Analisador e Simulador de Redes de Petri

Felipe G. de Oliveira Lino[1], Alexandre Sztajnberg[1,2]

[1]Departamento de Informática e Estatística / Instituto de Matemática e Estatística
[2]Programa de Pós-Graduação em Eletrônica / Faculdade de Engenharia
Universidade do Estado do Rio de Janeiro (UERJ)
felipelino44@gmail.com, alexszt@ime.uerj.com.br

Abstract. *Petri Nets can be used on the modeling and verification of several operation system features such as concurrency and process synchronization, conflict verification, resource sharing, among others. This paper describes a tool, on-going development, which allows describing and verifying Petri Nets with the support of a graphical interface. This tool, based on the ARP, is being developed in Java and employs modern object oriented techniques and design patterns. In this way extensions to this tool, such as time constrains and the introduction of new verification strategies can be easily plugged.*

Resumo. *As Redes de Petri têm aplicação na modelagem e verificação de várias características de sistemas operacionais como a concorrência e sincronização de processos, verificação de conflitos, compartilhamento de recursos, entre outros. Este artigo descreve uma ferramenta, em desenvolvimento, que permite a descrição e a verificação de Redes de Petri com o suporte de uma interface gráfica. Esta ferramenta, baseada no ARP, está sendo desenvolvida em Java e emprega técnicas modernas de orientação a objetos e "design patterns". Desta forma extensões a ferramentas, tais como, restrições temporais e a introdução de novas estratégias de verificação, poderão ser plugadas com certa facilidade.*

1. Introdução

Neste artigo apresentamos uma ferramenta visual para a descrição, simulação e verificação de Redes de Petri desenvolvida em Java. O modelo básico da representação interna de uma rede e as verificações são inspiradas no Analisador de Redes de Petri (ARP) [Maz90]. Em nossa implementação o modelo de representação interna do ARP foi adaptado para beneficiar-se da orientação a objetos, e uma interface gráfica foi proposta para facilitar e aumentar a interatividade em todas as etapas de seu uso.

Nas próximas seções destacamos os pontos importantes de nossa implementação, discutimos o modelo de objetos e os pontos possíveis de extensão. Ao longo do texto apresentamos também exemplos de alguns módulos da interface gráfica.

2. Redes de Petri

Redes de Petri (RP, neste texto) são essencialmente grafos *bipartite*, compostos por três elementos principais: lugares, transições e arcos, que inicialmente teve sua aplicação no estudo de autômatos [Mar05]. Devido ao seu potencial de modelagem sua aplicação se estendeu para modelagem de sincronização de processos, concorrência, conflitos, compartilhamento de recursos, entre outros. As RPs modelam basicamente dois

aspectos de um sistema: eventos e condições, bem como as relações entre eles. Através dessas características é possível observar certas condições em cada estado do sistema. Estas por sua vez, podem possibilitar a ocorrência de eventos que podem alterar o estado do sistema. (Por uma questão de espaço não discutiremos detalhes sobre a mecânica de operação das RP)

O formalismo matemático associado às RPs possibilita a verificação de propriedades dos sistemas modelados. Algumas propriedades são:

1. Uma rede é *limitada* se para todas as marcações acessíveis, a partir de uma marcação inicial, o número de fichas em qualquer lugar da rede não exceder *K* (inteiro).
2. Uma rede é *segura* se todos os seus lugares são seguros. Um lugar é seguro se seu número de fichas não exceder a *K*=1. Segurança é um caso especial da propriedade de limitação.
3. Uma rede é *reinicializável* ou *própria* se para todas as marcações possíveis da rede, existir uma seqüência de disparos que leve a rede a marcação inicial.
4. Uma rede é *conservativa* se o número total de fichas na rede se mantém.
5. Uma rede é *viva* quando todas as transições são vivas. Uma transição é viva se para toda marcação alcançável, existe uma seqüência de disparos, tal que a mesma torne-se habilitada.
6. Ocorre um *bloqueio* na rede quando uma transição ou conjunto de transições não dispara. Um caso especial de bloqueio é o *deadlock*, quando nenhuma transição está habilitada para disparo.

