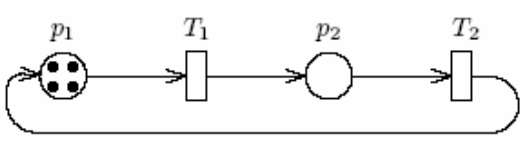


Figura 6. Rede de Petri Temporizada representando um sistema *timesharing*

2.3 Redes de Petri Estocásticas

A introdução de conceitos de temporização ao formalismo de redes de Petri usual amplia a gama de problemas que podem ser modelados. Entretanto, a introdução de noções de variáveis aleatórias pode enriquecer ainda mais o realismo do sistema, transformando o não-determinismo das redes não temporizadas em probabilidades no modelo temporizado. Para isso, a teoria de

processos estocásticos pode ser aplicada para extração de medidas de desempenho do modelo gerado.

A avaliação de desempenho de sistemas complexos, como redes de comunicação, arquiteturas computacionais, etc. utiliza como ferramenta a teoria das probabilidades. Em tais sistemas, normalmente os valores para determinados componentes não podem ser determinados com precisão, mas podem ser estimados, como, por exemplo, o número de peças que podem ser produzidas em uma fábrica, ou o tempo de duração de um determinado elemento do sistema. Também se sabe que em sistemas complexos, as partes possuem uma relação de dependência umas com as outras, e este tipo de dependência também pode ser conhecido.

Processos estocásticos são comumente utilizados para modelar e computar resultados e métricas de desempenho para sistemas complexos. Um processo estocástico pode ser visto como um conjunto de variáveis randômicas que compartilham um espaço de valores, normalmente chamado de "espaço de estados do processo". Geralmente o tempo é utilizado como parâmetro no processo, e pode ser indexado por um parâmetro discreto ou contínuo. Caso o parâmetro de tempo seja considerado contínuo, o processo é dito "processo estocástico de tempo contínuo". No caso de o parâmetro de tempo ter variação constante, o processo é então chamado de "processo estocástico de tempo discreto".

Um processo estocástico com um espaço discreto de estados é conhecido como uma "cadeia estocástica", e quando esta não possui memória, ou seja, o comportamento futuro do sistema independe dos estados passados, mas apenas do estado atual, esta é dita uma "Cadeia de Markov" ou *Discrete Time Markov Chain* (DTMC). Normalmente uma cadeia de Markov possui uma representação gráfica, na qual círculos representam os estados possíveis do sistema, enquanto arcos representam as transições entre dois estados.

Uma rede de Petri estocástica (Stochastic Petri Net – SPN) também se apresenta como um modelo baseado em estados, porém com um nível descritivo maior do que com cadeias de Markov. As SPNs adicionam tempo ao formalismo de redes de Petri, com a diferença de que os atrasos associados às transições são dados probabilisticamente e distribuídos exponencialmente. A característica de ausência de memória da distribuição exponencial implica na total imprevisibilidade no disparo de transições conflitantes, o que torna o modelo mais fiel ao sistema real. A análise probabilística de uma SPN pode ser realizada tanto em um determinado ponto no tempo a partir do seu estado inicial, chamada análise transiente, ou após um longo tempo de "funcionamento", análise esta conhecida como análise estacionária.

Quando transições imediatas também são incorporadas ao modelo de redes de Petri estocásticas, este passa a pertencer a uma outra classe de SPNs, conhecida como Redes de Petri Estocásticas Generalizadas, ou simplesmente GSPNs (*Generalized Stochastic Petri N*ets) [2]. Esta classe de redes permite descrições mais concisas de sistemas do que utilizando cadeias de Markov.

Capítulo 3

Ferramentas

Neste capítulo serão apresentadas algumas ferramentas utilizadas por este trabalho. Na Seção 3.1 oferece uma visão geral da *Petri Net Markup Language*, PNML, a Seção 3.2 apresenta o *framework* EZPetri, e, por fim, a Seção 3.3 discorre a respeito da ferramenta de modelagem, análise e simulação de redes de Petri estocásticas TimeNET.

3.1 Petri Net Markup Language

A popularização do uso de ferramentas de modelagem de redes de Petri levou a uma situação em que, na ausência de um padrão para representação das redes, cada ferramenta desenvolveu um formato próprio de representação dos seus modelos [3]. Isso torna impossível a importação de modelos entre ferramentas, o que traria diversos benefícios [14] através do reuso de modelos. Por exemplo, uma pessoa poderia desenvolver a rede em determinada ferramenta, realizar análises de desempenho em outra, e fazer verificações estruturais do modelo em uma terceira.

Sendo assim, foi iniciado o desenvolvimento de um formato de intercâmbio de arquivos. Na conferência sobre Aplicação e Teoria das Redes de Petri (*International Conference on Application and Theory of Petri Nets*), ocorrida em Aarhus, Dinamarca, no ano de 2000, diversas propostas baseadas em XML foram propostas. Dentre elas, a que mereceu maior destaque foi a *Petri Net Markup Language* (PNML) [14], cuja estrutura é baseada na *Extensible Markup Language* (XML) [15]. Diversos outros eventos ocorreram desde então, sempre focados na criação de um formato de representação de redes de Petri. Entretanto, até a atual data não existe um padrão que seja aceito em toda a comunidade de usuários de redes de Petri.

O PNML, embora ainda não seja aceito por todas as ferramentas existentes, já é adotado por algumas, como o EZPetri e o JARP. Além disso, o formato é aberto para futuras extensões e também suporta diferentes tipos de redes de Petri. Na PNML, características comuns a todos os tipos de redes de Petri são representadas por elementos XML previamente estabelecidos.

O projeto do PNML segue estritamente três princípios básicos [14]. *Flexibilidade*, ou seja, o formato deve ser capaz de representar qualquer tipo de rede de Petri, com suas extensões e características especificas. No PNML informações adicionais podem ser anexadas à rede propriamente dita, ou apenas aos elementos relacionados àquela informação. *Não-Ambiguidade,* que significa afirmar que a rede de Petri representada e seu respectivo tipo podem ser determinados unicamente pela sua representação em PNML. *Compatibilidade*, que garante que o máximo de informação possível seja compartilhado entre os diferentes tipos de rede de Petri

existentes. Um outro princípio fundamental definido em [20] é a *Legibilidade*. O formato deve permitir a leitura e edição das redes em qualquer editor de textos.

O PNML permite que, através de programas tradutores (compiladores), a rede possa ser transferida entre as diversas ferramentas, como ilustrado na Figura 7, na qual é representado o processo de comunicação entre redes mantidas nos formatos de duas diferentes ferramentas. De acordo com a proposta, o usuário final não precisaria manipular diretamente documentos PNML, pois tal tarefa é delegada aos compiladores entre os formatos.

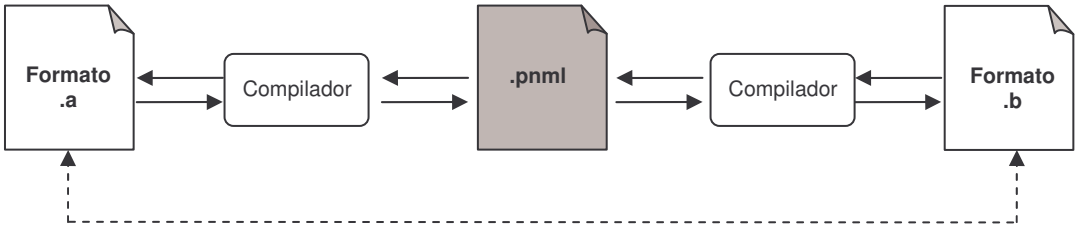


Figura 7. Comunicação entre diferentes formatos de representação

O PNML disponibiliza mecanismos de validação de documentos nesse formato utilizando DTDs (*Document Type Definitions*) associados a verificadores sintáticos de documentos XML. No PNML, estas DTDs são chamadas de PNTD (*Petri Net Type Definition*) e são definidos através de XML *Schemas* na linguagem *RELAX NG* [21]. Nela, é especificada a estrutura necessária para representar os diversos tipos de redes de Petri existentes. Apesar da orientação de que a cada nova extensão tenha uma PNTD correspondente, geralmente esta instrução não é seguida devido à relativa complexidade para gerar as gramáticas *RELAX NG*. Neste trabalho optou-se por utilizar o formato das redes *Place / Transition*, representando as características específicas de redes estocásticas através de elementos *toolspecific*, que serão detalhados mais adiante.

3.1.1 Estrutura de um documento PNML

Todo documento PNML possui o elemento `<pnml>` na raiz da estrutura XML em que o documento está estruturado. Esta *tag* indica que o arquivo contém informações no formato PNML. Um mesmo documento pode conter uma ou mais redes, cada uma delas envolvida pela *tag* `<net>`. Todos os elementos em um documento PNML devem possuir um atributo *id*, que pode ser utilizado para referenciar de forma única aquele objeto. Uma rede de Petri e sua respectiva representação em PNML podem ser vistas na Figura 8 e 9.

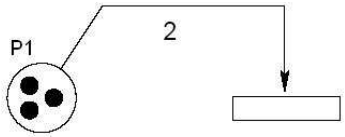


Figura 8. Representação gráfica de uma rede de Petri

```
<pnml>
   <net id="1" type="PTNet">
   <name>
      <text>Exemplo</text>
   </name>
   <place id="2">
      <graphics>
         <position x="15" y="15"/>
      </graphics>
      <name>
         <value>P1</value>
         <graphics>
            <offset x="-5" y="-4"/>
         </graphics>
      </name>
      <initialMarking>3</initialMarking>
   </place>
   <transition id="3">
      <name>
         <value>P1</value>
         <graphics>
            <offset x="0" y="-14"/>
         </graphics>
      </name>
      <graphics>
         <position x="60" y="20"/>
      </graphics>
      <toolspecific tool="tool" version="1.0">
         <type>Immediate</type>
      </toolspecific>
   </transition>
   <arc id="4" source="2" target="3">
      <graphics>
         <position x="30" y="5"/>
         <position x="60" y="5"/>
      </graphics>
      <inscription>
         <value>2</value>
         <graphics>
            <offset x="15" y="-2"/>
         </graphics>
      </inscription>
   </arc>
   </net>
</pnml>
```

Figura 9. Documento PNML representando a rede apresentada na Figura 8

A seguir será dada uma breve descrição sobre cada um dos elementos que podem ser utilizados para compor outros elementos presentes na rede modelada, são eles:

- **<**`name`**>:** esta *tag* é utilizada para adicionar um rótulo ao elemento ao qual ela esteja anexada, podendo assim ser utilizada para adicionar um maior significado ao objeto. Possui como elementos filhos `<value>`, que contém o valor a ser exibido, e o elemento `<graphics>`;
- **<**`graphics`**>:** Cada elemento possui alguma informação gráfica a ele associada, que pode ser a sua localização no caso de lugares e transições, ou até mesmo uma lista de posições que definem os pontos intermediários de um arco. A *tag* `<graphics>` pode ter como filhos os elementos `<position>` ou `<offset>`. A primeira deve ser utilizada quando se deseja determinar a posição absoluta do elemento, enquanto a segunda é

utilizada para indicar a posição relativa em relação ao elemento acima na hierarquia. Ambos possuem os atributos *x* e *y*, que indicam as coordenadas cartesianas do objeto.

Lugares e Transições
Uma rede pode conter zero ou mais lugares, tendo cada um deles suas informações encapsuladas em PNML pela *tag* `<place>`. Esta possui um atributo *id*, que deve ser único no modelo definido. Os elementos filhos desta *tag* são `<name>`, `<graphics>` e `<initialMarking>`. Os dois primeiros já foram explicados anteriormente, e a *tag* `<initialMarking>` contém a quantidade inicial de *tokens* presentes no lugar em questão. As transições possuem representação semelhante aos lugares em PNML, sendo representadas pela *tag* `<transition>`. Os elementos filhos desta *tag* são apenas `<name>` e `<graphics>`.

Arcos
A representação mais elementar de um arco em PNML pode ser obtida com apenas um elemento `<arc>`, contento três atributos: *id*, *source* e *target*. O atributo *id* é necessário, pois a PNML exige que todos os elementos possuam um identificador único, o que permite que existam dois ou mais arcos entre os mesmos elementos. As informações gráficas do arco são indicadas por uma lista de pontos, que correspondem aos pontos intermediários do arco. O elemento `<inscription>` representa as informações relativas ao rótulo do arco, com o `<offset>` indicando a sua posição em relação ao ponto de referência do arco.

Tag `toolspecific`
Devido à necessidade de armazenar informações específicas e restritas a algumas ferramentas, foi criada a *tag* `toolspecific`. Este elemento pode se situar hierarquicamente abaixo de qualquer elemento na estrutura do documento PNML, tendo o seu formato dependente da ferramenta. A ferramenta que gerou a informação, bem como sua versão, pode ser mantida nos atributos *tool* e *version*, respectivamente. As informações contidas em elementos `toolspecific` podem facilmente ser ignoradas por outras ferramentas, de modo que estas não devem interferir na estrutura geral do modelo. De maneira análoga, novos elementos `toolspecific` podem ser adicionados ao modelo com informações de outras ferramentas, sem comprometer a topologia da rede.

3.2 O Projeto EZPetri

O EZPetri consiste em um conjunto extensível de ferramentas que faz uso da tecnologia de *plug-ins* do Eclipse [17] a fim de criar uma plataforma modular para modelagem de redes de Petri, utilizando PNML como formato de representação interna do modelo. Com isso, permite a integração de ferramentas para redes de Petri existentes, bem como a implementação de novas funcionalidades.

O Eclipse é uma plataforma de integração de código aberto, lançada no ano de 2001 pela IBM e outras organizações. Fornece um ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE) para a linguagem Java, sendo ele mesmo desenvolvido nesta linguagem, podendo ser executado em diversas plataformas e sistemas operacionais. Entretanto, o Eclipse não está restrito apenas à linguagem Java, possuindo facilidades para a extensão da plataforma para integração de diversas ferramentas e linguagens, como C, C++, Pascal, PHP, etc. A ferramenta foi desenvolvida focada em apresentar um alto grau de extensibilidade, e, para isso,

utiliza a tecnologia de *plug-ins*. *Plug-ins* são unidades funcionais que podem ser facilmente acopladas e desacopladas ao sistema, a Figura 10 ilustra esta modularidade.

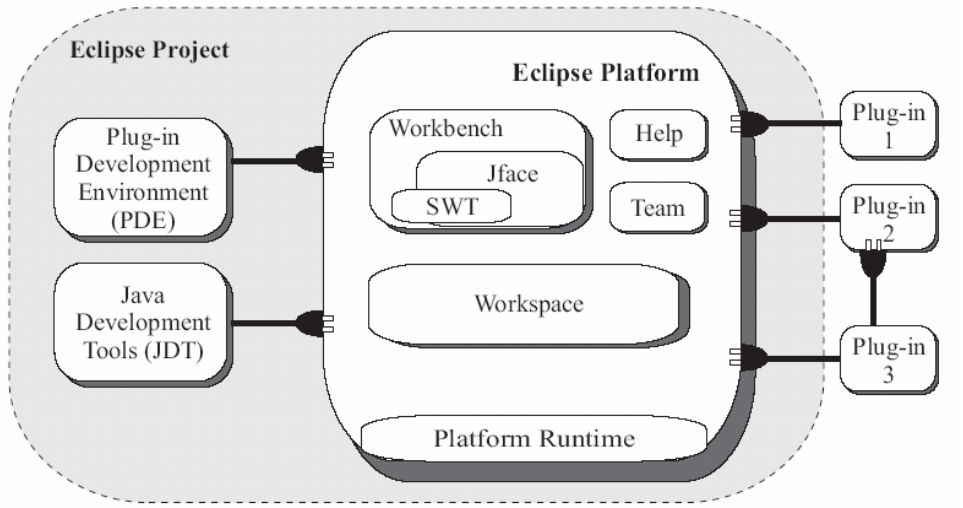


Figura 10. A arquitetura modular da plataforma Eclipse

Observando a Figura 10 podemos identificar na parte central da figura os componentes do núcleo do Eclipse, compartilhados pelos *plug-ins*. São eles o *Workbench*, o *Workspace* e o *Help*. O *plug-in* JDT possui uma série de outros *plug-ins* que disponibilizam os diversos componentes da IDE, como por exemplo editores, *views*, perspectivas, etc. O PDE fornece um ambiente integrado para o desenvolvimento de novos *plug-ins* para a própria plataforma Eclipse, que oferece inúmeras facilidades ao desenvolvedor. Demonstrando a grande capacidade de extensão da ferramenta, podemos observar no lado direito da figura uma série de *plug-ins*, associados não somente à plataforma Eclipse, mas também estendendo as funcionalidades de outros *plug-ins.*

A interface gráfica com o usuário (*Graphical User Interface* – GUI) do Eclipse está baseada no conceito de perspectivas (*perspectives*). Uma determinada perspectiva define um conjunto de visões (*views*) e editores (*editors*), voltados para os requisitos desejados pelo autor da extensão. As *views* são geralmente utilizadas para navegar por recursos, ou para alterar valores de determinado recurso. Os *editors* são usados para criar, editar e exibir conteúdos de interesse do usuário.

O projeto EZPetri faz uso da tecnologia de *plug-ins* do Eclipse, com a finalidade de prover um ambiente de suporte a edição e a simulação de redes de Petri, como também possibilitar a importação e exportação das redes de Petri modeladas entre diferentes ferramentas. O projeto também foi desenvolvido de forma modular, e tem como principal característica, assim como o próprio Eclipse, a capacidade de absorver novas funcionalidades.

Atualmente, possui integrados em seu ambiente o compilador *EZPetri PNML INA Compiler* (EPIC), voltado para a ferramenta de análise de redes de Petri *Integrated Net Analyzer* (INA), e também o compilador *EZPetri PNML PEP Compiler* (EPPC) para a ferramenta de modelagem, compilação, simulação e verificação de redes de Petri, o *Programming Environment based on Petri Nets* (PEP) [22]. Outros trabalhos vêm sendo integrados à plataforma, como o *Power Consumption Analysis Framework* (PCAF), que consiste em um ambiente para a análise do consumo de energia em componentes de hardware utilizando redes de Petri coloridas como linguagem de descrição formal de tais sistemas.

