

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**FERRAMENTA PARA SIMULAÇÃO VISUAL
DE REDES DE AUTÔMATOS ESTOCÁSTICOS
ATRAVÉS DO CÁLCULO DE ESTADOS
SUCESSORES E PREDECESSORES**

ALBERTO SALES E SILVA

Dissertação de Mestrado apresentada como requisito para obtenção do título de Mestre em Ciência da Computação pelo Programa de Pós-graduação da Faculdade de Informática.

Orientador: Paulo Henrique Lemelle Fernandes

**Porto Alegre
2011**

Dados Internacionais de Catalogação na Publicação (CIP)

S586f	Silva, Alberto Sales e Ferramenta para simulação visual de redes de autômatos estocásticos através do cálculo de estados sucessores e predecessores / Alberto Sales e Silva. – Porto Alegre, 2011. 97 f. Diss. (Mestrado) – Fac. de Informática, PUCRS. Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes. 1. Informática. 2. Simulação e Modelagem em Computadores. 3. Redes de Autômatos Estocásticos. I. Fernandes, Paulo Henrique Lemelle. II. Título. CDD 003.3
-------	--

**Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS**



TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Simulação Visual de Redes de Autômatos Estocásticos**", apresentada por Alberto Sales e Silva, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 21/03/2011 pela Comissão Examinadora:

Prof. Dr. Paulo Henrique Lemelle Fernandes -
Orientador

PPGCC/PUCRS

Prof. Dr. Fernando Luís Dotti -

PPGCC/PUCRS

Dr. Afonso Henrique Corrêa de Sales -

FACIN/PNPD

Homologada em 04 / 01 / 2012, conforme Ata No. 004 / 2012 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS

Campus Central

Av. Ipiranga, 6681 - P32 - sala 507 - CEP: 90619-900

Fone: (51) 3320-3611 - Fax (51) 3320-3621

E-mail: ppgcc@pucrs.br

www.pucrs.br/facin/pos

*“Não há nada mais difícil de controlar,
mais perigoso de conduzir ou mais incerto em seu
sucesso do que liderar a introdução
de uma nova ordem das coisas.”
(Maquiavel, O Príncipe)*

*“Deus criou os inteiros,
tudo então é o trabalho do homem.”
(Leopold Kronecker)*

AGRADECIMENTOS

Agradeço à minha esposa Nádia e ao meu filho George pela compreensão e paciência. Foram difíceis os momentos de dedicação à este trabalho, sem a colaboração da minha família não seria possível o desenvolvimento deste trabalho. Ao meu orientador que, pela sua simplicidade, facilitou o meu trabalho e por tudo que eu aprendi. Obrigado por ter me aceitado como orientando. Agradeço aos pesquisadores do grupo de pesquisas PEG-PUCRS, dr. Afonso Sales e dra. Thais Webber, por me apoiarem e me ajudarem com imensa paciência e dedicação quando me explanaram sobre o tema da pesquisa deste trabalho. Tenho imensa gratidão por vocês. Não posso esquecer de mencionar o colega dr. Ricardo Czekster por ter esclarecido alguns aspectos do mundo das bolinhas.

Muito obrigado a DELL pela bolsa fornecida que tanto me ajudou nesta jornada.

Aos demais colegas, professores e funcionários do programa de pós-graduação, obrigado pelo apoio, paciência e sugestões.

À minha esposa e filho dedico esta dissertação.

FERRAMENTA PARA SIMULAÇÃO VISUAL DE REDES DE AUTÔMATOS ESTOCÁSTICOS ATRAVÉS DO CÁLCULO DE DE ESTADOS SUCESSORES E PREDECESSORES

RESUMO

O objetivo deste trabalho é fornecer uma ferramenta para simulação visual de SAN. O formalismo SAN, através da ferramenta PEPS, utiliza soluções numéricas para calcular erros de avaliações condicionais ou comportamento não esperado de sistemas modelados através deste formalismo. Estas soluções numéricas são a base para os formalismos estruturados na medida em que fornecem resultados numéricos por meio de relações matemáticas. Complementar a eficiência de soluções numéricas com a simulação visual é bastante interessante, pois adiciona informações mais detalhadas sobre o modelo o que facilita que usuários acadêmicos iniciantes ou pesquisadores tenham um maior entendimento da aplicabilidade do formalismo estruturado. Esta dissertação descreve uma ferramenta para modelagem e simulação visual de SAN que em sua estrutura define um esquema de armazenamento compacto para a matriz de transição da cadeia de Markov e usa a álgebra tensorial para lidar com as multiplicações de vetores de base da matriz. Esta ferramenta permitirá aos usuários acadêmicos ou entusiastas do formalismo SAN manipular modelos sem a preocupação de um domínio profundo dos conceitos deste formalismo estruturado.

Palavras-chave: Redes de Autômatos Estocásticos; Soluções Numéricas; Simulação Visual.

TOOLS FOR VISUAL SIMULATION OF STOCHASTIC AUTOMATA NETWORKS USING CALCULATIONS OF PREDECESSORS AND SUCCESSORS STATES

ABSTRACT

This study aims to develop a tool to visually simulate Stochastic Automata Networks (SAN). The SAN formalism uses the PEPS tool for numerical solution, evaluating unexpected behaviors. These numerical solutions are the core for structured formalisms since they provide numerical results based on mathematical relationships. It is very interesting to complement the numerical solution with visual simulations because it adds more detailed information about models, making them more understandable for broader audiences of researchers and general users. The present dissertation describes a tool for modeling and visual simulation of SAN models, since its internal structure defines a compact storage schema (a Descriptor) for the transition matrix representing the underlying Markov Chain and uses tensor algebra to deal with the vector-descriptor multiplication. The proposed tool will allow both academic users and enthusiasts of the SAN formalism to manipulate models abstracting a deeper knowledge of this structured formalism.

Keywords: Stochastic Automata Networks; Numerical Solutions, Exact Simulation.

LISTA DE FIGURAS

Figura 2.1	O esquema do PEPA Workbench	29
Figura 2.2	Interface textual da ferramenta PEPS2009	31
Figura 3.1	Modelo SAN com dois autômatos e sua tabela de eventos [BRE04a]	35
Figura 3.2	Autômato global equivalente ao modelo SAN com dois autômatos [BRE04a]	36
Figura 3.3	Estrutura Modular do formato textual da SAN	40
Figura 3.4	Modelo de compartilhamento de recursos	43
Figura 3.5	Modelo do primeiro servidor disponível	45
Figura 4.1	Exemplo de representação de um espaço de estados \mathcal{S} por um MDD	50
Figura 4.2	Modelo SAN com 4 autômatos [SCO06]	51
Figura 4.3	Exemplo de MDD para modelo SAN com 4 autômatos [SCO06]	51
Figura 4.4	Modelo de compartilhamento de 4 recursos	52
Figura 4.5	Representação de um espaço de estados \mathcal{S} por um MDD, referente à Figura 4.4	53
Figura 4.6	Exemplo de MDD referente à Figura 4.4 com estados inatingíveis omitidos .	54
Figura 5.1	Modelo SAN com 2 autômatos.	57
Figura 5.2	Modelo SAN com 2 autômatos com taxa funcional f_1	60
Figura 5.3	Modelo SAN com 2 autômatos	62
Figura 5.4	Modelo SAN com 2 autômatos	64
Figura 5.5	Exemplo de SAN com 2 autômatos	65
Figura 6.1	Estrutura da ferramenta	69
Figura 6.2	Editor da ferramenta VisualSAN	72
Figura 6.3	Editor de taxas funcionais da ferramenta VisualSAN	73
Figura 6.4	A ferramenta VisualSAN	74
Figura 6.5	Ferramenta VisualSAN com modelo SAN	75
Figura 6.6	SAN com estados iniciais (círculo duplo) e estados sucessores (círculo pontilhado)	76
Figura 6.7	Modelo SAN como novos estados atuais e estados sucessores	76
Figura 6.8	exemplo de simulação de eventos sincronizantes	76
Figura 6.9	Exemplo de simulação de eventos sincronizantes e taxa funcional	77
Figura 6.10	Editor de taxas funcionais com exemplo referente a Figura 6.9	77
Figura 6.11	Exemplo de modelo SAN de eventos com taxa funcional	78
Figura 6.12	Exemplo de modelo SAN de eventos com taxa funcional simulando FAS . . .	79
Figura 6.13	Taxas funcionais referente ao modelo da Figura 6.12	79
Figura 6.14	Tela do editor do arquivo ".san"referente ao modelo da Figura 6.12	80
Figura A.1	Ambiente de edição da ferramenta VisualSAN	91
Figura A.2	Botão de edição de componentes	92

Figura A.3	Botão de criar estados	92
Figura A.4	Botão de criar transições	92
Figura A.5	Botão de deleção de componentes	92
Figura A.6	Botão para adicionar autômatos	93
Figura A.7	Botão para execução	93
Figura A.8	Botão acionador da tela de edição do arquivo ".san"	93
Figura A.9	Botão de gravação do modelo SAN em formato XML	93
Figura A.10	Adicionando autômato no ambiente de edição de SAN	94
Figura A.11	Adicionando estado no ambiente de edição de SAN	94
Figura A.12	Adicionando transição no ambiente de edição de SAN	95
Figura A.13	Identificando uma transição no ambiente de edição de SAN	95
Figura A.14	Transição inserida no ambiente de edição de SAN	96

LISTA DE TABELAS

Tabela 4.1	Lista de estados globais atingíveis do modelo da Figura 4.4	52
Tabela 5.1	Matriz de transição para estados sucessores	57
Tabela 5.2	Matriz de transição transposta, da Tabela 5.1, para estados predecessores . .	57
Tabela 5.3	Matriz de transição referente a Figura 5.2	62
Tabela 5.4	Matriz de transição transposta referente a Figura 5.2	63
Tabela 6.1	Lista de estados globais atingíveis do modelo da Figura 6.12	80

LISTA DE ALGORITMOS

Algoritmo 5.1 - Procedimento de obtenção de estados sucessores (s)	61
Algoritmo 5.2 - Procedimento de obtenção de estados predecessores ($s^{(l)}$)	63

LISTA DE SIGLAS

SAN	<i>Stochastic Automata Network</i>
RSS	<i>Reachable State Space</i>
PSS	<i>Product State Space</i>
MDD	<i>Multi-valued Decision Diagram</i>
PEPA	<i>Performance Evaluation Process Algebra</i>
CTMC	<i>Continuous-Time Markov Chain</i>
HBF	<i>Hardwell-Boieng Format</i>
FAS	<i>First Available Server</i>
BDD	<i>Binary Decision Diagram</i>
SPN	<i>Stochastic Petri Nets</i>
API	<i>Application Programming Interface</i>
FA	<i>Finite Automata</i>
PDA	<i>Pushdown Automata</i>
DFA	<i>Deterministic Finite Automata</i>

SUMÁRIO

1. INTRODUÇÃO	25
1.1 Simulação de Sistemas	26
1.1.1 Simulação Visual	26
1.1.2 Simulação Baseada em Soluções Numéricas	27
1.2 Objetivos	27
1.3 Estrutura dos Capítulos deste Trabalho	28
2. TRABALHOS RELACIONADOS	29
2.1 PEPAnets	29
2.2 PEPA Workbench	29
2.3 GreatSPN	30
2.4 <i>Performance Evaluation of Parallel Systems - PEPS</i>	30
3. REDES DE AUTÔMATOS ESTOCÁSTICOS	33
3.1 Autômatos Estocásticos	33
3.1.1 Autômatos Estocásticos sem Interação	34
3.1.2 Autômatos Estocásticos com Interação	34
3.2 Estados Locais e Globais	35
3.3 Transições Locais e Sincronizantes	36
3.4 Taxas Funcionais e Probabilidades Funcionais	37
3.4.1 Função de Atingibilidade e Funções de Integração	37
3.5 Modelagem de SAN na Ferramenta PEPS	38
3.5.1 A Interface Textual	39
3.5.2 Identificadores e Domínios	39
3.5.3 Eventos	41
3.5.4 Função de Atingibilidade	42
3.5.5 Resultados	42
3.5.6 Modelando um Exemplo de SAN	42
4. DIAGRAMAS DE DECISÃO MULTIVALORADA	49
4.1 Utilizando Diagramas de Decisão Multivalorada em SAN	50
4.1.1 Obtenção de Estados Atingíveis com Redução do Espaço de Estados	50

5. PROPOSTA DO ALGORITMO E ESTRUTURA DE DADOS	55
5.1 Algoritmo para Obtenção de Estados Sucessores e Predecessores	55
5.1.1 Matrizes de Transições para Representação de SAN	55
5.1.2 Estruturação do Algoritmo	56
5.1.3 Definições do Algoritmo	58
5.2 Estrutura de Dados Baseada em XML para Representação da SAN	65
5.2.1 Formato Proposto para a Estrutura de Dados	65
6. FERRAMENTA PARA SIMULAÇÃO VISUAL DE SAN	69
6.1 Objetivos da Ferramenta	70
6.2 Visão Geral	70
6.3 Arquitetura	70
6.3.1 Interface do Usuário	71
6.4 Análise e implementação da Ferramenta	73
6.5 Modelagem de SAN - Exemplificando Modelos na Ferramenta	74
6.5.1 Exemplo sem taxa funcional	75
6.5.2 Exemplo com taxa funcional	76
6.6 Exemplo de Obtenção dos Estados Atingíveis por meio de MDD	78
6.7 Ambiente de Desenvolvimento	80
6.8 Considerações	81
7. CONCLUSÃO	83
7.1 Contribuição	83
7.2 Trabalhos Futuros	84
REFERÊNCIAS BIBLIOGRÁFICAS	85
A. VISUALSAN	91
A.1 Ambiente de Edição	91
A.1.1 Componentes	91
A.2 Exemplo de Uso da Ferramenta	93
B. FERRAMENTAS	97
B.1 JAVACC e a Geração de <i>Parser</i>	97
B.1.1 Exemplo	97
B.2 JFLAP	98

1. INTRODUÇÃO

A complexidade no desenvolvimento de sistemas computacionais é um dos fatores preponderantes para o avanço das modelagens de sistemas utilizando formalismos estruturados. De fato, desenvolver sistemas têm se tornado bastante oneroso em relação à custos operacionais, o que faz com que o uso de formalismos para avaliação de desempenho de sistemas seja largamente utilizado. Para tal, formalismos baseados em métodos analíticos como Redes de Petri [PET81, MUR89, BAL94], Álgebra de Processos Estocásticos [CLA07] e Redes de Autômatos Estocásticos [FER98a, FER98b, PLA85, PLA91] estão sendo utilizados na modelagem de sistemas como sistemas distribuídos, redes de computadores, garantia de qualidade no desenvolvimento de *softwares*, etc.

As Redes de Autômatos Estocásticos (*SAN*) é um formalismo para a descrição e a avaliação de processos estocásticos baseados em métodos analíticos. O princípio básico da modelagem de *SAN* é a construção de sistemas como um conjunto de subsistemas que interagem entre si. Cada subsistema é modelado por um autômato e a interação entre autômatos é modelado por regras de transições relacionando os estados dos autômatos [PLA85].

Métodos analíticos podem ser aplicados para garantia de qualidade no desenvolvimento de um *software*, bem como ser utilizados na busca de soluções exatas ou muito aproximadas de um resultado desejado, pois permitem descrever também situações do mundo real com um nível de abstração maior do que a simulação, gerando modelos puramente matemáticos. Neste tipo de modelo, o funcionamento do sistema real é reduzido a relações matemáticas. Desenvolver métodos analíticos normalmente exige maior abstração de aspectos da realidade, se comparado a modelos de simulação [FER98a].

Uma característica a ser observada em um modelo *SAN* é o comportamento dos autômatos. A avaliação do desempenho e a análise do comportamento de uma *SAN* se dá pela verificação dos estados, que ao serem atingidos, representam processos realizados pelo sistema modelado. Estados sucessores e predecessores indicam, no espaço de estados de uma *SAN*, a trajetória e evolução dos padrões comportamentais do sistema modelado. Por ser baseado em cadeias de Markov [STE94, HAG02], no formalismo *SAN* a representação dos estados globais pode se tornar inviável devido ao problema da explosão de espaços de estados, o que consistirá em elevado consumo de memória para representar os diferentes estados aplicáveis de um modelo.

Para que seja possível o estudo do comportamento de sistemas grandes e complexos a fim de minimizar a explosão de espaço de estados, para alguns sistemas modelados nem sempre se pode evitar grandes espaços de estados, deve-se realizar uma modelagem levando-se em conta apenas a porção atingível do modelo (*Reachable State Space - RSS*) do espaço de estados produto (*Product State Space-PSS*). Em estudos iniciais o espaço de estados produto era armazenado em um vetor ou lista de estados, onde [BUC98], [CIA01a] e [MIN06] propuseram maneiras eficientes para descrever o espaço de estados atingíveis de modelos grandes e complexos para o formalismo *SPN* (*Stochastic Petri Nets*). Estas proposições se basearam na estrutura de dados chamada Diagramas de Decisão

Multivalorados (*Multi-valued Decision Diagram* - MDD) [KAM98], esta estrutura foi utilizada devido ao fato de ser compacta e eficaz para as operações comuns como inserções e buscas.

O espaço de estados produto é a combinação dos estados individuais de cada autômato. Dentre os estados que compõem este espaço, existem estados atingíveis e podem existir inatingíveis no âmbito global do sistema (dadas as transições que podem ser disparadas a partir de cada estado global). A obtenção do espaço de estados atingíveis pode se dar através de rotinas de percurso na cadeia de Markov que permeia o modelo estruturado, e seu armazenamento pode se dar através de um vetor de estados ou estruturas mais complexas como Diagramas de Decisão Multivalorados (ou outras técnicas de armazenamento).

Estudos referentes à aplicação de MDD em SAN [SAL09a] demonstram os benefícios na geração do espaço de estados atingíveis pois poderá haver uma diminuição do problema de explosão do espaço de estados nos modelos SAN através da eliminação dos estados redundantes, dependendo das estruturas de dados usadas na solução, sem que haja uma alteração na representação do espaço de estados.

A análise e estudo de sistemas complexos exigem um processamento robusto de dados e interpretação das informações fornecidas. Desenvolver sistemas com grandes quantidades de variáveis tem se tornado bastante custoso, o que pode inviabilizar a produção de *software* para atender as necessidades destes sistemas.

Uma maneira de reduzir custos operacionais é utilizar a simulação de sistemas. O uso de simulação nas áreas de conhecimento ligadas ao desenvolvimento computacional (*software* e *hardware*) vem a facilitar o controle da complexidade comumente encontrada na implementação destas aplicações possibilitando uma maior produtividade. Buscando uma maior aproximação da realidade e da interpretação dos dados dos sistemas analisados, foram realizados diversos estudos a fim de encontrar maneiras de se compreender estes dados com eficiência utilizando métodos analíticos, bem como a utilização de ambientes gráficos e de animação para o ensejo da simulação de sistemas [SAB06].

1.1 Simulação de Sistemas

Nas sub-seções a seguir serão apresentados, resumidamente, conceitos de simulação de sistemas através de simulação visual e soluções numéricas.

1.1.1 Simulação Visual

Ambientes de simulação visual oferecem recursos interativos para uma melhor observação dos resultados intermediários como, por exemplo, a evolução comportamental do modelo e variáveis [WAG96]. Neste contexto, os resultados de soluções numéricas poderão ser avaliados com uma maior aproximação da realidade do sistema modelado, uma vez que os resultados numéricos retornam índices quantitativos de desempenho. Soluções numéricas aliadas à simulação visual facilitam a avaliação de sistemas complexos proporcionando o aumento da produtividade no desenvolvimento de sistemas enfatizando a construção do modelo através de recursos gráficos oferecendo ao usuário

mecanismos para que este possa compreender e melhorar o sistema em estudo [OKE91], ao passo que apenas o uso de simulação visual, por ser baseada em um modelo de sistema executável, só permite uma avaliação rápida e superficial da qualidade do projeto [ZAP08].

A simulação faz uma abordagem e tratamento de sistemas dinâmicos, ou seja, o tratamento de sistemas ou objetos cujo comportamento é descrito num modelo, onde o tempo define a evolução do estado destes sistemas. Quando a simulação é interativa visual, é possível acompanhar e controlar esta evolução [FRE94]. A simulação de modelos SAN, por exemplo, pode ser realizada representando os grafos contidos neste formalismo e assim analisar o comportamento tal qual o formalismo modela. O controle, pelo usuário, possibilita a execução dos eventos respeitando, inclusive, a dependência das taxas funcionais pertencentes ao modelo representado. Para a perfeita aplicação do conceito de Simulação Visual Interativa são necessários quatro requisitos básicos, são eles [ROO93]:

- intervenção, possibilidade de interação com o modelo;
- inspeção, acesso a dados durante o experimento;
- especificação, requisito que especifica os parâmetros do modelo, tempo e variáveis necessárias para a execução do modelo;
- visualização, representação gráfica do comportamento do modelo e relacionamento de interesse.

1.1.2 Simulação Baseada em Soluções Numéricas

A Análise e Modelagem de Simulação é o processo de criação e experimentos com modelos matemáticos de um sistema real (sistema físico). A simulação é utilizada para visualizar comportamentos, para identificar problemas de sistemas reais, melhorar o desempenho de sistemas [BAN05] e utilizar recursos computacionais para avaliar um modelo numericamente, e por seguinte os dados são obtidos ordenadamente para estimar as verdadeiras características desejadas do modelo [LAW00].

A simplicidade de alguns sistemas permite que sejam utilizados métodos matemáticos para se obter informações exatas através de soluções analíticas. Entretanto, alguns modelos são bastante complexos para permitir o uso de soluções analíticas, para estes modelos deve-se aplicar a simulação baseada em soluções numéricas. Atualmente, diversas áreas aplicam os conceitos da simulação tais como: projeto e análise de sistemas, protocolos de redes ou definição dos requisitos de *hardware*, re-engenharia de processos de negócios, *etc.*

1.2 Objetivos

O objetivo principal deste trabalho é apresentar uma ferramenta gráfica para simular visualmente o formalismo SAN, bem como propor algoritmos para obtenção de estados sucessores e predecessores em modelos SAN. Outro objetivo deste trabalho é apresentar uma estrutura de dados, a ser utilizada na ferramenta proposta, para representação de modelos SAN a fim de permitir que outras

ferramentas possam utilizar uma SAN modelada na ferramenta. Além de obter os estados atingíveis por meio de algoritmos de soluções numéricas, o *software* a ser proposto deverá integrar-se às ferramentas PEPS [BRE07] e GTAexpress [CZE09] a fim de utilizar as soluções numéricas disponíveis nestas ferramentas.

1.3 Estrutura dos Capítulos deste Trabalho

Este trabalho está organizado como descrito a seguir: Capítulo 2 apresenta os trabalhos relacionados que utilizam ferramentas de simulação visual e interface textual aplicando formalismos. O Capítulo 3 trata-se de uma revisão teórica sobre o formalismo SAN, no Capítulo 4 é apresentado uma breve introdução de MDD. Nos Capítulos 5 e 6, principais focos desta dissertação, é apresentada a contribuição científica na forma de algoritmo para obtenção de estados sucessores e predecessores e estrutura de dados para representação de modelos SAN, bem como é apresentada a ferramenta de simulação visual de Redes de Autômatos Estocásticos. No Capítulo 7 é realizado um fechamento e são apresentados os trabalhos futuros. Os apêndices A e B expõem um tutorial de uso da ferramenta e uma introdução as ferramentas que suportaram o desenvolvimento da ferramenta proposta neste trabalho.

2. TRABALHOS RELACIONADOS

Este capítulo apresenta algumas ferramentas que visam simular formalismos tais como PEPA, SPN e SAN, sendo feita uma breve introdução nas características de cada ferramenta.

2.1 PEPAnets

PEPAnets [GIL02] é uma extensão de PEPA [POO96] permitindo que os componentes sejam usados como objetos de uma Rede de Petri Estocástica Colorida. Um componente de PEPA tem estados locais que podem executar ações sincronizadas, individualmente ou em cooperação com um outro componente. Estas atividades definem uma interface análoga à interface composta pelos métodos que podem ser invocados num objeto. Devido às limitações na natureza da informação sincronizada (distribuídos exponencialmente com retardos aleatórios) significa dizer que um modelo PEPA define uma Cadeia de Markov de Tempo-Contínuo (CTMC) .

2.2 PEPA Workbench

O propósito do PEPA Workbench [GIL94] é checar se o modelo PEPA é bem-formado, e gerar o processo Markoviano correspondente. Este *workbench* detecta falhas como *deadlocks* e cooperações que não envolvem participantes ativos. Em essência, o processo de translação do *workbench* aceita um modelo PEPA como entrada e produz uma matriz contendo o processo de Markov codificando um determinado modelo. Um exemplo deste relacionamento é demonstrado na Figura 2.1.

$$\begin{array}{c}
 P_1 \stackrel{def}{=} (run, r_1).P_2 \quad P_2 \stackrel{def}{=} (stop, r_2).P_1 \\
 P_1 \quad || \quad P_1
 \end{array}$$

↓

$$\begin{pmatrix} -2r_1 & r_1 & r_1 & 0 \\ r_2 & -r_1 - r_2 & 0 & r_1 \\ r_2 & 0 & -r_1 - r_2 & r_1 \\ 0 & r_2 & r_2 & -2r_2 \end{pmatrix}$$

Figura 2.1: O esquema do PEPA Workbench

Neste exemplo é demonstrado que o modelo PEPA terá dois componentes independentes que executará em paralelo, e não cooperantes. Cada componente é iniciado no estado P_1 , e cada tem

um estado significando que a matriz geradora para o processo de Markov correspondente terá quatro dimensões. Em caso de exemplos mais complexos e maiores, devido a esta complexidade de construir uma matriz do processo de Markov sem uso de software específico torna-se uma atividade inviável e errônea, desde que a matriz seja uma representação não estruturada do sistema desprovido de nomes, e somente contendo índices.