A capacidade de verificarem-se a presença ou ausência destas propriedades em uma rede permite a análise do sistema-alvo modelado. Em nossa ferramenta, além do suporte a edição gráfica de RPs, várias destas propriedades podem ser verificadas.

3. Trabalhos correlatos

Em [PNW06] é mantido um repositório sobre Redes de Petri. Entre as informações, uma lista abrangente de ferramentas é disponibilizada, juntamente com uma análise das características relevantes de cada uma delas. A Tabela 1 compara algumas ferramentas com objetivos e características semelhantes à nossa, ou seja, construídas em Java que trabalham com RP clássicas.

Tabela 1. Comparação de algumas ferramentas para Redes de Petri

Atributos	JARP	PetriTool	jPNS	Nosso trabalho
Composição visual	➨	➨	➨	➨
Simulação interativa	➨	➨	➨	➨
Simulação automática		➨	➨	
Geração de árvore alcançabilidade		➨		➨
Verificação de propriedades	➨	➨		➨
Resultados gráficos				➨
Persistir rede	➨	➨	➨	➨

O Analisador de Redes de Petri (ARP) [Maziero 1990], desenvolvido em Pascal para o sistema operacional DOS 3.0 contém várias ferramentas para o uso de RP, como um editor textual, módulos de verificação, e módulos para o uso de temporização e com temporização estendida. A estrutura do ARP, algoritmos para verificação, e partes do seu código foram utilizadas como base para a nossa ferramenta.

Em nossa ferramenta procuramos aliar a interatividade, desde o editor até o "token animation game", com a capacidade de simulação passo a passo, a geração da árvore de alcançabilidade, e a verificação de várias características. No estado atual, nossa ferramenta ainda não contempla a simulação e análise de RPs com características temporais (embora o arcabouço necessário esteja preparado, Seção 4.3), estocásticas ou de predicados no disparo de transições. Procuramos também facilitar a extensão através dos mecanismos de herança e reflexão estrutural de Java. As RP editadas são persistidas em XML, o que facilita também a interoperabilidade com outras ferramentas.

4. Implementação

Nossa implementação é baseada na tecnologia de orientação a objetos fazendo uso de técnicas como herança, encapsulamento e polimorfismo, com o objetivo de prover reusabilidade e extensibilidade ao código do programa. Utilizamos a linguagem Java. A parte gráfica utiliza o pacote *Swing* e a API *Forms* fornecida pela *JGoodies* [JGoodies 2006]. Outra API, também *open source*, utilizada foi a XStream [XStream 2006], uma biblioteca simples para serializar objetos (no caso, os objetos representando a RP sendo editada) em XML, bem como para recompor os objetos a partir de sua representação "flat" em XML. Para nossa implementação criamos um pequeno *framework* para suporte multilíngüe, basicamente fazendo uso de arquivos *properties*, classes de interface e uma classe que liga esses elementos. Atualmente o programa possui como idiomas opcionais o português brasileiro e o inglês.

4.1. Arquitetura e Estrutura do Código

A Figura 1 apresenta a arquitetura geral da ferramenta. A base da ferramenta é o modelo de objetos, que separa a parte gráfica e a análise. Através dessa separação é possível, por exemplo, reutilizar a *engine* de simulação e análise e adaptá-la a outra interface gráfica, ou utilizar a representação de uma RP com outras ferramentas de análise. O modelo de uma RP consiste de classes-base para representar os elementos lugar, transição, arco e Rede de Petri. Através de herança são criadas classes especializadas com atributos e métodos necessários para a interface gráfica. Este modelo é utilizado pelos outros módulos da ferramenta.