O EZPetri utiliza o formato PNML para representar internamente um modelo de rede Petri no sistema. Assim, une-se a facilidade de integração fornecida pelo Eclipse com o formato de intercâmbio de redes PNML. O projeto tem como principal objetivo fornecer à comunidade de

usuários de redes de Petri um ambiente extensível, que contribui para reduzir os problemas de compatibilidade entre as inúmeras ferramentas, tipos de rede de Petri e formato de arquivos existentes entre os usuários deste formalismo.

3.3 A Ferramenta TimeNET

O TimeNET é o resultado da união de um conjunto de ferramentas, com a finalidade de oferecer o suporte para a criação, edição, simulação e análise de redes de Petri estocásticas. É na verdade uma evolução da ferramenta DSPNexpress, cuja influência maior se deu por parte da ferramenta GreatSPN, mantida pela *Technische Univertät (TU) Dortmund*, da Alemanha. A ferramenta DSPNexpress já vinha sendo desenvolvida na *TU Berlin* desde o ano de 1991, e desde então já possuía poderosos algoritmos de análise.

O TimeNET 3.0, versão considerada por este trabalho, apresenta-se através da interface gráfica AGNES (*A Generic Net Editing System*), desenvolvida sobre o gerenciador de janelas X11 (disponível nas plataformas Sun, DEC Alpha e Linux). Na Figura 11 podemos visualizar um *screenshot* desta interface. A ferramenta é especialmente voltada para a análise transiente e estacionária de redes de Petri estocásticas e determinísticas, bem como da classe de redes de Petri generalizadas e estocásticas.

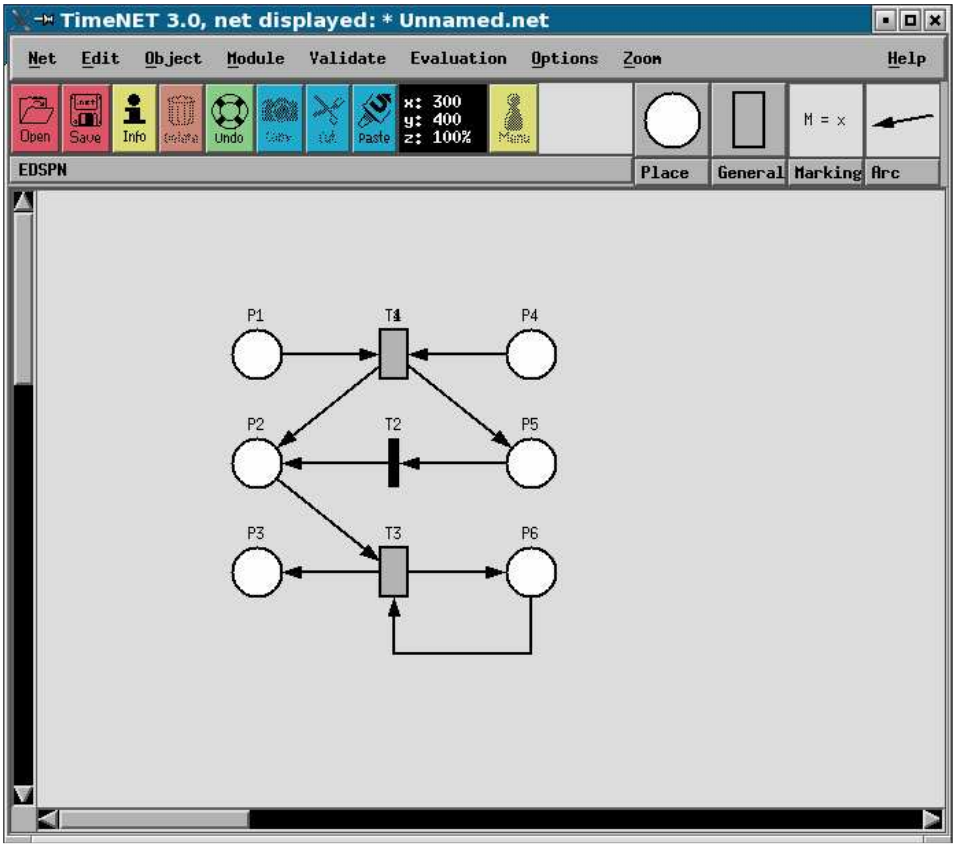


Figura 11. Interface gráfica do TimeNET 3.0

A ferramenta permite a criação de transições com tempos de disparo determinísticos, distribuídos exponencialmente e também com uma classe especial de distribuição, a distribuição expolinomial [12]. Esta distribuição em particular contém diversas distribuições conhecidas (e.g. determinística, uniforme, triangular, exponencial, etc.) e permite a aproximação de praticamente

qualquer outra distribuição. Uma definição detalhada da distribuição expolinomial foge ao escopo deste trabalho. Maiores detalhes podem ser obtidos em [12].

O TimeNET suporta a modelagem dos seguintes tipos de redes:

- **Redes de Petri Coloridas Hierarquicas (*Hierarchical Coloured Petri Nets* - HCPN)**
  Esta classe de redes de Petri pode conter *tokens* sem tipo, que não armazenam valores, bem como *tokens* coloridos, que, possuem tipos e, portanto, são capazes de armazenar valores.

- **Redes de Petri Estocásticas Fluidas (*Fluid Stochastic Petri Nets* - FSPN)**
  Este formalismo é uma extensão da classe de redes de Petri GSPN, no qual não só se conhece lugares que contém determinado número de *tokens*, como também permite que um ou mais lugares tenham uma quantidade contínua de fluído, que é representado por um número real não negativo, em vez de apenas um número discreto de *tokens*. Este "fluido" pode ser transferido através arcos fluidos, em um fluxo contínuo, tanto quanto a transição em questão permitir. Também é permitido que uma determinada quantidade de fluido seja depositada ou removida de uma vez. São bastante úteis na avaliação de sistemas que envolvem componentes como tempo, líquidos, temperaturas, etc. Maiores informações sobre este tipo de rede podem ser encontradas em [23].

- **Redes de Petri Determinísticas e Estocásticas Extendidas (*Extended Deterministic and Stochastic Petri Nets* - eDSPN)**
  Esta classe de redes de Petri consiste em uma extensão à classe GSPN. Nela, são permitidas transições com distribuições de disparo imediatas (zero), exponencialmente distribuídas, determinísticas, ou pertencer a classe de distribuições expolinomial, já citada anteriormente.

Estas classes de redes são compostas basicamente por elementos comuns às redes de Petri estocásticas, com algumas especializações. Em geral, uma rede de Petri estocástica no TimeNET é composta por Lugares, Transições, Arcos e Inibidores, e outros parâmetros. Este trabalho apresenta um maior enfoque na classe de redes eDSPN no TimeNET, pois a ferramenta utiliza esta mesma classe para representar um modelo que contenha o poder de modelagem de diversas sub-classes bem conhecidas, como GSPN e DSPN. Esta classe pode ser utilizada em seu ambiente tanto com escala de tempo contínua como também em tempo discreto.

A seguir é dada uma breve descrição acerca dos principais objetos disponíveis para a classe de redes eDSPN, extraída de [13]. A disponibilidade ou não dos componentes é determinada pelo tipo de algoritmo de análise aplicado ao modelo. Sendo assim, nem todos os objetos estarão sempre disponíveis para uso. Lugares são representados pela forma usual, como círculos, estando apenas disponível um tipo de lugar para esta classe. Um lugar possui os atributos:

- *name*, que consiste em uma string de identificação (todos os nomes de elementos existentes no modelo devem ser únicos).
- *marking*, indica a marcação inicial do lugar, e pode ser um número natural, ou, um parâmetro de marcação, detalhado mais adiante.

As transições de redes eDSPN podem ser imediatas, determinísticas, exponenciais ou gerais. Todas possuem um atributo *name*, que denota sua string de identificação no modelo. Transições imediatas são representadas graficamente como barras finas, e possuem os seguintes atributos:

- *weight*, é um número real que especifica a probabilidade de disparo da transição em relação a outras transições imediatas habilitadas no mesmo instante que estejam em conflito.

- *priority,* é um número natural que define uma precedência ante outros disparos de transições imediatas habilitadas simultaneamente. Possui valor padrão 1. Maiores valores para este campo denotam maior prioridade de disparo.
- *enabling function*, também chamada de guarda (*guard*) é uma expressão dependente de marcação, que deve ser avaliada como verdadeira para que a transição possa ser habilitada. Um exemplo de expressão dependente de marcação é #P1=1, a qual será avaliada como verdadeira se a quantidade de *tokens* no lugar P1 for igual a 1. Seu valor padrão vazio significa que a transição está habilitada para ser disparada.

As transições exponenciais são representadas graficamente como retângulos vazios, e possuem os seguintes atributos:

- *delay*, indica o atraso de disparo da transição e é exponencialmente distribuído. Por razões de consistência, os tempos de disparo de transições são especificados como *delays* para todos os tipos de transição.
- *server policy*, pode ser tanto *infinite server* como *single-server* (valor padrão), e determina a maneira na qual múltiplos clientes são atendidos. Transições com semântica *infinite server* podem ser habilitadas concorrentemente a elas mesmas tantas vezes quanto existam *tokens* de entrada disponíveis. No caso da semântica *single server* os tempos de disparo são determinados sequencialmente.
- *preemption policy*, determina o comportamento da transição quando torna-se habilitada novamente após ter sido desabilitada. Ou seja, este atributo define a política de memória adotada (explicada no capítulo anterior) pela transição. Pode assumir os valores PRD (*preemptive repeat different,* valor padrão, de significado idêntico à *race enabling policy*, descrita anteriormente) ou PRS (*preemptive resume*, correspondente ao *age memory policy*).
- *priority*, possui apenas significado para os algoritmos de análise que possuem escala de tempo discreta.

As transições determinísticas, como o próprio nome indica, possuem um atraso (*delay*) fixo, com valor padrão 1. Também possuem os atributos *priority* e *preemption policy*, e ambos apresentam as mesmas restrições impostas às transições exponenciais. São representadas como retângulos preenchidos com a cor preta.

Transições gerais, representadas graficamente como retângulos preenchidos com a cor cinza, apresentam o seu tempo de disparo descrito por uma função de probabilidade (*probability mass function*). Esta classe de funções engloba diversas distribuições (uniforme, triangular, exponencial truncada e finita discreta). Possui o valor padrão "UNIFORM(0.0,1.0);" a qual apresenta valores distribuídos uniformemente entre 0 e 1.

Marcações e tempos de disparos podem ser especificados diretamente nos lugares e transições correspondentes. Entretanto, algumas vezes é mais interessante definir parâmetros para valores usados constantemente, facilitando o entendimento e dando mais clareza ao modelo.

Parâmetros de marcação e de atraso funcionam basicamente como constantes, podendo ser inteiras no caso das marcações, com valor padrão 0, e reais no caso dos atrasos, que possuem valor padrão 1. Os parâmetros de marcação estão associados ao elemento "*marking parameters",* enquanto os parâmetros de atraso são chamados de "*delay parameters*".

Um outro tipo de objeto de muita importância em uma rede modelada no TimeNET são as medidas de desempenho. Estas, definem o que deve computado durante a análise, podendo armazenar, por exemplo, o número médio de *tokens* em um determinado lugar. As medidas possuem um nome associado, uma definição e um valor computado, o qual não pode ser editável

e só estará disponível caso uma avaliação seja executada com sucesso. O significado de uma determinada medida pode variar de acordo com o tipo de avaliação executada.

A definição de medidas de desempenho apresenta uma gramática especial [13], pois seu valor é composto de uma expressão, que pode conter números, parâmetros de atraso e de marcações, operadores algébricos, funções dependentes de marcação (*marking dependent functions*) e condições lógicas (*logic conditions*).

Funções dependentes de marcação, em definições de medidas de desempenho, apresentam-se na forma #Pn, cujo valor semântico indica o número de *tokens* existentes no lugar Pn. Condições lógicas normalmente contêm comparações entre funções dependentes de marcação e números. Uma breve descrição da gramática utilizada nas definições das medidas de desempenho é dada a seguir (para maiores detalhes, consultar [13]):

- `P{<condição_lógica>};`: Indica a probabilidade de <condição_lógica>
- `P{<condição_lógica_1> IF <condição_lógica_2>};`: Indica a probabilidade de <condição_lógica_1> sob a precondição de <condição_lógica_2> (probabilidade condicional)
- `E{<marc_func>};`: referencia o valor esperado da expressão dependente de marcação <marc_func>
- `E{<marc_func> IF <condição_lógica>};`: referencia o valor esperado da expressão dependente de marcação <marc_func> apenas se a condição <condição_lógica> seja avaliada como verdadeira.

Exemplos de medidas de desempenho são: E{#P5};, N/(5*P{#P2<3}; e P{#SendReq1>0};

A análise de redes eDSPN envolve o uso de avançados conceitos e métodos numéricos, cujos detalhes de implementação fogem ao escopo deste trabalho explicados mais adiante. Maiores detalhes podem ser encontrados em [12] [13]. A ferramenta TimeNET disponibiliza os seguintes tipos de análise:

- análise estacionária em tempo continuo (*Continuous Time Stationary Analysis*)
- análise transiente em tempo continuo (*Continuous Time Transient Analysis*)
- análise estacionária aproximativa (*Aproximative Stationary Analysis*)
- análise em tempo discreto (*Discrete Time Analysis*)
- simulação em tempo continuo (*Continuous Time Simulation*)

Este trabalho irá restringir-se às análises transientes em tempo contínuo. Este tipo de análise requer que transições temporizadas não exponenciais sejam determinísticas. Sendo assim, apenas DSPNs podem ser analisadas em um determinado estado transiente. Também requer que pelo menos uma transição determinística esteja habilitada em qualquer ponto no tempo. Maiores detalhes da implementação dos algoritmos podem ser obtidos em [24].

A análise transiente em tempo contínuo computa e exibe a solução de DSPNs, mostrando o comportamento da rede da marcação inicial (tempo zero) até um determinado ponto no tempo definido pelo usuário. Para que esta análise esteja habilitada, pelo menos uma medida de desempenho deve ser especificada no modelo.

3.3.1 O Formato NET

O TimeNET 3.0 salva as redes modeladas em arquivos texto com extensão ".net", utilizando o formato aqui batizado de NET. Este é composto de diversos elementos, organizados hierarquicamente, com variações em seus elementos, dependendo do tipo de rede modelada. Não foi encontrado nenhum tipo de documentação deste formato (a ausência de tal documentação foi confirmada com os autores da ferramenta), sendo necessário desenvolver um trabalho de geração de modelos e identificação dos possíveis componentes de cada tipo de rede, tornando possível identificar as diferenças entre eles. A definição das características do formato foi resultado de um processo de tentativa e erro realizado neste trabalho, analisando os arquivos gerados pela ferramenta, e introduzindo variações nos elementos que compõem a rede. Os resultados obtidos são apresentados nos parágrafos seguintes.

O primeiro elemento existente em uma rede salva pelo TimeNET identifica qual o tipo da rede representada. Em seguida, é listada a coleção de objetos que compõem a rede, acompanhados dos seus respectivos parâmetros (*name*, *id*, etc.). Um exemplo de uma rede EDSPN representada no formato NET pode ser visto abaixo:

```
EDSPN (
(
 objects = (
 rew_item (name = U1; ... ),
 ...
 Place (name = P1; id = 851; ... ),
 ...
 DetTrans ( ... ),
 ...
 Arc (name = A3; id = 862;  ... ),
 ...
 Inhibitor (name = I7; id = 866; ... )
))
)
```

Todos os elementos presentes no arquivo possuem em comum alguns parâmetros, enquanto outros são específicos de cada classe de elementos. Nas pesquisas desenvolvidas neste trabalho, foram identificados os seguintes itens, comuns a todos os elementos, em qualquer das redes consideradas (EDSPN, HCPN e FSPN). Apesar do foco deste trabalho ser as redes EDSPN, os outros tipos de rede também foram considerados a fim de facilitar a integração total desses outros elementos dentro do projeto no futuro. Daremos a seguir uma breve descrição dos elementos que compõem estas redes, ressaltando que os dados que se seguem foram determinados experimentalmente, e talvez não reflitam todas as possibilidades apresentadas pela ferramenta. A **Tabela 1** apresenta uma descrição dos parâmetros existentes, acompanhados dos respectivos tipos identificados, bem como uma breve descrição funcional. Faz-se necessário também a apresentação das restrições nos valores identificadas no decorrer do trabalho, uma vez que basta que apenas um detalhe não esteja condizente com o formato, para que um erro fatal seja lançado pela ferramenta, provocando a finalização de todas as instâncias do programa em execução. A identificação de tais restrições configurou um extenso trabalho de tentativa e erro, visto que nenhuma mensagem sobre qual item gerou o erro é fornecida pela ferramenta quando da abertura de novos arquivos.

Tabela 1. Parâmetros comuns a elementos do formato NET

Nome do Parâmetro	***Tipo***	***Semântica***	***Restrições***
name	Texto	Indica o nome do objeto	Deve iniciar com uma letra, e deve conter apenas letras e números
id	Inteiro	Identifica unicamente o objeto	Deve ser único em todo o modelo
scale	Real x Real	Indica a escala (zoom) do objeto na visualização da rede	Composto por uma tupla, com valor padrão (0.2,0.2)
attributes	Conjunto de parâmetros, no formato: "nome = valor;"	Depende da classe do elemento. Lista argumentos relativos ao elemento em questão, como o peso de um arco ou a marcação inicial de um lugar	Alguns atributos são envoltos por aspas duplas, enquanto outros não o são
relatives	Conjunto de tuplas Inteiro x Inteiro	Indica a posição do nome em relação ao objeto (em pixels)	___
position	Inteiro x Inteiro	Indica a posição absoluta do objeto (em pixels)	___
orientation	Inteiro	Especifica a orientação do objeto na interface gráfica	___

Nas próximas seções serão informados detalhes sobre cada uma das classes de elementos que compõem uma rede de Petri no TimeNET, inclusive detalhando a lista de parâmetros do grupo *attributes* particulares a cada classe.

Lugares

Os lugares numa rede de Petri do TimeNET são identificados pela palavra "*Place*", existindo ainda variações dependendo do tipo de rede gerada. Os padrões de lugares identificados, e seus respectivos tipos de redes nos quais estão presentes podem ser visto na **Tabela 2**. A **Tabela 3** indica os atributos identificados para esta classe de elementos.