2.3 GreatSPN

O GreatSPN é um *software* para modelagem, validação, avaliação de performance de sistemas distribuídos usando Redes de Petri Generalizadas e suas extensões coloridas: *Stochastic Well-formed Nets* [BAA01]. A ferramenta fornece um *framework* amigável para experimentos com redes de Petri temporizadas baseadas em técnicas de modelagem. Ela implementa algoritmos de análise eficientes para permitir seu uso em aplicações bastante complexas, não apenas em aplicações mais simples.

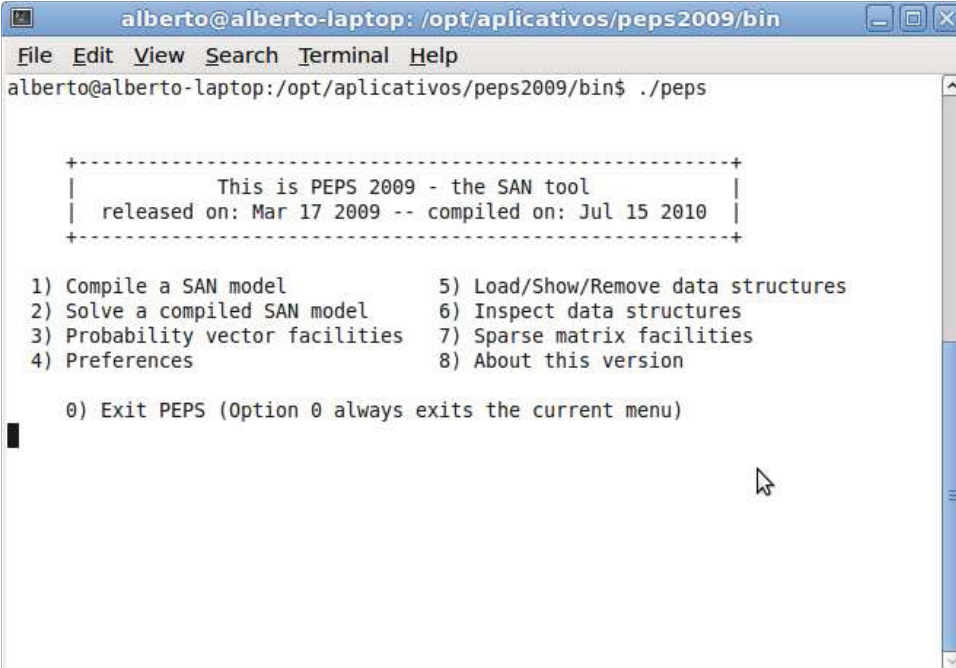
As principais funcionalidades do GreatSPN são: (i) permitir visualizar representações gráficas das propriedades estruturais; (ii) a definição de sincronização e especificações estocásticas, parâmetros, e medidas de desempenho; (iii) simulação com interação gráfica de sincronização e modelos estocásticos; (iv) geração de gráficos de atingibilidade; (v) solução Markoviana para estados-constantos assim como avaliação transitória de desempenho explorando eficiência, técnicas numéricas para matrizes esparsas; (vi) módulos de simulação para interação de eventos que em cooperação com a interface gráfica fornece simulação com interação, com animação gráfica do modelo e movimento de símbolos, com passo-a-passo, e progressão de tempo para frente e para trás, reprogramação arbitrária de eventos; dentre outras características.

2.4 Performance Evaluation of Parallel Systems - PEPS

O PEPS é uma ferramenta composta por módulos que fazem a descrição, compilação e solução de modelos SAN. Implementado com uma interface textual (Figura 3.3) usando a linguagem de programação C++ podendo ser usado em qualquer plataforma de sistemas operacionais, sendo que suas versões foram testadas apenas para plataformas Linux e Solaris. Algumas características da ferramenta são;

- Descrição textual compacta de modelos SAN de tempo contínuo;
- Solução estacionária de modelos usando métodos Arnoldi, GMRES e Potência Iterativa;
- Otimização numérica sobre dependências funcionais, pré-computação diagonal, pré-condicionamento e álgebra de agregação de autômatos;
- Avaliação de resultados;
- Agregação semântica de SAN com réplicas;

- Solução numérica usando vetores de probabilidades com o tamanho do espaço de estados atingíveis;
- Fornecer ao usuário uma seleção de métodos de solução iterativa para calcular vetor de probabilidades transitória e estacionária e uma série de soluções numéricas;
- Inspeccionar estruturas de dados como o espaço de estados atingíveis, o vetor de probabilidade estacionária (e comparar dois deles) e o descritor SAN; e
- Resultados de saída e os arquivos de estrutura de dados: os resultados, os vetores de probabilidade, SAN descritor, formato HBF do descritor compatível com o *software* Marca.



```

alberto@alberto-laptop: /opt/aplicativos/peps2009/bin
File Edit View Search Terminal Help
alberto@alberto-laptop:/opt/aplicativos/peps2009/bin$ ./peps

+-----+
|          This is PEPS 2009 - the SAN tool          |
| released on: Mar 17 2009 -- compiled on: Jul 15 2010 |
+-----+

1) Compile a SAN model           5) Load/Show/Remove data structures
2) Solve a compiled SAN model    6) Inspect data structures
3) Probability vector facilities  7) Sparse matrix facilities
4) Preferences                   8) About this version

0) Exit PEPS (Option 0 always exits the current menu)

```

Figura 2.2: Interface textual da ferramenta PEPS2009

A ferramenta PEPS implementa dois conjuntos de métodos para solução analítica de modelos SAN. O primeiro conjunto trabalha com o formato Kronecker e o segundo conjunto trabalha com o formato esparsa (HBF).

O formato Kronecker que utiliza o método *shuffle* e cuja idéia básica é multiplicar o vetor de probabilidades com o conjunto de matrizes que compõem o Descritor Markoviano. De fato, cada pequena matriz multiplica uma parte do vetor. A soma de todas as multiplicações das pequenas matrizes é uma multiplicação completa do vetor-descritor. Este método de multiplicação foi implementado em duas versões. A primeira trabalha com vetor estendido (com o tamanho global do espaço de estado). O segundo trabalha com vetor esparsa (somente o tamanho do espaço de estados atingíveis). Para o formato HBF, foi implementado uma multiplicação do gerador de vetor infinitesimal usando uma representação de matriz esparsa para o gerador infinitesimal.

3. REDES DE AUTÔMATOS ESTOCÁSTICOS

O formalismo SAN foi proposto por Plateau [PLA85] e o objetivo básico é representar um sistema por meio de uma coleção de sub-sistemas com um comportamento independente (transições locais) e interdependências ocasionais (taxas funcionais e eventos sincronizantes) sob os pressupostos das Cadeias de Markov [STE94]. Cada sub-sistema é descrito como um autômato estocástico, conforme a Figura 3.1.

Este formalismo é uma forma compacta de descrever realidades complexas, otimizando ainda a busca de soluções estacionárias [FER98b], utilizando-se das noções de estado e de transições entre estados para representação dos eventos, assim como as Cadeias de Markov. Os eventos em SAN podem estar associados a um único autômato, ou a vários ao mesmo tempo, ocorrendo de acordo com taxas específicas. Dá-se o nome de estado local ao estado individual de cada autômato e estado global como sendo a combinação de todos os estados locais de cada autômato componente da SAN. Dessa forma, o formalismo de redes de autômatos estocásticos é particularmente interessante para a modelagem de sistemas distribuídos, nos quais as unidades de processamento não interagem entre si a todo instante [FER98a]. Um autômato é um modelo matemático de um sistema com entradas e saídas discretas [HOP00].

O formalismo de redes de autômatos estocásticos pode ser utilizado tanto para modelos em escala de tempo discreto quanto para modelos em escala de tempo contínuo. O termo estocástico é devido ao fato de que o tempo é tratado como uma variável aleatória, que na escala de tempo contínua obedece a uma distribuição exponencial [FER98b].

3.1 Autômatos Estocásticos

Uma SAN é uma coleção de componentes chamados de autômatos estocásticos. Os Autômatos estocásticos representam um modelo matemático de um sistema que possui entradas e saídas discretas. O sistema pode se encontrar em qualquer um dentre o número finito dos estados do sistema ou das configurações internas. O estado interno em que o sistema se encontra sumariza as informações sobre entradas anteriores e indica o que é necessário para determinar o comportamento do sistema para as entradas seguintes [HOP00]. Partindo desta definição pode-se descrever um autômato estocástico como um conjunto finito de estados e um conjunto finito de transições entre esses estados. A Figura 3.1 é uma representação gráfica de um modelo de Redes de Autômatos Estocásticos com dois autômatos: $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(2)}$.

As sub-seções 3.1.1 e 3.1.2 são fortemente baseadas nas pesquisas de Brigitte Plateau e William J. Stewart - "Stochastic Automata Network" [STE96], e fornecerão subsídios para que algumas regras sejam estabelecidas a fim de auxiliar no desenvolvimento dos algoritmos necessários para a criação da aplicação objeto de estudo desta dissertação.

3.1.1 Autômatos Estocásticos sem Interação

Considerando o caso de um sistema que pode ser modelado por dois autômatos estocásticos complementamente independentes, cada qual pode ser representado por uma Cadeia de Markov à escala de Tempo Contínuo (CTMC). Assume-se que o primeiro autômato, denotado por $\mathcal{A}^{(1)}$, tenha n_1 estados e matriz de transições de probabilidades estocásticas dado por $P^1 \in \mathcal{R}^{n_1 \times n_1}$. Similarmente, denota-se $\mathcal{A}^{(2)}$ como o segundo autômato; n_2 , o número de estados em sua representação e $P^2 \in \mathcal{R}^{n_2 \times n_2}$, sua matriz de transições de probabilidades estocásticas. Os estados de todo (bi-dimensional) o sistema podem ser representados pelo par (i, j) onde $i \in 1, 2, \dots, n_1$ e $j \in 1, 2, \dots, n_2$. De fato, a matriz de transição de probabilidades estocásticas do sistema bi-dimensional é dado pelo produto tensorial das matrizes P^1 e P^2 . Se, em vez de ser representado por duas cadeias de Markov de tempo-discreto, os autômatos estocásticos forem caracterizados por Cadeias de Markov de tempo-contínuo com gerador infinitesimal, $Q^{(1)}$ e $Q^{(2)}$, respectivamente, o gerador infinitesimal do sistema bi-dimensional é dado pela soma tensorial das matrizes de $Q^{(1)}$ e $Q^{(2)}$.

Continuando com o exemplo, seja $\pi_i^{(1)}(t)$ a probabilidade de que o primeiro autômato está no estado i no tempo t e $\pi_j^{(2)}(t)$ a probabilidade do segundo autômato está no estado j , no tempo t . Então a probabilidade que, no tempo t , o primeiro está no estado i e o segundo está no estado j é simplesmente o produto $\pi_i^{(1)}(t) \times \pi_j^{(2)}(t)$. Além disso, a distribuição das probabilidades de todo o sistema (bi-dimensional) é dado pelo produto tensorial dos dois vetores de probabilidades individuais, $\pi^{(1)} \in \mathcal{R}^{n_1}$ e $\pi^{(2)} \in \mathcal{R}^{n_2}$, resumido em: $\pi^{(1)} \otimes \pi^{(2)}$. Maiores detalhes sobre produto de tensores pode ser visto em [FER98a].

3.1.2 Autômatos Estocásticos com Interação

Há basicamente duas maneiras em que os autômatos estocásticos interagem entre si:

1. A taxa com que uma transição pode ocorrer em um autômato pode ser uma função do estado de outro autômato. Estas transições são chamadas de transições funcionais;
2. Uma transição em um autômato pode forçar uma transição para ocorrer em um ou mais autômatos, chamadas de transições sincronizadas, estas transições são disparadas por eventos sincronizantes, de fato, um único evento sincronizante irá, geralmente, causar o disparo de múltiplas transições sincronizantes. Transições sincronizantes também podem ser funcionais.

Os elementos da matriz que representam qualquer autômato estocástico são todos constantes, i.e., números reais não negativos, ou funções do espaço de estados globais para número não negativos. Taxas de transição que dependem somente dos estados do próprio autômato, e não dos estados de um outro autômato qualquer, são para todos os efeitos, taxas de transições constantes. Uma transição sincronizante tanto pode ser funcional como constante. Num determinado autômato, transições que não são sincronizantes são ditas transições locais.

Se um dado estado de um autômato estocástico contém transições funcionais, as taxas destas transições devem ser avaliadas de acordo com as suas fórmulas definidas e o estado global atual

dos autômatos inseridos na função. Em certos exemplos pode ser vantajoso avaliar estas taxas funcionais uma única vez e armazenar os valores obtidos em um *array* que pode ser recuperado como e quando eles forem necessários. Em outros casos, talvez seja melhor deixá-los em sua forma original e reavaliar a fórmula a cada vez.

Se a taxa de transição for constante ou funcional, é importante observar que somente o estado local do autômato é afetado se estiver num evento local. Portanto, todas as informações relativas às taxas de transições, constantes e funcionais, podem ser manipuladas dentro desse autômato (assumindo apenas que esse autômato tem um conhecimento do estado global do sistema).

3.2 Estados Locais e Globais

Os estados de uma SAN sintetizam a informação referente às entradas passadas que serão necessárias para determinar o comportamento do sistema para entradas posteriores [FER98b]. O estado local é o estado individual de cada autômato do modelo SAN que combinado com estados locais de outros autômatos forma o estado global. A Figura 3.1 modela um sistema através de dois autômatos, onde o autômato $\mathcal{A}^{(1)}$ possui três estados ($0^{(1)}, 1^{(1)}$ e $2^{(1)}$), o segundo autômato ($\mathcal{A}^{(2)}$) possui dois estados ($0^{(2)}$ e $1^{(2)}$). Neste modelo, os eventos e_1, e_2, e_3 e e_5 são eventos locais e e_4 é sincronizante, sendo que o evento e_5 contém uma taxa funcional f que define uma interação entre os autômatos. O conceito de taxa funcional será detalhado na Seção 3.4.

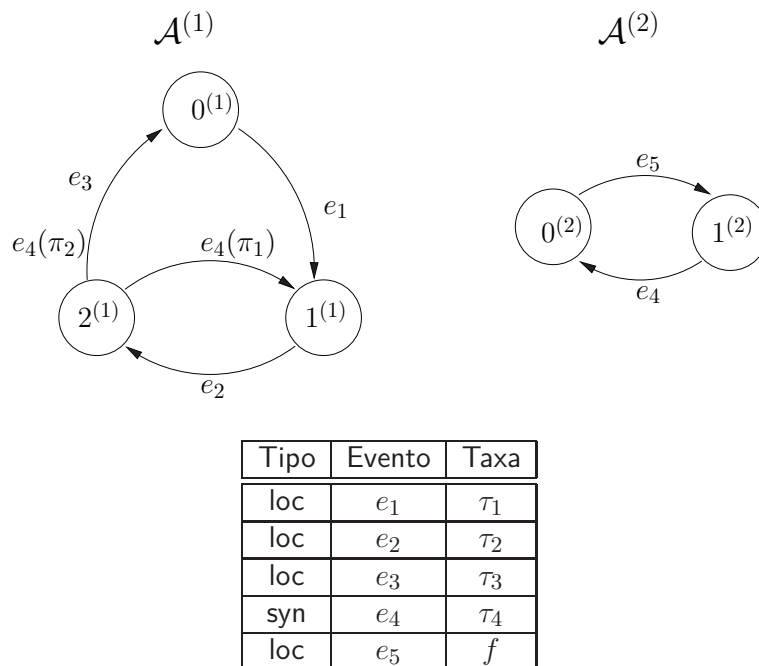


Figura 3.1: Modelo SAN com dois autômatos e sua tabela de eventos [BRE04a]

Os estados globais de uma rede de autômatos estocásticos são definidos como a combinação de todos os estados locais de cada autômato. Estes estados são classificados ainda quanto a sua atingibilidade. Quando todos os autômatos de uma rede encontram-se em estados locais que, em

conjunto, correspondem a uma situação fisicamente possível, diz-se tratar de um estado global atingível. No entanto, é comum a situação em que dois ou mais estados locais, embora estejam individualmente corretos, no seu conjunto correspondam a uma situação irreal do sistema físico que se está modelando.

A Figura 3.2 apresenta uma cadeia de Markov equivalente à SAN modelada na Figura 3.1. Neste autômato cada estado é a combinação de uma dupla de estados locais de cada autômato. Por exemplo, quando a SAN representada na Figura 3.1 encontra-se no estado $2^{(1)}$ do autômato $\mathcal{A}^{(1)}$ e no estado $0^{(2)}$ do autômato $\mathcal{A}^{(2)}$, esta situação equivale ao estado global $2^{(1)}0^{(2)}$ do autômato modelado na Figura 3.2, f representa a taxa funcional relacionada ao evento e_5 .

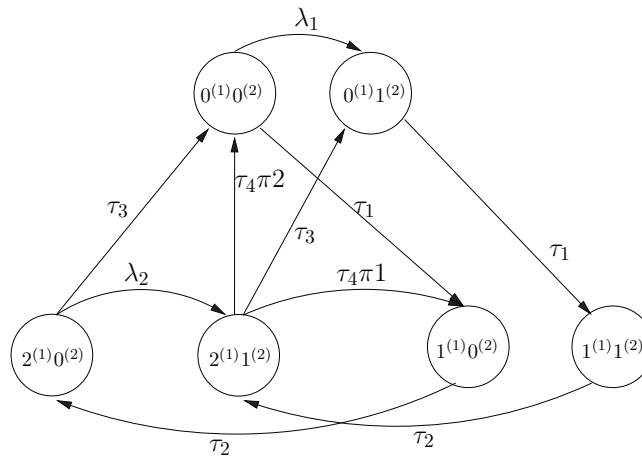


Figura 3.2: Autômato global equivalente ao modelo SAN com dois autômatos [BRE04a]

3.3 Transições Locais e Sincronizantes

As transições são construções que indicam a possibilidade de mudança entre um estado e outro. No entanto, cada transição necessita ter ao menos um evento associado a ela para que essa possa ser disparada [BRE04b]. As transições locais de um autômato alteram o estado global pela mudança de estado em apenas um autômato, enquanto as transições sincronizantes alteram o estado global pela mudança de estado em dois ou mais autômatos simultaneamente [PLA91, FER99]. As transições locais permitem que os autômatos tenham comportamento paralelos através de eventos locais, mudando somente o estado local do autômato em que ocorreram e não sofrem interferências por parte dos estados dos outros autômatos. Mesmo os eventos sendo independentes, eles são disparados um de cada vez [FER99].

As transições sincronizadas permitem que seja representado sincronismo no disparo de transição entre os autômatos, constituindo eventos sincronizantes entre os mesmos [PLA91, BRE02]. Esta característica a torna um pouco mais sofisticada, pois podem trocar o estado de dois ou mais autômatos simultaneamente [FER98a].

3.4 Taxas Funcionais e Probabilidades Funcionais

Além dos eventos sincronizantes, taxas e probabilidades funcionais são utilizadas para representar possíveis interações entre autômatos sem alterar o estado de todos os autômatos envolvidos. Neste caso, as taxas funcionais modificam apenas o estado do autômato a qual esta pertence e não os estados dos autômatos aos quais ela depende, ou seja, o evento pode ser dependente do estado local de outro autômato para ser disparado como evento sincronizante. Deve-se ressaltar que essas taxas podem ser definidas por funções que são avaliadas conforme os estados atuais do modelo SAN [FER98a]. Estas funções levam em consideração os estados atuais dos autômatos de um modelo variando seu valor conforme os estados em que se encontram os autômatos envolvidos na função [FER98b].

No autômato $\mathcal{A}^{(2)}$ da Figura 3.1, usa-se uma função para definirmos a taxa de transição do evento local e_5 . A taxa do evento e_5 não é uma taxa constante, mas sim uma taxa funcional f descrita pela notação SAN. A interpretação da função pode ser visualizada com a avaliação de uma expressão não tipificada de linguagem de programação (e.g., linguagem Java). Cada comparação é avaliada para valor 1 (*verdadeiro*) e valor 0 (*falso*) [SAL09a].

Definindo-se a função f como segue abaixo, temos então três taxas de transições distintas dependendo do estado em que se encontra o modelo.

$$f = \begin{cases} \lambda_1 & \text{se o autômato } \mathcal{A}^{(1)} \text{ está no estado } 0^{(1)} \\ 0 & \text{se o autômato } \mathcal{A}^{(1)} \text{ está no estado } 1^{(1)} \\ \lambda_2 & \text{se o autômato } \mathcal{A}^{(1)} \text{ está no estado } 2^{(1)} \end{cases}$$

Como pode-se observar na definição da função f , a taxa de ocorrência da transição do estado $0^{(2)}$ para o estado $1^{(2)}$ é igual a λ_1 (caso o autômato $\mathcal{A}^{(1)}$ esteja no estado $0^{(1)}$), e igual a λ_2 (caso o autômato $\mathcal{A}^{(1)}$ esteja no estado $2^{(1)}$), e a transição não ocorrerá caso o autômato $\mathcal{A}^{(1)}$ esteja no estado $1^{(1)}$. As probabilidades de cada evento podem ser expressas por funções tal qual as taxas de ocorrência. A definição de funções usadas para expressar as probabilidades funcionais são exatamente iguais as funções usadas para definir as taxas de ocorrência e permite associar a um evento diferentes valores. Na Figura 3.1, pode-se observar o uso de probabilidades para as transições entre os estados $2^{(1)}$ e $0^{(1)}$ com probabilidade π_2 , e $2^{(1)}$ e $1^{(1)}$ com probabilidade π_1 . Estas probabilidades definem uma provável escolha. Assim, quando houver a ocorrência do evento e_4 , o autômato $\mathcal{A}^{(1)}$ cujo estado atual seja o $2^{(1)}$ transitará para o estado 0 com a probabilidade π_2 ou para o estado $1^{(1)}$ com probabilidade π_1 , a soma das taxas de probabilidades deve ser igual a 1.

3.4.1 Função de Atingibilidade e Funções de Integração

Segundo [BRE04b], as Funções de Atingibilidade e Integração são outros dois tipos utilizados em SAN. As expressões que definem estas funções são descritas da mesma forma que as taxas e probabilidades funcionais. Entretanto, esses dois tipos de funções desempenham papéis diferenciados conforme descrição a seguir.

Função de Atingibilidade: Devido à representação em SAN ser de forma modular e o autômato global (equivalente à cadeia de Markov) se constituído pela combinação de todos os autômatos do modelo, é necessário especificar uma função que determina os estados atingíveis do autômato global de um modelo em SAN.

A definição de quais estados podem ser atingíveis (ou alcançáveis) em um modelo em SAN é dada pela função de atingibilidade. Essa função é definida usando-se as mesmas regras adotadas para a definição de taxas e probabilidade funcionais.

Ainda segundo [BRE04b], a noção de função de atingibilidade fica mais clara ao se imaginar, por exemplo, um modelo de compartilhamento de recursos, onde se tem N clientes disputando R recursos. Este sistema pode ser modelado em SAN usando um autômato com dois estados para cada cliente. O estado $0^{(i)}$ representa que o recurso não está sendo utilizado pelo cliente i , enquanto o estado $1^{(i)}$ representa que o recurso está em uso pelo cliente i . É fácil imaginar que, se o número R de recursos for menor do que o número N de clientes, o estado global que representa todos os clientes utilizando um recurso não poder ser atingido, pois este estado não corresponde a realidade do modelo. Os estados que possuem tal característica são chamados de estados inatingíveis e devem ser eliminados do modelo através da função de atingibilidade, pois a probabilidade do modelo encontrar-se em algum destes estados é igual a zero.

A função de atingibilidade correta para o modelo de compartilhamento de recurso descrito é a seguinte:

$$reachability = nb[\mathcal{A}^{(1)} \dots \mathcal{A}^{(N)}]1^{(i)} \leq R$$

Funções de Integração: Define-se para a obtenção de resultados numéricos sobre o modelo em SAN. As funções de integração avaliam qual a probabilidade do modelo em SAN encontrar-se em um determinado estado.

Segundo [BRE04a], pode-se compor funções de integração que levem em conta a probabilidade do modelo se encontrar em um conjunto de estados, podendo assim obter índices de desempenho e confiabilidade do modelo. Essas funções de integração são avaliadas sobre o *vetor de probabilidade* que contém a probabilidade do modelo se encontrar em cada um dos estados pertencentes a ele.

Um exemplo de Funções de integração, tendo em mente o modelo de compartilhamento de recursos exposto anteriormente é dado pela função u , onde se quer descobrir a probabilidade do autômato $\mathcal{A}^{(1)}$ não estar utilizando o recurso, *i.e.*, encontrar-se no estado $0^{(1)}$.

$$u = \left(st(\mathcal{A}^{(1)} == 0^{(1)}) \right)$$

Em SAN todas as funções são modeladas da mesma forma, a forma de como são empregadas é que as diferenciam [BRE04a].