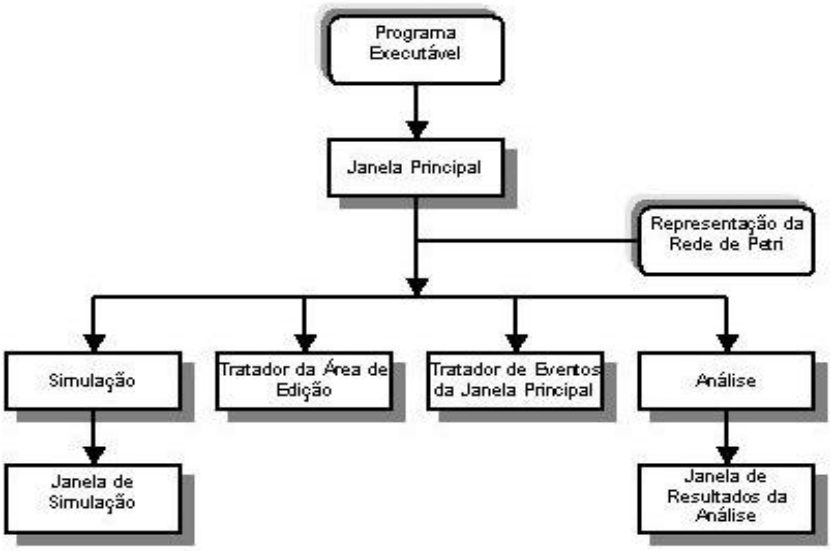

Figura 1. Arquitetura da ferramenta

Na camada seguinte temos, então, os módulos que operam sobre o modelo de objetos da rede através da captura de eventos e emissão de notificações para os módulos de exibição. Assim, temos o Tratador da Área de Edição (*PetriNetEditorCanvas*), Tratador da Janela Principal (*MenuActionListener*, *ButtonActionListener*, *PlaceActionListener*, *ArcActionListener* e *TransActionListener*), Simulação (*SimulationAction*), Análise (*AnalyzerAction*).

Na última camada temos os módulos de exibição: Janelas de Simulação e Resultados da Análise, que apresentam o resultado dos eventos de simulação e análise.

Observa-se que as classes referentes à análise e simulação independem da interface gráfica e precisam apenas das informações contidas nos objetos-base do modelo de uma rede. A estrutura do código se divide em quatro tipos: (i) classes-base, usadas para trafegar informações entre a interface gráfica e a *engine*; (ii) classes da interface com usuário; (iii) classes da *engine*; e (iv) classes mediadoras. Na Figura 2, por exemplo, temos o diagrama de classes do caso da *engine* de simulação. As ações que dirigem a simulação são capturadas através da classe *SimulationAction*, que também notifica as ações para a interface gráfica, classe *SimulationWindow*, e para a classe que representa a rede estendida, *PetriNetGraph* (que contém informações de posicionamento, cor, etc. dos elementos básicos da rede). A simulação, propriamente dita é realizada pela classe *ImplSimulator*, que opera sobre a classe *ImplPetriNetBase*. Diagrama semelhante se aplica ao caso da análise. Detalhes sobre o código, pacotes Java e classes podem ser encontrados em [Lin07].

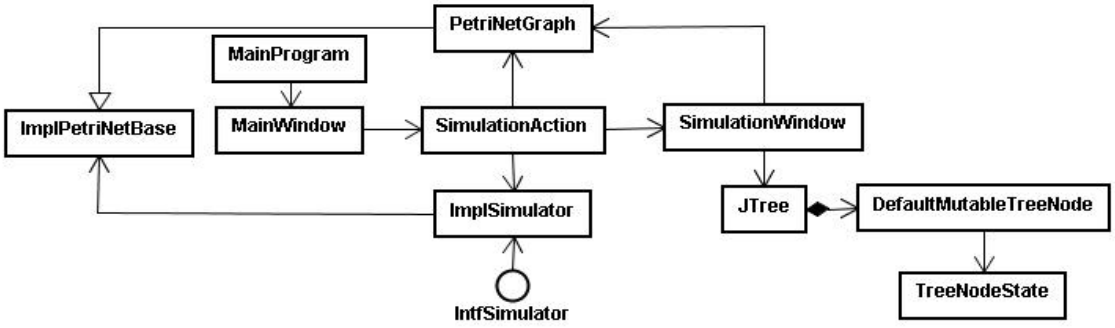

Figura 2. Separação entre a Simulação e o Modelo da Rede

4.2. Interface Gráfica

A Figura 3 apresenta alguns exemplos de uso das ferramentas, destacando-se o editor gráfico. (a) apresenta detalhes da edição de arcos e transições. (b) e (c) apresentam a janela principal, com os menus de edição/ação, redes editadas e uma janela de simulação. (d) apresenta resultado de análise.