Tabela 2. Tipos de lugares e suas respectivas classes

Tipo de Lugar	***EDSPN***	***FSPN***	***HCPN***
Place	Sim	Sim	Sim
PPlace	Não	Não	Sim
EPlace	Não	Não	Sim
FluidPlace	Não	Sim	Não

Tabela 3. Lista de atributos de lugares. Os itens marcados com * apresentam seu valor envolvido por aspas duplas

Lista de atributos	***Valor***
*marking**	Inteiro ou Parâmetro de marcação
*capacity**	Inteiro
unused	Booleano
*capacity**	Inteiro
*minval **	Real
*maxval**	Real

A função exercida pelo campo *marking* é indicar qual o número de marcações (*tokens*) existentes no lugar inicialmente. O campo *capacity* representa a capacidade máxima de marcações suportada pelo lugar em questão, enquanto os campos *minval* e *maxval* indicam as quantidades máximas e mínimas de "fluído" para um determinado *FluidPlace*.

Transições

Transições são identificadas pela palavra "*Trans*", e, assim como os lugares, possuem variações dependendo do tipo de rede em que se encontram. Os padrões de transições encontrados nos modelos gerados são listados na **Tabela 4**. Os atributos identificados em transições encontram-se indicados na **Tabela 5**.

Tabela 4. Tipos de transições e suas respectivas classes

Nome / Tipo de Rede	***EDSPN***	***FSPN***	***HCPN***
DetTrans	Sim	Sim	Sim
ExpTrans	Sim	Sim	Sim
ImmTrans	Sim	Sim	Sim
TimedTrans	Não	Não	Sim
GenTrans	Sim	Sim	Não

Tabela 5. Lista de atributos de transições. Os itens marcados com * apresentam seu valor envolvido por aspas duplas

Lista de atributos	***Valor***
*delay**	Expressão, Identificador de parâmetro de delay ou valor Real
IsMD	Booleano
fire_pol	"PRD" ou "PRS"
Priority	Inteiro
*selcol**	Texto
inf_serv	Booleano
*weight**	Real
*md_enable**	Real ou Expressão
isDMD	Booleano
isFMD	Booleano
*global_guard**	Expressão
Bindings	Lista de Expressões
server_type	"infinite" ou "finite"
distr	"Deterministic" ou "Exponential"
Unused	Booleano

*pmfunc**	Expressão de distribuição

Arcos

Os arcos, e também os inibidores, numa rede de Petri possuem uma informação não existente nos outros elementos: a origem do arco e o seu destino. Arcos no TimeNET são identificados pela palavra "Arc", e suas variações são indicadas na. **Tabela 6**.

Tabela 6. Tipos de arcos e suas respectivas classes

	EDSPN	***FSPN***	***HCPN***
Arc	Sim	Sim	Sim
DArc	Não	Não	Sim
PArc	Não	Não	Sim
EArc	Não	Não	Sim
FluidArc	Não	Sim	Não
Inhibitor	Sim	Sim	Sim

Numa rede EDSPN, arcos e inibidores possuem um atributo chamado *multi*, que representa a multiplicidade do arco, e possui o valor padrão 1. Este campo também pode assumir como valor um parâmetro de marcação. Um exemplo de uso pode ser de um arco com origem em um lugar de nome P1, e com multiplicidade #P1, irá consumir todos os *tokens* presentes no lugar P1 quando a transição de destino do arco for disparada.

Os elementos do tipo *Arc* não possuem os parâmetros *orientation* e *position*, presentes nos outros elementos. Em lugar destes, possui o parâmetro *positions*, o qual denota as coordenadas do arco desde a origem até o destino, bem como seus vértices intermediários.

Cada um dos tipos de arcos citados na **Tabela 6** possui uma lista de atributos específica separados por ";" (ponto e virgula), e assim como os elementos já apresentados, é restringida pelo tipo da rede em que se encontra. Os atributos identificados são listados na **Tabela 7**.

Tabela 7. Lista de atributos de arcos. Os itens marcados com * apresentam seu valor envolvido por aspas duplas.

Lista de atributos	***Valor***
*multi**	Inteiro
isMD	Booleano
isDMD	Booleano
distribution	Nome de distribuição
*first_param**	Real
second_param	Texto ou Real
Boundc	Valores pré-definidos no formato

Parâmetros

Nesta seção serão apresentados os elementos que, embora não sejam comuns ao formalismo de redes de Petri, estão presentes nos modelos gerados pelo TimeNET. Este trabalho resumiu-se a estudar os elementos pertinentes à classe de redes eDSPN: são eles: os parâmetros de marcação, os parâmetros de atraso e as medidas de desempenho, representados no formato por *MarkPar*, *DelayPar* e *rew_item*, respectivamente. Outros elementos existentes nas redes HCPN e FSPN (*Comment, Result, Def, TextPar, Frame, Process,* etc.) não serão detalhados por este trabalho.

Os parâmetros dos modelos de redes de Petri gerados no TimeNET possuem estrutura similar à encontrada nos lugares e transições, diferenciando-se apenas pela lista de atributos de cada um deles, apresentados na **Tabela 8**.

Tabela 8. Lista de atributos dos parâmetros das redes eDSPN. Os itens marcados com * apresentam seu valor envolvido por aspas duplas.

Parâmetro	***Atributo***	***Valor***
MarkPar	value	Inteiro
DelayPar	IsMD	Booleano
	delay*	Real
Rew_item	result*	"*yes*" ou "*no*"
	reward_def*	Expressão

3.3.2 O formato TN

Este formato, diferentemente do formato NET, possui uma gramática bem definida e encontra-se documentado, sendo utilizado em versões anteriores do TimeNET. Inicialmente, o formato TN não fazia parte do compilador desenvolvido neste projeto[2]. Porém, após a constatação de que os algoritmos de análise do TimeNET não estavam aptos a interpretar redes no formato NET, mas apenas no antigo formato TN, foi tomada a decisão de incluir a opção de gerar arquivos neste formato pelo compilador, embora a leitura dos mesmos não tenha sido implementada.

A inclusão do formato não afetou os módulos que já haviam sido desenvolvidos, devido a arquitetura modular assumida no projeto, e também pelo fato da gramática deste formato estar bem documentada em [13]. Um exemplo de rede no formato TN pode ser visto na Figura 12. Observa-se que o arquivo apresenta uma estrutura bem definida, em que na parte inicial é descrito o nome do arquivo, o tipo da rede e a quantidade de lugares, transições, parâmetros de marcação e atraso bem como medidas de desempenho.

Em seguida é apresentada a lista de parâmetros de marcação, a lista de lugares da rede e a lista de parâmetros de atraso. A discriminação das transições inclui a definição dos arcos de entrada e saída das mesmas. A última parte do arquivo é composta dos parâmetros específicos de cada tipo de transição, e, por fim, as medidas de desempenho existentes no modelo, especificadas no mesmo formato apresentado na seção 3.2.

[2] Informações fornecidas pelo autor da ferramenta indicavam o abandono deste formato, sendo substituído pelo formato NET, textualmente mais conciso.

```
-- FILE mm1_10_queue.TN CONTAINING STRUCTURAL DESCRIPTION OF A NET

NET_TYPE: EDSPN
DESCRIPTION:              ?
PLACES:                    4
TRANSITIONS:              3
DELAY_PARAMETERS:         2
MARKING_PARAMETERS:       1
REWARD_MEASURES:           2

-- LIST OF MARKING PARAMETERS (NAME, VALUE, (X,Y)-POSITION):
MARKPAR K 10 0.73 3.82

-- LIST OF PLACES (NAME, MARKING, (X,Y)-POSITION (PLACE & TAG)):

PLACE P1 K 0.66 -0.15 1.91 -0.06
...

-- LIST OF DELAY PARAMETERS (NAME, VALUE, (X,Y)-POSITION):
DELAYPAR service 0.125 0.73 3.62

-- LIST OF TRANSITIONS
--       (NAME, DELAY, ENABLING DEPENDENCE, KIND, FIRING POLICY, PRIORITY,
--       ORIENTATION, PHASE, GROUP, GROUP_WEIGHT,
--       (X,Y)-POSITION (TRANSITION, TAG & DELAY), ARCS)

TRANSITION T2 1 SS IM RE 1 0 1 0 1.000000 2.74 1.91 -0.03 -0.15 0 0
INPARCS 2
1 FilaAtendimento 0
1 ServidorLivre 0
OUTPARCS 1
1 ServidorOcupado 0
INHARCS 0
...

-- DEFINITION OF PARAMETERS:

-- MARKING DEPENDENT FIRING DELAYS FOR EXP. TRANSITIONS:

-- MARKING DEPENDENT FIRING DELAYS FOR DET. TRANSITIONS:

-- PROBABILITY MASS FUNCTION DEFINITIONS FOR GEN. TRANSITIONS:

-- MARKING DEPENDENT WEIGHTS FOR IMMEDIATE TRANSITIONS:

-- ENABLING FUNCTIONS FOR IMMEDIATE TRANSITIONS:

-- MARKING DEPENDENT ARC CARDINALITIES:

-- REWARD MEASURES:

MEASURE ServerUtil
P{#P2=0};

MEASURE Avg
E{#P1};

-- END OF SPECIFICATION FILE
```

Figura 12. Exemplo de rede de Petri no formato TN

Capítulo 4

Extensão da PNML para representação de redes de Petri estocásticas

Nesta seção serão apresentadas as extensões para incorporar na PNML a representação das redes de Petri estocásticas definidas pela ferramenta TimeNET. Todas as características utilizadas exclusivamente por redes estocásticas ficam definidas dentro de *tags* `toolspecific`, as quais podem estar localizadas logo abaixo da hierarquia da rede propriamente dita, ou dentro de elementos específicos.

Neste trabalho, optou-se por utilizar uma versão simplificada do padrão PNML proposto em [20] para as redes *Place / Transition*, visando uma representação com o máximo de simplicidade, concisão e precisão possível. Foram acrescentados elementos `toolspecific` para o devido encapsulamento de informações restritas às redes de Petri estocásticas. Devemos ressaltar que nem todas as informações mantidas nas *tags* `toolspecific` serão utilizadas ou exibidas no ambiente de edição, sendo muitas vezes necessárias apenas para manter a consistência na tradução entre os diferentes formatos utilizados no compilador desenvolvido. Os elementos `toolspecific` criados nos modelos devem ser manipulados apenas por este projeto. Caso haja a necessidade de adicionar informações por outras ferramentas, devem ser criados novos elementos `toolspecific`, diferenciando-se dos já existentes pelo atributo *tool*.

De modo geral, todos os tipos de elementos tiveram à sua estrutura adicionadas as *tags*, `type`, `scale` e `relatives`, que correspondem aos elementos homônimos da estrutura correspondente no formato NET. Optou-se pela representação redundante de algumas informações, como, por exemplo, a marcação inicial de lugares (indicado pelo atributo *marking* e também pelo elemento `initialMarking`) a fim de simplificar o processo de tradução.

Na estrutura criada, lugares e transições apresentam representações semelhantes. Neles, foram adicionados alguns elementos ao formato de PNML já utilizado pelo EZPetri em seu editor de redes *Place / Transition*, relativos à exibição da rede no editor desenvolvido neste trabalho. As informações sobre as características temporais do elemento em questão ficam envolvidas por outra *tag* `toolspecific`. Um exemplo da proposta de representação em PNML de um lugar em uma rede de Petri estocástica pode ser visto a seguir.

```
<place id="1160096859390">
   <initialMarking>
      <value>0</value>
   </initialMarking>
   <name>
      <value>label</value>
      <graphics>
         <offset x="40" y="0"/>
      </graphics>
   </name>
   <graphics>
      <position x="386" y="125"/>
   </graphics>
   <toolspecific tool="EZPetri" version="1.0">
      <size height="40" width="40"/>
   </toolspecific>
   <toolspecific tool="EZPetri" version="1.0">
      <preferedSize height="40" width="40"/>
   </toolspecific>
   <toolspecific tool="TimeNET" version="3.0">
      <type>Place</type>
      <attributes marking="0"/>
      <scale x="0.2" y="0.2"/>
      <relatives>
         <offset x="0.0" y="0.0"/>
      </relatives>
      <orientation value="0"/>
   </toolspecific>
</place>
```

Todos os lugares são criados com *tags* `place`, enquanto as transições pertencentes à classe de redes eDSPN são representadas por *tags* para seus respectivos tipos (`immtransition`, `dettransion`, `gentransition` e `exptransition`). Apesar da diferença nos nomes das *tags* responsáveis pela representação das transições, todas apresentam a mesma estrutura. Esta diferenciação deu-se por uma limitação imposta pelo *framework* de mapeamento utilizado, o qual será explicado mais adiante. Uma transição exponencial é representada em PNML da seguinte forma:

```
<exptransition id="850">
   <name>
         <value>T3</value>
         <graphics>
               <offset x="0" y="264" />
         </graphics>
   </name>
   <graphics>
         <position x="607" y="348" />
   </graphics>
   <toolspecific tool="TimeNET" version="3.0">
         <type>ExpTrans</type>
         <attributes inf_serv="false" IsMD="false" selcol="blue" priority="0"
                     delay="arrival" fire_pol="PRD">
               <bindings />
         </attributes>
         <scale x="0.2" y="0.2" />
         <relatives>
               <offset x="0.0" y="264.0" />
         </relatives>
         <orientation value="0" />
   </toolspecific>
</exptransition>
```

Observe que existem dois tipos distintos de *tags* `toolspecific` definidos. Um deles, é responsável por manter as informações relacionadas à exibição do modelo no ambiente de modelagem do EZPetri (com o atributo `tool` = "EZPetri"). O outro (com atributo `tool` = "TimeNET") contém informações restritas às representações do elemento no formato NET ou TN. Dentre estas informações estão, por exemplo, a lista de atributos (indicada pela *tag* `attributes`) e o tipo do elemento (*tag* `type`).

Arcos são representados indicando-se apenas a origem e o destino do mesmo (atributos *source* e *target*), bem como seu peso (elemento `inscription` e atributo *multi*). A lista de vértices intermediários do arco é indicada dentro da *tag* `graphics`. Os diversos tipos de arcos diferenciam-se entre si apenas pela lista de atributos e pelo tipo indicado na *tag* `type` (*Arc, FluidArc, Inhibitor*, etc.). Um exemplo da representação de arcos no formato desenvolvido pode ser visualizado a seguir.

```
<arc id="857" source="855" target="837">
      <inscription>
            <value>1</value>
      </inscription>
      <graphics>
            <position x="970" y="348" />
            <position x="970" y="148" />
            <position x="970" y="135" />
      </graphics>
      <toolspecific tool="TimeNET" version="3.0">
            <type>Arc</type>
            <name>A13</name>
            <scale x="0.25" y="0.25" />
            <relatives>
                  <offset x="0.0" y="0.0" />
                  <offset x="-50.0" y="0.0" />
            </relatives>
            <endpoints>
                  <position x="913.15" y="348.3" />
                  <position x="355.499" y="142.8" />
            </endpoints>
            <attributes multi="1" IsMD="false" />
      </toolspecific>
</arc>
```

A representação de arcos no formato definido possui um elemento que não está presente nos outros elementos da rede, a *tag* `endpoints`. Este elemento deve conter apenas duas instâncias do elemento `position`, cada qual com dois atributos, *x* e *y*, com a função de manter as coordenadas das posições de origem e destino do arco. Optou-se por manter esta informação em um elemento isolado, diferente da *tag* `graphics`, devido ao fato de essa informação não ser compartilhada por todos os formatos de saída.