3.5 Modelagem de SAN na Ferramenta PEPS

Esta seção descreverá o uso da ferramenta PEPS 2007 para a implementação de exemplos de modelagem SAN e para tal será necessário a apresentação dos módulos que compõem este pacote de

software. Este *software* é composto por alguns módulos que implementam os passos para a resolução do modelo. Os módulos são agrupados em três fases: descrição, compilação e solução. A fase de descrição é composta pelos módulos de interface. O módulo de estrutura dos dados compõe a fase de compilação e a fase de solução se preocupa com a solução dos modelos. Todas as informações acerca da ferramenta PEPS 2007 foram extraídas de [BRE07] e do manual de instruções da própria ferramenta¹.

3.5.1 A Interface Textual

O módulo de interface textual para descrever modelos mantém a característica chave do formalismo SAN: ser modular. O PEPS 2007 incorpora uma abordagem baseada em grafos que é próximo aos modelos semânticos. Nesta abordagem cada autômato é representado por um grafo, em que os nodos são os estados e os arcos representam transições pela ocorrência de eventos. Esta descrição textual foi mantida simples, extensível e flexível.

- É bastante simples, porque há poucas palavras reservadas, o suficiente para delimitar os diferentes níveis de modularidade;
- Extensível porque a definição do modelo SAN é executado hierarquicamente;
- Flexível porque a inclusão de estruturas de replicação permite a reutilização de autômatos idênticos, e a construção de autômatos repetindo blocos de estado com o mesmo comportamento.

A descrição da SAN é composta por 5 blocos, conforme a Figura 3.3 que são facilmente localizados com os seus delimitadores² (em negrito). As outras palavras reservadas do PEPS, linguagem de entrada, são indicadas com a fonte itálica. Os símbolos "<" e ">" indicam informação mandatória para ser definida pelo usuário. Os símbolos "{" e "}" indicam informação opcional.

3.5.2 Identificadores e Domínios

O primeiro bloco, identificadores (*identifiers*), contém todas as declarações dos parâmetros: valores numéricos, funções, ou conjuntos de índices (domínios) a ser usado pela replica na definição modelo. Um identificador (**< id_name >**) pode ser qualquer *string* de caracteres alfanuméricos. Os valores numéricos e funções são definidas de acordo com a sintaxe da linguagem C. Em geral, as expressões são similares as expressões matemáticas com lógicas e operadores aritméticos. Os argumentos destas expressões podem ser números constantes, identificadores de autômatos ou identificadores de estados. Neste último caso, as expressões são funções definidas no espaço de estados do modelo SAN. Por exemplo, "o número de autômatos no estado n0" (que dado um resultado inteiro) pode ser expresso como "nb n0". Uma função que retorna o valor 4 se dois autômatos (*A1* e

¹Este manual pode ser encontrado em www-id.imag.fr/Logiciels/peps/

²A palavra delimitadores é usada para indicar símbolos necessários, tendo uma posição fixa no arquivo.

```

identifiers
  < id_name >=< exp >;
  < dom_name >=[ i..j ];
events
  // without replication
  loc < evt_name > (< rate >)
  syn < evt_name > (< rate >)
  // with replication
  loc < evt_name > [replication_domain](< rate >)
  syn < evt_name > [replication_domain](< rate >)
partial reachability=< exp >;
network < net_name > (< type >)
  aut < aut_name > [replication_domain]
  stt < stt_name > [replication_domain](reward)
  to(< stt_name > [replication_domain]/f_cond)
    < evt_name > [replication_domain](< prob >)/f_cond
  . . .
  < evt_name > [replication_domain](< prob >)/f_cond
  . . .
  from < stt_name >
  to(< stt_name > [replication_domain]/f_cond)
    < evt_name > [replication_domain](< prob >)/f_cond
  . . .
  stt < stt_name > [replication_domain](reward)
  to(< stt_name > [replication_domain]/f_cond)
    < evt_name > [replication_domain](< prob >)/f_cond
  . . .
  aut < aut_name > [replication_domain] . . .
results
  < res_name >=< exp > ;

```

Figura 3.3: Estrutura Modular do formato textual da SAN

A_2) estão em diferentes estados, e o valor 0, caso contrário, é expresso como $(stA_1! = stA_2) * 4$. Operadores de comparação retornam o valor "1" para o resultado verdadeiro ou o valor "0" para o resultado falso.

Como pode-se observar na declaração de identificadores, de acordo com a Figura 3.3, é possível declarar quantos forem necessários e a sua definição é a seguinte:

- "*< id_name >*" É um identificador de expressão que começa com uma letra e é seguida por uma seqüência de letra ou números. O comprimento máximo de uma expressão é 128 caracteres;
- "*< dom_name >*" É um identificador de domínio. É um conjunto de índices. Um domínio pode ser por um intervalo "[1..3]", por um lista "[1,2,3]" ou por lista de intervalos "[1..3,5,7..9]". Identificadores podem ser usados para definir um intervalo "[1..ID1,5,7..ID2]", onde ID1 e ID2 são identificadores com valores constantes. Em todos os casos, o domínio deve respeitar uma

ordem crescente de valores;

- “< *exp* >” É um número real ou uma expressão matemática. Um número real tem um dos seguintes formatos: Um número inteiro, tal como “12345”; um número real com ponto flutuante, tal como “12345,6789”; um número real com mantissa e expoente, tal como “12345E(ou e)+(ou -)10”ou “12345.6789e+100”.

Conjuntos de índices são usados para definir números de eventos, autômatos ou estados que podem ser descritos como replicações. Um grupo de autômatos replicados de \mathcal{A} com o conjunto em índices [0..2; 5; 8..10] define o conjunto contendo os autômatos $\mathcal{A}[0]$; $\mathcal{A}[1]$; $\mathcal{A}[2]$; $\mathcal{A}[5]$; $\mathcal{A}[8]$; $\mathcal{A}[9]$; e $\mathcal{A}[10]$.

3.5.3 Eventos

O bloco de eventos define cada evento do modelo dado:

- seu tipo (local ou sincronizando);
- seu nome (um identificador);
- seu índice (uma constante ou função previamente definida no bloco de identificadores).

Adicionalmente, eventos podem ser replicados usando os conjuntos de índices (domínios). Esta facilidade pode ser usada quando eventos com os mesmos índices aparecem em um conjunto de autômatos.

Formato do bloco **events**

events

loc < *evt_name* > [*replication_domain*](< *rate* >)

syn < *evt_name* > [*replication_domain*](< *rate* >)

...

-
- “*loc*” define o tipo de evento como um evento local;
 - “*syn*” define o tipo do evento como evento sincronizado;
 - “< *evt_name* >” o identificador do evento inicia com uma letra e é seguida por uma sequência de letras ou números. O tamanho máximo de um identificador é de 128 caracteres.
 - “[*replicatoin_domain*]” é um conjunto de índices. O *replication_domain* deve ter um identificador definido no bloco de identificadores. Um evento pode ser replicado em até três níveis. Cada nível é definido por *replication_domain*. Por exemplo, um evento replicado em dois níveis é definido como < *evt_name* > [*replication_domain*][*replication_domain*];

- “< *rate* > define a taxa dos eventos. Deve ser um identificador de expressão declarado no bloco de identificadores.

3.5.4 Função de Atingibilidade

O bloco de *reachability* é uma função definindo o espaço de estados atingíveis do modelo SAN. Usualmente, é uma função *Boolean*, retornando um valor diferente de zero para os estados de \hat{S} , conjunto completo dos estados de um autômato, que pertence a \mathcal{S} , espaço de estados produto de um modelo SAN. Um modelo onde todos os estados são atingíveis tem a função de atingibilidade definida como qualquer constante diferente de zero, e.g., o valor 1. Opcionalmente, a função de atingibilidade parcial pode ser definida pela adição da palavra reservada “*partial*”. Neste caso, somente um subconjunto de \mathcal{S} é definido, e o total de \mathcal{S} será computado pela ferramenta PEPS 2007.

3.5.5 Resultados

Neste bloco são definidas as funções usadas para computar os índices de desempenho do modelo. Os resultados dado pelo PEPS 2007 são os valores integrais dessas funções com a distribuição estacionária do modelo. Este bloco é opcional.

Formato do bloco **results**

results

```
< evt_name > = < exp > ;
...
```

-
- < *res_name* > é um simples identificador;
 - < *exp* > é uma expressão matemática.

3.5.6 Modelando um Exemplo de SAN

Nesta seção serão apresentados dois exemplos para ilustrar a capacidade da modelagem e a eficácia computacional da ferramenta PEPS 2007. Para cada exemplo, um modelo SAN genérico é descrito.

Um Modelo de Compartilhamento de Recursos

O primeiro exemplo é um modelo tradicional de compartilhamento de recursos, onde N processos distintos compartilham uma certa quantidade (R) de recursos distintos. Cada um destes processos alternam entre um estado de dormência e utilização do recurso. Quando um processo deseja mover do estado de dormência para o estado de utilização encontra o processo R já utilizando os recursos,

então o processo falha ao acessar o recurso e retorna ao estado adormecido. Note que quando $R = 1$, este modelo reduz ao problema de exclusão mútua usual. Analogamente, quando $R = N$ todos os processos são independentes e não há nenhuma restrição em acessar os recursos. Indica-se λ_i para ser a taxa em que processo i acorda do estado adormecido desejando usar o recurso, e μ_i , a taxa em que este mesmo processo libera o recurso.

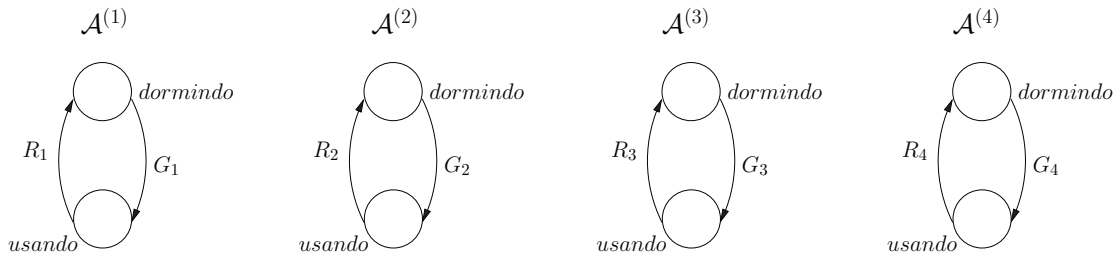


Figura 3.4: Modelo de compartilhamento de recursos

Na representação SAN (Figura 3.4), cada processo é modelado por um autômato com dois estados $\mathcal{A}^{(i)}$, e os dois estados são *dormindo* e *usando*. Indica-se $st\mathcal{A}^{(i)}$ para denotar o estado atual do automato $\mathcal{A}^{(i)}$. Será também introduzida a função

$$f = \delta\left(\sum_{i=1}^N \delta(st\mathcal{A}^{(i)} = usando) < R\right)$$

onde $\delta(b)$ é uma função inteira que terá o valor 1 se o Boolean b é verdadeiro, e o valor 0 caso contrário. Portanto, esta função f terá o valor 1 quando acesso ao recurso é permitido e terá valor 0, caso contrário. A Figura 3.4 fornece uma ilustração gráfica deste modelo. Nesta representação cada autômato $\mathcal{A}^{(i)}$ tem dois eventos locais.

- G_i que corresponde ao i -ésimo processo obtendo um recurso, com taxa $\lambda_i f$;
- R_i que corresponde ao i -ésimo processo liberando um recurso, com taxa μ_i .

A representação textual (arquivo .san) descrevendo este modelo é:

```
//===== RS model version 1 ===== //
                        N=4, R=2
//=====
identifiers
R          = 2;    // quantidade de recursos
mu1       = 6;    // taxa para deixar um recurso para o processo 1
lambda1   = 3;    // taxa para solicitacao de recursos para o processo 1
f1        = lambda1 * (nb usando < R);
```

```

mu2      = 5;    // taxa para deixar um recurso para o processo 2
lambda2  = 4;    // taxa para solicitacao de recursos para o processo 2
f2       = lambda2 * (nb usando < R);
mu3      = 4;    // taxa para deixar um recurso para o processo 3
lambda3  = 6;    // taxa para solicitacao de recursos para o processo 3
f3       = lambda3 * (nb usando < R);
mu4      = 3;    // taxa para deixar um recurso para o processo 4
lambda4  = 5;    // taxa para solicitacao de recursos para o processo 4
f4       = lambda4 * (nb usando < R);

events

loc G1 (f1) // evento local G1 com taxa f1
loc R1 (mu1) // evento local R1 com taxa mu1
loc G2 (f2) // evento local G2 com taxa f2
loc R2 (mu2) // evento local R2 com taxa mu2
loc G3 (f3) // evento local G3 com taxa f3
loc R3 (mu3) // evento local R3 com taxa mu3
loc G4 (f4) // evento local G4 com taxa f4
loc R4 (mu4) // evento local R4 com taxa mu4

// somente os estados onde a maioria é R
reachability = (nb usando <= R);
// recursos que estão sendo usados são atingíveis
network rs1 (continuous)
aut A1
  stt dormindo
    to(usando) G1
  stt usando
    to(dormindo) R1
aut A2
  stt dormindo
    to(usando) G2
  stt usando
    to(dormindo) R2
aut A3
  stt dormindo
    to(usando) G3
  stt usando
    to(dormindo) R3
aut A4
  stt dormindo

```

```

    to(usando) G4
    stt usando
    to(dormindo) R4
results
// probabilidade para todos os recursos que estão sendo utilizados
    full      = nb usando == R;
// probabilidade para todos os recursos que estão sendo disponíveis
    empty     = nb usando == 0;
// probabilidade que o primeiro processo usa o recurso
    use1      = st P1 == usando;
// número médio de recursos ocupados
    average   = nb usando;

```

Não será possível usar replicadores para definir todos os quatro autômatos deste exemplo. De fato, o uso de replicadores somente é possível se todos os autômatos são idênticos, que não é o caso aqui desde que cada autômato tenha eventos diferentes (com taxas diferentes).

Primeiro Servidor Disponível (FAS)

Neste segundo exemplo será considerado uma fila com chegadas exponenciais comuns e um número finito (C) de servidores ordenados ($C_i, i = 1 \dots C$) e distintos. Quando um cliente chega, é servido pelo primeiro servidor disponível, *i.e.*, se C_1 está disponível, o cliente é servido por ele, caso contrário se C_2 está disponível o cliente é servido por ele, e assim por diante. Este comportamento de fila não é monotônico, então, até pode-se determinar, onde não há nenhuma solução à forma-produto para este modelo. O modelo SAN descrevendo esta fila é apresentado na Figura 3.5. A técnica básica para modelar esta fila é considerar cada servidor com um autômato com dois estados (estados ocupado e livre). A chegada em cada servidor é expressa por um evento local (chamado L_i) com uma taxa funcional que é diferente de zero e igual a λ , se todos servidores precedentes estão ocupados, e zero caso contrário. Em um dado momento, somente um servidor, o primeiro disponível, tem uma taxa de chegada diferente de zero. O fim do serviço de cada servidor é simplesmente um evento local (D_i) com taxa constante μ_i .

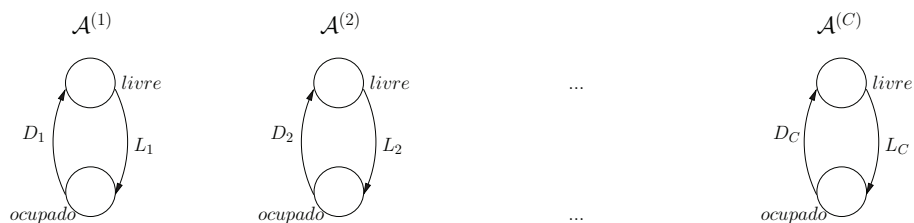


Figura 3.5: Modelo do primeiro servidor disponível

O formato textual na ferramenta PEPS 2007 para este modelo é o seguinte:

```

//===== FAS model =====
//          (with functions) C=4
//=====

identifiers
    C      = 4;
    lambda = 5;
    mu1    = 6;
    f1     = lambda;
    mu2    = 5;
    f2     = (st C1 == ocupado) * lambda;
    mu3    = 4;
    f3     = (nb[C1..C2] ocupado == 2) * lambda;
    mu4    = 3;
    f4     = (nb[C1..C3] ocupado == 3) * lambda;

events
    loc L1 (f1)
    loc D1 (mu1)
    loc L2 (f2)
    loc D2 (mu2)
    loc L3 (f3)
    loc D3 (mu3)
    loc L4 (f4)
    loc D4 (mu4)

reachability = 1;
network fas (continuous)
    aut C1
        stt livre
            to(ocupado) L1
        stt ocupado
            to(livre) D1
    aut C2
        stt livre
            to(ocupado) L2
        stt ocupado
            to(livre) D2
    aut C3
        stt livre
            to(ocupado) L3
        stt ocupado

```

```
        to(livre) D3
aut C4
    stt livre
        to(ocupado) L4
    stt ocupado
        to(livre) D4
results
full      = nb ocupado == C;
empty     = nb ocupado == 0;
use1      = st P1 == ocupado;
average   = nb ocupado;
```


4. DIAGRAMAS DE DECISÃO MULTIVALORADA

Este capítulo apresenta uma breve introdução sobre uma estrutura de dados conhecida como MDD que pode representar funções de entrada e saída multi-valorada. O MDD é uma generalização da estrutura de BDD (Diagramas de Decisão Binária) [AND99].

Um BDD é um grafo acíclico direto, que surgiu como uma estrutura de dados alternativa, utilizada em *Stochastic Petri Nets* (SPN), onde se armazena apenas as partes atingíveis de um modelo SPN, utilizando valores binários em seus nodos, estes valores são 0 e 1. Diferentemente, o MDD pode armazenar em seus nodos, valores inteiros quaisquer, não se restringindo a valores binários [SAL09a].

Um MDD é a representação de uma árvore de decisão como o BDD, onde sub-árvores são agregadas. Mais precisamente, MDD é uma árvore de decisões reduzida que não contém qualquer nó em redundância, neste contexto um nó não é único, se é uma réplica de um outro nó, e redundante, se todos os seus nós filhos são idênticos. Através de variáveis de ordenação, estes dois requisitos confirmam que MDDs fornecem uma representação canônica de funções de inteiros [KAM98].

Segundo [SAL09a], na Figura 4.1 nós não terminais são representados por círculos e nós terminais são representados por quadrados. Um dado estado $x = (x^{(1)}, \dots, x^{(N)})$ de \mathcal{S} é elemento de um subconjunto representado por um N -nível MDD se e somente se o caminho pelo MDD, iniciando no nó de nível N , seguindo abaixo a indicação $x^{(1)}$, alcance o nó terminal 1. Arcos e nós pontilhados são caminhos que direcionam somente para o nó terminal 0.

A seguir serão apresentadas algumas propriedades pertinentes à MDD [SAL09a]:

- *Nós são organizados em $N + 1$ níveis;*
- *O nível N tem somente um único nó não-terminal (a raiz), considerando que os níveis de $N - 1$ tem um ou mais nós não-terminais;*
- *O Nível 0 tem dois nós terminais: 0 e 1;*
- *Um nó não-terminal p no nível l contém n_l arcos apontando para os nós no nível $l - 1$. Um arco do nó p da posição $x^{(l)} \in \mathcal{S}^{(l)}$, que aponta para o nó q é denotado por $p[x^{(l)}] = q$;*
- *Não existem nós que sejam duplicados. Dois nós p não-terminal distintos e q no nível l são duplicatas se $\forall x^{(l)} \in \mathcal{S}^{(l)}, p[x^{(l)}] = q[x^{(l)}]$.*

A Figura 4.1 apresenta um MDD que representa um espaço de estados \mathcal{S} , subconjunto de um sistema dividido em quatro sub-sistemas [SAL09b]. Um sub-sistema $\mathcal{S}^{(4)}$ com quatro estados $\{0, 1, 2, 3\}$, o sub-sistema $\mathcal{S}^{(3)}$ e $\mathcal{S}^{(1)}$ com dois estados $\{0, 1\}$, e o sub-sistema $\mathcal{S}^{(2)}$ com três $\{0, 1, 2\}$.

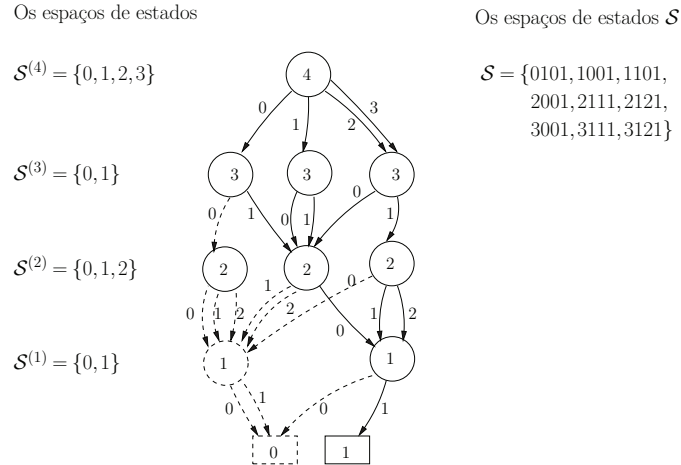


Figura 4.1: Exemplo de representação de um espaço de estados \mathcal{S} por um MDD

Formalmente, no MDD M é uma tupla $(V; r; E; var; D)$, onde V é um conjunto de vértices contendo vértice terminal 1 e uma raiz $r \in V, E \subseteq V \times V$ é um conjunto de arestas tal que $(V; E)$ forma um grafo direcionado acíclico com r como origem e 1 como o direcionador para todos os caminhos máximos no grafo. Além disso, $var : V \rightarrow \{1, \dots, n+1\}$ é uma identificação para todos os nós com um índice variável tal que $var(1) = n+1$ e D é uma identificação $D_{u,v}$ em todas as arestas (u, v) chamado de *edge domain* das arestas [AND00].

4.1 Utilizando Diagramas de Decisão Multivalorada em SAN

A explosão do espaço de estados do formalismo estruturado SAN é uma realidade pelo fato deste formalismo ser baseado em Cadeias de Markov que mesmo para sistemas menos complexos pode vir a ter uma explosão no conjunto de estados. A estruturação do formalismo SAN permitiu diminuir esta explosão de estados em modelagens baseadas em Cadeias de Markov, porém, como dito anteriormente, ainda ocorre a explosão de estados na modelagem de sistemas reais.

A atual forma de geração e armazenamento do espaço de estados atingíveis para o formalismo SAN utiliza uma abordagem baseada em Vetor de Booleanos. Cada possível estado do modelo requer uma posição que o represente nesse vetor. Se um estado é determinado como atingível, a posição correspondente do vetor é assinalada como verdadeiro, caso contrário é assinalada como falso. Portanto, o consumo de memória para este procedimento é considerado custoso em relação a representação de sistemas grandes e de alta complexidade [SCO06].

4.1.1 Obtenção de Estados Atingíveis com Redução do Espaço de Estados

Aplicar o MDD em SAN emerge como uma alternativa interessante para diminuir os problemas de explosão de espaço de estados, na medida em que o MDD armazenará somente o conjunto de espaços atingíveis de uma SAN. O formalismo SAN tem a característica de ser modular, tornando fácil assumir que cada estado do modelo SAN seja inserido em um nível da estrutura do MDD.

A Figura 4.3 apresenta um modelo MDD com uma provável estruturação do modelo SAN com 4 autômatos da Figura 4.2.

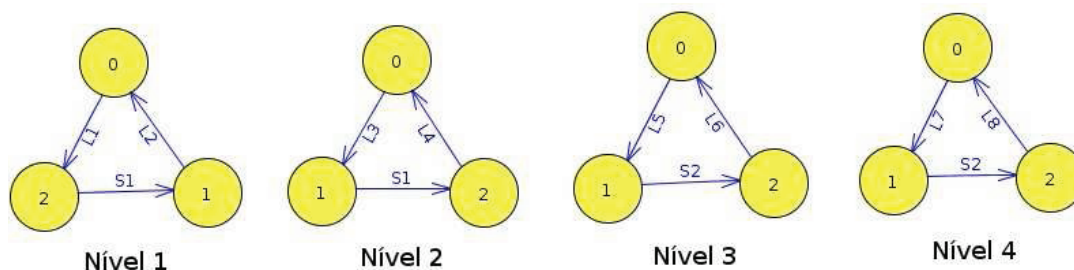


Figura 4.2: Modelo SAN com 4 autômatos [SCO06]

Na Figura 4.3, o MDD disponibiliza os autômatos de maneira que a sua disposição seja feita do nó raiz até os nós terminais da estrutura.