4.3. Extensões

As classes que modelam o núcleo de uma Rede de Petri possuem atributos e métodos atualmente não utilizados pelo aplicativo já visando uma extensão. Por exemplo, o código já está preparado para Redes de Petri Temporizadas segundo o modelo de Merlin [Merlin 1976]. A classe *TransitionBase* possui os atributos: *CurvaDensidade*; *StaticEarliestFiringTime* e *StaticLatestFiringTime*, facilitando a configuração destas informações.

Novas análises podem ser adicionadas consultando-se a representação matricial da Rede de Petri, fornecida pelos métodos *getInputMatrix*, *getOutputMatrix* e *getIncidenceMatrix*. Para mais verificações de propriedades, devem ser alteradas ou estendidas as classes *PetriNetAnalyzer* e *PetriNetProperties*. Todas as verificações feitas atualmente na rede, são executadas durante a construção da árvore de alcançabilidade, outras podem ser feitas após a construção da árvore o que exige que ela seja percorrida algumas vezes. Como as classes de simulação e análise da Rede foram construídas apenas usando-se a modelagem dos elementos básicos da rede, elas são completamente reutilizáveis.

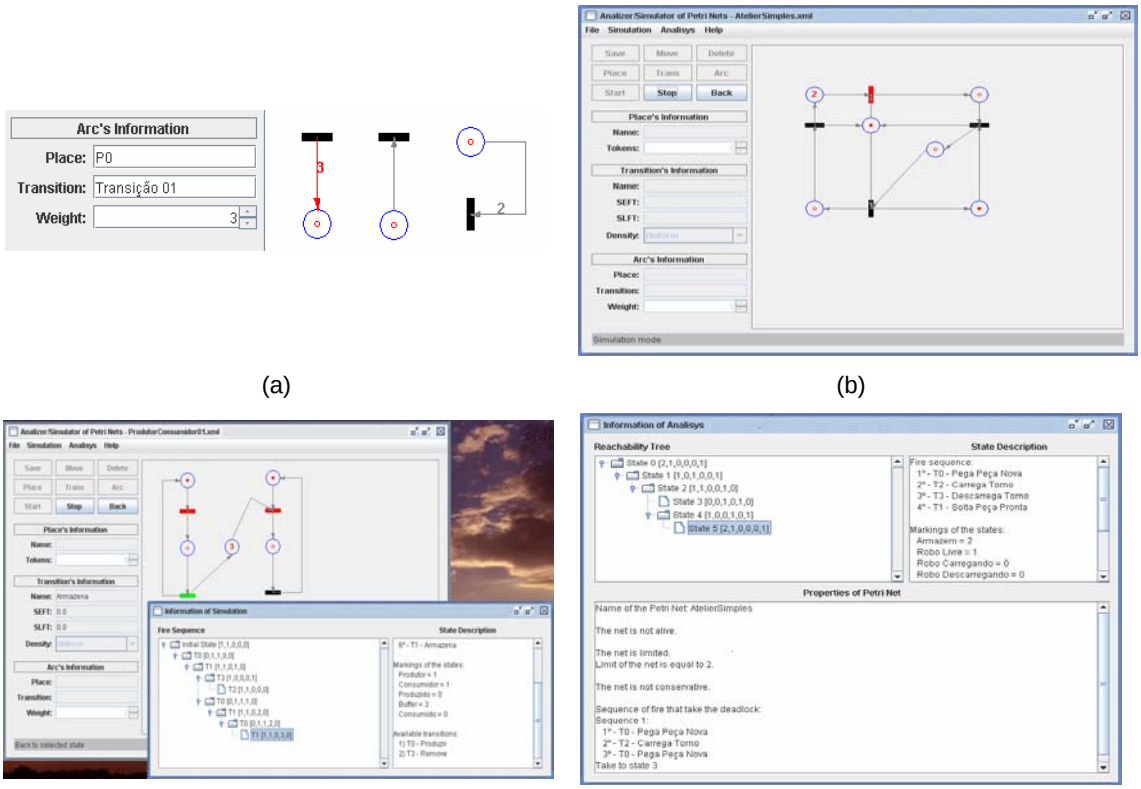

Figura 3. Interface gráfica, exemplos de uso

O formato do arquivo XML gerado para persistir uma rede editada assemelha-se à estrutura da classe *PetriNetGraph*, uma vez que o arquivo persistido é espelho da classe. Contudo, pelas classes envolvidas na formação de uma RP possuírem apenas os atributos essenciais, o arquivo gerado torna-se parecido com outras representações de RP em XML, como *PNML* por exemplo [PNML 2005]. Assim sendo, é simples capturar os atributos principais de um arquivo em um formato qualquer e colocá-los na representação de XML de nossa implementação. O inverso também é possível, mas sempre dependerá da complexidade do formato do arquivo alvo.