Por fim, apresentamos a representação dos parâmetros da rede, são eles: os parâmetros de marcação, de atraso e as medidas de desempenho, representados respectivamente pelos elementos `MarkPar`, `DelayPar` e `rew_item`. Estes compartilham a mesma estrutura, diferenciando-se entre si apenas pelo nome da *tag* e pela lista de atributos. Uma vez que todos se localizam hierarquicamente abaixo de um elemento `toolspecific`, situado logo abaixo do nó raiz da rede (o elemento `net`) estes não precisam fazer uso de novos elementos `toolspecific` (à exceção das informações de exibição), pois possuem informações relevantes apenas à ferramenta TimeNET. Um parâmetro de atraso representado em PNML é mostrado a seguir.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<pnml>
   <net id="n1" type="EDSPN">
      ...
      <toolspecific tool="TimeNET" version="3.0">
            <delayPar id="1160851294221">
                  <graphics>
                        <position x="99" y="489"/>
                  </graphics>
                  <toolspecific tool="EZPetri" version="1.0">
                        <size height="20" width="20"/>
                  </toolspecific>
                  <toolspecific tool="EZPetri" version="1.0">
                        <preferedSize height="20" width="20"/>
                  </toolspecific>
                  <name>service</name>
                  <relatives>
                        <offset x="0.0" y="0.0"/>
                  </relatives>
                  <attributes IsMD="false" delay="0.125"/>
                  <scale x="0.2" y="0.2"/>
                  <orientation value="0"/>
            </delayPar>
            ...
      </toolspecific>
   </net>
</pnml>
```

Capítulo 5

Implementação

Decidiu-se dividir o ambiente de criação, edição e análise em módulos separados, seguindo os princípios utilizados pelo EZPetri, segundo os quais as funcionalidades devem ser agrupadas em *plug-ins* para a plataforma Eclipse, visando um maior isolamento entre os módulos. O primeiro módulo desenvolvido foi o compilador para as diferentes linguagens trabalhadas no projeto: PNML; TN e NET. Os detalhes da implementação do compilador serão demonstrados na Seção 5.1. O segundo módulo desenvolvido foi o ambiente para edição e criação de redes de Petri estocásticas eDSPN integrado ao ambiente Eclipse / EZPetri, apresentado na Seção 5.2. E, por fim, foi desenvolvido o ambiente de análise de redes eDSPN, utilizando as funcionalidades implementadas na ferramenta TimeNET, apresentado na Seção 5.3.

5.1 Compilador

O compilador foi inicialmente desenvolvido para realizar apenas a tradução (nas duas direções) entre os formatos NET e PNML, pois, como ressaltado na Seção 4.3, este seria o formato a ser adotado nas próximas versões da ferramenta. Além disso, não era conhecido o fato de que o único formato aceito pelos *scripts* de análise existentes no TimeNET é o formato TN. O desenvolvimento do compilador para tradução do formato NET consumiu grande parte do tempo desenvolvimento, devido ao fato de não haver nenhuma documentação existente para este formato.

A falta de uma gramática bem documentada dificultou bastante o trabalho, sendo necessária a geração de vários modelos, para que pudesse ser criado um mapeamento entre os elementos e sua respectiva representação no modelo. O compilador foi desenvolvido tendo como foco não só as redes eDSPN, mas também as redes FSPN e HCPN. Isto se deu devido ao fato de que, inicialmente, este trabalho deveria abranger a todos os tipos de redes existentes na ferramenta TimeNET. Entretanto, tal nível de profundidade mostrou-se impossível de ser realizado dentro das limitações de prazo para desenvolvimento do projeto. A decisão de especializar o trabalho nas redes eDSPN foi tomada após ter sido iniciado o desenvolvimento do compilador, sendo mantido o suporte às redes FSPN e HCPN, ainda que não tenham sido extensivamente testados e validados.

A arquitetura utilizada pelo compilador foi feita de forma que tornasse mais fácil a adição de novos formatos de saída, pois todos os processos de compilação utilizam uma representação intermediária da rede a partir da qual são gerados os arquivos de saída. A Figura 13 demonstra

esta arquitetura, na qual, a tradução para qualquer formato apresenta a mesma seqüência de passos. O arquivo de entrada é lido e as informações sobre a rede de Petri e seus elementos são extraídas para criação de um objeto `PetriNetTN`, mantido em memória. Este objeto encapsula todos os elementos da rede, tornando possível que a partir dele sejam geradas as saídas para os outros formatos.

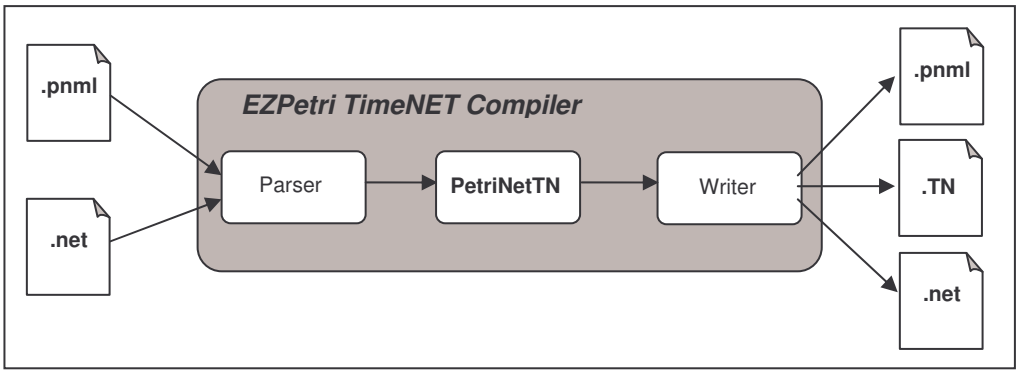


Figura 13. Formatos de entrada e saída do compilador desenvolvido

5.1.1 Arquitetura

O *plug-in* EPTC divide-se basicamente em duas partes: o *core*, que contém as classes relacionadas aos processos de compilação e geração de arquivos; e o pacote *ui*, que trata das interfaces com o usuário no ambiente EZPetri. O EPTC apresenta a seguinte estrutura de pacotes, ilustrada na Figura 14:

- ***com.ezpetri.compiler.eptc.core.entities*:** contém as classes que compõem a representação em Java de um rede de Petri estocástica do Timenet.
- ***com.ezpetri.compiler.eptc.core.io.net*:** contém as classes responsáveis por realizar a gravação dos arquivos de saída no formato NET.
- ***com.ezpetri.compiler.eptc.core.io.pnml*:** contém as classes responsáveis por realizar a gravação dos arquivos de saída no formato PNML.
- ***com.ezpetri.compiler.eptc.core.io.tn*:** contém as classes responsáveis por realizar a gravação dos arquivos de saída no formato TN.
- ***com.ezpetri.compiler.eptc.core.parser.net*:** contém as classes que compõem o parser de arquivos no formato NET.
- ***com.ezpetri.compiler.eptc.core.parser.pnml*:** contém as classes que compõem o parser de arquivos no formato PNML.
- ***com.ezpetri.compiler.eptc.core.util*:** Classes de uso comum a todo o sistema.
- ***com.ezpetri.compiler.eptc.ui.popup.actions*:** contém as classes de tratamento da iteração do usuário com a interface gráfica.

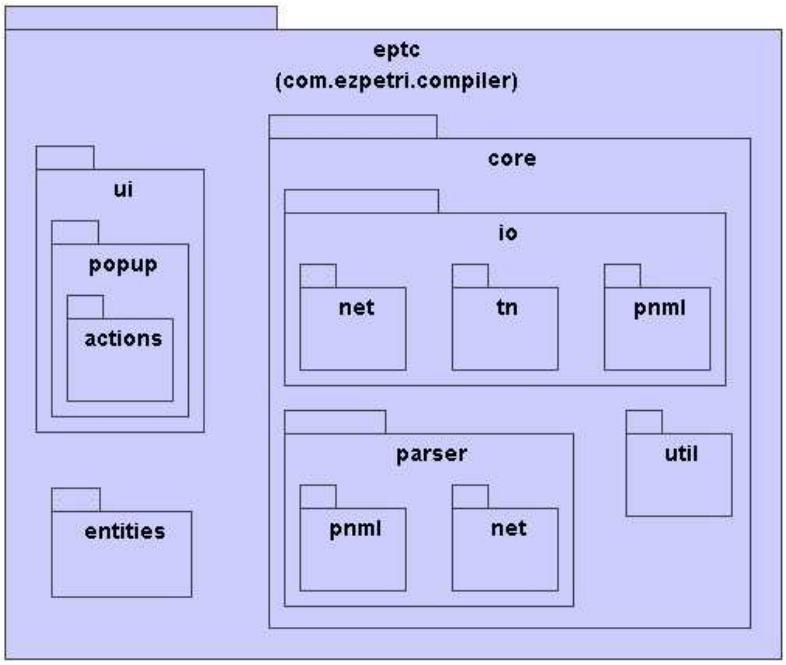


Figura 14. Diagrama de pacotes do compilador EPTC

A utilização do compilador se dá de maneira análoga à utilizada nos compiladores EPPC e EPIC, já integrados ao EZPetri. O compilador foi implementado utilizando o padrão de projeto *Strategy*, devido aos benefícios relacionados a polimorfismo trazidos por esta abordagem. Este modelo de integração, indicado na Figura 15, centraliza as operações de tradução na classe `Eptc`, responsável pela devida instanciação das classes de *parsing* e de *writing*.

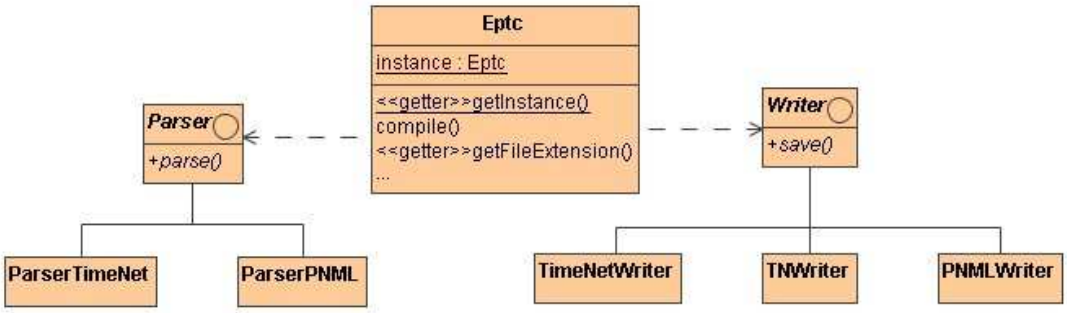


Figura 15. Visão geral do compilador EPTC

A seguir será detalhado o pacote *entities*, cujos elementos representam o modelo definido no compilador de uma rede de Petri estocástica do TimeNET.

Modelo (pacote *entities*)

Contém as classes que representam as entidades básicas do sistema. Ou seja, cada classe representa uma categoria de elementos que compõem uma rede de Petri gerada pelo TimeNET. As classes que compõem este pacote são listadas a seguir:

- **`PetriNetTN`**: representa uma rede de Petri com todos os seus possíveis elementos;
- **`Node`**: classe com elementos comuns às classes `Transition` e `Place`;

- **`Place`**: representa um lugar de uma rede de Petri;
- **`Transition`**: representa uma transição de uma rede de Petri;
- **`Arc`**: representa um arco de uma rede de Petri;
- **`Parameter`**: representa os parâmetros existentes em redes de Petri geradas pelo TimeNET;
- **`Coordinate`**: representa um conjunto de coordenadas cartesianas x e y.

O diagrama de classes do referido pacote Java é apresentado na Figura 16.

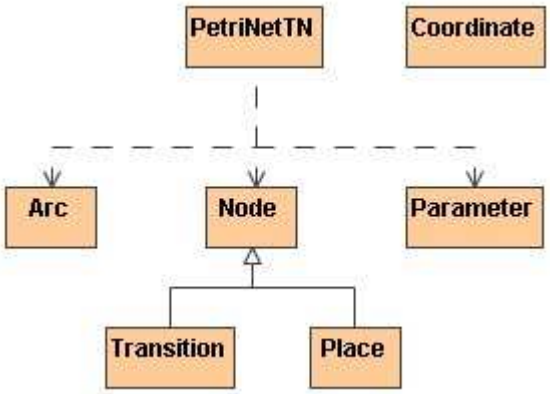


Figura 16. Diagrama de Classes do pacote *entities*

5.1.2 Parsing

Um *parser* foi desenvolvido para cada um dos formatos de entrada: um para o formato NET, que utiliza expressões regulares Java; e outro para o formato PNML, que utiliza a *Application Programming Interface* (API) *Java Document Object Model* (JDOM) [25]. Ambos apresentam basicamente duas etapas, a primeira consiste em agrupar os elementos de acordo com seu tipo (Lugares, Transições, Arcos, etc.), e a segunda em extrair as informações de cada um dos elementos. A seguir apresentamos as etapas no processo de *parsing* de arquivos NET e arquivos PNML.

As classes envolvidas nos processos de leitura e *parsing* de arquivos estão encapsuladas no pacote ***com.ezpetri.compiler.eptc.core.parser*** (Figura 17)**.** Todos os *parsers* criados devem implementar a interface `Parser` existente neste pacote. A classe `ParserFactory` é responsável por gerar instâncias de classes que implementem a classe `Parser`, de acordo com a extensão do arquivo de entrada (.net ou .pnml). No trecho de código a seguir, extraído da classe `ParserFactory`, podemos observar como é escolhida a classe que irá realizar o *parse* do arquivo. Tal escolha é feita baseada na extensão do arquivo de entrada, e, em seguida, uma instância da classe selecionada é acessada, e a partir dela é executado o método `parse`, cuja assinatura é definida na interface `Parser`.

```
if ( fileExtension.equals( "net" ) ) {
    parserClassName =
        "com.ezpetri.compiler.eptc.core.parser.net.ParserTimeNet";
} else if ( fileExtension.equals( "pnml" ) ) {
    parserClassName =
        "com.ezpetri.compiler.eptc.core.parser.pnml.ParserPNML";
}

...

Class parserClass = Class.forName( parserClassName );
Class[] args = {};
Method instanceMethod = parserClass.getMethod( "getInstance", args );
builtParser = ( Parser ) instanceMethod.invoke( builtParser, args );
...
```

Figura 17. O pacote *com.ezpetri.compiler.eptc.core.parser*, e seus respectivos elementos

Nenhum dos parsers desenvolvidos implementa nenhum tipo de validação léxica e estrutural da gramática utilizada em medidas de desempenho. Os valores definidos pelos usuários tanto nos arquivos de entrada como também no ambiente de modelagem produzido (explicado mais adiante), são tratados como um valor textual, sem valor semântico. Tal validação e tratamento fazem parte das propostas para trabalhos futuros a serem realizados.

Parsing de arquivos NET

As classes utilizadas para o *parsing* de um arquivo NET são agrupadas no pacote ***com.ezpetri.compiler.eptc.core.parser.net***. O processo consiste na execução de uma seqüência de passos bem definida, indicada no diagrama de seqüência da Figura 18. O método `getNetElements` da classe `ParserTimeNet` é o responsável por agrupar os referidos elementos existentes no arquivo de entrada, de acordo com seus tipos. Este método recebe como argumento o conteúdo do arquivo de entrada como uma única *String*, e agrupa os elementos em grupos utilizando expressões regulares da API Java. Tal separação se faz necessária devido às diferenças, muitas vezes bastante sutis, existentes entre as classes de elementos nos arquivo. Alguns exemplos de expressões utilizadas podem ser vistas nas **Tabelas** 9 e 10.

Tabela 9. Tipos de lugares e as respectivas expressões regulares utilizadas.

Tipos de Lugares	*Expressão Regular*
Place	"Place (.*),"
PPlace	"PPlace (.*),"
EPlace	"EPlace (.*),"
FluidPlace	"FluidPlace (.*),"

Tabela 10. Tipos de Transições e as respectivas expressões regulares utilizadas.

Tipos de Transições	*Expressão Regular*
DetTrans	"DetTrans (.*),"
ExpTrans	"ExpTrans (.*),"
ImmTrans	"ImmTrans (.*),"
TimedTrans	"TimedTrans (.*),"
GenTrans	"GenTrans (.*),"

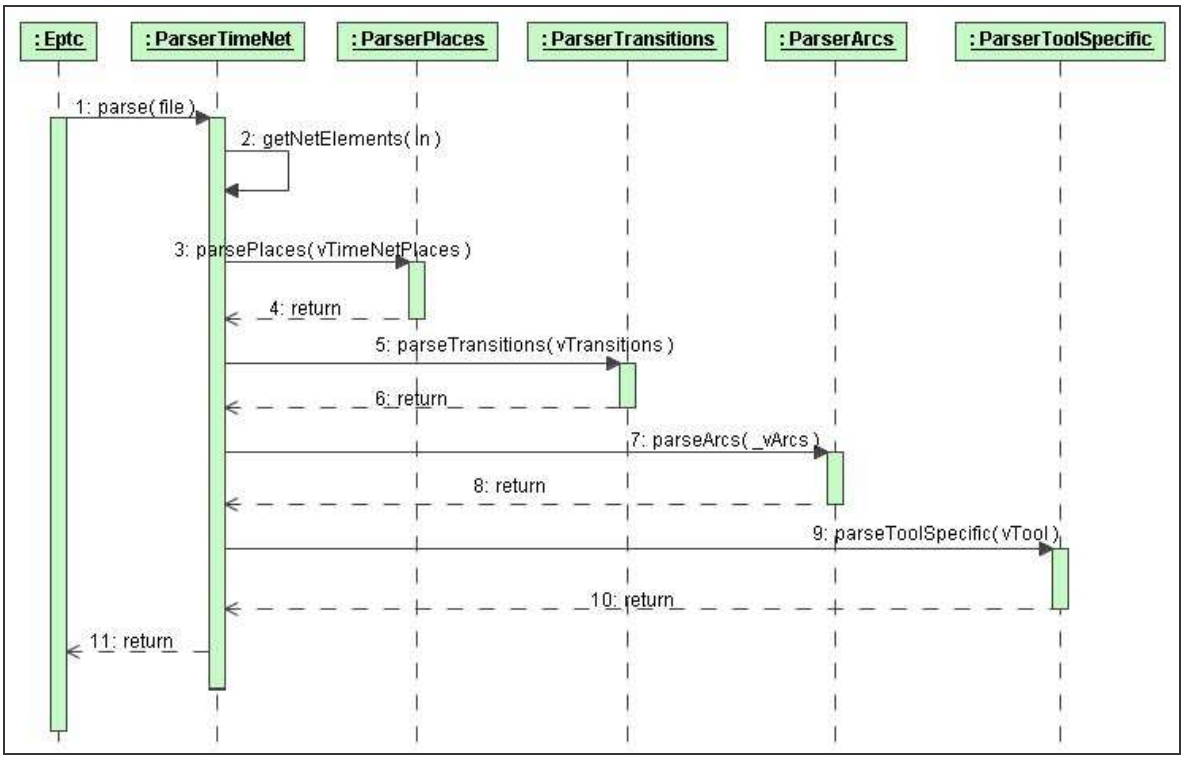


Figura 18. Diagrama de seqüência indicando os passos envolvidos no *parsing* de arquivos NET. Chamadas a sub-rotinas foram omitidas para simplificar o diagrama

Uma vez que as linhas de texto encontram-se separadas de acordo com seus respectivos tipos de elementos, entra em ação a segunda etapa do processo de *parsing*, que é a extração das informações contidas em cada uma das linhas do arquivo de entrada. Esta etapa também utiliza expressões regulares.

O processo de extração de informações é feito de maneira seqüencial, acompanhando a estrutura do arquivo e as definições dos elementos apresentadas na Seção 3.3.1. Uma vez extraídas as informações do documento, é criado um novo objeto representando o elemento analisado.

Parsing de arquivos PNML

O *parsing* de documentos PNML foi desenvolvido utilizando a API JDOM, que oferece inúmeras facilidades para o tratamento de arquivos XML. As classes envolvidas no processo encontram-se encapsuladas no pacote ***com.ezpetri.compiler.eptc.core.parser.pnml.*** De maneira análoga à utilizada na leitura de arquivos NET, os elementos da rede representados no documento PNML de entrada são inicialmente agrupados de acordo com seus respectivos tipos, e em seguida são passados como parâmetros para as classes responsáveis pela tradução dos elementos PNML em

objetos Java do modelo desenvolvido. Um diagrama de seqüência representando as etapas do processo pode ser visualizado na Figura 19. Nele, podemos perceber a ausência do método `getNetElements`, pois, a API JDOM já agrupa os elementos existentes no modelo de acordo com com suas respectivas *tags*. Por exemplo, a chamada ao método `getChildren("place")` a partir do elemento raiz ("*net*"), retorna a coleção de elementos `place` presentes no arquivo PNML.

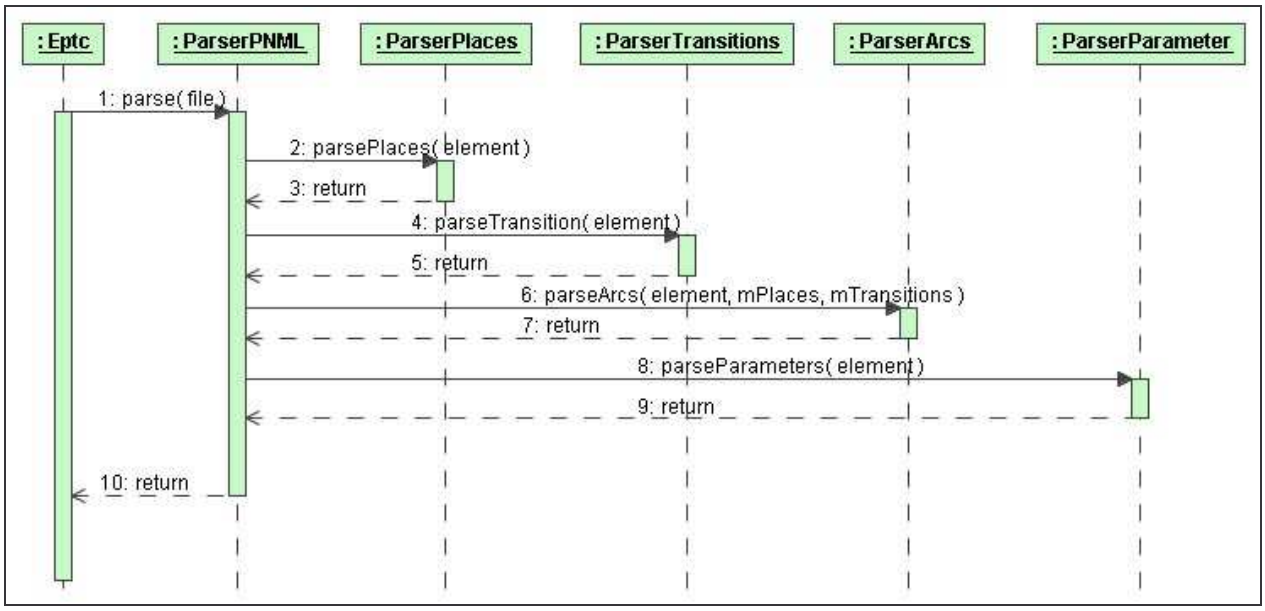


Figura 19. Diagrama de seqüência indicando as etapas envolvidas no *parsing* de arquivos PNML. Chamadas às sub-rotinas foram omitidas para simplificar o diagrama

Um trecho do código responsável pela extração das informações gerais da rede contida no documento PNML é indicado abaixo. Podemos observar como o uso da API JDOM se dá de modo bastante intuitivo. Este trecho demonstra a extração das informações da *tag* `net`, contida nos atributos *id* e *type*, bem como na *tag* filha `name`.