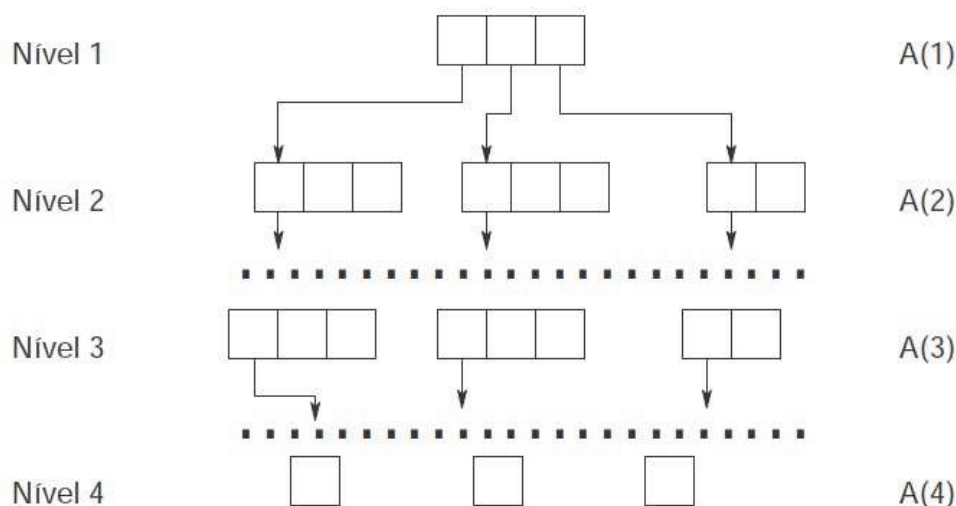


Figura 4.3: Exemplo de MDD para modelo SAN com 4 autômatos [SCO06]

Uma comparação entre modelos utilizando MDD e vetor de booleanos poder ser vista em [SCO06] com maiores detalhes. O intuito desta seção é de apenas apresentar um breve esboço com relação a aplicação do conceito de MDD sobre modelos SAN. Em [SAL09a, SCO06] foram apresentadas técnicas que permitem uma melhor adequação do espaço de estados onde é possível ter um armazenamento dos estados atingíveis através de MDD, os estados inatingíveis (ou mesmo não acessíveis) serão eliminados do RSS resultante. Neste sentido, o uso de MDD permite uma melhor adaptação para se obter estados atingíveis a partir de um determinado estado, o que é bastante interessante em relação ao estudo do comportamento de sistemas modelados com o formalismo SAN.

Em sua estrutura, o MDD não representa os estados inatingíveis. Esta afirmação de que os estados não atingíveis não são representados na porção de espaço de estado é verdadeira, mas não para todos os modelos. Nem todos os modelos tem alguma redução do espaço de estados. Por

exemplo, há modelos (como o modelo ASP - *Alternate Service Pattern*) [BOL98] em que o espaço de estados produto (PSS) é o mesmo que o espaço de estados atingíveis (RSS). Logo, neste caso, não há redução alguma do espaço de estados. Mas sim, em outros casos, o RSS de um modelo é menor (e às vezes, muito menor) do que o PSS do modelo. Desta forma, têm-se uma melhor eficiência no armazenamento de um dado conjunto de estados.

Exemplo de Obtenção de Estados Atingíveis Através do Uso de MDD

Nesta seção será apresentado um exemplo que demonstrará o uso das ferramentas **PEPS** e o algoritmo **RSS-Finder**, que aplica as técnicas de MDD, para efeitos de comparação de resultados de ambas as ferramentas. O objetivo é verificar se os resultados apresentados são efetivamente os mesmos.

O primeiro exemplo, Figura 4.4, é um modelo de compartilhamento de recursos tradicional, conforme descrito na seção 3.5.6, onde $N = 4$ e $R = 2$.

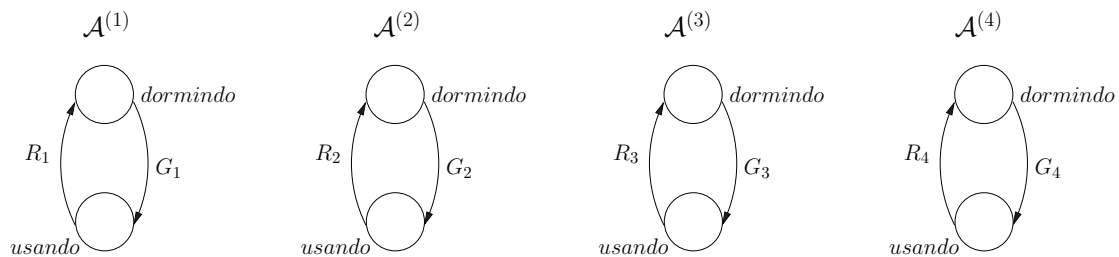


Figura 4.4: Modelo de compartilhamento de 4 recursos

Para este modelo serão obtidos os estados atingíveis, apresentados na Tabela 4.1, utilizando a ferramenta PEPS em sua configuração mais básica baseada no método de potência. Segundo a ferramenta, 11 dos 16 possíveis estados globais são atingíveis.

EG	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$	$\mathcal{A}^{(3)}$	$\mathcal{A}^{(4)}$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
12	1	1	0	0

Tabela 4.1: Lista de estados globais atingíveis do modelo da Figura 4.4

A Tabela 4.1 aponta quais os estados globais atingíveis do modelo referente à Figura 4.4, **EG** representa estes estados. Como pode-se observar na tabela, cada coluna corresponde a um autômato e as linhas representam os estados globais. É possível observar ainda que as linhas correspondentes a posição 7, 11, 13, 14 e 15 não foram apresentadas na tabela por serem estados globais inatingíveis.

Para efeito de comparação será aplicado no modelo da Figura 4.4 o algoritmo **RSS-Finder** onde será obtido a árvore MDD correspondente ao espaço de estados do modelo, apresentada na Figura 4.5. Mesmo que a aplicação deste algoritmo em alguns modelos de sistemas não surta efeito na redução do espaço de estados, pode-se notar que na Figura 4.5 é apresentada uma estrutura demonstrando, através das linhas tracejadas, que representam os estados inatingíveis, a redução do espaço de estados. Além do ganho por conta do baixo custo computacional devido à possível redução do espaço de estados o resultado será o mesmo apresentado pela ferramenta PEPS2007.

Os espaços de estados

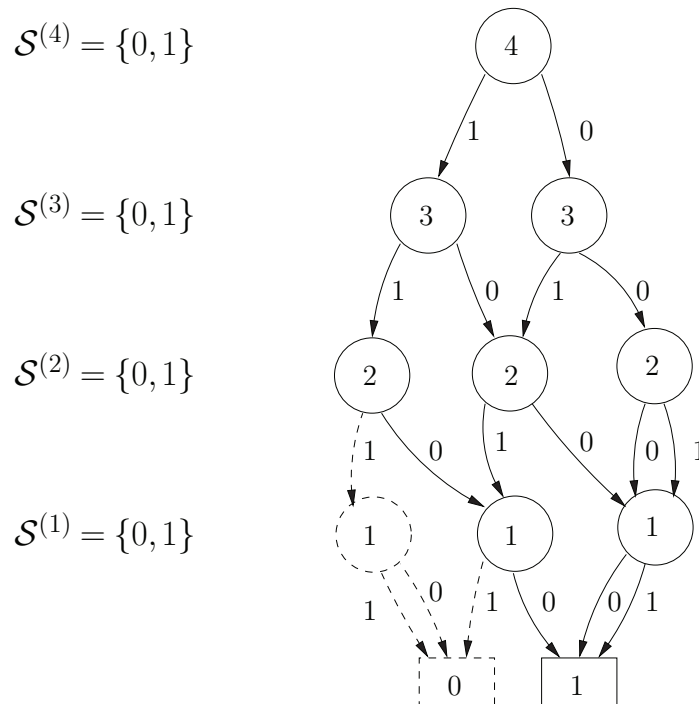


Figura 4.5: Representação de um espaço de estados \mathcal{S} por um MDD, referente à Figura 4.4

Na Figura 4.5, os estados globais formados pelas arestas rotuladas $\{1,1,1,1\}$, $\{1,1,1,0\}$, $\{1,1,0,1\}$, $\{1,0,1,1\}$ e $\{0,1,1,1\}$ são direcionados ao nó terminal 0 indicando que estes estados são inatingíveis. Para todos os efeitos estas linhas tracejadas serão sempre omitidas, uma vez que o algoritmo apenas manipula em suas listas os estados atingíveis. A Figura 4.6 é uma apresentação da árvore com os estados inatingíveis omitidos. Assim, temos os estados globais formados pelas arestas $\{1,1,0,0\}$ (em destaque pontilhado), $\{1,0,1,0\}$, $\{1,0,0,0\}$, $\{1,0,0,1\}$, $\{0,1,1,0\}$, $\{0,1,1,0\}$, $\{0,1,0,0\}$, $\{0,1,0,1\}$, $\{0,0,0,0\}$, $\{0,0,0,1\}$, e $\{0,0,1,1\}$, que são direcionados ao nó terminal 1, indicando que estes são os estados atingíveis do modelo. Comparando estes resultados com os dados apresentados na Tabela 4.1 percebe-se que são exatamente os mesmos. Desta forma, o uso de MDD

garante, para alguns modelos, a redução do espaço de estados como demonstrado na Figura 4.6.

Os espaços de estados

$$\mathcal{S}^{(4)} = \{0, 1\}$$

$$\mathcal{S}^{(3)} = \{0, 1\}$$

$$\mathcal{S}^{(2)} = \{0, 1\}$$

$$\mathcal{S}^{(1)} = \{0, 1\}$$

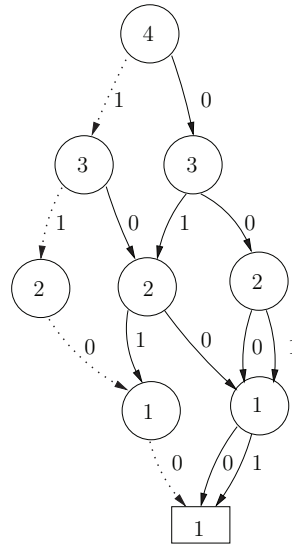


Figura 4.6: Exemplo de MDD referente à Figura 4.4 com estados inatingíveis omitidos

A utilização de MDD em SAN surgiu como uma técnica que viabiliza a obtenção dos estados atingíveis que aproveita a vantagem da estrutura modular do formalismo SAN, que também permite representar a matriz de taxa de transição de CTMC por meio de uma soma de produtos generalizados Kronecker [SAL09a].

5. PROPOSTA DO ALGORITMO E ESTRUTURA DE DADOS

Neste capítulo será apresentada uma abordagem dos algoritmos para cálculos de estados sucessores e predecessores que serão aplicados na ferramenta de simulação visual para o formalismo SAN baseada em eventos, onde extrai-se o conjunto de eventos para este modelo e constrói-se a função de transição para representar as mudanças de estados possíveis. Na seção 5.2 é descrito o processo de estruturação de dados para a representação de modelos SAN na linguagem XML.

5.1 Algoritmo para Obtenção de Estados Sucessores e Predecessores

O comportamento de cada autômato, $\mathcal{A}^{(i)}$, para $i = 1, \dots, N$, é descrito por um conjunto de matrizes quadradas, de ordem n_i . Estas matrizes que compõem o Descritor Markoviano¹ permite a identificação das transições entre os estados através de tensores. Baseado neste conceito pode-se representar o conjunto dos eventos locais e sincronizantes em uma matriz de transição², representando o espaço de estados que é obtido pelo produto cartesiano da dimensão de todos os autômatos de uma SAN, definido como espaço de estados produto \hat{S} [SAL09a]. No caso do modelo da Figura 3.1 o \hat{S} é $3 \times 2 = 6$.

Ressalta-se que o foco deste trabalho é o desenvolvimento de uma aplicação para simulação visual de SAN com algoritmos embutidos para o cálculo de eventos sucessores bem como os predecessores. Portanto, não faz parte do escopo a apresentação do conceito detalhado do Descritor Markoviano e seus tensores, maiores detalhes em [FER98a, WEB09, SAL09b, CZE10]. A próxima seção faz uma abordagem simplificada de como representar autômatos em matrizes.

5.1.1 Matrizes de Transições para Representação de SAN

Para representar o conjunto de eventos locais, $\xi = e_1, e_2, e_3, e_5$ do modelo, referente a Figura 3.1 apresentada na Seção 3.2, em matrizes de transições é necessário disponibilizar uma matriz quadrada de ordem n igual à quantidade de estados do autômato onde os eventos locais ocorrem. No caso dos eventos e_1, e_2, e_3, e_5 , será representado, como a seguir, uma matriz quadrada de ordem 3 para o autômato $\mathcal{A}^{(1)}$ e ordem 2 para o autômato $\mathcal{A}^{(2)}$.

$$\mathcal{M}_l^{(1)} = \begin{array}{c|ccc} & 0^{(1)} & 1^{(1)} & 2^{(1)} \\ \hline 0^{(1)} & 0 & e_1 & 0 \\ 1^{(1)} & 0 & 0 & e_2 \\ 2^{(1)} & e_3 & 0 & 0 \end{array} \quad \mathcal{M}_l^{(2)} = \begin{array}{c|cc} & 0^{(2)} & 1^{(2)} \\ \hline 0^{(2)} & 0 & e_5 \\ 1^{(2)} & 0 & 0 \end{array}$$

De acordo com a definição do Descritor Markoviano em [FER98a, CZE10, WEB09, SAL09b] deve-se representar os eventos locais identificando-os no autômato e a partir destes construir o tensor

¹Uma representação algébrica para modelos SAN.

²Matriz de transição é o gerador infinitesimal da cadeia de Markov de um modelo original SAN.

com as matrizes das transições locais, $\mathcal{M}_l^{(1)}$, correspondentes aos autômatos. Os tensores acima se referem aos autômatos $\mathcal{A}^{(i)}$ da Figura 3.1.

Uma vez construído o tensor local, será identificado o conjunto de eventos sincronizantes, para os autômatos $\mathcal{A}^{(i)}$ da Figura 3.1 há apenas o evento $\xi = e_4$, e construído os tensores das transições sincronizadas que representam a ocorrência dos eventos sincronizantes em cada autômato. Desta forma, tem-se os tensores das transições sincronizantes $\mathcal{M}_{e_4^+}^{(i)}$ correspondentes aos autômatos $\mathcal{A}^{(i)}$.

$$\mathcal{M}_{e_4^+}^{(1)} = \begin{array}{c|ccc} & 0^{(1)} & 1^{(1)} & 2^{(1)} \\ \hline 0^{(1)} & 0 & 0 & 0 \\ 1^{(1)} & 0 & 0 & 0 \\ 2^{(1)} & \pi_2 & \pi_1 & 0 \end{array} \quad \mathcal{M}_{e_4^+}^{(2)} = \begin{array}{c|cc} & 0^{(2)} & 1^{(2)} \\ \hline 0^{(2)} & 0 & 0 \\ 1^{(2)} & e_4 & 0 \end{array}$$

Note que o autômato $\mathcal{A}^{(2)}$ quando transposto para suas respectivas matrizes pode-se verificar que a taxa de ocorrência vai estar contida em apenas uma das matrizes e as demais apenas indicarão a ocorrência, pois realizar-se-á um produto entre as matrizes e a taxa assim não é modificada. A partir desta estruturação das matrizes será apresentado o algoritmo para a obtenção dos estados sucessores e predecessores (antecessores).

5.1.2 Estruturação do Algoritmo

Tomando a matriz de transições $\mathcal{M}^{(i)}$ como referência será definido que as transições entre os estados serão representadas pela posição i, j , sendo que i representa uma linha da matriz e j uma coluna. Os elementos de uma posição i, j representam a taxa para sair do estado representado pela linha i e ir para o estado representado pela coluna j : **(i)** saber quais são os sucessores significa percorrer uma linha da matriz; **(ii)** saber quais são os predecessores significa percorrer uma coluna desta matriz.

Para a obtenção dos estados sucessores e predecessores deve-se verificar e identificar os estados globais a partir da execução dos autômatos pertencentes a uma SAN, listando os estados através de uma varredura no espaço de estados de um autômato. A identificação dos estados sucessores se dá basicamente pela observação das seguintes características: **(i)** um estado x é sucessor de um estado y quando existe uma transição com taxa não nula que pode ocorrer no estado y e leva ao estado x ; **(ii)** analogamente, um estado y é predecessor de um estado x quando existe uma transição com taxa não nula que pode ocorrer no estado y e leva ao estado x . O conjunto de estados sucessores indica os estados para onde se pode ir, e o conjunto de estados predecessores indicam os estados para onde se pode partir em um modelo SAN.

A seguir temos uma transcrição dos autômatos da Figura 5.1 para matrizes de transição tanto para auxiliar na descoberta dos estados sucessores, apresentados na Tabela 5.1, quanto para estados predecessores, apresentados na Tabela 5.2.

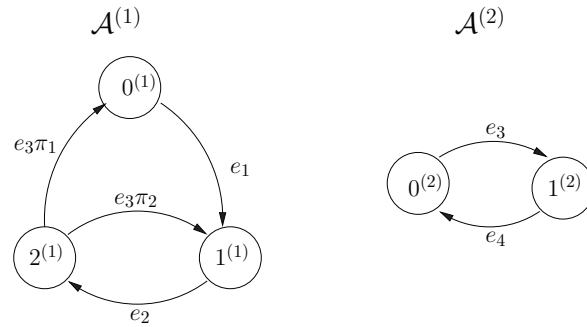


Figura 5.1: Modelo SAN com 2 autômatos.

		0 ⁽¹⁾	1 ⁽¹⁾	2 ⁽¹⁾
$\mathcal{M}_q^{(1)}$	0 ⁽¹⁾	0	e_1	0
	1 ⁽¹⁾	0	0	e_2
	2 ⁽¹⁾	e_3	e_3	0

		0 ⁽²⁾	1 ⁽²⁾
$\mathcal{M}_q^{(2)}$	0 ⁽²⁾	0	e_4
	1 ⁽²⁾	e_3	0

Tabela 5.1: Matriz de transição para estados sucessores

		0 ⁽¹⁾	1 ⁽¹⁾	2 ⁽¹⁾
$\mathcal{M}_t^{(1)}$	0 ⁽¹⁾	0	0	e_3
	1 ⁽¹⁾	e_1	0	e_3
	2 ⁽¹⁾	0	e_2	0

		0 ⁽²⁾	1 ⁽²⁾
$\mathcal{M}_t^{(2)}$	0 ⁽²⁾	0	e_3
	1 ⁽²⁾	e_4	0

Tabela 5.2: Matriz de transição transposta, da Tabela 5.1, para estados predecessores

Como pôde-se observar foi realizado a transcrição do modelo SAN da Figura 5.1 para matrizes de transição, da Tabelas 5.1 e 5.2. Estas matrizes tem as respectivas linhas e colunas de acordo com a quantidade de estados de cada autômato do modelo fornecido.

A descoberta dos estados sucessores, conjunto não vazio dos estados y , tais que possua uma transição do estado x para algum estado y , se dá pela verificação das linhas e colunas da matriz de transições, como se pode observar na Tabela 5.1. E a descoberta dos estados predecessores, conjunto não vazio dos estados x , tais que possua uma transição do estado y para algum estado x , será feita pela verificação das colunas e linhas da matriz de transição, para este caso é importante observar que realizando a transposição da matriz representada na Tabela 5.1, demonstrada na Tabela 5.2, o procedimento de descoberta dos estados predecessores será tal qual para o conjunto de estados sucessores.

Exemplificando: para saber qual o estado sucessor do estado atual de um autômato deve-se percorrer, na matriz quadrada obtida, cada linha e coluna e então identificar se há eventos. Caso haja eventos, é a indicação de que a transição do estado representado pela linha leva ao estado representado pela coluna pelo disparo do evento encontrado de acordo com a taxa do evento, se a taxa do evento for constante. Caso o evento contenha uma taxa funcional é preciso avaliar a função para verificar se o resultado da função não é nula, se não for nula então o evento pode ser disparado. O evento e_1 encontrado na intersecção da linha 0 com a coluna 1 do autômato $\mathcal{A}^{(1)}$ quando disparado determina que o estado 1 deste autômato será o sucessor do estado 0.

5.1.3 Definições do Algoritmo

Considerando que o espaço de estados produto \hat{S} de um modelo é composto por um conjunto de estados globais \tilde{s} , estados locais e uma coleção finita $\Pi = \{e_1, \dots, e_p\}$ de P eventos (eventos locais e sincronizantes são todos do conjunto Π). Considerando ainda que SAN contém transições que associam os estados entre si, uma função de transição ϕ de um autômato $\mathcal{A}^{(i)}$ permite o disparo de eventos de acordo com a avaliação das taxas vinculadas aos eventos.

A aplicação da função de transição $\phi(s^{(l)}, e_1)$ é dada por um evento e_1 operando sobre cada estado local $s^{(l)}$ em $\tilde{s} = \{s^{(1)}; \dots; s^{(l)}\}$, mudando ou não o estado global. Uma ativação de eventos deve verificar todos os autômatos envolvidos, indicando que um evento $e \in \Pi$ está habilitado (ou desabilitado) considerando cada estado local $s^{(l)} \in \hat{S}$ para cada autômato $\mathcal{A}^{(i)}$ na rede. Um evento é dito disparável no estado local $s^{(l)}$ se há uma transição identificada com e de $s^{(l)}$ no autômato i . Se e é um evento sincronizante e disparável em um estado local, isto não implica que ele é, no momento, disparável no estado global. A ocorrência dos eventos muda o estado global \tilde{s} para outro estado global \tilde{r} , a função de transição determina a permanência no mesmo estado global caso não haja a ocorrência de um dado evento.

A seguir serão descritas as definições informais, cujo objetivo é demonstrar algumas características para a estruturação de funções a serem aplicadas em modelos SAN para a obtenção de estados sucessores e predecessores.

Definição 1: a função de obtenção de estados sucessores e predecessores $\phi(s^{(l)}, e)$ retorna um novo estado local resultando do evento e disparando sobre o estado local $s^{(l)}$. O novo estado local retornado pode ser o mesmo $s^{(l)}$, se e não o atinge, se caso não atingir a função deverá retornar 0.

Definição 2: um evento e é dito estar habilitado no estado global $\tilde{s} \in \hat{S}$, se e somente se $\phi(s^{(l)}, e) = r$, onde $s \neq r$, e $r \in \hat{S}$. Analogamente, um evento é dito estar desabilitado no estado \tilde{s} , se e somente se $\phi(s^{(l)}, e) = r$, e $s = r$.

Definição 3: ξ é a lista de eventos que se encontra na posição $(\mathcal{M}_{s^{(l)}, j}^{(i)})$ da matriz.

Definição 4: $\mathcal{M}_{s^{(l)},j}^{(i)}$ representa o (i,j) -ésimo elemento da matriz, sendo i o autômato, $s^{(l)}$ é a linha que corresponde ao estado de onde foi disparado um determinado evento e j a coluna da matriz.

Definição 5: $O^{(e)}$ é conjunto dos autômatos afetados pelo evento e .

Considerando que um evento e é disparável no estado global \tilde{s} se e somente se para todos os autômatos envolvidos pelo evento e , o conjunto de estados sucessores (ϖ_s) para o evento e não são vazios ($\forall i \in [1..N], \phi(s^{(l)}, e) \neq \emptyset$).

Uma aplicação da função de obtenção de estados sucessores e predecessores no produto do espaço de estados $\hat{\mathcal{S}}$, PSS, formados pelos estados atuais da rede, deve resultar no conjunto de estados sucessores ou predecessores, se e somente se r não for vazio e a função de sucessão/predecessão $\phi(s^{(l)}, e) = r^{(l)}$. A aplicação da função $\phi(s^{(l)}, e)$ atribuída ao evento e_1 , do autômato apresentado na Figura 5.2, indicará qual o estado sucessor para cada estado global $\tilde{s} \in \hat{\mathcal{S}}$ e $s^{(l)} = \{s^{(1)}; \dots; s^{(l)}\}$, determinando qual será o comportamento do sistema prevendo se o estado atual terá o seu estado sucessor ativado.

Considerando o estado local $s^{(l)}$ do autômato $\mathcal{A}^{(i)}$. A aplicação da função ϕ resulta no estado $r^{(l)}$ significando que o evento está habilitado para o estado local $s^{(l)}$. Se a aplicação da função ϕ resulta vazio, conseqüentemente o evento e é desabilitado para este estado local $s^{(l)}$.

Analogamente, dado um estado global $\tilde{s} = \{s^{(1)}; \dots; s^{(l)}\}$, a função de transição local pode ser visualizada genericamente como $\phi(s^{(l)}, e) = \tilde{r}$ se e está habilitado para \tilde{s} , ou $\phi(s^{(l)}, e) = \tilde{s}$ se e_p está desabilitado para \tilde{s} .

Algoritmo para Obtenção dos Estados Sucessores

O Algoritmo 5.1 mostra o procedimento de obtenção de estado sucessor, de acordo com as proposições tratadas nas seções anteriores, pressupondo a realização de um disparo de evento, onde o autômato e o estado são informados pelo usuário e o autômato pertencente à ϖ_s (conjunto dos estados sucessores) é analisado para disparar um evento e_p e verificar os estados sucessores através da execução da função de sucessão ϕ , considerando o exemplo na Figura 5.2.

O fato de uma SAN ser constituída de autômatos compostos por estados que por sua vez possuem transições com eventos locais ou sincronizantes e considerando o fato de haver dependências entre eles por meio das taxas funcionais que determina a ocorrência dos eventos. O atendimento a estas características garantirá uma simulação coerente do disparo de eventos implicando em um tratamento mais elaborado por parte do algoritmo.

Para a explicitação do algoritmo proposto será utilizado a SAN apresentada na Figura 5.2, para que se tenha um melhor entendimento das funções pertencentes ao Algoritmo 5.1.