5. Considerações finais

A ferramenta apresentada está em desenvolvimento. O núcleo do código foi estruturado para facilitar a inclusão de novas características, tanto em relação ao modelo da rede, quanto às análises e a própria interface gráfica. Como exemplo, estamos trabalhando na inclusão de características temporais nas RP editadas. Como discutido, outras extensões como RP estocásticas ou redes coloridas podem ser adicionadas.

A ferramenta desenvolvida está sendo utilizada no curso de Sistemas Operacionais no DICC/UERJ e no curso de Sistemas Concorrentes do PEL/UERJ. Através desta ferramenta os alunos simulam e verificam exemplos clássicos como "produtores-cosumidores", "leiores-escritores", "ceia dos filósofos" e outros programas concorrentes desenvolvidos ao longo do curso. A ferramenta também está sendo utilizada para simular protocolos de comunicação em que a expressividade das RPs clássicas é suficiente. Esperamos também atrair novos alunos para continuar o projeto.

O código da ferramenta será aberto para a comunidade interessada em trabalhar em extensões e, obviamente a ferramenta será disponibilizada livremente. Na medida em que a ferramenta seja consolidada, iremos disponibilizar a mesma, e seu respectivo código, através de um sítio na Internet. Contribuições ao projeto serão bem-vindas.

Agradecimentos. Gostaríamos de agradecer o Prof. Carlos Maziero (PUC-PR) pelo código fonte do ARP e sua documentação. Agradecemos o apoio parcial da Faperj (APQ1 E-26/171.130/2005).

Referências

Cardoso, J. e Valette, R. (1997). "Redes de Petri", Editado na UFSC.

de Souza Leão, J. L. (2004). "Programação e Validação de Sistemas Multitarefa – Capítulo 5 Redes de Petri", VI , 76 p, 29,7cm (Rio de Janeiro) COPPE/UFRJ, http://www.gta.ufrj.br/%7Eleao/coe717-2004-1/.

Gamma, E., Helm, R., Johnson, R. e Vlissides, J. (2003) "Design Patterns Elements of Reusable Object-Oriented Software", Edited by Addison Wesley.

Lino, F. G (2007). "Implementação de uma ferramenta gráfica para Redes de Petri", Monografia de Graduação, DICC/IME – UERJ.

JGoodies (2006). http://www.jgoodies.com.

Marranghello, N. (2005). "Redes de Petri: Conceitos e Aplicações", http://www.dcce.ibilce.unesp.br/~norian/cursos/mds/ApostilaRdP-CA.pdf.

Maziero, C. (1990). "ARP - Analisador de Redes de Petri". http://www.ppgia.pucpr.br/~maziero/diversos/petri/.

Maziero, C. (1990). "Um ambiente para a análise e simulação de sistemas modelados por redes de Petri", Tese de Mestrado, Universidade Federal de Santa Catarina..

Merlin, P.M. e Farber, D.J (1976). "Recoverability of Communication Protocols – Implication of a Theoretical Study". IEEE Transactions on Communications, págs.1036-1043.

Peterson, J. L. (1981). "Petri Net Theory and the Modeling of Systems", Prentice-Hall, .J., 1981, ISBN: 0-13-661983-5.

Petri Nets World (2006). http://www.informatik.uni-hamburg.de/TGI/PetriNets/ TGI group at the University of Hamburg, Germany.

PNML (2005). Software and Systems Engineering – High-level Petri Nets, Part 2: Transfer Format. International Standard ISO/IEC 15909-2. Working Draft Version 0.9.0, June 2005. (Submitted for a ISO/IEC SC7 WD/CD registration and CD ballot).

XStream (2006). http://xstream.codehaus.org.