```
...
Document doc = builder.build(file);

Element rootElement = doc.getRootElement();
petriNet = new PetriNetTN();

Element netElement = rootElement.getChild("net");

petriNet.setId(netElement.getAttributeValue("id"));
petriNet.setName(netElement.getChildTextNormalize("name"));

String type = netElement.getAttributeValue("type");
...
```

O processo de tradução de documentos PNML para o modelo de classes apresenta algumas peculiaridades, devido à possibilidade do arquivo ter sido gerado por outra ferramenta. Sendo assim, uma seqüência de validações faz-se necessária, levando em consideração alguns pontos principais:

1. verificação se os IDs de cada um dos elementos da rede são válidos no contexto da ferramenta TimeNET (IDs em modelos gerados pela ferramenta devem ser obrigatoriamente números inteiros);
2. verificação se os nomes dos elementos representados no documento PNML são nomes válidos na ferramenta TimeNET (nomes válidos devem iniciar com uma letra).

As duas validações acima são implementadas fazendo uso de expressões regulares Java, nos métodos da classe `Util`, `isValidID` e `isValidName`. Estes métodos e suas respectivas expressões de validação são listados a seguir:

```
public static boolean isValidID(String id) {
     return id.matches("\\d+");
}

public static boolean isValidName(String name) {
     boolean result = name.matches("[a-zA-Z]+([0-9]|[a-zA-Z]|_)*");
     return result;
}
```

No caso de serem encontrados identificadores ou nomes inválidos, estes são substituídos por IDs e nomes válidos. Estas modificações devem ser mantidas para que, durante a leitura de elementos que possuem referências a outros elementos da rede, como por exemplo os arcos, não sejam introduzidas inconsistências no modelo gerado. Para que isso seja evitado, um mapeamento entre o valor original e o novo valor é criado para cada ID ou nome modificado, possibilitando manter a consistência da rede gerada mesmo quando estas correções são necessárias.

Outras validações também são feitas durante a análise da estrutura dos elementos PNML do modelo. Em termos gerais, estes elementos gerados externamente têm as propriedades específicas do TimeNET inicializadas com valores padrão.

5.1.3 Writing

A partir do momento em que o processo de *parsing* da entrada se encerra, tem-se um objeto `PetriNetTN` pronto para ser persistido, seja no formato NET, TN ou mesmo PNML. As classes envolvidas na gravação de arquivos estão encapsuladas no pacote ***com.ezpetri.compiler.eptc.core.io***, ilustrado na Figura 20**.**

Figura 20. O pacote *com.ezpetri.compiler.eptc.core.io*,
e seus respectivos elementos

As classes de gravação de arquivos devem implementar a interface `Writer` existente neste pacote. A instanciação dessas classes é feita de maneira análoga à utilizada na etapa de *parsing* dos arquivos de entrada, através da classe `WriterFactory`. A classe `WriterFactory` é

responsável por gerar instâncias das classes de gravação de arquivos, de acordo com a extensão do arquivo de saída desejado (.net, .TN ou .pnml). O trecho do código da *Factory* capaz de identificar qual a classe a ser instanciada é listado a seguir.

```
public Writer buildWriter( String fileExtension ) throws Exception {
  ...
  if ( fileExtension.toLowerCase().equals( "net" ) ) {
    writerClassName = "com.ezpetri.compiler.eptc.core.io.net.TimeNetWriter";
  }
  else if ( fileExtension.toLowerCase().equals( "pnml" ) ) {
    writerClassName = "com.ezpetri.compiler.eptc.core.io.pnml.PNMLWriter";
  }
  else if ( fileExtension.toLowerCase().equals( "tn" ) ) {
    writerClassName = "com.ezpetri.compiler.eptc.core.io.tn.TNWriter";
  }
  ...
  Class writerClass = Class.forName( writerClassName );
  Class[] args = {};
  Method instanceMethod = writerClass.getMethod( "getInstance", args );
  builtWriter = ( Writer ) instanceMethod.invoke( builtWriter, args );
  ...
}
```

De uma forma geral, a arquitetura utilizada trouxe diversas facilidades ao processo de geração de arquivos, uma vez que, a partir de um mesmo modelo mantido em memória, é possível gerar uma saída do modelo no formato desejado. O modelo foi desenvolvido de modo a manter todas as informações necessárias para qualquer um dos formatos definidos, ficando a cargo da implementação da interface `Writer` a seleção das informações relevantes para a saída.

A geração de arquivos NET consumiu mais tempo de desenvolvimento que a geração de arquivos do tipo TN. A principal dificuldade no processo de geração de arquivos deu-se pelo fato de não haver uma gramática definida para o formato, como também pela não existência de mensagens de erro no TimeNET (a ferramenta finaliza todas as instâncias do programa em execução no momento da abertura de um arquivo inválido), sendo necessário um longo processo de tentativa e erro até chegar à solução final. Nos testes realizados com os arquivos gerados pelo compilador, a ferramenta TimeNET apresentou-se bastante rigorosa com relação aos tipos de dados existentes no arquivo, bem como em relação à estrutura dos elementos que compõem a rede. Um tratamento especial foi necessário para as unidades de coordenadas gráficas utilizadas: *pixels* para os formatos NET e PNML; e polegadas para o formato TN.

A geração de arquivos PNML fez uso da API JDOM devido às facilidades oferecidas por esta biblioteca para a manipulação, criação e persistência de documentos XML. Uma estrutura interna, representando o modelo da rede, é construída de forma modular para cada classe de elementos, sendo, por fim, persistida através do JDOM em um arquivo texto.

5.1.4 Interface com o usuário

O *plug-in* desenvolvido fornece uma interface com o usuário intuitiva, adicionando novas funcionalidades ao menu de contexto do EZPetri "*Compile To*". Este menu surge após o usuário clicar sobre um arquivo existente no *workspace*, caso este possua extensão pnml, net (TimeNET), pnt (INA) e ll_net (PEP). No caso de arquivos com extensão net, o *plug-in* adiciona a opção "PNML (EPTC)", e, quando o arquivo selecionado possuir extensão pnml, o *plug-in* acrescenta as opções "TN (EPTC)" e "NET (EPTC)". Estas opções são exibidas na Figura 21.

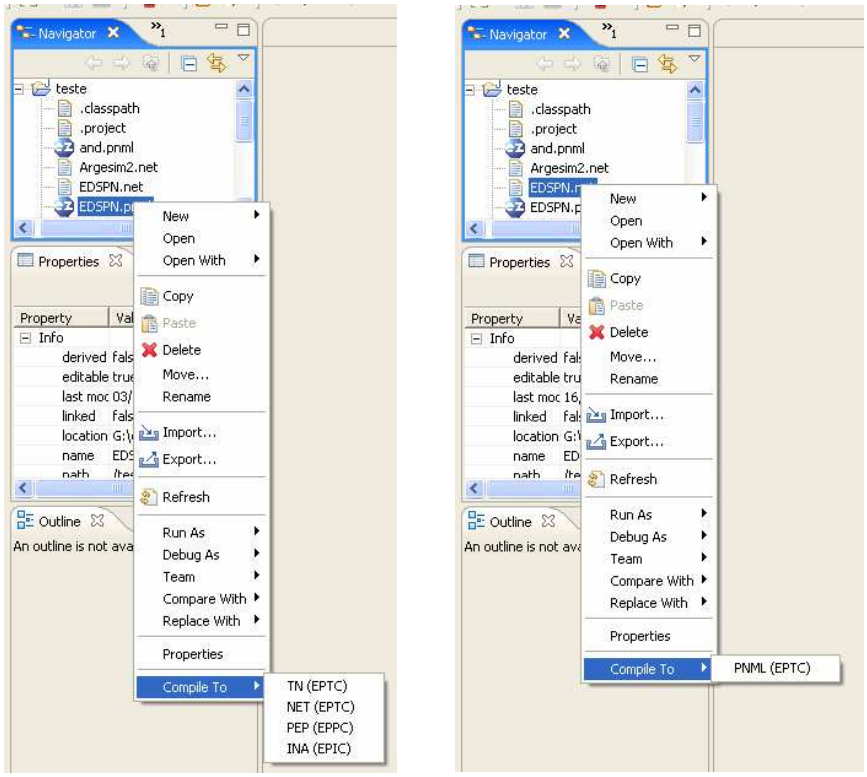


Figura 21. Opções adicionadas no *context menu* "*Compile To*" para arquivos com extensão (a) pnml e (b) net

A criação das novas opções de compilação ao menu "*Compile To*" foi feita definindo no arquivo plugin.xml os pontos de extensão afetados pelo *plug-in* EPTC. O trecho do arquivo correspondentes à contribuição do *plug-in* à interface gráfica do Eclipse para a geração de arquivos com extensão TN é mostrado abaixo.

```
...
<extension  point="org.eclipse.ui.popupMenus">
    ...
    <objectContribution
         id=" com.ezpetri.compiler.eptc.contribution3"
         nameFilter="*.pnml"
         objectClass="org.eclipse.core.resources.IFile">
       <action
           label="TN (EPTC)"
           class="com.ezpetri.compiler.eptc.ui.popup.actions.CompilePNMLToTN"
           menubarPath="com.ezpetri.compiler.compileTo/compilers"
           enablesFor="1"
           id=" com.ezpetri.compiler.eptc.ui.popup.actions.CompilePNMLToTN">
       </action>
    </objectContribution>
</extension>
...
```

As pré-condições de uso do *plug-in* EPTC ficam indicadas no arquivo MANIFEST.MF, cujo conteúdo é exposto a seguir. Dentre as diversas informações contidas neste arquivo, encontramos o elemento `Require-Bundle`, que define qual a lista de *plug-ins* necessários para o

uso do *plug-in* atual. Caso algum deles não esteja instalado, o *plug-in* atual não será carregado pela plataforma Eclipse.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Eptc Plug-in
Bundle-SymbolicName: com.ezpetri.compiler.eptc; singleton:=true
Bundle-Version: 1.0.0
Bundle-Vendor: EZPetri (DSC)
Bundle-Localization: plugin
Require-Bundle: jdom_plugin,
 org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources
Eclipse-AutoStart: true
Bundle-ClassPath: ezpetri_eptc.jar
Bundle-Activator: com.ezpetri.compiler.eptc.EptcPlugin
Export-Package: com.ezpetri.compiler.eptc,
 com.ezpetri.compiler.eptc.core
```

Pode-se observar que dentre os *plug-ins* listados, existe um que não pertence à plataforma Eclipse, o jdom_plugin. Este *plug-in* foi criado durante o desenvolvimento do compilador para possibilitar o uso da API JDOM pelo *plug-in* EPTC. Esta apresentou-se como a alternativa mais viável, uma vez que não foi possível encapsular a biblioteca de funções do JDOM dentro do próprio *plug-in* EPTC. Não houve dificuldade na criação deste *plug-in*, pois a API JDOM apresenta-se na forma de um arquivo .jar, e a própria IDE Eclipse possui ferramentas capazes de gerar um *plug-in* a partir de arquivos com esta extensão.

5.2 Ambiente para modelagem de redes eDSPN

O módulo que compõe o editor de redes eDSPN foi desenvolvido na forma de um *plug-in* da plataforma Eclipse. O ambiente configura uma modificação do editor de redes de Petri já existente no EZPetri, sendo adicionados novos elementos específicos das redes EDSPN, bem como as modificações nos elementos já existentes para possibilitar a representação de uma rede EDSPN de forma satisfatória. A interface gráfica com o usuário é feita através de uma nova aba, adicionada à perspectiva do EZPetri apenas quando o modelo definido no arquivo PNML aberto seja do tipo "EDSPN". Um "*Wizard*", ou seja, um guia para auxiliar a criação dos novos arquivos a serem utilizados no editor também foi desenvolvido.

5.2.1 Arquitetura

Inicialmente foi realizada uma tentativa de implementar o editor utilizando a tecnologia de *fragments* (fragmentos) existente no Eclipse. Um fragmento pode ser visto como uma parte de um *plug-in*, ou "*plug-in* alvo" (*target plug-in*). As funcionalidades disponibilizadas pelo fragmento são unidas às funções já existentes no *target plug-in*. Sendo assim, o uso de fragmentos adicionando funcionalidades ao *plug-in* já existente (o *Place / Transition Editor*) seria o ideal. Entretanto, esta abordagem foi abandonada, pois as novas funcionalidades exigiam modificações nas classes existentes no editor para que pudessem devidamente representar as características das redes EDSPN definidas pela ferramenta TimeNET. Sendo assim, optou-se por desenvolver um novo *plug-in*, tomando como base a implementação do *Place / Transition Editor*.

O projeto do editor organiza-se nos pacotes listados a seguir:

- ***com.ezpetri.editor.timenet.actions***: contém as classes que representam as ações que podem ser realizadas pelo usuário no editor (Seleção de elementos, zoom, etc.).
- ***com.ezpetri.editor.timenet.command***: contém as classes responsáveis pelas ações de criação e remoção de elementos no modelo.
- ***com.ezpetri.editor.timenet.figures***: agrupa as classes que determinam a aparência dos elementos no editor.
- ***com.ezpetri.editor.timenet.model***: contém as classes correspondentes ao modelo lógico de cada um dos elementos disponíveis para edição.
- ***com.ezpetri.editor.timenet.part***: contém as classes de tratamento das ações para cada elemento de representação.
- ***com.ezpetri.editor.timenet.policies***: classes com o tratamento e comportamento gráfico dos elementos.
- ***com.ezpetri.editor.timenet.util***: classes com funções auxiliares e mapeamentos.

Os mecanismos de controle da disposição gráfica dos elementos permaneceram os mesmos utilizados no editor de redes *P/T Net* existentes no EZPetri, apenas sendo adicionados os novos elementos. O editor utiliza o *Grafical Editing Framework* (GEF), um *plug-in* que auxilia a criação de interfaces gráficas no ambiente Eclipse. O uso do GEF implica na adoção do padrão de projeto *Model View Controller* (MVC), o qual define que funcionalidades diferentes devem ser implementadas em classes diferentes, a fim de manter uma maior modularidade bem como manter um maior isolamento entre a interface gráfica e o modelo lógico.

Optou-se por não utilizar o modelo lógico de representação de redes de Petri estocásticas já existente no módulo do compilador EPTC, visando manter um maior isolamento entre os elementos do projeto. A utilização do mesmo modelo acarretaria na criação de um modelo sobrecarregado, com informações que não seriam relevantes para cada módulo isoladamente. Por exemplo, o modelo do compilador ficaria sobrecarregado com informações relacionadas à exibição dos elementos na tela do editor, o que apenas acrescentaria mais complexidade ao modelo. Da mesma forma, o modelo gráfico não requer informações que são utilizadas durante o processo de tradução entre os diferentes formatos.

A classe `NetModel` representa o modelo lógico da rede de Petri e agrupa todos os elementos específicos de redes EDSPN que possam ser representados graficamente. A composição do modelo lógico é bastante similar à utilizada no compilador, sendo simplificada para manter apenas três tipos de parâmetros: parâmetros de marcação; parâmetros de atraso; e medidas de desempenho. Estes são os únicos parâmetros considerados em redes eDSPN no TimeNET.

Para realizar as etapas de tradução entre o documento PNML e o modelo lógico de representação foi utilizado o *framework* existente no ambiente chamado *EZPetri Translator*. Este *framework* utiliza um mecanismo de mapeamento entre objetos Java e elementos PNML. Através dele, é possível gerar um modelo de rede de Petri em objetos Java a partir de um arquivo PNML, como também efetuar o processo inverso, ou seja, persistir uma rede de Petri representada em objetos Java em um documento PNML. Um diagrama indicando a utilização do *EZPetri Translator* pelo editor gráfico de redes eDSPN é dado na Figura 22.