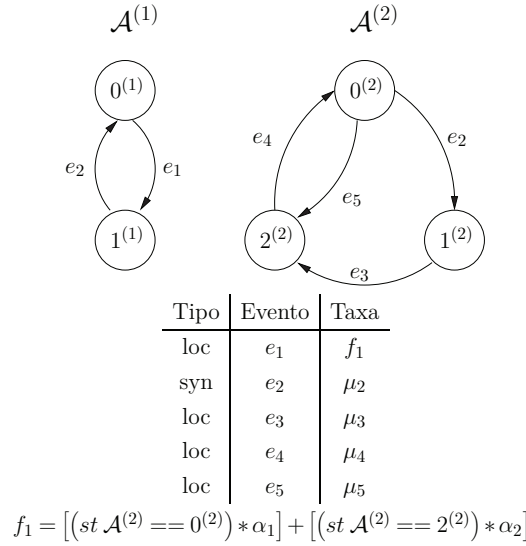


Figura 5.2: Modelo SAN com 2 autômatos com taxa funcional f_1

Inicialmente, o algoritmo deve realizar uma varredura nas matrizes de transições que representam os autômatos da SAN, a Tabela 5.3 apresenta estas matrizes referente à Figura 5.2, para que assim seja realizada uma verificação das linhas e colunas (linhas 4 e 6) e então descobrir, por meio do estado atual, $s^{(l)}$, fornecido pelo usuário, quais eventos podem ser disparados. O estado atual de um autômato é representado na tabela de transição por uma das linhas existentes na matriz. A cada vez que é encontrado um ou mais eventos no (i,j) -ésimo elemento da matriz, a lista de eventos ξ é atualizada, processo realizado na linha 8. Esta lista será verificada para avaliar se o evento desta lista se encontra inserido na lista de eventos analisados (linha 9), enquanto a lista não for vazia serão realizados os procedimentos avaliação dos disparos dos eventos. A função *pick()* tem o propósito de pegar cada evento contido em ξ (linha 10) para verificar se o mesmo já foi analisado. Na linha 11 é verificado se o evento pertence a lista de eventos analisados, uma vez analisado o evento é ignorado para os demais autômatos, e ainda é realizada uma avaliação para identificar se o evento tem taxa funcional (procedimentos executados da linha 15 até a 19) para que seja realizados os cálculos necessários para o disparo do evento, as taxas funcionais exigem um tratamento especial no algoritmo pelo fato de ser necessário uma avaliação léxica e sintática em sua estrutura.

De acordo com a Figura 5.2, o modelo apresenta dois autômatos com 2 e 3 estados, respectivamente, com taxa funcional definida para o evento e_1 . Com a identificação da taxa funcional o algoritmo realizará a solução da função estabelecida (linha 14), no caso do exemplo fornecido, ($f_1 = [(st \mathcal{A}^{(2)} == 0^{(2)}) * \alpha_1] + [(st \mathcal{A}^{(2)} == 2^{(2)}) * \alpha_2]$), será feita uma varredura na lista de autômatos verificando se o autômato $\mathcal{A}^{(2)}$ se encontra no estado $0^{(2)}$ ou se o autômato $\mathcal{A}^{(2)}$ se encontra no estado $2^{(2)}$. Caso se encontre em um dos 2 estados no autômato $\mathcal{A}^{(2)}$ da rede, será calculada a expressão matemática. Finalmente, a taxa funcional será analisada para verificar se o resultado é diferente de zero e assim permitir ou não o disparo do evento com a taxa funcional (e_1). Note que se não houver taxa funcional e o evento for local, o mesmo será disparado imediatamente.

Para os eventos sincronizantes presentes na rede será realizado uma varredura na lista de autômatos (linha 26) e verificar se todos os estados de onde se origina a transição são os estados atuais, e assim realizar o disparo destes eventos. Por fim, caso seja realizado o disparo do evento a lista de estados sucessores será atualizada assim como a lista de eventos analisados e em seguida retornado os estados sucessores.

Algoritmo 5.1: - Procedimento de obtenção de estados sucessores (s)

```

1:  $\varpi_s$  é a lista de estados sucessores ( $s$ )
2:  $\Theta$  é a lista de eventos analisados
3: {procurando os eventos na matriz de transições em cada autômato existente na SAN}
4: for  $i = 0$  to  $N - 1$  do
5:   for  $j = 0$  to  $n_{coluna_i} - 1$  do
6:     {verifica se elemento da linha e coluna da matriz ( $\mathcal{M}$ ) é diferente de 0}
7:     if  $(\mathcal{M}_{s^{(i)},j}^{(i)} \neq 0)$  then
8:       {adicionando evento(s) de cada posição da matriz na lista de eventos}
9:        $\xi \leftarrow \text{addList}(\mathcal{M}_{s^{(i)},j}^{(i)})$ 
10:      while  $\xi \neq \emptyset$  do
11:         $e \leftarrow \text{pick}(\xi)$  {função para obter um elemento da lista de eventos}
12:        if  $(e \notin \Theta) \vee (i = \min(O(e)))$  then
13:           $r \leftarrow s$ 
14:          dispara  $\leftarrow$  false
15:          {taxas funcionais em eventos são avaliadas}
16:          if  $(e$  tem taxa funcional  $)$  then
17:            Tx  $\leftarrow f_e(s)$ 
18:          else
19:            Tx  $\leftarrow \tau_e$ 
20:          end if
21:          if  $(Tx \neq 0)$  then
22:            dispara  $\leftarrow$  true
23:            if  $(e$  é evento local  $)$  then
24:               $r^{(i)} \leftarrow j$ 
25:            else
26:              {procurando cada evento envolvido em  $O(e)$ }
27:              for  $l \in O(e)$  do
28:                if  $(l \neq i)$  then
29:                   $r^{(i)} \leftarrow \phi(s^{(l)}, e)$  {obtendo os estados sucessores em função de  $e$ }
30:                  if  $(r^{(i)} = \emptyset)$  then
31:                    dispara  $\leftarrow$  false
32:                    break
33:                  end if
34:                end if
35:              end for
36:            end if
37:          end if
38:          if dispara then
39:             $\varpi_s \leftarrow \text{add}(r)$ 
40:          end if
41:           $\Theta \leftarrow \text{add}(e)$ 
42:        end if
43:      end while
44:    end if
45:  end for
46: end for
47: return  $\varpi_s$  {retornando os estados sucessores}

```

Exemplo de Uso do Algoritmo para Obtenção dos Estados Sucessores

A seguir, é demonstrado a obtenção dos estados sucessores através do disparo de transições dos estados locais atuais (estado atualmente posicionado ou selecionado) nos autômatos do modelo. Para isto, deve-se criar a matriz de transição do respectivo modelo (Figura 5.2), a Tabela 5.3 repre-

senta esta matriz. Os eventos locais serão disparados através da função ϕ , gerando os respectivos estados sucessores.

$$\mathcal{M}_q^{(1)} = \begin{array}{c|cc} & 0^{(1)} & 1^{(1)} \\ \hline 0^{(1)} & 0 & e_1 \\ 1^{(1)} & e_2 & 0 \end{array} \quad \mathcal{M}_q^{(2)} = \begin{array}{c|ccc} & 0^{(2)} & 1^{(2)} & 2^{(2)} \\ \hline 0^{(2)} & 0 & e_2 & e_5 \\ 1^{(2)} & 0 & 0 & e_3 \\ 2^{(2)} & e_4 & 0 & 0 \end{array}$$

Tabela 5.3: Matriz de transição referente a Figura 5.2

Dado que o estado local atual do autômato $\mathcal{A}^{(1)}$ seja o estado $1^{(1)}$ e o estado local atual para o autômato $\mathcal{A}^{(2)}$ seja o estado $0^{(2)}$, conforme indicado na Figura 5.3 em destaque tracejado, se houver disparo das respectivas transições será retornado os seguintes estados: $(0^{(1)}, 1^{(2)})$, em destaque pontilhado na figura. Note que o disparo das transições é simultâneo por ser tratar de um evento sincronizante, e_2 .

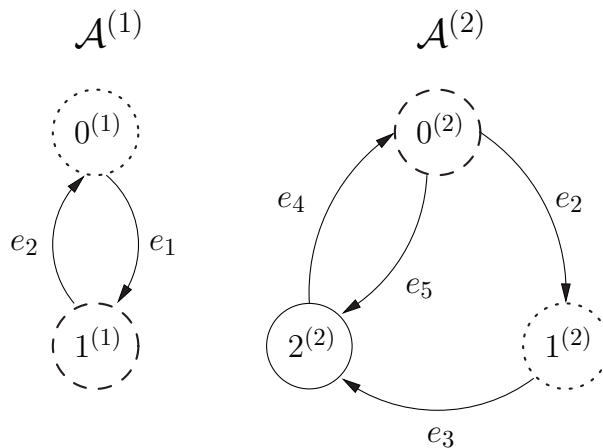


Figura 5.3: Modelo SAN com 2 autômatos

Algoritmo para Obtenção dos Estados Predecessores

Todos os procedimentos para se obter os estados predecessores de uma Rede de Autômatos de Estocásticos é bastante semelhante aos procedimentos de obtenção de estados sucessores, exceto pelo fato de se ter uma matriz de transição transposta.

O Algoritmo 5.1 mostra o procedimento de obtenção do estado sucessor, para que se obtenha os estados predecessores deve-se realizar os procedimentos citados na seção 5.1.1, ou seja, encontrar a matriz transposta da matriz de ordem n que representa a Rede de Autômatos Estocásticos e assim com a aplicação do Algoritmo 5.2 sobre a matriz transposta será obtido os estados predecessores.

Algoritmo 5.2: - Procedimento de obtenção de estados predecessores ($s^{(l)}$)

```

1: { $\varpi_s$  é a lista de estados predecessores ( $s^{(l)}$ )}
2: { $\Theta$  é a lista de eventos analisados}
3: {procurando os eventos na matriz de transições}
4: for  $i = 0$  to  $N - 1$  do
5:   for  $j = 0$  to  $ncoluna_i - 1$  do
6:     {verifica se elemento da linha e coluna da matriz transposta ( $\mathcal{M}^t$ ) é diferente de 0}
7:     if ( $\mathcal{M}_{s^{(l)},j}^{t(i)} \neq 0$ ) then
8:       {adicionando os eventos de cada posição da matriz transposta ( $\mathcal{M}^t$ ) na lista de eventos}
9:        $\xi \leftarrow \text{addList}(\mathcal{M}_{s^{(l)},j}^{t(i)})$ 
10:      while  $\xi \neq \emptyset$  do
11:         $e \leftarrow \text{pick}(\xi)$  {função para obter um elemento da lista de eventos}
12:        if ( $e \notin \Theta$ ) || ( $i = \min(O(e))$ ) then
13:           $r \leftarrow s$ 
14:          dispara  $\leftarrow$  false
15:          {taxas funcionais em eventos são avaliadas}
16:          if ( $e$  tem taxa funcional ) then
17:            Tx  $\leftarrow f_e(s)$ 
18:          else
19:            Tx  $\leftarrow \tau_e$ 
20:          end if
21:          if (Tx  $\neq 0$ ) then
22:            dispara  $\leftarrow$  true
23:            if ( $e$  é evento local ) then
24:               $r^{(i)} \leftarrow j$ 
25:            else
26:              {procurando cada evento envolvido em  $O(e)$ }
27:              for  $l \in O(e)$  do
28:                if ( $l \neq i$ ) then
29:                   $r^{(l)} \leftarrow \phi(s^{(l)}, e)$  {obtendo os estados predecessores em função de  $e$ }
30:                  if ( $r^{(l)} = \emptyset$ ) then
31:                    dispara  $\leftarrow$  false
32:                    break
33:                  end if
34:                end if
35:              end for
36:            end if
37:          end if
38:        end while
39:      end if
40:    end if
41:  end for
42: end for
43: return  $\Delta_s$  {retornando os estados predecessores}

```

Desta forma, a Tabela 5.4 apresenta a transposição da matriz de transição apresentada na Tabela 5.3.

$$\mathcal{M}_q^{(1)} = \begin{array}{c|cc} & 0^{(1)} & 1^{(1)} \\ \hline 0^{(1)} & 0 & e_2 \\ 1^{(1)} & e_1 & 0 \end{array} \quad \mathcal{M}_q^{(2)} = \begin{array}{c|ccc} & 0^{(2)} & 1^{(2)} & 2^{(2)} \\ \hline 0^{(2)} & 0 & 0 & e_4 \\ 1^{(2)} & e_2 & 0 & 0 \\ 2^{(2)} & e_5 & e_3 & 0 \end{array}$$

Tabela 5.4: Matriz de transição transposta referente a Figura 5.2

Para um estado ser predecessor de outro através de um evento com taxa ou probabilidade funcional é necessário que a taxa (e eventualmente a probabilidade) tenham valores não nulos

considerando o estado de origem (o candidato a predecessor).

Exemplo de Uso do Algoritmo para Obtenção dos Estados Predecessores

A seguir, é demonstrado a obtenção dos estados predecessores através do disparo de transições dos estados locais atuais (estado atualmente posicionado ou selecionado) nos autômatos do modelo. Para isto, deve-se criar a matriz de transição transposta do respectivo modelo (Figura 5.2), a Tabela 5.4 representa esta matriz. Os eventos locais serão disparados através da função ϕ , gerando os respectivos estados predecessores.

Tomando a Figura 5.4 como exemplo, será aplicado o algoritmo 5.2 para descobrir quais os estados predecessores dos estados locais atuais dos autômatos. Dado que o estado local atual do autômato $\mathcal{A}^{(1)}$ seja o estado $1^{(1)}$ e o estado local atual para o autômato $\mathcal{A}^{(2)}$ seja o estado $2^{(2)}$, conforme indicado na Figura 5.4 em destaque pontilhado, se houver disparo dos eventos e_1 e e_3 , e estes disparos aplicados na matriz transposta da Tabela 5.4, serão retornados, respectivamente, os seguintes estados: $(0^{(1)}, 2^{(2)})$ e $(1^{(1)}, 1^{(2)})$. Neste caso, dado a ocorrência destes dois eventos têm-se o estado predecessor $(0^{(1)}, 1^{(2)})$, conforme mostrado em destaque tracejado na Figura 5.4.

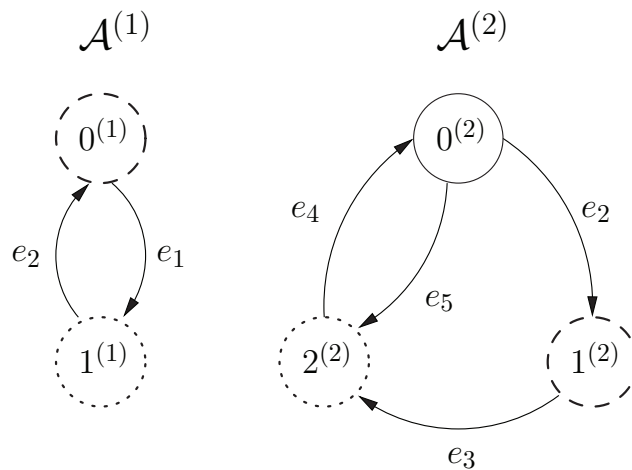


Figura 5.4: Modelo SAN com 2 autômatos

Considerações

Observando os aspectos estruturais dos modelos SAN, percebe-se os estados globais e o efeito dos eventos sobre eles. A obtenção dos estados sucessores e predecessores é um passo importante para se avaliar o comportamento de sistemas, e estes estados indicam um possível comportamento esperado em parte do sistema modelado.

5.2 Estrutura de Dados Baseada em XML para Representação da SAN

Neste capítulo será apresentado a estrutura de dados que representará um modelo descrito no formalismo SAN. Esta estrutura de dados tem como finalidade facilitar uma eventual transação entre aplicações que se utilizam do formalismo estruturado SAN.

Para representar em um formato estruturado um modelo baseado em SAN será utilizado a XML (*eXtended Markup Language*) [BER00]. O XML é um formato para a criação de documentos com dados organizados de forma hierárquica. É um conjunto de regras para a codificação de documentos em forma estruturada e legível por máquina [BER00]. Projetado para enfatizar a simplicidade, generalidade e usabilidade nas transações de informações entre aplicações dentro e fora da *Internet*. Maiores detalhes sobre como se estrutura um arquivo no formato XML pode ser encontrado em [BRY97, BER00].

Por ser uma formato de dados textual, a XML permite a SAN representar os autômatos do modelo de forma prática. Para representar a SAN em um formato XML primeiro é necessário definir a estrutura hierárquica que implicará em definir qual *tag* será a principal, bem como definir o corpo da estrutura do modelo SAN em XML com suas respectivas *tags*. O tópico a seguir define a estrutura básica desta representação.

5.2.1 Formato Proposto para a Estrutura de Dados

A Figura 5.5 apresenta uma SAN com 2 autômatos que servirá como base para exemplificar a transposição de um modelo SAN para o padrão de arquivos XML.

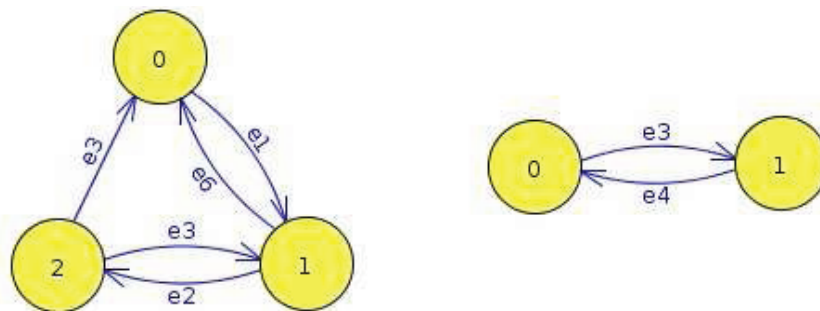


Figura 5.5: Exemplo de SAN com 2 autômatos

A estrutura do arquivo XML será composto pelas seguintes blocos de informações:

- **san:** define o bloco principal (raiz) incluindo a estruturação que o sistema terá, aqui teremos todas as propriedades de um modelo SAN, como por exemplo estados, transições e taxas. Uma SAN poderá conter n estados e transições;
- **states:** são os estados que a SAN possui. Neste bloco será descrito o posicionamento gráfico de cada estado, deverá conter um bloco interno que descreverá as posições referente a cada estado;

- **transitions:** o conjunto de transições definidos nos estados de uma SAN;
- **rates:** correspondem as taxas definidas para cada transição;
- **events:** define os eventos de cada transição disponível no modelo.
- **x,y:** define os posicionamento de cada estado, no editor gráfico, disponível no modelo.

A seguir, têm-se um exemplo do arquivo XML para a SAN apresentada na Figura 5.5. Observa-se que para um modelo descrito em XML estar sintaticamente correto, existem bibliotecas que validam o arquivo a partir de um esquema (*XML Schema Definition*, ou XSD) predefinido. Uma vantagem de se usar XML para descrever uma SAN é a possibilidade de validar arquivos sintaticamente [CZE10].

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<san id="1" name="san">
  <!--The list of automata.-->
  <automaton id="0">
    <!--The list of states.-->
    <state id="0">
      <x>169.0</x>
      <y>110.0</y>
      <name>0</name>
    </state>
    <state id="1">
      <x>298.0</x>
      <y>116.0</y>
      <name>1</name>
    </state>
    <state id="2">
      <x>233.0</x>
      <y>262.0</y>
      <name>2</name>
    </state>
    <!--The list of transitions.-->
    <transition>
      <from>0</from>
      <to>1</to>
      <event>e1</event>
      <rate>t1</rate>
    </transition>
  </automaton>
</san>
```

```

<from>1</from>
  <to>0</to>
  <event>e6</event>
  <rate>t2<\rate>
</transition>
<transition>
  <from>1</from>
  <to>2</to>
  <event>e2</event>
  <rate>t3<\rate>
</transition>
<transition>
  <from>2</from>
  <to>1</to>
  <event>e3</event>
  <rate>t4<\rate>
</transition>
<transition>
  <from>2</from>
  <to>0</to>
  <event>e3</event>
  <rate>t5<\rate>
</transition>
</automaton>
<automaton id="1">
  <!--The list of states.-->
  <state id="0">
    <x>169.0</x>
    <y>110.0</y>
    <name>0</name>
  </state>
  <state id="1">
    <x>298.0</x>
    <y>116.0</y>
    <name>1</name>
  </state>
  <!--The list of transitions.-->
  <transition>
    <from>0</from>

```

```
<to>1</to>
<event>e3</event>
<rate>t1<\rate>
</transition>
<transition>
  <from>1</from>
  <to>0</to>
  <event>e4</event>
  <rate>t2<\rate>
</transition>
</automaton>
</san>
```

O uso da XML garantirá uma melhor integração com outras ferramentas, por ser uma linguagem padronizada de transação de dados entre aplicações, que por ventura venham a utilizar modelos SAN para medidas de desempenho. A ferramenta VisualSAN deverá ter em sua implementação a aplicação do XML como meio de realizar transações entre pacotes de *software*. Estas transações poderão ser de simples envio de dados até mesmo a estrutura completa de SAN.

6. FERRAMENTA PARA SIMULAÇÃO VISUAL DE SAN

Este capítulo apresenta a estruturação da ferramenta para apoio a modelagem de sistemas por meio do formalismo estruturado SAN. Esta estrutura foi definida para aplicação do algoritmo proposto no Capítulo 5 que é um dos objetivos deste trabalho além da integração com algoritmos para avaliação de modelos SAN por meios de métodos analíticos, *GTExpress*, e o algoritmo baseado em MDD para obtenção de estados atingíveis com redução de espaço de estados, *RSS-Finder*. O termo *RSS-Finder* é apenas uma referência à técnica apresentada no Capítulo 4.

A Figura 6.1 apresenta a estrutura e identifica as relações dos módulos que fazem parte da ferramenta. A figura apresenta, ainda, que a estrutura é composta por módulos de edição dos autômatos SAN, edição de funções, módulo de integração com software que fornecem os algoritmos baseados em soluções numéricas, *GTExpress* [CZE09] e *RSS-Finder* [SAL09a], e o editor auxiliar. O editor auxiliar servirá para complementar o modelo SAN para ser utilizado pelas ferramentas *GTExpress* e *RSS-Finder*.

Ainda é possível observar na Figura 6.1 que o usuário manipulará os módulos que interagem diretamente com o ambiente de simulação SAN. No módulo **editor de SAN** o usuário pode interagir para criar redes de autômatos estocásticos. No módulo editor de funções, o usuário informará as taxas constantes ou funcionais e no editor auxiliar o usuário poderá inserir informações que complementam o arquivo ".san", este utilizado pelas ferramentas *GTExpress* e *RSS-Finder*.

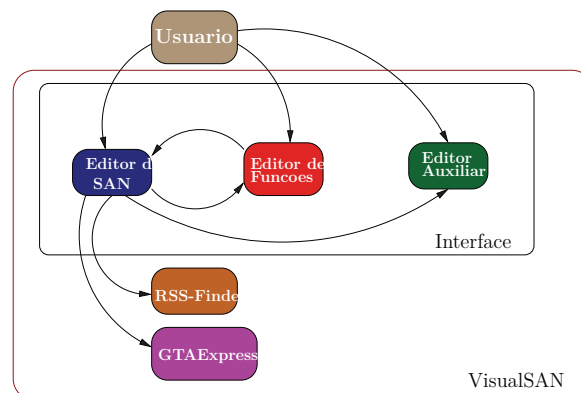


Figura 6.1: Estrutura da ferramenta

Para o desenvolvimento da ferramenta **VisualSAN** foram necessárias adaptações bastante abrangentes na ferramenta *JFlap* tais como: permitir a modelagem de vários autômatos, de acordo com a concepção do formalismo SAN, por meio da inserção de vetores, inclusão de algoritmos de análise léxica, sintática e semântica baseados na ferramenta **JavaCC** [SUN10], inclusão de interatividade utilizando controles para simular as tabelas de eventos e suas taxas e a disponibilização dos algoritmos definidos no Capítulo 5.

6.1 Objetivos da Ferramenta

Projetar e implementar sistemas complexos implica em resolver problemas como a interação de processos com comportamentos inesperado ou que não interajam entre si a todo instante. Desenvolver sistemas com estas características pode ser tão propenso a erros e custosos que se torna interessante fazer uso de ferramentas que permitam simular o comportamento de sistemas reais. Pensando nisto, este trabalho tem como objetivo disponibilizar uma ferramenta que dê suporte a modelagem visando a avaliação do desempenho e a verificação do comportamento de sistemas. O uso de ferramenta para modelagem auxilia no desenvolvimento de projetos, pois habilitam projetos específicos a serem testados e aperfeiçoados de modo a proporcionar o máximo de desempenho e ao mesmo tempo proporcionar a eliminação de problemas inerentes ao projeto, tais como taxas funcionais mal formadas, modelos SAN mal formados, ou seja que não atendam as definições estabelecidas no formalismo SAN, e possíveis estados absorventes.

Adicionado ao benefícios da interatividade que a simulação visual propõe, esta ferramenta terá um propósito educacional permitindo que acadêmicos possam aplicar os conceitos do formalismo SAN em pesquisas que necessitem de auxílio na avaliação dinâmica de modelos SAN, bem como analisar os comportamentos do sistema em estudo.

6.2 Visão Geral

O objetivo deste projeto é produzir um ambiente que contenha:

- uma interface gráfica para executar modelos SAN utilizando a linguagem JAVA;
- um simulador que permita a representação de grafos tais como os fornecidos pelo formalismo estruturado SAN e que execute alguma funcionalidade e características básicas de avaliação de desempenho;
- interação com a ferramenta *PEPS2007* por meio do *software GTAExpress*.
- interação com o algoritmo para obtenção de estados atingíveis *RSS-Finder*.

Esta ferramenta é um *software* experimental com funcionalidades para manipular modelos baseados no formalismo SAN. Este pacote gráfico poderá ser utilizado principalmente como uma forma de auxiliar na aprendizagem dos conceitos básicos de Redes de Autômatos Estocásticos.

6.3 Arquitetura

A arquitetura deste *software* é baseada no pacote *JFLAP* e *JavaCC*. o *JFLAP* é um *software* para pesquisas com linguagens formais com tópicos que incluem autômatos finitos não-determinísticos, autômatos de pilha não-determinísticos, máquinas de Turing (*multi-tape*), vários tipos de gramáticas, *parsers*. Além de construir e testar para estas finalidades citadas, o *JFLAP* permite experimentos

que podem utilizar conversões entre os tópicos citados, como por exemplo converter um modelo de autômatos finitos não-determinísticos para autômatos finitos determinísticos e deste para expressões regulares ou gramática regular [ROD06].

O *JavaCC* auxilia na criação de compiladores para linguagens simples, gerando um analisador sintático. Este programa aceita como entrada uma gramática e transforma-a num programa *Java* capaz de analisar um arquivo e dizer se satisfaz ou não as regras especificadas em sua gramática [DEL10]. O *JavaCC* é um produto de propriedade da *Sun Microsystems* e liberado sob a licença *Berkeley Software Distribution* (BSD).

As características inerentes à modelagem de sistemas baseada em SAN foram incorporadas à ferramenta de maneira que as regras básicas definidas pelo formalismo como manutenção de N autômatos (incluindo estados e transições), inserções de taxa funcionais, controle de execução das transições do tipo *para frente* e *para trás* e apresentação de resultados baseados nos métodos analíticos disponíveis na ferramenta PEPS2007.

As principais funcionalidades do pacote de *software* VisualSAN são: (i) visualizar representações gráficas das propriedades estruturais; (ii) simulação, com interação gráfica, de sincronização; (iii) soluções numéricas baseadas em tensores através da integração com o *GTAExpress*; (iv) utilizar métodos mais avançados para soluções estacionárias e transitórias baseados em algoritmos esparsos [STE94] e *Split* fornecidos pelo *GTAExpress*; (v) módulos de simulação para interação de eventos que em cooperação com a interface gráfica fornece simulação com interatividade e com animação gráfica do modelo, com passo-a-passo, e progressão de tempo para frente e para trás; (vi) apresentação de estado atingíveis por meio de MDD (será utilizado algoritmo (*RSS-Finder*) disponível em [SAL09b]).

Processos estocásticos de um sistema podem ser representados por meio de derivações gráficas do formalismo SAN. Um sistema é modelado em SAN através de componentes que representam a sua estrutura interna. A interação destes componentes, que pela própria definição da SAN são multi-dimensionais, determina o comportamento do sistema como um todo ou define o comportamento de apenas um objeto da coleção. Segundo [TAS10], cada comportamento dos componentes é definido pela coleção de variáveis necessárias para descrever um sistema, estas variáveis correspondem ao estado do mesmo em certo instante de tempo. A característica multi-dimensional possibilita que o sistema seja analisado granularmente, ou seja, o comportamento do sistema poderá ser avaliado individualmente, porém o comportamento geral também pode ser avaliado uma vez que ele é afetado pelo comportamento individual dos sub-sistemas.

6.3.1 Interface do Usuário

A interface com o usuário é composta pelos editores de SAN e de funções que serão utilizados para a modelagem de sistemas e criação de taxas funcionais respectivamente. A simplicidade da interface da ferramenta torna a interação com o usuário direta, ou seja, o usuário influenciará na estruturação de modelos SAN.

Editor de SAN

O editor de SAN permite o usuário criar modelos baseados no formalismo estruturado SAN. O editor tem em sua composição objetos que representam os autômatos e sua estrutura interna: estados, eventos e transições. Os autômatos que representam sistemas ou subsistemas, os estados representando processos ou subsistemas no autômato e as transições, atreladas com os eventos, que representam a mobilidade de um estado para outro no modelo.

Este editor, cuja interface é bastante simples, apresentado na Figura 6.2 possibilita a nomeação dos autômatos, estados e os eventos. Esta nomeação associa os objetos da rede com as características de sistemas reais. No editor está incluso funcionalidades para persistir o modelo em formato ".san", deletar objetos da modelagem, edição, alteração de nomes (rótulos) e reposicionamento de objetos no editor.

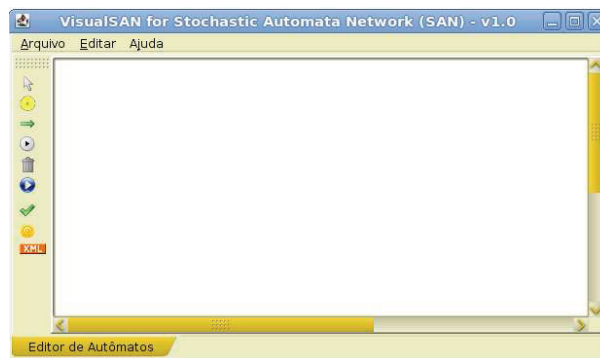


Figura 6.2: Editor da ferramenta VisualSAN

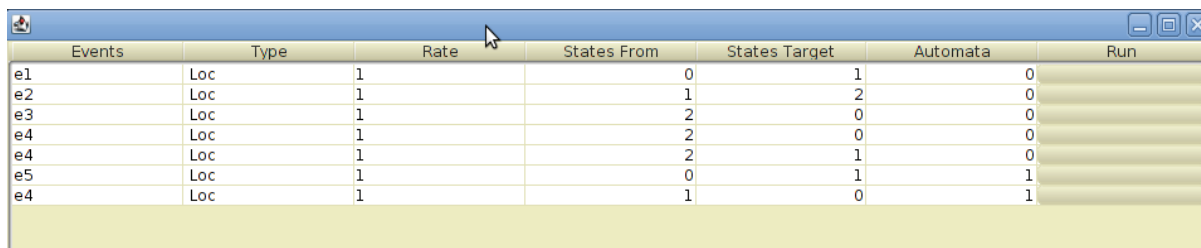
No processo de criação dos autômatos, no editor, há restrições que garantem um modelo bem formado de SAN. Isto é, a SAN não será simulada se um dos estados não tem transição associada se conectando a ele ou partindo dele para outro estado.

A execução do procedimento de simulação visual desta ferramenta será baseada no algoritmo proposto no Capítulo 5. Este algoritmo é composto por várias rotinas, dentre elas estão as rotinas de criação das matrizes de transição (que utiliza o modelo projetado no editor), de verificação de eventos locais e sincronizantes que influenciam diretamente na simulação, de simulação (redesenho) dos objetos no editor para indicar mudanças de estados e de verificação das taxas funcionais. Na seção seguinte, será abordado o funcionamento do sistema por meio das taxas funcionais.

Editor de Taxas Funcionais

O editor de taxas funcionais, Figura 6.3 tem como objetivo receber do usuário informações sobre a composição das taxas funcionais da modelagem. A partir desta informação o sistema processará a taxa realizando um desmembramento, se necessário, para calcular as expressões contidas, como neste exemplo: $f_1 = [(st\mathcal{A}^{(2)} == 0^{(2)}) * \alpha_{11}] + [(st\mathcal{A}^{(2)} == 2^{(2)}) * \alpha_{12}]$. Este cálculo se baseia nas definições pré-estabelecidas pelo *software* PEPS2007, e para que a transição dependente da taxa

funcional seja disparada o resultado da expressão deve ser 1; o valor 0 impede o disparo da transição dependente da taxa funcional associada. As taxas funcionais informadas serão avaliadas para garantir que as mesmas sejam bem formadas. A seguir o exemplo dado será descrito detalhadamente, bem como a sua execução.



Events	Type	Rate	States From	States Target	Automata	Run
e1	Loc	1	0	1	0	
e2	Loc	1	1	2	0	
e3	Loc	1	2	0	0	
e4	Loc	1	2	0	0	
e4	Loc	1	2	1	0	
e5	Loc	1	0	1	1	
e4	Loc	1	1	0	1	

Figura 6.3: Editor de taxas funcionais da ferramenta VisualSAN

A avaliação da taxa funcional, como dito anteriormente, deverá ser realizada com o auxílio da ferramenta *JavaCC*. Este pacote de *software*, desenvolvido para estudo de compiladores, tem em seus mecanismos internos analisadores que avaliam uma expressão quanto a sua sintaxe e quanto a sua estrutura léxica. Para o exemplo citado, será avaliado a primeira expressão $[(st.A^{(2)} == 0^{(2)}) * \alpha_{11}]$ que, segundo a definição da ferramenta PEPS2007, verificará se o autômato $\mathcal{A}^{(2)}$ se encontra no estado $0^{(2)}$, denotado pela instrução *st*. Caso seja verdadeiro, então será retornado 1, do contrário será retornado o valor 0 como resultado da expressão. Esta avaliação ainda realiza um cálculo aritmético da expressão onde a mesma utiliza o resultado da avaliação lógica para então multiplicá-la pela taxa constante α_{11} , bem como pela taxa α_{12} , e finalmente somando os resultados da multiplicação para assim, se o valor for diferente de 0, disparar o evento vinculado a esta taxa funcional.

6.4 Análise e implementação da Ferramenta

Nesta seção é descrito o processo de desenvolvimento do simulador *VisualSAN*. Foi escolhida a linguagem *Javatm*, por ser Orientada a Objetos, facilitando a representação de objetos que visam se comportar tal qual a realidade. Ademais, a orientação a objetos é muito utilizada pela notória capacidade de modelar objetos com propriedades de herança e encapsulamento.

A ferramenta *VisualSAN*, apresentada na Figura 6.4 foi desenvolvida para experimentos, especificamente, com o formalismo SAN. Este pacote fornece um ambiente desenvolvido utilizando a API 2D da linguagem JAVA. Este ambiente disponibiliza um editor de autômatos que contém as seguintes características funcionais:

- Componente que adiciona autômatos ao modelo no editor;
- Função que permite inserir elementos que representam estados possibilitando a definição de rótulos e o posicionamento livre no editor;

- Função para inserção de transições entre os estados, é livre a inclusão de n transições entre os estados;
- Gerar da tabela de eventos a matriz de transição para cada autômato manipulado no editor;
- Comando para determinar a execução da simulação do modelo.

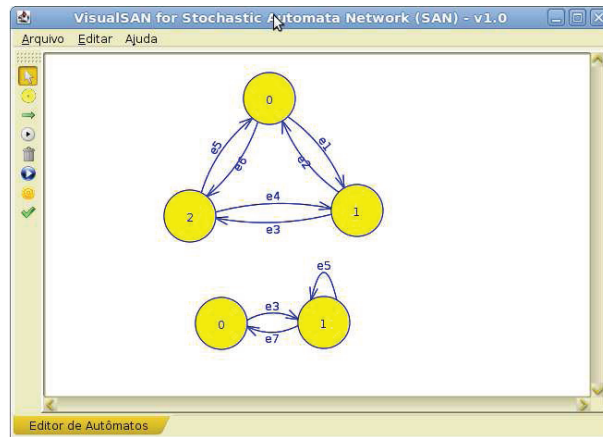


Figura 6.4: A ferramenta VisualSAN

Para o correto funcionamento da simulação proposta pela ferramenta é necessário que, a partir da criação dos autômato no editor, sejam criadas as matrizes de transição, trabalhadas no Capítulo 5. Estas matrizes terão uma representação muito importante, pois será através delas que haverá a manipulação dos autômatos cujos disparos dos eventos habilitarão as interações entre os componentes (estados, eventos e transições com ou sem taxa funcional).

Nas Seções 3.1, 5.1.2 e 5.1.3 foram apresentadas as regras básicas de funcionamento de modelos de sistemas baseados em SAN e a estruturação do algoritmo. Este *software* teve o conceito estrutural e funcional baseado nestas regras estabelecidas.

Para a avaliação das taxas funcionais, relacionadas aos eventos participantes dos autômatos do modelo, foi necessário o desenvolvimento de um compilador simples que fosse capaz de avaliar expressões algébricas, expressões lógicas e de realizar avaliação sintática para averiguar se a taxa funcional é válida, para tal foi utilizado o *software JavaCC* que possibilitou ao editor as capacidades de avaliações lógicas, matemáticas e sintáticas.

6.5 Modelagem de SAN - Exemplificando Modelos na Ferramenta

O uso da ferramenta *VisualSAN* é bastante simples, recursos para inserção de autômatos, estados e transições estão disponíveis através de componentes gráficos que são sugestivos quanto à sua funcionalidade. Nas próximas subseções serão propostos exemplos com e sem taxa funcional, assim serão explorados as funcionalidades da ferramenta. É importante destacar que os exemplos são meramente hipotéticos.

6.5.1 Exemplo sem taxa funcional

A Figura 6.5 apresenta uma modelagem de SAN com dois autômatos com 3 e 2 estados respectivamente. As Figuras 6.6, 6.7 e 6.8 apresentam a execução dos autômatos. Como pode-se observar é apresentado nas figuras os estados atuais e os estados sucessores, que formam respectivamente, os estados globais atuais (combinação dos estados atuais) e sucessores.

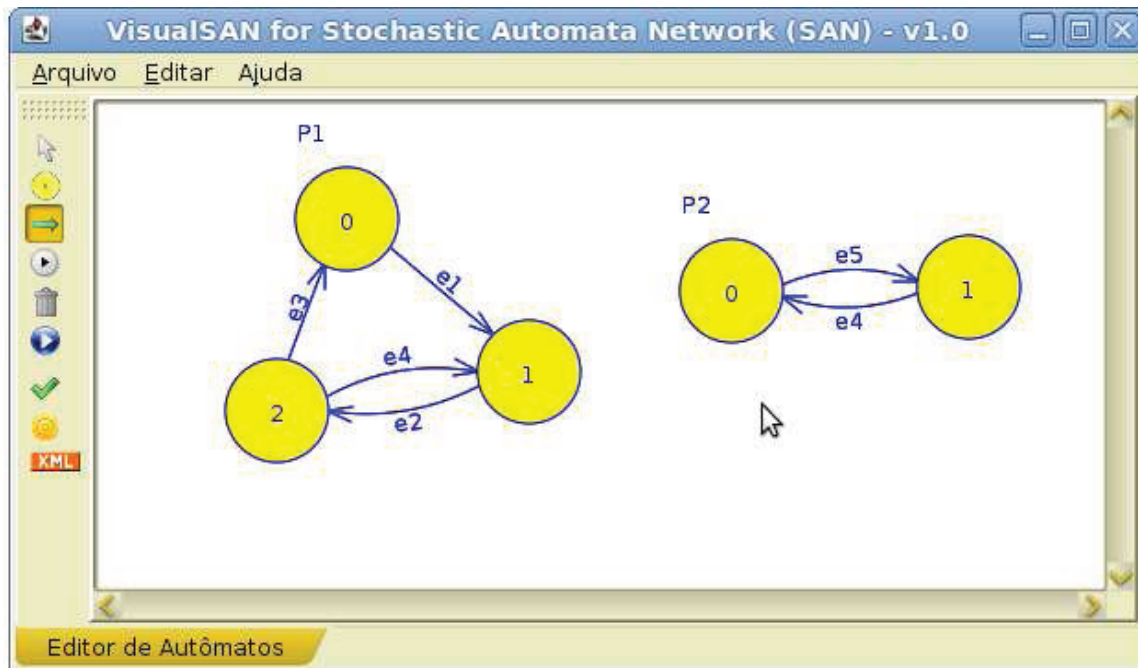


Figura 6.5: Ferramenta VisualSAN com modelo SAN

Por uma questão de melhor visualização as Figuras 6.6, 6.7 e 6.8 apresentam apenas os autômatos sendo executados. Na Figura 6.6 podemos notar que ao ser executada, por ser a primeira execução, o sistema apresenta os estados iniciais como sendo os estados $0^{(1)}$ (do primeiro autômato) e $0^{(2)}$ (do segundo autômato) identificados com um círculo duplo e os estados sucessores identificados com um círculo pontilhado.

A Figura 6.7 apresenta uma mudança de estados globais de $0^{(1)}, 0^{(2)}$ para os estados globais $1^{(1)}, 0^{(2)}$, através da mudança do estado local de $0^{(1)}$ para o estado $1^{(1)}$, e os respectivos estados sucessores $2^{(1)}$ e $1^{(2)}$. Sempre que há mudança de estados haverá uma alteração na formação dos estados globais.

Para o evento sincronizante e_4 dos autômatos apresentados na Figura 6.8, cujas transições interligam os estados $1^{(1)}$ e $2^{(1)}$ e os estados $1^{(2)}$ e $0^{(2)}$, quando executado levam aos estados sucessores $1^{(1)}$ e $0^{(2)}$, o próximo estado global, indicados pelo estado com círculo pontilhado.

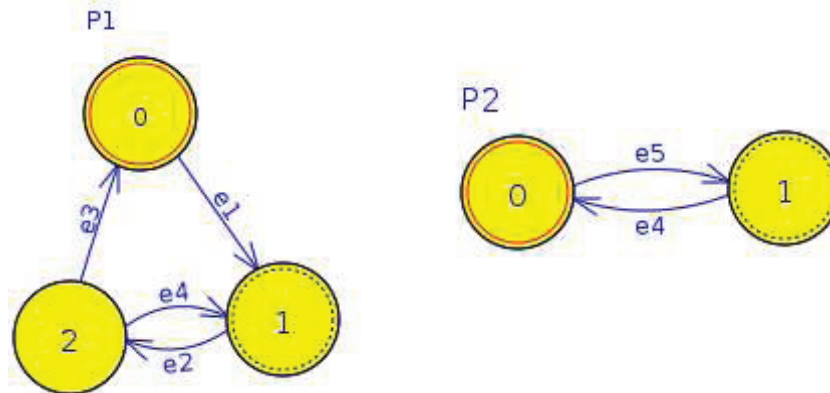


Figura 6.6: SAN com estados iniciais (círculo duplo) e estados sucessores (círculo pontilhado)

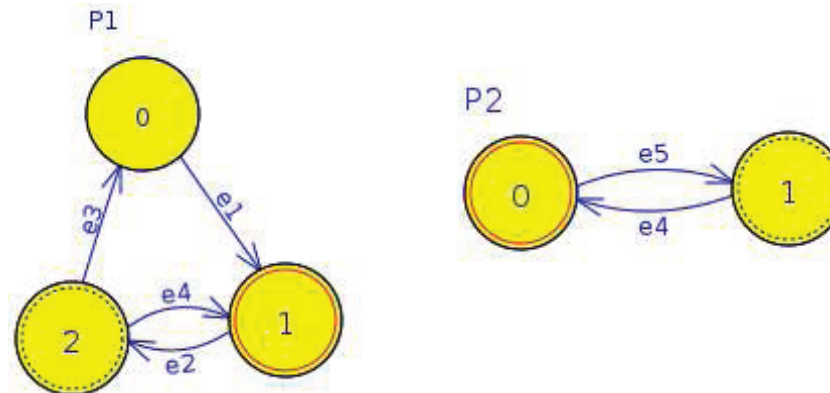


Figura 6.7: Modelo SAN como novos estados atuais e estados sucessores

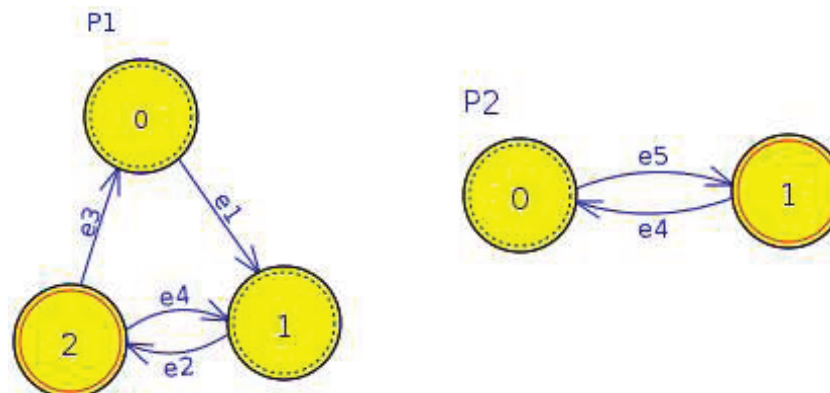


Figura 6.8: exemplo de simulação de eventos sincronizantes

6.5.2 Exemplo com taxa funcional

O uso de taxas funcionais em modelagem SAN define uma interdependência entre os autômatos. Na ferramenta proposta, as taxas funcionais são incluídas nos eventos locais e/ou sincronizantes.

A Figura 6.9 foi modelada para exemplificação de eventos sincronizantes e evento com taxa funcional e_2 , demonstrada na Figura . Como explicitado na sub-seção 6.3.1 as taxas funcionais foram implementadas na ferramenta *VisualSAN* com o uso do pacote de *software JavaCC*.

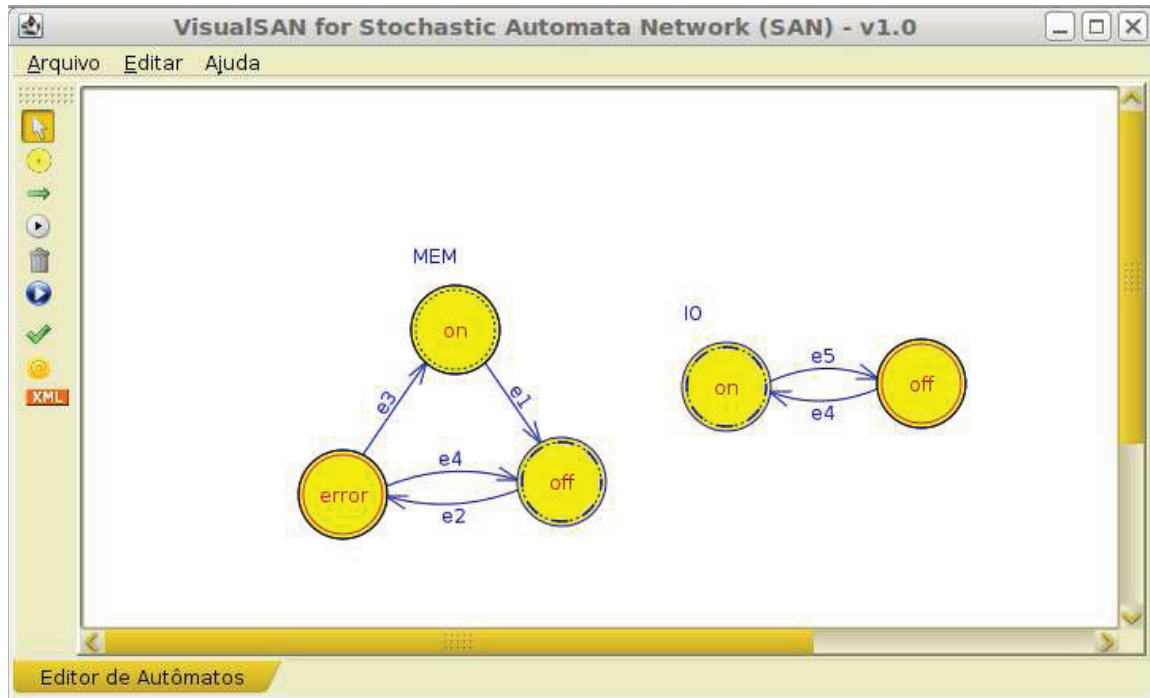


Figura 6.9: Exemplo de simulação de eventos sincronizantes e taxa funcional

Events	Type	Rate	States From	States Target	Automata	Run
e1	loc	1	0	1	0	
e2	loc	(st IO==off)	1	2	0	
e3	loc	1	2	0	0	
e4	syn	1	2	1	0	
e5	loc	1	0	1	1	
e4	syn	1	1	0	1	

Figura 6.10: Editor de taxas funcionais com exemplo referente a Figura 6.9

Conforme a Figura 6.10 foi editado na tabela de eventos o campo *Rate* referente às taxas dos eventos. Neste campo foi incluso a taxa funcional (st IO==off) para o evento e_2 . Isto implica em dizer que o autômato identificado como "IO" deverá estar com o estado "off" selecionado como o estado atual (em destaque com círculo duplo) para que a transição associada ao evento e_2 possa ser disparada. Caso o estado "off" do autômato "IO" seja o atual a taxa funcional será avaliada e tendo como resultado o valor 1, indicando que a condição imposta pela taxa foi satisfeita o que poderá causar o disparo do evento e_2 mudando o estado global atual (*error, off*) para o estado global sucessor (*off, on*), conforme a Figura 6.11. Na Figura 6.9 pode-se verificar que o sistema apresenta

também os estados predecessores do estado atual, estados circulado com linha tracejado, "off" do autômato MEM e o estado "on" do autômato IO. Os estados sucessores dos estados atuais "off" e "on" não estão sendo identificados na Figura 6.11 em consequência de: (i) de acordo com a taxa definida para e_2 o autômato IO deve estar com o estado "off" como sendo o estado atual, (ii) o fato do autômato IO ter apenas dois estados impede do sistema identificar o estado "off" como sendo o estado sucessor, o processo do algoritmo determina a escolha para que este estado seja identificado com sendo o estado predecessor.

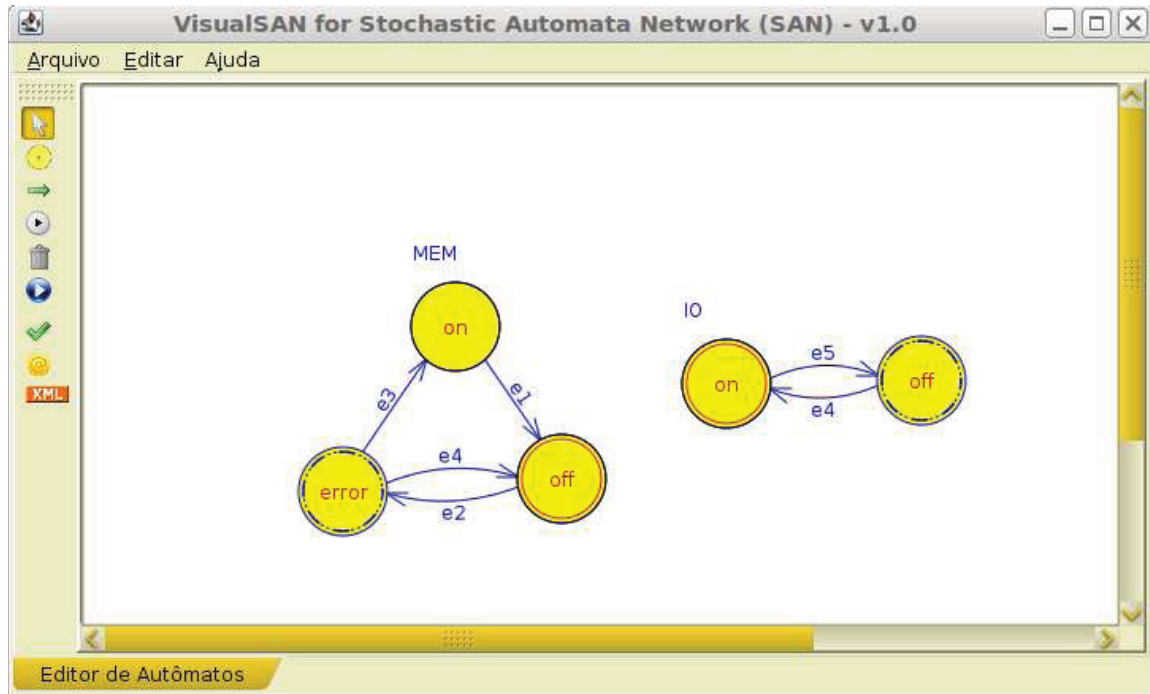


Figura 6.11: Exemplo de modelo SAN de eventos com taxa funcional

6.6 Exemplo de Obtenção dos Estados Atingíveis por meio de MDD

O exemplo citado nesta seção tem como objetivo apresentar o uso da ferramenta para modelagem de SAN e consequente obtenção dos estados atingíveis. A Figura 6.12 ilustra uma modelagem referente ao exemplo citado no Capítulo 2 seção 3.5.6 cujo sistema representa um FAS. Esta modelagem faz uso de taxas funcionais definidas através do editor de taxas funcionais representado na Figura 6.13.

As taxas inseridas, visualizadas na Figura 6.13, neste modelo para o evento $g1$ é 5, para o evento $r1$ é 6, para o evento $g2$ é $(stC1 == ocupado) * 5$, para o evento $r2$ é 5, para o evento $g3$ é $(nb[C1..C2]ocupado == 2) * 5$, para o evento $r3$ é 4, para o evento $g4$ é $(nb[C1..C3]ocupado == 3) * 5$ e para o evento $r4$ é 3, sendo que todos os eventos são locais.

Para se obter os estados atingíveis do exemplo citado basta solicitar para a ferramenta através do componente (botão) responsável por disparar o procedimento de criação do arquivo ".san". Este

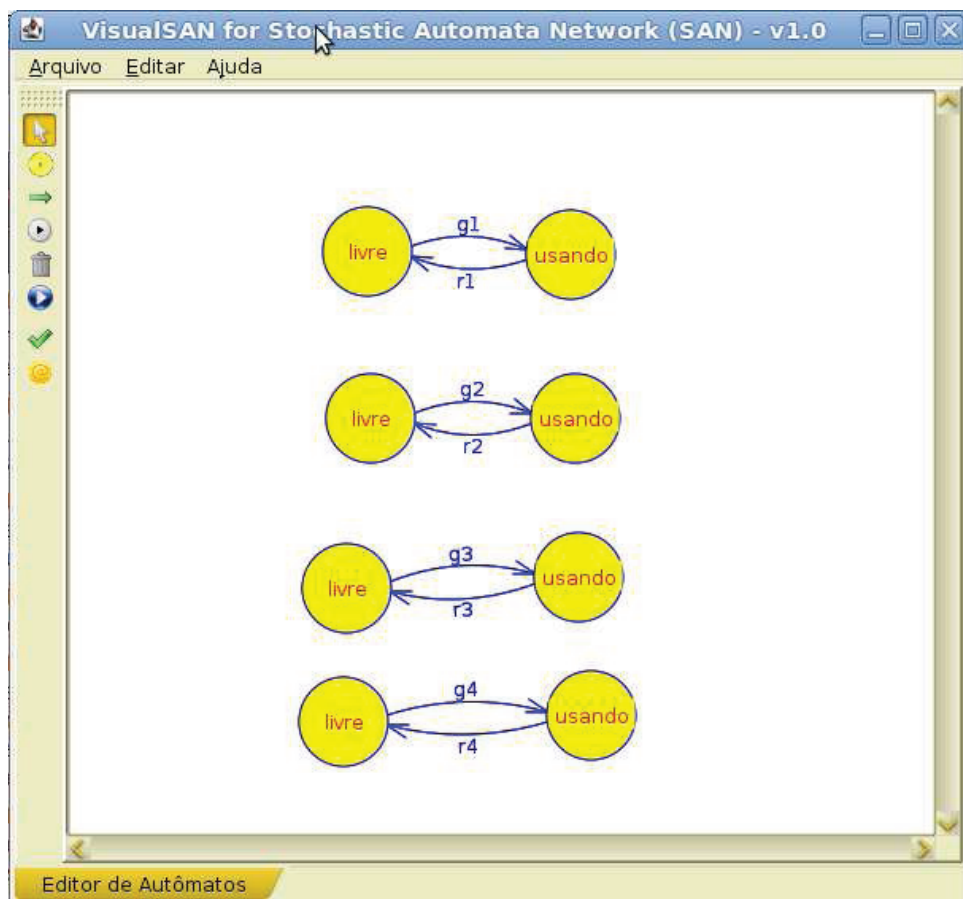


Figura 6.12: Exemplo de modelo SAN de eventos com taxa funcional simulando FAS

Events	Type	Rate	States From	States Target	Automata	Run
g1	Loc	5	0	1	0	
r1	Loc	6	1	0	0	
g2	Loc	(st C1 == ocupado) * 5	0	1	1	
r2	Loc	5	1	0	1	
g3	Loc	(nb [C1..C2] ocupado==2) * 5	0	1	2	
r3	Loc	4	1	0	2	
g4	Loc	(nb [C1..C3] ocupado==3) * 5	0	1	3	
r4	Loc	3	1	0	3	

Figura 6.13: Taxas funcionais referente ao modelo da Figura 6.12

arquivo será criado segundo as especificações da ferramenta PEPS2007 citadas devidamente na seção 3.5 do Capítulo 2. Ao ser solicitado a criação do arquivo ".san" é apresentada a tela, Figura 6.14 com as informações necessárias advindas no ambiente de edição de autômatos, bem como do editor de taxas funcionais. Este arquivo será então processado pelo algoritmo *RSS-Finder* e caso a estrutura do arquivo esteja correta é apresentado, os estados atingíveis para o modelo ora representado. Os estados atingíveis da SAN que representa o modelo FAS são demonstrados na Tabela 6.1, também ilustrado na Figura 6.14.

EG	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$	$\mathcal{A}^{(3)}$	$\mathcal{A}^{(4)}$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Tabela 6.1: Lista de estados globais atingíveis do modelo da Figura 6.12

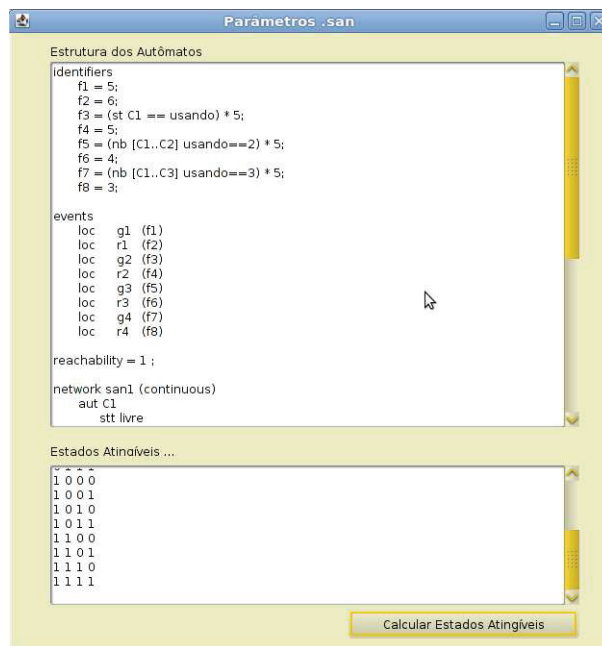


Figura 6.14: Tela do editor do arquivo ".san" referente ao modelo da Figura 6.12

6.7 Ambiente de Desenvolvimento

O pacote de *software* VisualSAN utilizou a linguagem JAVA em sua versão 1.6.0_20 em ambiente Linux/Ubuntu versão 10.10 codinome *the Maverick Meerkat*. O computador utilizado para produzi-lo tinha a seguinte configuração: Quad core Intel-i7 64 bits - Centrino, 6 GB de RAM. Mesmo sendo desenvolvido utilizando ambiente Linux este *software* por ser desenvolvido com a linguagem JAVA poderá ser utilizado em qualquer sistema operacional que permita a instalação do JAVA e que execute a versão do Java acima citada.

6.8 Considerações