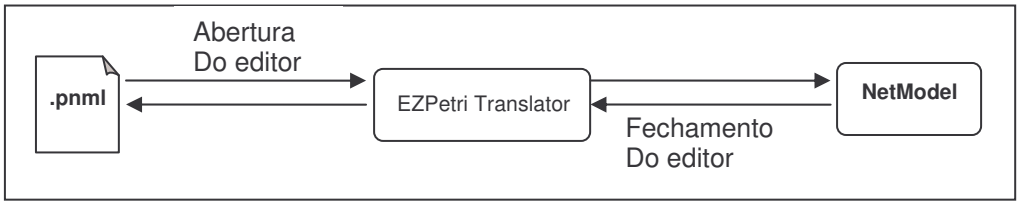


Figura 22. Etapas na tradução do documento PNML no modelo lógico de representação gráfica

Os processos de tradução em ambos os sentidos compartilham um mesmo documento XML contendo os mapeamentos. Um trecho do documento de mapeamentos criado pode ser visto na Figura 23. Maiores informações sobre o *EZPetri Translator* bem como sobre a criação de arquivos de mapeamento, podem ser encontradas em [26]

```
<!-- Immediate Transition -->
<entity tag-name="immtransition" class="ImmTransitionModel" package="model">
  <property attribute="id" name="id" />

  ...

  <entity tag-name="toolspecific[@tool='TimeNET' and @version='3.0']">
    <property name="elemType" tag-child="type/text()" default-value="ImmTrans" />
    <entity tag-name="attributes">
      <property name="priority" attribute="priority" />
      <property name="weight" attribute="weight" />
      <property name="enablingFunction" attribute="md_enable" />
      <property name="isMD" attribute="IsMD" default-value="false" />
      <property name="selcol" attribute="selcol" default-value="blue" />
      <property name="firingPolicy" attribute="fire_pol" default-value="PRS" />
      <entity tag-name="bindings" />
    </entity>

    ...

    <entity tag-name="orientation">
      <property name="orientation" attribute="value" default-value="0" />
    </entity>
  </entity>
</entity>
```

Figura 23. Trecho do arquivo de mapeamentos referente ao mapeamento entre o elemento PNML e a classe de transições imediatas, ilustrando o mapeamento dos elementos *toolspecific*

Entretanto, o *framework* apresenta algumas limitações e dificuldades de uso, uma vez que não existem instruções sobre como devem ser elaborados os mapeamentos. Uma outra grande dificuldade foi a falta de clareza nas mensagens de erro no *parsing* dos arquivos de mapeamento geradas pelo *EZPetri Translator*, uma vez que as mesmas não identificavam em qual *tag* estava localizado o erro. Estas dificuldades só puderam ser transpostas com a realização da depuração do código-fonte do *framework*, disponibilizado junto com o restante do código do EZPetri sob licença GPL (*Gnu Public License*).

5.2.2 Interface com o usuário

A interface com o usuário é dada por uma nova aba adicionada à perspectiva EZPetri, com um editor com de redes de Petri similar ao *P/T Net Editor*, porém com novas possibilidades de elementos em sua paleta. As opções disponibilizadas foram:

- Lugar (*Place*)
- Arco (*Arc*)
- Transições Imediatas (*Immediate Transition*)
- Transições Exponenciais (*Exponencial Transition*)
- Transições Determinísticas (*Deterministic Transition*)
- Transições Gerais (*General Transition*)
- Parâmetro de Marcação (*Marking Parameter*)
- Parâmetro de Atraso (*Delay Parameter*)
- Medidas de Desempenho (*Reward Measure*)

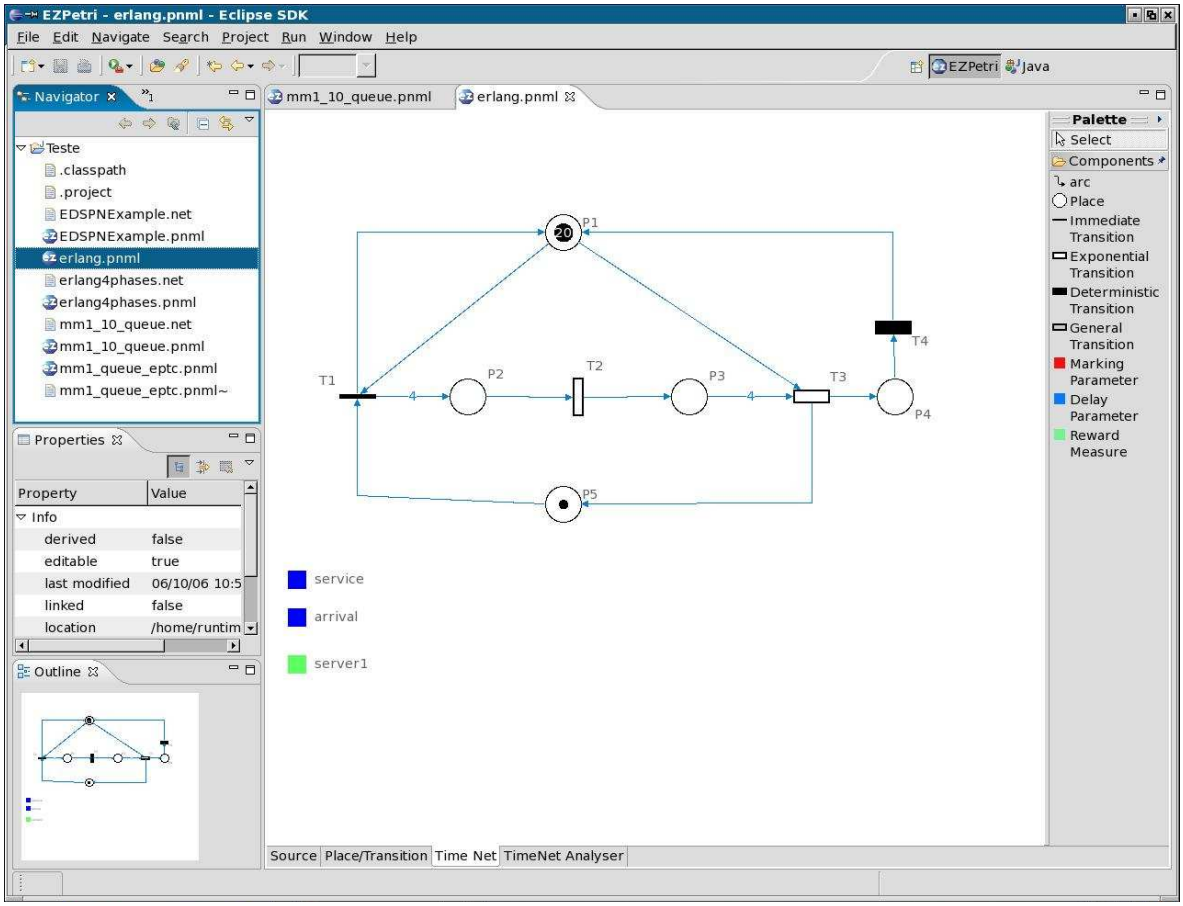


Figura 24. Tela do editor de redes eDSPN integrado ao EZPetri e ao Eclipse

A Figura 24 apresenta um *screenshot* da tela do editor de redes eDSPN desenvolvido. Cada um dos objetos listados possui uma representação gráfica exclusiva e uma lista de propriedades específica, disponível para edição ao usuário através da *Properties View*. A Figura 25 mostra as propriedades disponíveis para edição pelo usuário em uma transição exponencial.

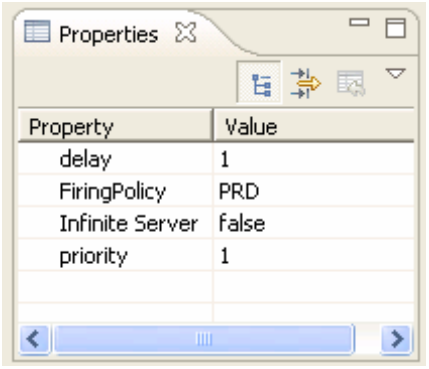


Figura 25. Propriedades listadas para edição quando uma transição exponencial é selecionada no editor

O "*Wizard*" para auxiliar o usuário no processo de criação de arquivos é definido pelo trecho do arquivo plugin.xml listado abaixo. Esta nova funcionalidade é exibida na Figura 26.

```
...
<extension
      point="org.eclipse.ui.newWizards">
   <category
         id="com.ezpetri.editor.timenet.wizard"
         name="TimeNET Wizards"/>
   <wizard
         category="com.ezpetri.editor.timenet.wizard"
         class="com.ezpetri.editor.timenet.wizards.NewEDSPNFileWizard"
         icon="icons/sample.gif"
         id="com.ezpetri.editor.timenet.wizards.NewEDSPNFileWizard"
         name="EDSPN pnml file"/>
</extension>
...
```

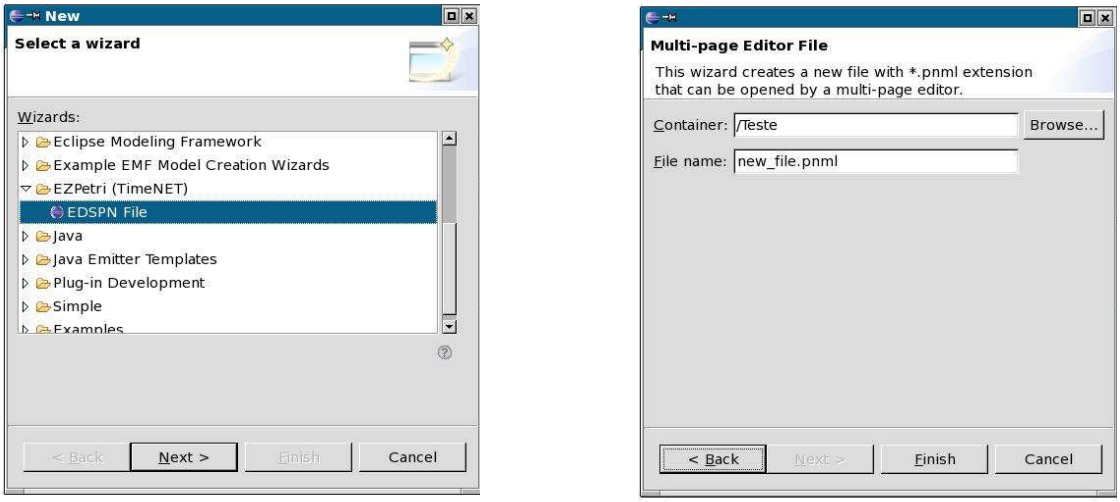


Figura 26. Assistente de criação de arquivos PNML para redes de Petri estocásticas no Eclipse

5.3 Ambiente de Análise de Redes eDSPN

Nesta seção serão apresentados os detalhes da implementação do ambiente para análises de redes de Petri eDSPN. A iniciativa de criação deste ambiente surgiu da observação da grande aceitação da ferramenta TimeNET na comunidade acadêmica como um poderoso mecanismo para realização de análises de redes de Petri estocásticas. Isto se deve, principalmente, à complexa matemática implementada por seus desenvolvedores nos algoritmos de análise desenvolvidos, com especial atenção às redes eDSPN. Os algoritmos de análise também envolvem um complexo processo de geração e compilação de códigos em linguagem C durante a execução dos algoritmos.

O ambiente apresentado por este trabalho não implementa estes algoritmos, em vez disso, o modelo desenvolvido no ambiente é traduzido para o formato aceito pela ferramenta TimeNET. Esta ferramenta é chamada automaticamente e os resultados computados são devidamente exibidos para o usuário sem que se faça necessário sair do ambiente de modelagem do EZPetri / Eclipse.

Este tipo de integração faz-se possível devido às características da implementação da ferramenta TimeNET, a qual, apesar de possuir uma interface gráfica, ainda apresenta uma interface de chamada aos seus *scripts* de análise através de interface de linha de comando, ainda que esta forma de utilização não apresente nenhum tipo de documentação ou suporte por parte dos autores. Um outro trabalho que explora esta possibilidade pode ser visto em [27].

O trabalho aqui apresentado não implementa uma interface para realização de todos os tipos de análise e simulações disponíveis na ferramenta TimeNET, ficando restrito às análises transientes de redes eDSPN. Maiores informações a respeito da análise transiente de redes de Petri estocásticas realizadas pela ferramenta TimeNET podem ser vistos na Seção 3.3 e em outros trabalhos [28] [24].

A implementação iniciou-se por um trabalho de rastreamento dos *scripts* responsáveis pelas análises desejadas, através do acompanhamento das iterações entre o processo da interface gráfica Agnes do TimeNET e a interface de linha de comando. Uma vez identificados quais os *scripts* responsáveis por cada um dos tipos de análise pesquisados, iniciou-se um processo de mapeamento entre os parâmetros oferecidos pela interface gráfica da ferramenta e os parâmetros passados para os algoritmos. Estas informações foram cruzadas com os códigos-fonte dos *scripts* de chamada (implementados em *shell script*) a fim de fortalecer e garantir um correto mapeamento entre os parâmetros.

Nas seções seguintes será apresentada a arquitetura do *plug-in*, cuja estrutura e nomenclatura de classes acompanha o padrão estabelecido pelo EZPetri.

5.3.1 Arquitetura

O *plug-in* apresenta uma estrutura bastante simples, visando à modularidade e facilidade de integração de novos algoritmos de análise no futuro.

O *plug-in* apresenta o diagrama de pacotes ilustrado na Figura 27. A Figura 28 indica os passos seguidos na implementação do mecanismo analisador de redes estocásticas. O pacote *analyzer* contém apenas a classe `TimeNetAnalyzer`, a qual realiza as chamadas aos *scripts* de análise da ferramenta TimeNET. A seqüência descrita inclui a leitura do arquivo PNML selecionado, que é traduzido para o formato TN utilizando o compilador desenvolvido (EPTC). O arquivo TN criado é salvo no diretório MODELS utilizado pelo TimeNET (é uma exigência da ferramenta que os arquivos a serem analisados estejam localizados neste diretório).

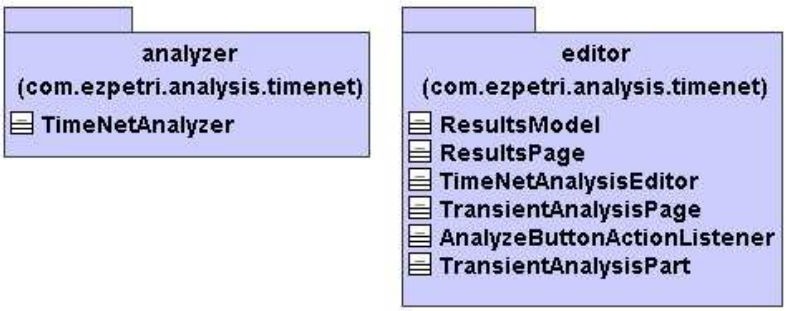


Figura 27. Classes e pacotes do módulo de análise transiente de redes de Petri estocásticas

Uma vez criado o arquivo TN, é iniciada a preparação da chamada ao TimeNET. Isso é feito utilizando a classe `java.lang.ProcessBuilder`, a qual permite que outros processos sejam executados a partir da linha de comando. Uma vez instanciado o processo, uma *thread* de leitura (com o método `readerThread()`) é criada para efetuar o registro do resultado da execução da análise iniciada, armazenando sua saída em uma variável para que seja posteriormente mostrada ao usuário. Por fim, é feita a leitura dos resultados da análise com o método `readResultsFile()` dos parâmetros determinados pelas "*reward measures*", definidas no modelo. Estas medidas são armazenadas pelo TimeNET em um arquivo com o mesmo nome do arquivo TN, mas com a extensão ".RESULTS". O conteúdo deste arquivo é lido e mantido em uma variável temporária.

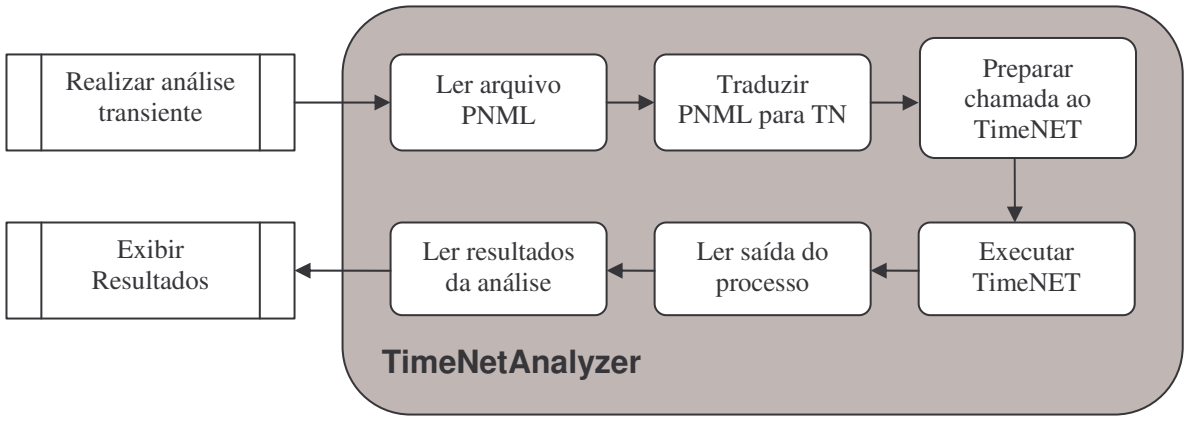


Figura 28. Seqüência de execução do analisador de redes de Petri estocásticas

A exibição dos resultados no ambiente Eclipse é descrita na próxima seção.

5.3.2 Interface com o usuário

A interface com o usuário do módulo de análise é implementada através de um *MultiPageEditor*, com apresentação similar à utilizada no EZPetri para análises de redes Place/Transition através da ferramenta externa INA. De maneira análoga aos outros módulos apresentados, este *plugin* também filtra os arquivos abertos de acordo com a sua extensão bem como o tipo da rede definida no modelo (atributo `type` igual a "EDSPN").

O editor criado possui apenas dois tipos de páginas: a primeira sendo a página de definição dos parâmetros da análise selecionada (transiente em tempo contínuo), exibida na Figura 29; e a segunda corresponde às telas de exibição dos resultados, mostrada na Figura 31.

O primeiro e mais importante parâmetro disponível para edição é o *Transient instant of time*, que determina o tempo de análise considerado, medido em unidades de tempo do modelo. O

segundo, *Precision*, corresponde à precisão requerida para análise. O campo *output form* determina qual a forma de exibição desejada (curva ou pontos). O campo *stepsize for output* controla os pontos para os quais resultados intermediários são computados e exibidos. O resultado pode ser obtido de duas formas, tanto pela opção *repeating Jensen's method* ou por *computing the matrix exponentials*. O valor em *cluster size* determina o número de passos para os quais uma randomização é executada, no caso de randomizações repetidas. O campo *internal stepsize* pode ser ajustado para controlar os pontos de discretização internos.