Esta ferramenta foi desenvolvida com o propósito de comprovar o funcionamento dos algoritmos propostos no Capítulo 5 deste trabalho e ainda fornecer uma ferramenta capaz de simular modelos SAN calculando estados sucessores e predecessores. Os estados sucessores e predecessores, num primeiro momento, apenas são identificados na SAN modelada através de círculos pontilhados e tracejados, respectivamente. Devido à complexidade em aliar os algoritmos a representação gráfica em 2D, para que a mesma fosse capaz de simular visualmente os elementos de uma SAN, foi utilizado a linguagem Java para o desenvolvimento do ambiente de edição dos autômatos. Nesta linguagem foram desenvolvidas todas as funcionalidades baseadas nos algoritmos para cálculo dos estados sucessores e predecessores, além dos próprios algoritmos para controlar o processamento das imagens que representam os autômatos. Ainda foram necessários o uso de ferramentas de terceiros para o desenho dos autômatos, cálculo das taxas funcionais e para o cálculo dos estados atingíveis. Estas ferramentas e os componentes do editor de autômatos serão comentadas nos apêndices deste trabalho.

7. CONCLUSÃO

Este trabalho apresentou, no Capítulo 5, as definições dos algoritmos para descobertas de estados sucessores e predecessores, a definição da estrutura de dados que representará uma SAN no formato XML e a ferramenta *VisualSAN*. Cabe ressaltar, que as integrações entre a ferramenta proposta neste trabalho com as ferramentas GTAexpress e o PEPS ainda não foram concluídas pois há a necessidade de fazer novas adaptações, tais como: aprimorar o aplicativo para estruturação arquivo ".san" para que o *software* PEPS possa utilizá-lo e a integração com o *software* para a obtenção dos estados atingíveis dos antecessores e predecessores. Em trabalhos futuros será feito um aperfeiçoamento da ferramenta para adequação das necessidades citadas anteriormente, bem como inclusão de alguma funcionalidade ainda não identificada.

Também foi apresentado neste trabalho o desenvolvimento de algoritmos para a obtenção de estados sucessores e predecessores, nota-se que para a obtenção destes estados foi necessário abordar procedimentos que permitissem o disparo de transições, verificação de estados locais e estados globais. Em cada estado global alguns eventos são habilitados, mudando o estado global. Entretanto, nem todos os eventos podem ocorrer a partir de um dado estado global. Podemos notar que no algoritmo implementado é disponibilizado funções que não necessariamente são utilizadas em soluções para implementações de modelos SAN atuais.

Os estados sucessores e predecessores são fundamentais para a simulação gráfica de modelos SAN, pois permitem validar se um modelo é bem formado através da ocorrência dos eventos dos autômatos pertencentes à Rede de Autômato Estocásticos. Nota-se, que com a simulação da execução do algoritmo é possível prever os estados de todos os autômatos envolvidos, o que viabiliza a resolução de modelos, segundo os critérios do formalismo SAN. Este algoritmo foi base para o desenvolvimento da aplicação gráfica, descrita no Capítulo 7, que será utilizada para simulação e permitirá observar o comportamento de sistemas reais.

Algumas dificuldades foram encontradas ao longo do desenvolvimento deste trabalho. A criação do ambiente para se desenhar os autômatos foi superado com o uso da ferramenta JFLAP, embora houve outras dificuldades na adaptação para se manipular uma rede de autômatos. O algoritmo para avaliação das taxas funcionais foi outro problema bastante difícil a ser resolvido pois o formato das expressões exigidas pelo formalismo possui características bastante peculiares. Para facilitar o desenvolvimento de analisadores léxicos foi utilizado o pacote de software JAVACC, as adaptações para o uso desta ferramenta também foi bastante trabalhosa devido às características específicas do formalismo SAN.

7.1 Contribuição

Este trabalho contribui para a comunidade acadêmica no sentido de fornecer uma ferramenta que implemente uma simulação visual com o uso de algoritmos para o cálculo de estados sucessores

e predecessores. A utilização desta ferramenta pelos acadêmicos proporcionará para os leigos uma melhor aprendizagem do formalismo SAN e para os usuários mais avançados uma possibilidade de simulação de sistemas com recursos da SAN, tais como: uso de taxas funcionais, eventos sincronizados e simulação para trás (*backward*) e para frente (*forward*) (para visão geral do comportamento do modelo).

7.2 Trabalhos Futuros

Dentro da especificidade deste trabalho, o desenvolvimento de algoritmo para obtenção de estados sucessores/predecessores e a simulação visual de modelos SAN, foi possível identificar alguns trabalhos futuros. A seguir são descritos, sumariamente, estes trabalhos:

- *Software* para conversão de modelos SAN para modelos baseados em Redes de Petri, que permite modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos. Este *software* deverá permitir a simulação visual de modelos desenvolvidos em SAN e convertidos para redes de Petri;
- Integrar na ferramenta o conceito de simulação por meio de modelos matemáticos de maneira que seja possível exercitar o modelo numericamente, inserindo entradas e avaliando o comportamento do sistema modelado [TAS10];
- Implementar o conceito *Model-Checking* para verificação de modelos onde se possa verificar de maneira automática a validade e propriedades acerca do comportamento de sistemas reativos [ROM03];
- Desenvolver pacote de *software* que realize a conversão de modelos SAN simulados na ferramenta VisualSAN para modelos representados em MDD.

REFERÊNCIAS BIBLIOGRÁFICAS

- [AJM84] M. Ajmone-Marsan, G. Conte e G. Balbo. “A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems”. *ACM Transactions on Computer Systems*, vol. 2-2, Maio 1984, pp. 93–122.
- [AND99] H. R. Andersen. “An Introduction to Binary Decision Diagrams”. *Lecture Notes*, IT University of Copenhagen, 1999.
- [AND00] H. R. Andersen, T. Hadzic, J. N. Hooker e P. Tiedemann. “A Constraint Store Based on Multivalued Decision Diagrams”. *Lecture Notes in Computer Science*, vol. 4741, 2007, pp. 118–132.
- [BAA01] J. M. Ilie, S. Baarir, M. Beccuti, C. Delamare, S. Donatelli, C. Dutheillet, G. Franceschinis, R. Gaeta e P. Moreaux. “Extended SWN Solvers in GreatSPN”. In: 1st International Conference on Quantitative Evaluation of Systems (QEST04), setembro 2008, pp. 324–325.
- [BAN05] J. Banks, J. S. Nelson E D. M. Nicol. “Discrete-Event System Simulation”. Prentice Hall, vol. 4, 2005, 622p.
- [BAL94] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli e G. Franceschinis. “Modelling with Generalized Stochastic Petri Nets”. *Wiley Series in Parallel Computing*, John Wiley and Sons, 1994.
- [BEN03a] A. Benoit, L. Brenner, P. Fernandes, B. Plateau e W. J. Stewart. “The PEPS Software Tool”. *Lecture Notes in Computer Science*, vol. 2794, Setembro 2003, pp. 98–115.
- [BEN03b] A. Benoit. “Méthodes et algorithmes pour l'évaluation des performances des systèmes informatiques à grand espace d'états”, Tese de Doutorado, Institut National Polytechnique de Grenoble, France, 2003, 170p.
- [BER00] A. Bergholz. “Extending Your Markup: An XML Tutorial”. In: *IEEE Internet Computing*, vol. 4, 2000, pp. 74–79.
- [BOL98] G. Bolch, S. Greiner, H. Meer e K. S. Trivedi. “Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications”. John Wiley and Sons, 1998.
- [BRA03] J. T. Bradley, N. J. Dingle, S. Gilmore e W. J. Knottenbelt. “Derivation of passage-time densities in PEPA models using ipc: the imperial PEPA compiler”. In: 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS), 2003, pp. 344–351.

- [BRE02] L. Brenner, P. Fernandes e C. G. P. Alegretti. “Redes de Autômatos Estocásticos: um formalismo para avaliação de desempenho e confiabilidade de sistemas”. PUCRS-PPGCC, nb. 25, 2002, 21p.
- [BRE04a] L. Brenner. “Agregação em Redes de Autômatos Estocásticos”, dissertação de mestrado, PUCRS-FACIN-PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, 2004, 109p.
- [BRE04b] L. Brenner, P. Fernandes e A. Sales. “Avaliação de Desempenho de Sistemas Paralelos”. In: 4a. Escola Regional de Alto Desempenho, vol. 4, 2004, pp. 97-120.
- [BRE07] L. Brenner, P. Fernandes, B. Plateau e I. Sbeity. “PEPS 2007 - Stochastic Automata Networks Software Tool”. In: Proceedings of the Fourth International Conference on Quantitative Evaluation of Systems (QEST '07), setembro 2007, pp. 163–164.
- [BRY97] M. Bryan. “An Introduction to the Extensible Markup Language (XML)”. World Wide Web Consortium, dezembro 1997.
- [BUC98] P. Buchholz e P. Kemper. “On generating a hierarchy for GSPN analysis”. *ACM Performance Evaluation Review*, vol. 26 (2), 1998, pp. 5–14.
- [CIA01a] G. Ciardo. “Distributed and Structured Analysis Approaches to Study Large and Complex Systems”. Lectures on Formal Methods and Performance Analysis vol. 2090, 2001, pp. 344–374.
- [CLA07] A. Clark, S. Gilmore, J. Hillston e M. Tribastone. “Stochastic Process Algebras”. *Lecture Notes in Computer Science*, vol. 4486, 2007, pp. 132–179.
- [CZE09] R. M. Czekster, P. Fernandes e T. Webber. “GTAexpress: a Software Package to Handle Kronecker Descriptors”. *Proceedings of the 6th International Conference on Quantitative Evaluation of Systems (QEST '09)*, September 2009, pp. 281–282.
- [CZE10] R. M. Czekster. “Solução numérica de descritores Markovianos a partir de re-estruturações de termos tensoriais”, Tese de Doutorado, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, 2010, 195p.
- [DEL10] M. E. Delamaro. “Como construir um compilador utilizando ferramentas Java”. Novatec Editora, São Paulo, SP, Brasil, 2004, 308p.
- [FER98a] P. Fernandes. “Méthodes Numériques pour la Solution de Systèmes Markoviens à Grand Espace d'États”, Tese de Doutorado, Institut National Polytechnique de Grenoble, France, 1998, 260p.
- [FER98b] P. Fernandes, W. J. Stewart e B. Plateau. “Efficient Descriptor-Vector Multiplication in Stochastic Automata Networks”. *Journal of the ACM*, vol. 45 (3), 1998, pp. 381–414.

- [FER99] P. Fernandes, C. G. P. Alegretti e R. Hessel-Jungblut. "Avaliação de Desempenho Através de Redes de Autômatos Estocásticos de Sistemas Paralelos". *VII ERI - Escola Regional de Informática*, 1999, pp. 159–184.
- [FER08] P. Fernandes, J. M. Vincent e T. Webber. "Perfect Simulation of Stochastic Automata Networks". *International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'08)*. *Lecture Notes in Computer Science*, vol. 5055, 2008, pp. 249–263.
- [FRE94] C. M. S. Freitas. "Uma abordagem unificada para análise exploratória e simulação interativa visual", tese de doutorado, Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciência da Computação, Brasil, 1994, 147p.
- [GIL94] S. Gilmore e J. Hillston. "The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling". In: *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, vol. 794, maio 1994, pp. 353–368.
- [GIL02] S. Gilmore, J. Hillston e M. Ribaud. "PEPA nets: A structured performance modelling formalism", In: *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*. *Lecture Notes in Computer Science*, vol. 2324, abril 2001, pp. 111–130.
- [GIL03] S. Gilmore, J. Hillston, L. Kloul e M. Ribaud. "PEPA nets: A structured performance modelling formalism". *Performance Evaluation*, vol. 54, outubro 2003, pp. 79–104.
- [HAG02] O. Häggström. "Finite Markov Chains and Algorithmic Applications". Cambridge University Press, 2002, 124p.
- [HOP00] J.E. Hopcroft e J.D. Ullman. "Introduction to Automata Theory, Languages and Computation". Addison-Wesley, 2.ed, 2000, 312p.
- [JUN01] R. Jungblut-Hessel, B. Plateau, W. J. Stewart e B. Ycart. "Fast simulation for Road Traffic Network", *RAIRO Operations Research*, vol. 35 (2), 2001, pp. 229–250.
- [KAM98] T. Kam, T. Villa, R. K. Bryaton e A. Sangiovanni-Vincentelli. "Multi-valued decision diagrams: theory and applications", *Multiplie-Valued Logic* vol. 4(1-2), 1998, pp. 9–62.
- [LAW00] J. Banks, J. S. Nelson e D. M. Nicol. "Simulation Modeling and Analysis". McGraw Hill, 2004, 800p.
- [MIN06] A. S. Miner. "Saturation for a General Class of Models". In: *IEEE Transactions on Software Engineering* vol. 32 (8), 2006, pp. 559–570.

- [MUR89] T. Murata. "Petri Nets: Properties, Analysis and Applications". In: Proceedings of IEEE, vol. 4, 1989, pp. 541–580.
- [OKE91] R. M. O’Keefe e I. L. Pitt. "Interaction With a Visual Interactive Simulation and the Effect of Cognitive Style", *European Journal of Operational Research*, vol. 54 (3), 1991, pp. 339–348.
- [PET81] J. L. Peterson. "Petri Net Theory and the Modeling of Systems". Prentice Hall PTR, 1981, 290p.
- [PLA85] B. Plateau. "On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms". In: ACM SIGMETRICS - Conference on Measurements and Modeling of Computer Systems, 1985, pp. 147–154.
- [PLA91] B. Plateau e K. Atif. "Stochastic Automata Networks for Modelling Parallel Systems". In: IEEE Transactions on Software Engineering, vol. 17 (10), outubro 1991, pp. 1093–1108.
- [POO96] R. Pooley e J. Hillston. "Refining internal choice in PEPA model". Proceedings of the Twelfth UK Performance Engineering Workshop, 1996, pp. 49–64.
- [ROD06] S. Rodger, T. Finley e P. Linz. "JFLAP". SIGCSE 2006 Workshop, março 2006.
- [ROM03] D. R. Vasconcelos. "Análise de Estratégias Utilizando Verificação Formal de Modelos", dissertação de mestrado, PUCRS-FACIN-PPGCC, Pontifícia Universidade Católica do Rio de Janeiro (PUCRS), Brasil, 2003.
- [ROO93] T. Murata. "A User-Centered Paradigm for Interactive Simulation". In: Society for Modeling and Simulation International (SCS), vol. 60 (3), 1993, pp. 168–177.
- [SAB06] F. Sabbadini e M.J.F. de Oliveira. "Simulação interativa visual: uma ferramenta para tomada de decisão". Anais do III Simpósio de Excelência em Gestão e Tecnologia, 2006.
- [SAL03] A. Sales. "Formalismos Estruturados de Modelagem para Sistemas Markovianos Complexos", dissertação de mestrado, PUCRS-FACIN-PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, 2003, 138p.
- [SAL09a] A. Sales e B. Plateau. "Reachable State Space Generation for Structured Models which use Functional Transitions". In: Proceedings of the 6th International Conference on the Quantitative Evaluation of Systems (QEST 2009), 2009, pp. 269–278.
- [SAL09b] A. Sales. "Réseaux d’Automates Stochastiques: Génération de l’espace d’états atteignables et Multiplication vecteur-descripteur pour une sémantique en temps discret", Tese de Doutorado, Institut National Polytechnique de Grenoble, France, 2009, 260p.

- [SCO06] A.P.S. Scolari. "Utilização de Diagramas de Decisão Multivalorados para Representação do Espaço de Estados Atingíveis em Redes de Autômatos Estocásticos", dissertação de mestrado, PUCRS-FACIN-PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, 2006, 87p.
- [STE94] W. J. Stewart. "Introduction to the numerical solution of Markov chains". Princeton University Press, 1974, 539p.
- [STE96] B. Plateau e W. J. Stewart. "Stochastic Automata Networks: Product Forms and Iterative Solutions", Rapport de Recherche - INRIA, nb. 2939, 1996.
- [SUN10] S. Microsystems. "JavaCC - The Java Compiler Compiler", julho 2004, 21p.
- [TAS10] D. Taschetto. "Precisão de Simulações para Soluções de Modelos Estocásticos", dissertação de mestrado, PUCRS-FACIN-PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, 2010, 91p.
- [ZAP08] I. S. Zapreev. "Model checking Markov chains : techniques and tools", Tese de Doutorado, University of Twente, Enschede, 2008.
- [WAG96] P. Wagner, C. Freitas e F. Wagner. "A New Paradigm for Visual Interactive Modelling and Simulation". Proceedings of the 8th European Simulation Symposium, 1996, pp. 142–145.
- [WEB03] T. Webber. "Alternativas para o Tratamento Numérico Otimizado da Multiplicação Vetor-Descritor", dissertação de mestrado, PUCRS-FACIN-PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, 2003, 98p.
- [WEB09] T. Webber. "Reducing the impact of state space explosion in Stochastic Automata Networks", Tese de Doutorado, Pontifícia Universidade Católica do Rio Grande do Sul - FACIN-PPGCC, Brasil, 2009, 120p.

A. VISUALSAN

Neste apêndice serão apresentados os componentes da ferramenta VisualSAN quanto às funcionalidades e utilização.

A.1 Ambiente de Edição

O ambiente é composto por dois componentes principais. A barra de tarefa, situada à esquerda, onde se encontra os botões com funções de criação dos autômatos, estados e transições. O editor de autômatos onde se desenha a estrutura completa de uma SAN. A Figura A.1 apresenta o ambiente de edição da ferramenta VisualSAN. A descrição para estes componentes será dada na próxima sub-seção.

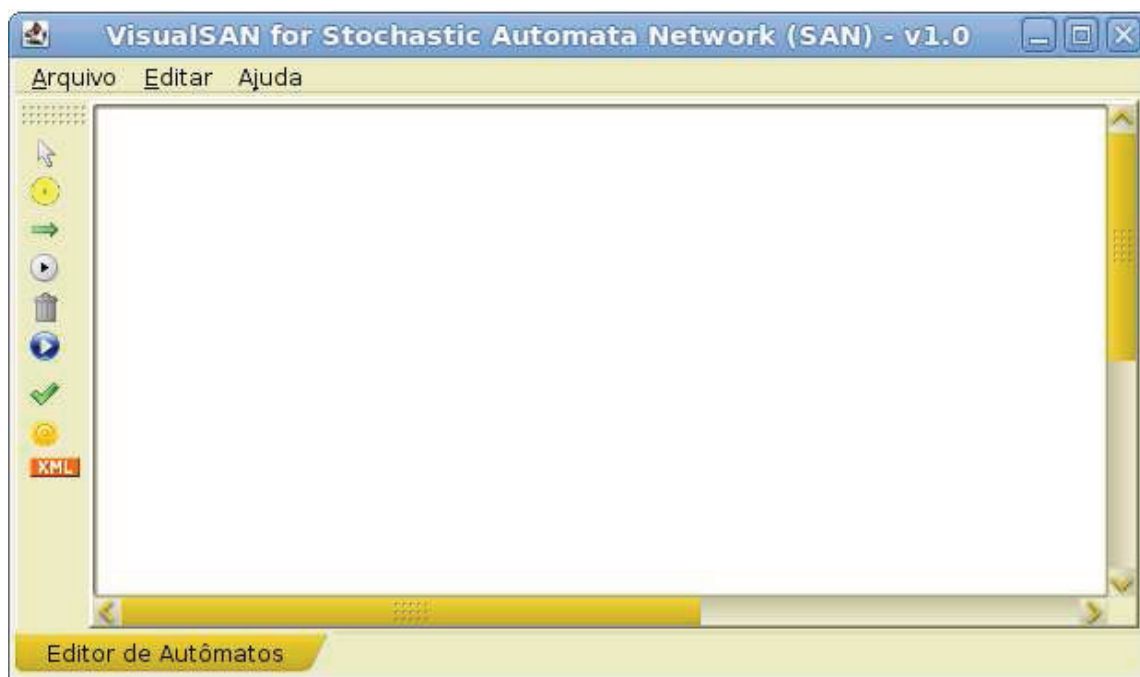


Figura A.1: Ambiente de edição da ferramenta VisualSAN

A.1.1 Componentes

Nesta sub-seção serão apresentados os componentes que fazem parte da estrutura composicional do software. Estes componentes são o meio de interação com o usuário da ferramenta.

A Figura A.2 é o componente para edição de estados e transições e com função de *drag-and-drop*. Este componente permite alterar rótulo de estados e transições, utilizando o botão oposto, e

arrastar qualquer elemento da SAN.



Figura A.2: Botão de edição de componentes

Na Figura A.3 têm-se o componente para inserção de estados. Para utilizá-lo o usuário deverá selecioná-lo e em seguida clicar no editor para gerar um novo estado.



Figura A.3: Botão de criar estados

O componente para inserção de transições está representado na Figura A.4. Para inserir uma transição o usuário deverá selecionar este componente e então clicar em um estado, para gerar uma transição, e em seguida selecionar um estado qualquer. É permitido gerar transições entre estados ou em apenas um estado.



Figura A.4: Botão de criar transições

Para deleção de estados e transições é disponibilizado o componente da Figura A.5. Sua utilização é simples: basta selecioná-lo e então clicar no componente do editor que o mesmo será deletado do ambiente.



Figura A.5: Botão de deleção de componentes

A Figura A.6 é utilizada para adicionar autômatos no editor. Selecione-o e informe o nome do autômato a ser adicionado no ambiente de edição de autômatos.

Na Figura A.7 temos a ação de execução. Este componente disponibiliza a execução do modelo SAN. Ao ser acionado será apresentado a tela com uma tabela de eventos correspondentes ao modelo SAN projetado no ambiente.



Figura A.6: Botão para adicionar autômatos



Figura A.7: Botão para execução

Componente que cria o arquivo “.san”, representado pela Figura A.8 baseado no modelo estabelecido no editor de autômatos. Quando acionado é apresentado a tela com uma estrutura básica do arquivo “.san” funcional. O usuário poderá implementar modificações se necessário, editando taxas, função de atingibilidade e as funções usadas para computar os índices de desempenho do modelo.



Figura A.8: Botão acionador da tela de edição do arquivo “.san”

A Figura A.9 Componente que cria o arquivo XML baseado no modelo estabelecido no editor de autômatos. Quando acionado é realizada uma gravação do modelo no formato XML conforme estabelecido no Capítulo 5, seção 5.2.



Figura A.9: Botão de gravação do modelo SAN em formato XML

A.2 Exemplo de Uso da Ferramenta

A interface principal do ambiente de edição de SAN aceita a inclusão de n autômatos, a restrição da quantidade de autômatos depende apenas no recurso computacional a disposição. Neste ambiente pode-se inserir autômatos e manuseá-los quanto a edição (alteração) de seus componentes (estados e transições) e de suas informações (rótulos de estados e transições).

A inclusão de autômatos é feita ativando-se o botão de adicionar autômatos (Figura A.6) na barra de componentes do editor (Figura 6.2). A Figura A.10 mostra o acionamento do componente e a janela de identificação do autômato criado.

Após a inserção e identificação de um autômato é possível criar estados. Os estados são criados acionando o componente de criação de estados (Figura A.3 que uma vez acionado habilita-se a



Figura A.10: Adicionando autômato no ambiente de edição de SAN

inserção de estados no ambiente de edição, a identificação de um estado é feita automaticamente através de numeração dada. Esta numeração é quantificada de acordo com a quantidade de estados inseridos. A Figura A.11 ilustra este procedimento.

As transições só poderão ser inseridas após a inserção dos estados. Para realizar este procedimento deve-se selecionar o componente de inclusão de transição (Figura A.4). A Figura A.12 ilustra o processo inicial, observa-se que o componente é selecionado (em destaque na barra de tarefas), que exige uma seleção de um estado, onde se origina a transição, e um outro estado, onde se destina a transição. A Figura A.13 demonstra a solicitação do rótulo da transição, na Figura A.14 tem-se o procedimento finalizado.

Depois da inclusão dos autômatos, o formalismo SAN exige pelo menos dois, a execução já pode ser iniciada. Para tanto, o sistema permite a configuração de taxas funcionais para os eventos dos autômatos. O que pode ser feito acionando o botão desta funcionalidade (Figura A.7). Outro procedimento que pode ser disparado é o de obtenção de estados atingíveis da SAN modelada, bastando selecionar o componente específico desta função. É importante frisar que para se obter os estados atingíveis o usuário do sistema deverá se atentar para as taxas funcionais, se caso o modelo exige as mesmas devem ser informadas. Caso contrário o sistema define as taxas com o valor 1. No Capítulo 6 é feita uma simulação completa de modelos SAN.

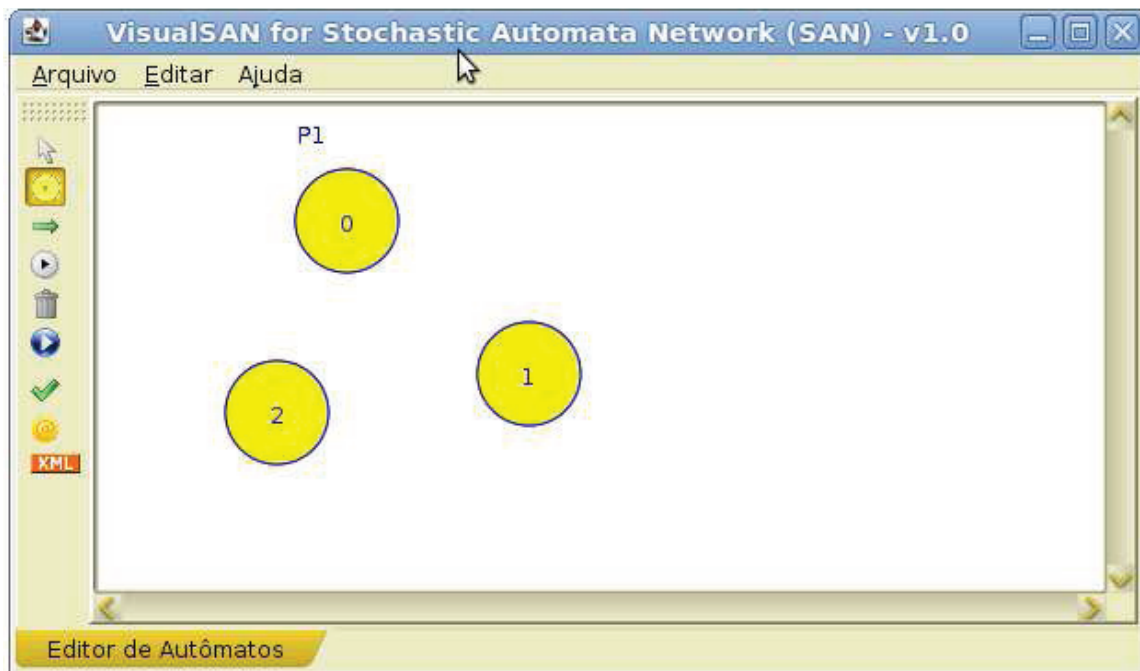


Figura A.11: Adicionando estado no ambiente de edição de SAN

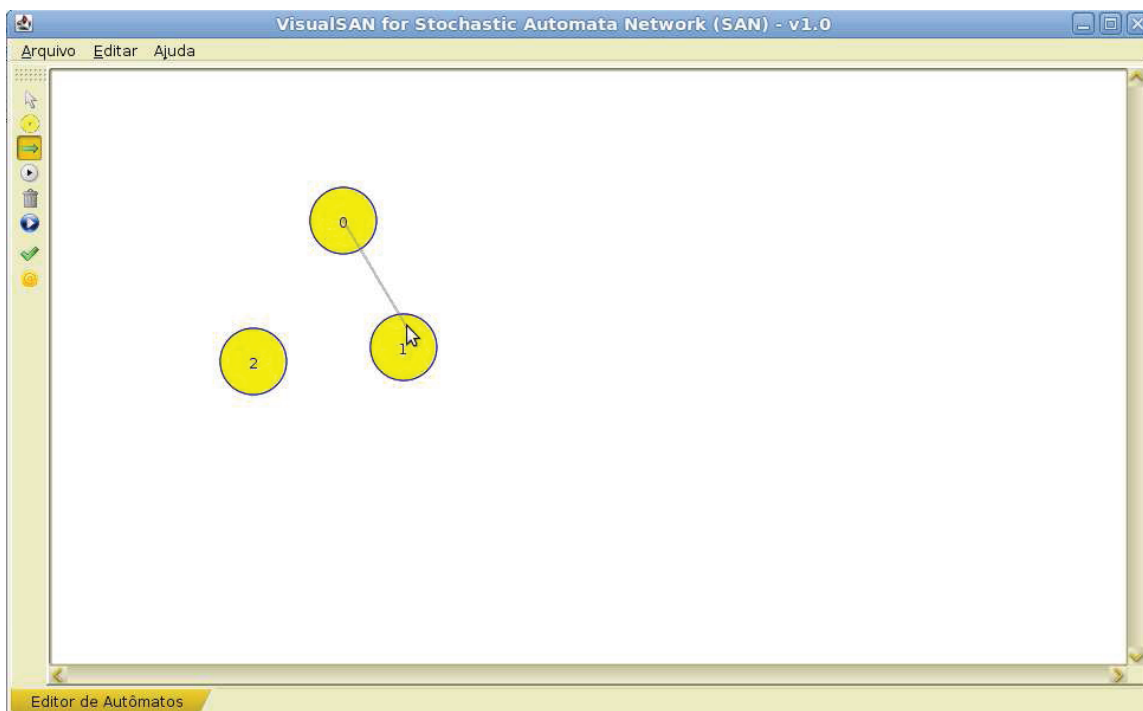


Figura A.12: Adicionando transição no ambiente de edição de SAN

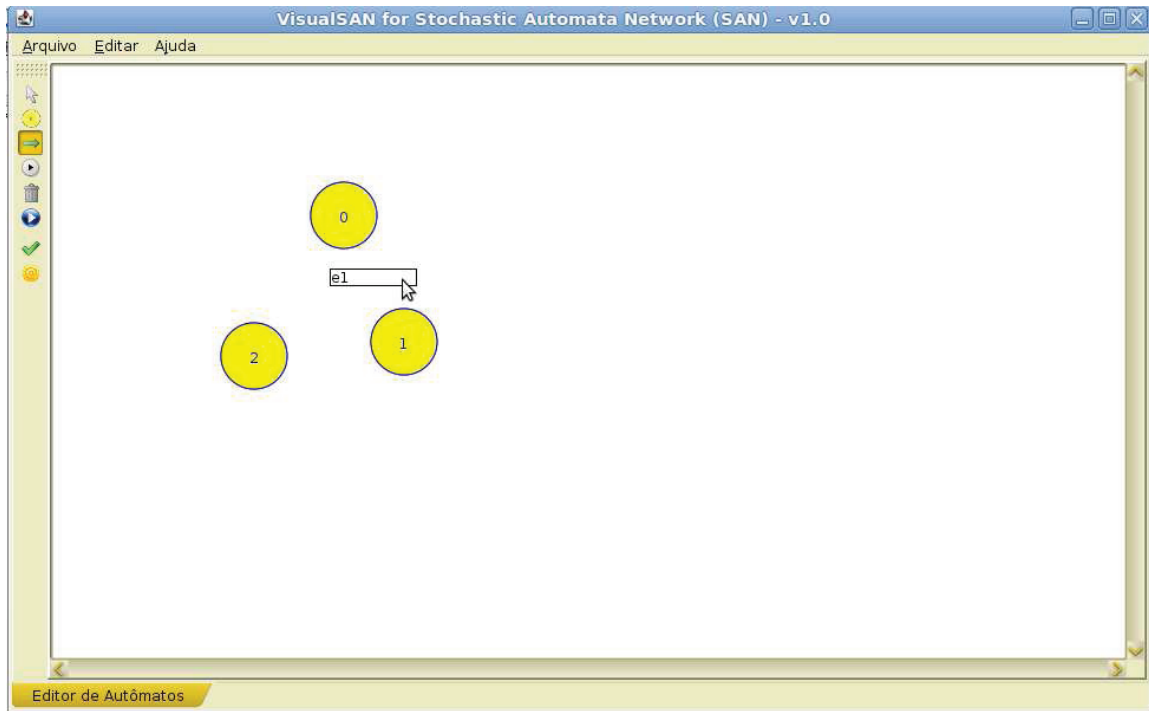


Figura A.13: Identificando uma transição no ambiente de edição de SAN

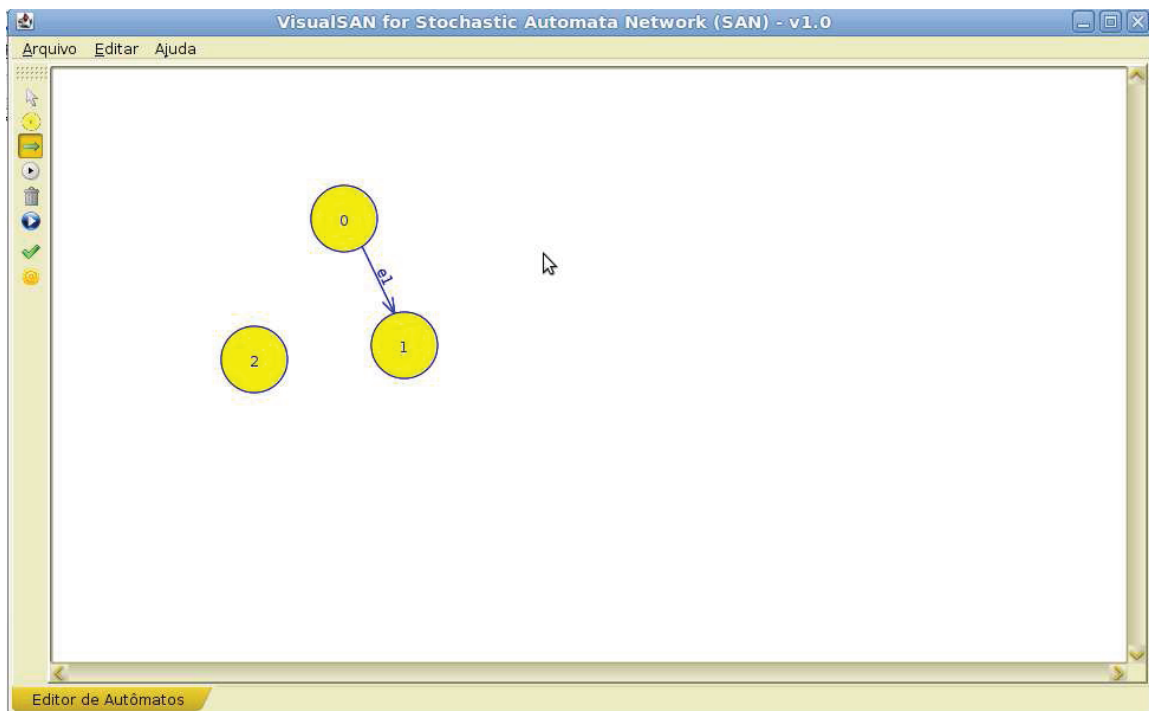


Figura A.14: Transição inserida no ambiente de edição de SAN

B. FERRAMENTAS

Nas próximas seções será feita uma abordagem das ferramentas JAVACC e JFLAP destacando algumas das características mais básicas de cada uma.

B.1 JAVACC e a Geração de Parser

O *JavaCC* [SUN10] é um gerador de *parser* e gerador de analisador léxico. Analisadores léxicos e *parsers* são componentes de *software* para lidar com entrada de caracteres sequenciais. Compiladores e interpretadores incorporam analisadores léxicos e *parsers* para decifrar arquivos contendo programas ou sentenças lógicas. Foi desenvolvido em Java e usa a notação de expressões regulares e formalismo de *Backus-Naur* que especifica a sequência legítima de tipos de *tokens* em entradas livres de erros.

A entrada para o analisador léxico é uma sequência de caracteres - representado pelo objeto *InputStream* ou um objeto *Reader*. A saída do analisador léxico é determinado pelo *JAVACC* sendo uma sequência de objetos *Tokens*. *Tokens* é resultado da quebra de um sequência de caracteres em uma subsequência de caracteres. Assim com a saída, a entrada também é fixada e é uma sequência de objetos *Tokens*.

B.1.1 Exemplo

Um exemplo para ilustrar o uso do *JAVACC* pode ser uma expressão tal como: $90 + 42$. Esta expressão pode ser avaliada pelo *JAVACC* pela observação da entrada de dados, neste caso, o exemplo citado será avaliado como uma subsequência de números e operadores existentes na expressão. Abaixo um exemplo básico do uso do *JAVACC*.