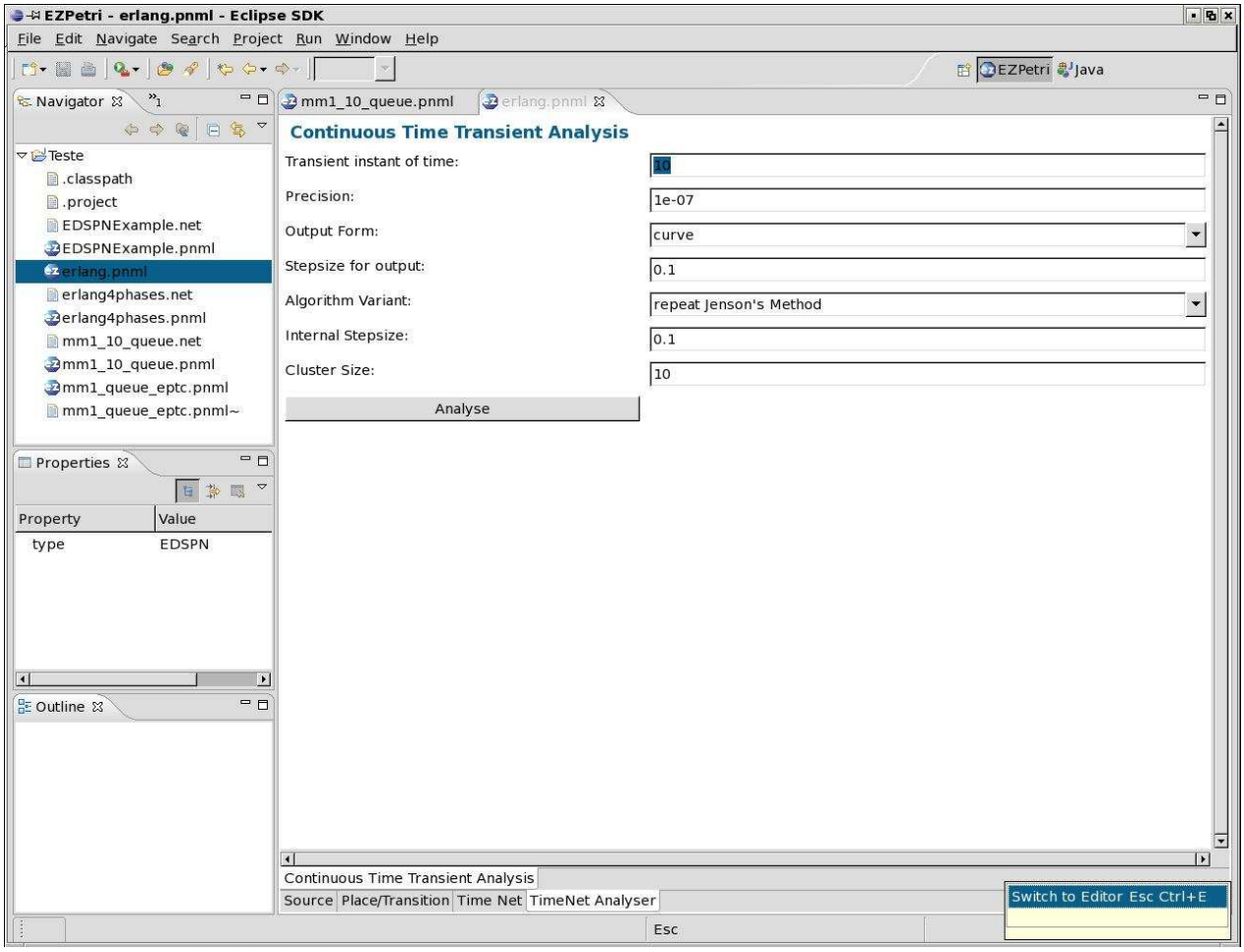


Figura 29. Página de edição dos parâmetros da análise transiente em tempo contínuo

Os parâmetros fornecidos pelo usuário são utilizados para gerar a *String* de chamada ao processo de análise do TimeNET. A cada análise realizada com sucesso, uma nova página é adicionada ao editor apresentando ao usuário com a saída gerada pelo processo e o conteúdo do arquivo de resultados. No caso das análises transientes, também é exibido o gráfico gerado pelo *script* de análise da ferramenta (Figura 30). Tal gráfico, é gerado automaticamente pelo algoritmo de análise através da ferramenta GNUPlot, portanto, não foi necessário implementar tal funcionalidade no ambiente desenvolvido. Sendo assim, permite-se ao usuário ter acesso a todas as informações relativas à análise, sem que seja necessário sair do ambiente EZPetri.

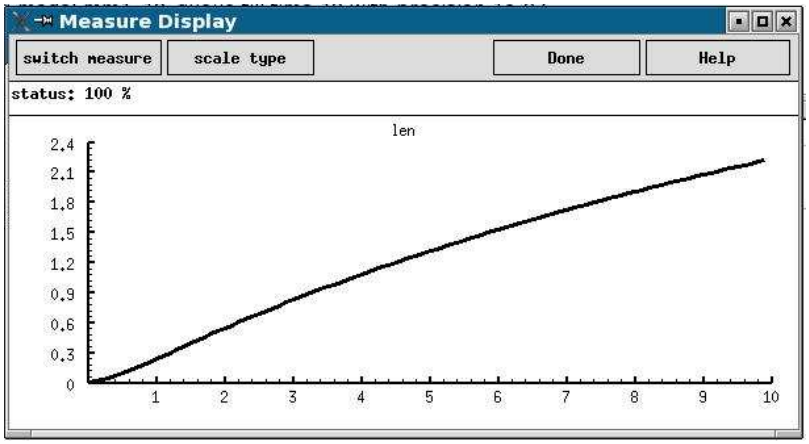


Figura 30. Gráfico gerado pela ferramenta TimeNET nas análises transientes executadas com sucesso

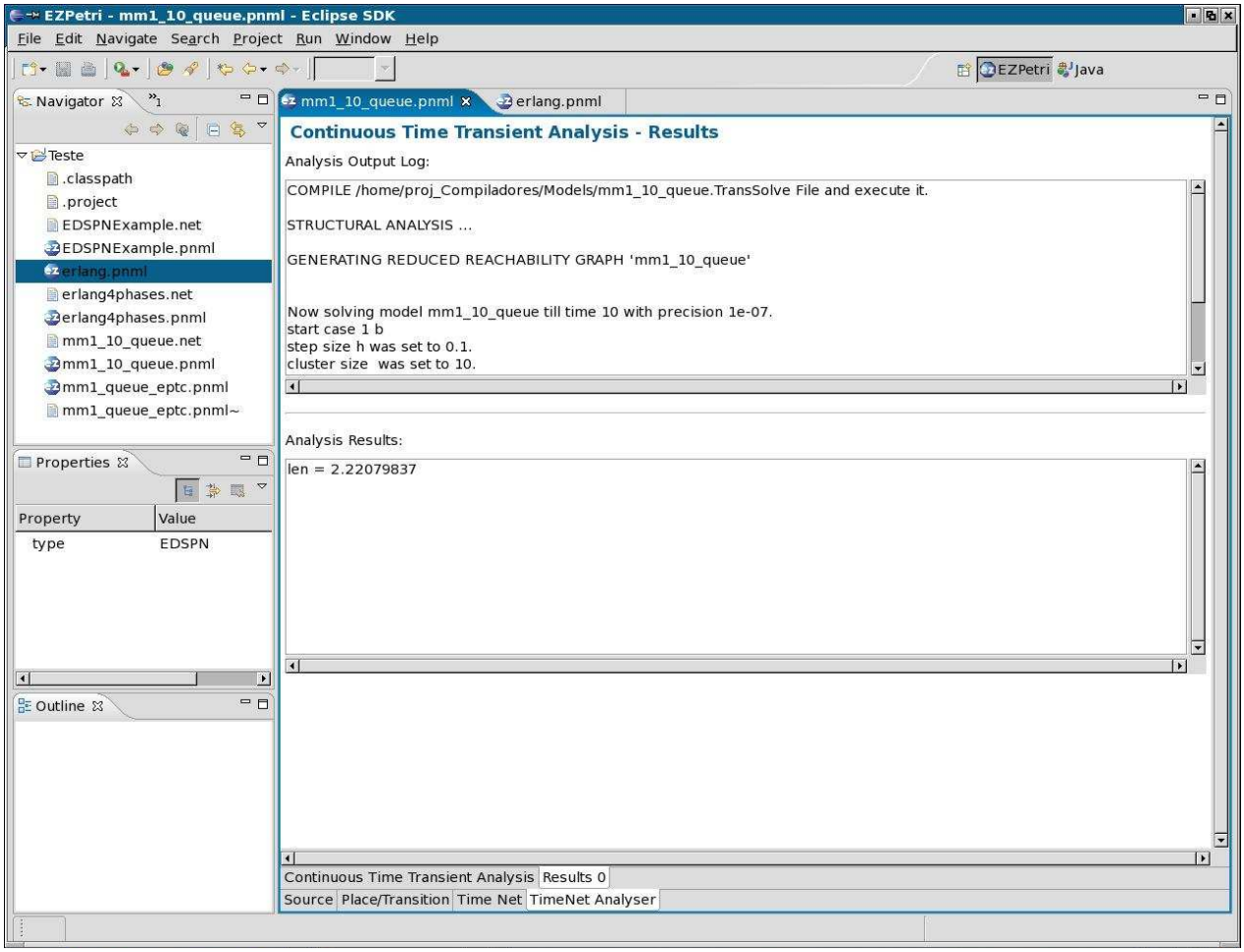


Figura 31. Página com os resultados da análise adicionada ao *MultiPageEditor* do *plug-in* de análise

A cada nova análise realizada, novas instâncias da classe `ResultsModel` são criadas. A classe `ResultsModel` encapsula o modelo de resultados desenvolvido, em que os únicos aspectos relevantes considerados foram o resultado da execução do comando de análise e os resultados dela obtidos. Então, é definida uma nova página, cujo modelo de resultados é o objeto `ResultsModel` criado.

Capítulo 6

Estudo de Caso: Sistema de Filas M/M/1/K

Neste capítulo, será apresentado um exemplo de uso do ambiente desenvolvido, realizando a modelagem e análise transiente de um sistema de filas M/M/1/K. Um modelo de filas M/M/1/K possui iid[3] tempos de intervalo entre chegadas (λ), e também iid tempos de serviço (μ), ambos distribuídos exponencialmente. O sistema possui apenas um único servidor e utiliza uma disciplina de atendimento *First In First Out* (FIFO), apresentando a limitação de suportar apenas um número K de clientes no sistema. Se um novo cliente chegar e o sistema já tiver atingido sua capacidade máxima, o novo cliente será dado como perdido, i.e. desviado para fora do sistema, não podendo mais a ele retornar. Tal sistema pode ser modelado como uma Cadeia de Markov de Tempo Contínuo (CTMC), e representado graficamente pelo diagrama de *state-transition-rate* exibido na Figura 32. Este tipo de diagrama, serve para dar uma noção do fluxo de probabilidades entre os estados do sistema, identificando os possíveis estados do sistema, que, neste caso, denotam o número de clientes no sistema e as taxas de chegada e de serviço em cada um dos estados possíveis. Um sistema de filas M/M/1/K pode ser visto como um sistema de nascimento e morte puro, podendo ser utilizado para modelar sistemas reais (como por exemplo, roteadores), uma vez que possuem *buffer* sempre finito.

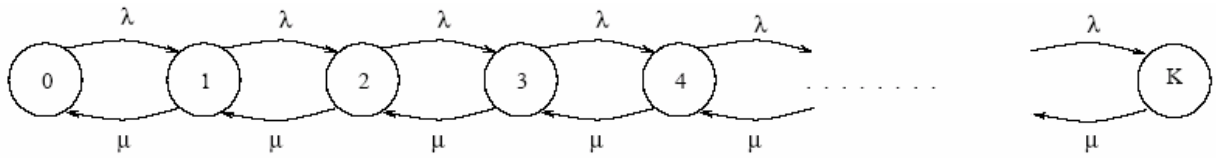


Figura 32. CTMC para um sistema de filas M/M/1/K

Este modelo de fila foi especificado no ambiente de edição de eDSPN desenvolvido, como a rede de Petri indicada na Figura 33, na qual as transições exponenciais T1 e T3 correspondem às chegadas e atendimentos dos clientes na fila, respectivamente, e ambos utilizam a semântica *single-server*. O lugar FilaAtendimento indica a fila de clientes em espera por atendimento,

[3] *independent and identically distributed* (i.i.d.): variáveis em uma seqüência ou coleção de variáveis randômicas são ditas iid se todas possuem a mesma distribuição de probabilidade e são mutuamente independentes.

enquanto o lugar ServidorOcupado representa o cliente em atendimento num dado instante de tempo, e o lugar ServidorLivre indica a disponibilidade do recurso. A transição T2 é do tipo imediata, e é utilizada para alocação do recurso no sistema. Os parâmetros considerados no modelo são: K=10, λ=0.1 e μ=0.125.

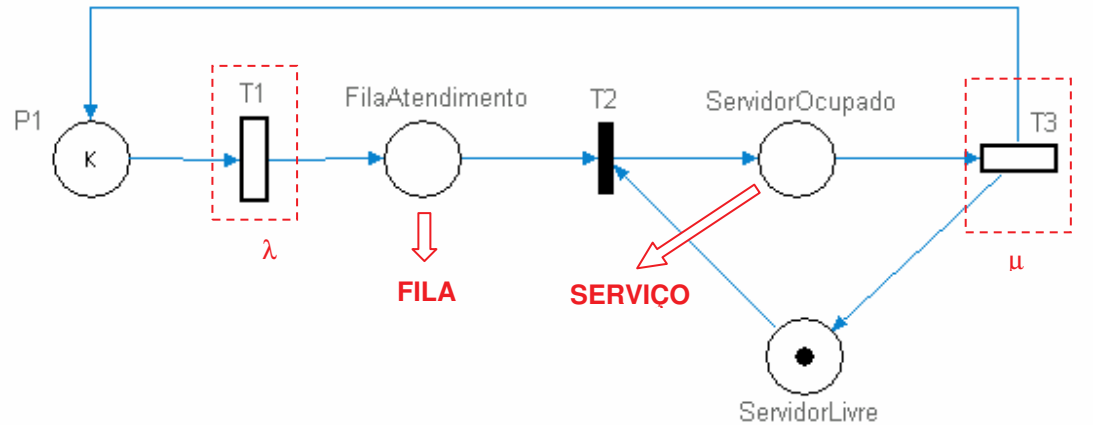


Figura 33. Rede de Petri modelada para representar um modelo de filas M/M/1/K

Uma vez criada a rede, é preciso determinar quais as medidas de desempenho que serão consideradas no sistema. As medidas aqui consideradas são: o tamanho médio da fila durante o intervalo de tempo selecionado (*Avg*) e a taxa de utilização do recurso em questão (*ServerUtil*). Para isso, dois *reward measures* são adicionados à rede. As expressões utilizadas para os parâmetros criados seguem a semântica definida pela ferramenta TimeNET, descrita em [13], e são listadas a seguir.

Avg: **E{#P1}**
ServerUtil: **P{#P2=0}**

Criados os parâmetros de desempenho, a análise do modelo propriamente dita pode ser realizada. Para isso, é selecionada a aba "*TimeNET Analyzer*" do ambiente, e definidos os parâmetros para que a análise seja iniciada. Os parâmetros da análise realizada foram:

Parâmetro	Valor
Transient instant of time	10
Precision	1e-07
Output form	Curve
Stepsize for output	0.1
Algorithm variant	repeat Jenson's method
Internal stepsize	0.1
Cluster size	10

Uma vez que a análise seja executada, um gráfico (Figura 34) com a distribuição dos parâmetros de desempenho no intervalo selecionado é exibido, e uma nova página, contendo a saída do algoritmo de análise executado e o conteúdo do arquivo .RESULTS, é apresentada. O arquivo. RESULTS possui os valores calculados para os parâmetros selecionados. Os valores calculados relativos à utilização do servidor do sistema (parâmetro *ServerUtil*) e para a média de *tokens* na fila de espera (parâmetro *Avg*) foram 0,97646677 e 6,05551463, respectivamente. A partir destes números podemos concluir que a probabilidade do servidor estar ocupado durante o

tempo de observação é da ordem de 97,64%, enquanto o comprimento médio da fila de espera é 6,05 *tokens*.

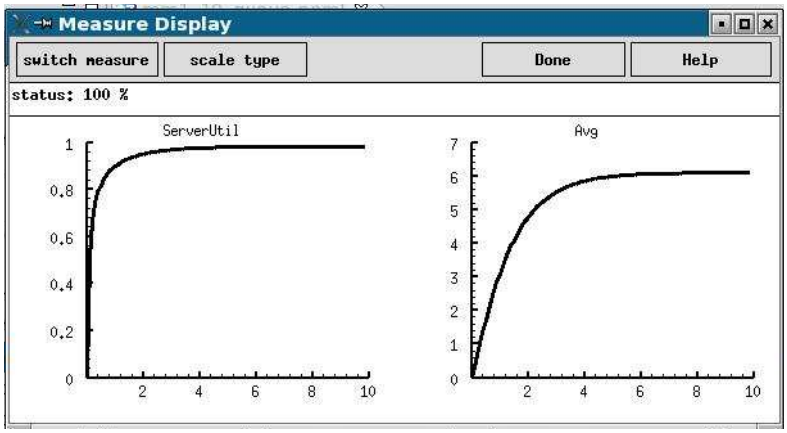


Figura 34. Gráfico indicando o comportamento das varáveis de desempenho

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho apresentou uma extensão ao projeto EZPetri, cuja principal motivação consiste em suprir a lacuna existente na comunidade de usuários de redes de Petri de uma ferramenta capaz de comunicar-se com outras, utilizando para isso uma arquitetura modular e o formato PNML como formato de representação dos modelos.

O trabalho apresentado contribui com o projeto EZPetri, integrando ao ambiente uma ferramenta para redes de Petri estocásticas TimeNET. Como resultado desta integração, foram apresentados os três módulos do ambiente: um compilador capaz de traduzir arquivos do formato PNML para os formatos de representação utilizados pela ferramenta TimeNET (TN e NET), e vice-versa; um editor de redes *Extended Deterministic and Stochastic Petri Nets* – eDSPN, tipo de rede utilizada pelos principais algoritmos de análise do TimeNET; e uma interface para realização de análises transientes em tempo contínuo entre o EZPetri e a ferramenta TimeNET, possibilitando a modelagem e a análise de redes eDSPN de dentro do ambiente EZPetri. Ainda foram desenvolvidas duas contribuições menores, a primeira delas consiste de uma extensão ao formato PNML para a devida representação de redes de Petri estocásticas. A segunda, a descrição dos elementos e atributos existentes no formato NET, proprietário da ferramenta TimeNET, e que até então não possuía nenhum tipo de documentação.