```

/* adder.jj - Adding up numbers */
options {
    STATIC = false;
}
PARSER BEGIN(Calculadora)
class Adder {
    static void main( String[] args ) throws ParseException, TokenMgrError
    {
        Calculadora somar = new Calculadora (System.in ) ;
        parser.Iniciar() ;
    }
}
PARSER END(Calculadora)
SKIP : { " " }
SKIP : { "\n" | "\r" | "\r\n" }

```

```

TOKEN : { < ADICAO : "+" > }
TOKEN : { < NUMERO : (["0"-"9"])+ > }
void Iniciar() :
{
}
{
  <NUMERO>
  (
    <ADICAO>
    <NUMERO>
  )*
  <EOF>
}

```

Este exemplo basicamente contém a identificação dos *tokens*, **ADICAO** e **NUMERO**. O procedimento a ser realizado pelo *JAVACC* é o seguinte: (i) o *parser* irá identificar, na expressão fornecida, se há espaços em branco() ou se há entradas iguais a "\n" ou "\r" que serão ignorados e não serão avaliados, (ii) o *parser* irá efetuar a substituição na expressão onde houver o símbolo "+" pela palavra "ADICAO" e onde houver números será efetuado a substituição pela palavra "NUMERO". Sendo assim, considerando a expressão $90 + 42$ esta será processada pela ferramenta *JAVACC* apresentando a seguinte saída: **NUMERO, ADICAO, NUMERO**.

Como dito anteriormente, a sequência de caracteres fornecida foi quebrada em *tokens* e avaliada em subsequência de caracteres que representam cada elemento da expressão permitindo que haja uma interpretação destes *tokens*, pela ferramenta, a fim de realizar procedimentos computacionais como a criação de pseudo-linguagens, sistemas para avaliação de expressões matemáticas.

B.2 JFLAP

A ferramenta *JFLAP* [ROD06] é um editor de autômatos para diferentes tipos de máquinas. Este pacote de ferramenta *open-source* é distribuído livremente sob licença da *Duke University*. O intuito desta ferramenta é ser utilizada como uma ajuda na aprendizagem dos conceitos básicos de Linguagens Formais e Teoria dos Autômatos. *JFLAP* permite que usuários possam criar e operar em autômatos, gramática, *L-systems*, e expressões regulares; o termo estruturado é usado para referir a qualquer autômato, gramática, *L-system*, ou expressões regulares. *JFLAP* oferece os seguintes grupos majoritários de operações para aplicar em estruturas:

- **Explorar a estrutura de linguagem** - *JFLAP* tem a capacidade de simular sequências de entrada de autômatos não-determinísticos, construir tabelas de análise e árvores de análise para gramáticas, e renderizar sucessivas expansões de *L-systems*. Os autômatos representados nesta ferramenta são autômatos finitos (FA), autômatos de pilha (PDA) e máquinas de *Turing multi-tape*.

- **Conversão entre Estruturas Equivalentes** - Uma vasta gama de habilidades de conversão está disponível, por exemplo, expressão regular para FA, FA não-determinísticos para FA determinísticos (DFA), PDA para gramática, gramática para PDA, DFA para DFA minimizado, gramática livre de contexto de forma normal da gramática de Chomsky, e outros.
- **Diversas Análises de Estruturas** JFLAP também oferece algumas ferramentas de análise para apresentar diversas propriedades de estruturas, como destacar λ -transição, estados não-determinísticos, e determinar a equivalência de dois FAs.

Esta ferramenta é disponibilizada de forma a permitir modificações por parte de interessados para representar qualquer realidade onde se aplica o conceito de autômatos.