A utilização da linguagem Java tornou possível a usuários não habituados com ambientes *Unix-like* possam criar redes estocásticas com os mesmos parâmetros existentes na ferramenta TimeNET, atualmente apenas disponível para o Unix. O compilador desenvolvido permite também que modelos criados no EZPetri sejam exportados para o formato aceito pelo TimeNET e seus algoritmos de análise.

Um dos módulos desenvolvidos, o *EZPetri TimeNET Compiler* (EPTC), vem sendo utilizado como uma ferramenta em um projeto desenvolvido pelo DSC para a Companhia Hidroelétrica do São Francisco, no projeto “Desenvolvimento de uma metodologia de modelagem e de avaliação da confiabilidade/disponibilidade da infra-estrutura de comunicação”. Isto só foi possível graças a arquitetura de baixo acoplamento adotado no projeto do compilador, que permite a sua adaptação para uso fora do ambiente EZPetri.

Como trabalhos futuros, teríamos a inclusão de outros métodos de análise existentes na ferramenta TimeNET ao ambiente, como, por exemplo, análises estacionárias em tempo discreto e contínuo. Também seria de grande importância a implementação de uma validação das expressões inseridas pelo usuário nos parâmetros da rede, seguindo a semântica já definida pela ferramenta EZPetri.

O formato PNML proposto neste trabalho não seguiu totalmente os padrões previamente adotados no EZPetri. Uma revisão no formato de representação em PNML criado, a fim de compatibilizá-lo com os outros formatos já incorporados ao EZPetri, como os adotados pelas ferramentas INA e PEP, seria de grande valia, pois possibilitaria exportar os modelos de rede de Petri estocásticas criados para estas ferramentas. Claro, que nestes casos, apenas os elementos da rede suportados por estas ferramentas seriam exportados. Aqueles específicos dos modelos estocásticos seriam ignorados.

A inclusão de outros tipos de redes, como as *Hierarchically Coloured Petri Nets* (HCPN) e as *Fluid Stochastic Petri Nets* (FSPN) ao editor desenvolvido está previsto como trabalhos futuros. Esta extensão ampliará ainda mais as funcionalidades disponibilizadas pelo *framework* EZPetri.

Os artefatos produzidos neste trabalho podem ser encontrados no endereço tcc.dsc.upe.br/20062/ezpetri_timenet.zip

Bibliografia

[1] MACIEL, P.R.M. *Introdução às Redes de Petri e Aplicações*. X Escola de Computação, Campinas, SP. 1996.

[2] MARSAN, M.A., BALBO, G., CONTE, G., et al. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, John Wiley and Sons, 1995.

[3] ARCOVERDE, A., ALVES, G., LIMA, R., MACIEL, P., OLIVEIRA, M., Barreto, R.. *EZPetri: A Petri Net interchange framework for Eclipse based on PNML*. In: ISoLA 2004 - 1st International Symposium on Leveraging Applications of Formal Methods. Paphos, Cyprus. 2004.

[4] HEUSER C.A. *Modelagem Conceitual de Sistemas*. (1a. Edição, EBAI - 1988) 2a. Edição, Campinas: UNICAMP (IV Escola Brasileiro-Argentina de Informática), 150 págs., 1991.

[5] RAMCHANDANI C. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Cambridge, MA, 1974.

[6] MERLIN P. M., FARBER D. J. *Recoverability of communication protocols: Implications of a theoretical study*. IEEE Transactions on Communications, 24(9):1036–1043, 1976.

[7] SIFAKIS J. *Petri nets for performance evaluation*. In H. Beilner and E. Gelenbe, editors, Proc. 3rd Intern. Symp. IFIP, páginas 75–93, 1978.

[8] JARP: *Petri Nets Analizer*. Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, http://jarp.sourceforge.net. 2001.

[9] CPNTools. *Computer Tool for Coloured Petri Nets*. http://wiki.daimi.au.dk/cpntools/. Acessado em: 02/11/2006.

[10] STARKE, P. H., ROCH, S. *INA - Integrated Net Analyzer*. Humbolt Universitat zu Berlin, Institut für Informatik. 1999.

[11] KINDLER, E., WEBER, M. *The petri net kernel: An infrastructure for building Petri net tools*. Software Tools For Technology Transfer; DOI 10.1007/s100090100055, Springer Verlag Online First, 2001.

[12] ZIMMERMANN, A.; KELLING, C.; GERMAN, R. HOMMEL, G.: *TimeNET – Toolkit for Evaluating Non-Markovian Stochastic Petri Nets*. Humbolt Universitat zu Berlin, Institut für Informatik. Disponível em: http://pdv.cs.tu-berlin.de/~timenet/. Acessado em: 10/02/2006.

[13] *TimeNET 3.0 User Manual*. Disponível em: http://pdv.cs.tu-berlin.de/~timenet/. Acessado em: 11/02/2006.

[14] BILLINGTON, J., CHRISTENSEN, S., VAN HEE, K. E., KINDLER, E., KUMMER, O., PETRUCCI, L., POST, R., STEHNO, C., WEBER, M.: *The Petri Net Markup Language: Concepts, Technology, and Tools*. In: W. M. P. van der Aalst and E. Best, eds., Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands. 2003.

[15] World Wide Web Consortium (W3C). *Extensible Markup Language* (XML). Disponível em: http://www.w3.org/XML/. Acessado em: 16/03/2006.

[16] *Java Programming Language*. Disponível em: http://java.sun.com. Acessado em: 05/03/2006.

[17] *Eclipse IDE*. Disponível em: http://www.eclipse.org. Acessado em: 05/03/2006.

[18] PETRI, C.A., *Kommunication mit Automaten*. Schriften des IIM Nr.2, Institut für Instrumentelle Mathematik, Bonn, 1962. Traduzida para o inglês como: *Communication with Automata*, Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, Vol.1, Suppl.1, 1966.

[19] MURATA, T. *Petri Nets: Properties, Analysis and Applications*. In: Proceedings of the IEEE, v. 77, n. 4, 1989.

[20] WEBER, M., KINDLER, E. *The Petri Net Markup Language*. In: Petri net Technology Communication Systems. Advances in Petri Nets 2002.

[21] Oasis Commitee. *RELAX NG Specification*. The Organization for the Advancement of Structured Information Standards, 2001. Disponível em: http://www.relaxng.org/spec-20011203.html. Acessado em: 03/05/2006.

[22] GRAHLMANN, B. *The PEP Tool*. Humbolt Universitat zu Berlin, Institut für Informatik. Disponível em: http://theoretica.informatik.uni-oldenburg.de/~pep/. Acessado em: 23/03/2006.

[23] TRIVEDI, K.S., KULKARNI, V.G. *FSPNs: Fluid Stochastic Petri Nets*. In: Proceedings of the 14th International Conference on Application and theory of Petri Nets, Lecture Notes in Computer Science, 1993.

[24] GERMAN R., MITZLA, J. *Transient analysis of deterministic and stochastic Petri nets with TimeNET*, in Proc. 8th Int. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools and MMB (LNCS 977), Heidelberg, Germany, 1995, pp. 209-223.

[25] *Java Document Object Model*. Disponível em: http://www.jdom.org/. Acessado em: 11/02/2006.

[26] ALBUQUERQUE, G.A. *EZPetri - Um Ambiente para redes de Petri no Eclipse*. 2004. Monografia (Graduação) – Curso de Engenharia da Computação, Departamento de Sistemas Computacionais, Universidade de Pernambuco, Recife, 2004.

[27] BEUTEL, B. *Integration of the Petri Net Analysator TimeNET into the Model Analysis Evironment MOSEL*. Master's thesis, Dept. of Computer Science, University of Erlangen-Nürnberg, April 2003.

[28] ZIMMERMANN, A., GERMAN, R., FREIHEIT, J., HOMMEL, G. *TimeNET 3.0 Tool Description*. Int. Conf. on Petri Nets and Performance Models (PNPM 99), Zaragoza, Spain, 1999.

Apêndice A

Gramática do formato TN

Apresentaremos aqui a descrição do formato TN, utilizado pela ferramenta TimeNET como formato de comunicação entre a interface com o usuário e os algoritmos de análise, e por isso também utilizado como formato de saída por este trabalho. Esta gramática foi extraída do manual do usuário do TimeNET 3.0, disponível em http://pdv.cs.tu-berlin.de/~timenet/. Linhas iniciadas com "--" indicam um comentário e são ignoradas.

```
"--FILE" <model name>".TN CONTAINING STRUCTURAL DESCRIPTION OF A NET"

"NET_TYPE:"             "GSPN" | "DSPN" | "EDSPN" | "CDSPN"
"DESCRIPTION:"          <optional comment>
"PLACES:"               <number of places>
"TRANSITIONS:"          <number of transitions>
"DELAY_PARAMETERS:"     <number of delay parameters>
"MARKING_PARAMETERS:"   <number of marking parameters>
"RESULT_PARAMETERS:"    <number of reward measures>

"-- LIST OF MARKING PARAMETERS (NAME, VALUE, (X,Y)-POSITION)"

{"MARKPAR" <name> <integer value> <X position> <Y position>}

"-- LIST OF PLACES (NAME, MARKING, (X,Y)-POSITION (PLACE & TAG))"

{"PLACE" <name> (<marking parameter> | <integer value>) <X position>

 <Y position> <X position of name> <Y position of name>}

"-- LIST OF DELAY PARAMETERS (NAME, VALUE, (X-Y)-POSITION)"

{"DELAYPAR" <name> ("<MD>" | <real value>) <X position> <Y position>}

"-- LIST OF TRANSITIONS"

"-- (NAME, DELAY, ENABLING DEPENDENCE, TYPE, FIRING POLICY, PRIORITY,"
"-- ORIENTATION, PHASE, GROUP, GROUP_WEIGHT,"
"-- (X,Y)-POSITION (TRANSITION, TAG & DELAY), ARCS)"

{"TRANSITION" <name> (<delay parameter> | "<MD>" | <real value>)
          ("IS" | "SS") ("EXP" | "DET" | "GEN" | "IM") ("RS" | "RA" | "RE")
          <priority> <orientation> <phase> <group> <group weight>
          <X position> <Y position> <X position of name> <Y position of name>
          <X position of delay> <Y position of delay>
"INPARCS" <number of input arcs>
<list of input arcs>
```

```
"OUTPARCS" <number of output arcs>
<list of output arcs>
"INHARCS" <number of inhibitor arcs>
<list of inhibitor arcs>}

"-- DEFINITION OF PARAMETERS:"

{ "DELAYPAR_DEF" <parameter name> <param_def> }

"-- MARKING DEPENDENT FIRING DELAYS FOR EXP. TRANSITIONS:"

{ "EXP_DELAY" <transition name> <md_exp_delay> }

"-- MARKING DEPENDENT FIRING DELAYS FOR DET. TRANSITIONS:"

{ "DET_DELAY" <transition name> <md_det_delay> }

"-- PROBABILITY MASS FUNCTION DEFINITIONS FOR GEN. TRANSITIONS:"

{ "GEN_DELAY" <transition name> <pmf_def> }

"-- MARKING DEPENDENT WEIGHTS FOR IMMEDIATE TRANSITIONS:"

{ "WEIGHT" <transition name> <md_weight> }

"-- ENABLING FUNCTIONS FOR IMMEDIATE TRANSITIONS:"

{ "FUNCTION" <transition name> <md_enable> }

"-- MARKING DEPENDENT ARC CARDINALITIES:"

{ "CARDINALITY" {"INPARC" | "OUTPARC" | "INHARC"}
                 <transition name> <place name> <md_arc_mult> }

"-- REWARD MEASURES:"

{ "MEASURE" <name of reward measure> <reward_def> }

"-- END OF SPECIFICATION FILE"
```

Símbolos:

```
<list of arcs> : { (<multiplicity of arc> | "<MD>")
      <connected place> <number of intermediate points>
      {<X position of intermediate point> <Y position of intermediate point>}}

"INPARC"  input arc
"OUTPARC" output arc
"INHARC"  inhibitor arc
```

Abreviações:

"GSPN" (*Generalized Stochastic Petri Net*)
O modelo pode conter transições imediatas e temporizadas exponencialmente.

"DSPN" (*Deterministic and Stochastic Petri Net*)
O modelo pode conter transições imediatas, temporizadas exponencialmente, e determinísticas.

"CDSPN" (*Concurrent Deterministic and Stochastic Petri Net*)
Similar a DSPN, mas permite que mais de uma transição determinística esteja habilitada para uma marcação.

"EDSPN" (*Extended Deterministic and Stochastic Petri Net*)
O modelo pode conter transições imediatas, temporizadas exponencialmente, determinísticas e generalizadas.

"<MD>" (*marking dependent*)
O valor atual depende da marcação indicada.

"IS" (*infinite server*)
A semântica da transição é *infinite server*.

"SS" (*single server*)
A semântica da transição é *single server*.

"EXP" (*exponential*)
O atraso de disparo da transição é distribuído randomicamente com uma função de distribuição exponencial.

"DET" (*deterministic*)
O atraso de disparo da transição é fixo.

"GEN" (*generalized*)
O atraso de disparo da transição é distribuído randomicamente com uma função de distribuição definida pelo usuário.

"IM" (*immediate*)
A transição dispara sem nenhum atraso.

"RS" (*race with resampling*)
Cada mudança de marcação faz com que a transição calcule um novo tempo de disparo a partir da sua distribuição.

"RA" (*race with age memory*)
A transição recalcula um novo tempo de disparo somente após ter sido disparada. O tempo gasto por uma transição enquanto esteve habilitada nunca será perdido.

"RE" (*race with enabling memory*)
A transição recalcula um novo tempo de disparo somente após ter sido disparada ou venha a ser habilitada novamente. Caso a transição esteja desabilitada, o tempo gasto enquanto esteve habilitada é perdido.

<*priority*>
A prioridade deve ser informada apenas para transições imediatas. Em uma dada marcação, apenas as transições com maior prioridade entre todas que estejam estruturalmente habilitadas podem ser disparadas.

<*orientation*>
Fornece a orientação para a representação gráfica de uma transição.

Utilização dos símbolos:

"*symbol*"	símbolo terminal
<*symbol*>	símbolo não-terminal
expr1 | *expr2*	Expressão 1 ou Expressão 2
{*expression*}	Qualquer número de ocorrências da Expressão (incluindo nenhuma)
[*expression*]	Expressão opcional
<md_exp_delay>	Atraso de transição exponencial dependente de marcação
<md_det_delay>	Atraso de transição determinística dependente de marcação
<md_weight>	Atraso de transição imediata dependente de marcação
<md_enable>	*Guard* de transição imediata dependente de marcação
<md_arc_mult>	Multiplicidade de arco de entrada, saída ou inibidor dependente de marcação
<reward_def>	Definição de medida de desempenho
<param_def>	Definição de um paramêtro de atraso que dependa de outros parâmetros
<pmf_def>	Função de distribuíção de atraso de uma transição generalizada

Definições de sintaxe:

```
<md_exp_delay>        : <if_expr>
<md_det_delay>        : <if_expr>
<md_weight>           : <if_expr>
<md_enable>           : <logic_condition> ";"

<md_arc_mult>         : <if_expr>
<reward_def>          : <expression> ";"
<param_def>           : <expression> ";"
<pmf_def>             : <pmf_definition> ";"
<pmf_definition>      : "DETERMINISTIC(" <real_constant> ");"
                      | "UNIFORM(" <real_constant> "," <real_constant> ");"
                      | <pmf_expression> ";"
<pmf_expression>      : <pmf_expression> "+" <pmf_expression>
                      | <pmf_expression> "-" <pmf_expression>
                      | <impulse>
                      | <rectangle>
<impulse>             : [<real_constant> ["*"]] "I[" <real_constant> "]"
<rectangle>           : [ <expolynomial>["*"]]
                        "R[ " <real_constant> "," <real_constant> "]"
<expolynomial>        : <expolynomial> "+" <expolynomial>
                      | <expolynomial> "-" <expolynomial>
                      | "(" <expolynomial> ")"
                      |[-] <real_constant> [["*"] "x"[∧" <integer_constant>]]
                        [["*"]"e ∧ (-" <real_constant> ["*"]"x)"]
<if_expr>             : { "IF" <logic_condition> ":" <expression> }
                        "ELSE" <expression> ";"
                      | <expression> ";"
<expression>          : <real_value>
                      | "-“ <expression>
                      | "(" <expression> ")"
                      | <expression> <num_op> <expression>
<real_value>          : <real_parameter>
                      | <real_constant>
                      | <rew_item> (<reward_def> only)
                      | <integer_value>
<real_parameter>      : <identifier>
<real_constant>       : <digit> { <digit> } "." { <digit> } [ <exponent> ]
                      | { <digit> } "." <digit> { <digit> } [ <exponent> ]
                      | <digit> { <digit> } <exponent>
<exponent>            : ( "E" | "e" ) ( "+" | "-" | "" ) <digit> { <digit> }
<rew_item>            : "P{" <logic_condition> "}"
                      | "P{" <logic_condition> "IF" <logic_condition> "}"
                      | "E{" <marc_func> "}"
                      | "E{" <marc_func> "IF" <logic_condition> "}"
<logic_condition>     : <comparison>
                      | "NOT" <logic_condition>
                      | "(" <logic_condition> ")"
                      | <logic_condition> "OR" <logic_condition>
                      | <logic_condition> "AND" <logic_condition>
<comparison>          : <mark_func> <comp_oper> <mark_func>
<comp_oper>           : "=" | "/=" | ">" | "<" | ">=" | "<="
<mark_func>           : <mark_func> <num_op> <mark_func>
                      | "(" <mark_func> ")"
                      | <integer_value>
<num_op>              : "+" | "-" | "*" | "/" | "∧"
<integer_value>       : <integer_constant>
                      | <integer_parameter>
                      | <marking> (not inside a parameter definition)
<integer_constant>    : <digit> { <digit> }
<integer_parameter>   : <identifier>
<marking>             : "#" <place_name>
<place_name>          : <identifier>
<identifier>          : <letter> { <letter> | <digit> }
<letter>              : "a" | .. | "z" | "A" | .. | "Z"
<digit>               : "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```