

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

HUGO VARES VIEIRA

GERAÇÃO DE CASOS DE TESTE
PARA A INTERFACE DE USUÁRIO DE
SISTEMAS DE GERÊNCIA DE WORKFLOW

Porto Alegre

2008

HUGO VARES VIEIRA

**GERAÇÃO DE CASOS DE TESTE
PARA A INTERFACE DE USUÁRIO DE
SISTEMAS DE GERÊNCIA DE WORKFLOW**

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, pelo Programa de Pós-Graduação em Ciência da Computação da Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador:

Prof. Dr. Duncan D. A. Ruiz

Porto Alegre

2008

Dados Internacionais de Catalogação na Publicação (CIP)

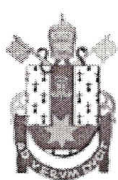
V658g Vieira, Hugo Vares
 Geração de casos de teste para a interface de usuário de
 sistemas de gerência de workflow / Hugo Vares Vieira. – Porto
 Alegre, 2008.
 72 f.

 Diss. (Mestrado) – Fac. de Informática, PUCRS
 Orientador: Prof. Dr. Duncan D. A. Ruiz

 1. Informática. 2. Redes de Petri. 3. Interface com o usuário.
 4. Workflow. 5. Processos de Negócios. I. Ruiz, Duncan D. A.
 II. Título.

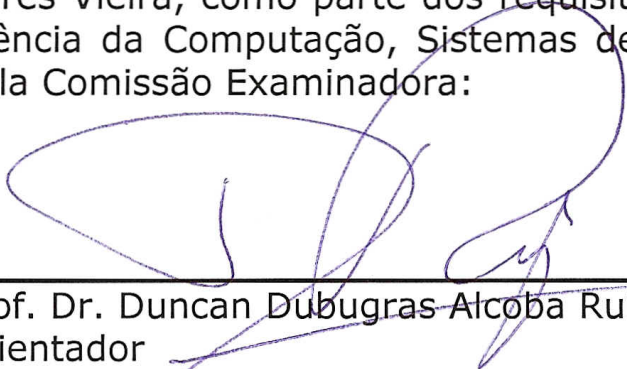
CDD 005.431

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

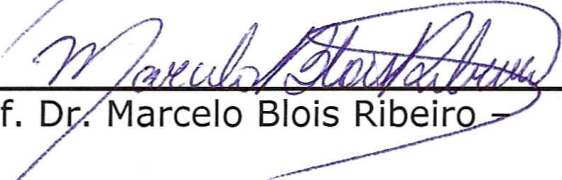


TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Geração de Casos de Testes Para a Interface de Usuário de Sistemas de Gerência de *Workflow***", apresentada por Hugo Vares Vieira, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 17/12/2007 pela Comissão Examinadora:


Prof. Dr. Duncan Dubugras Alcoba Ruiz –
Orientador

PPGCC/PUCRS



Prof. Dr. Marcelo Blois Ribeiro –

PPGCC/PUCRS


Prof. Dr. Flávio Moreira de Oliveira –

FACIN/PUCRS

Homologada em 19/08/08, conforme Ata No. 017/08 pela Comissão Coordenadora.


Prof. Dr. Fernando Gehm Moraes
Coordenador.



PUCRS

Campus Central

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

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

E-mail: ppgcc@inf.pucrs.br

www.pucrs.br/facin/pos

À minha família e à minha querida namorada Juci.

AGRADECIMENTOS

Ao Prof. Dr. Flávio Oliveira um agradecimento especial por sempre acreditar no potencial desta pesquisa, pelo apoio dado em todos os momentos e por servir de exemplo a ser seguido.

Ao meu orientador Prof. Dr. Duncan Dubugras Ruiz por toda a disponibilidade, interesse, atenção e dedicação.

À empresa Hewlett-Packard pelo apoio durante todo o tempo do mestrado.

A todos os integrantes que estão neste momento, ou passaram, pelos grupos GPIN e CPTS, os quais certamente influenciaram positivamente neste trabalho.

À minha família, por todo o seu amor, apoio incondicional e por estarem sempre ao meu lado.

Ao meu amor por toda sua generosidade e compreensão.

Aos amigos que nas horas difíceis proporcionaram conforto e tranquilidade para conclusão deste trabalho.

Às demais pessoas que torceram ou de alguma forma ajudaram a tornar este trabalho realidade.

RESUMO

O teste de um sistema de gerenciamento de *workflow* se faz necessário no momento em que há a necessidade de detecção de erros na execução de sistemas que apóiam processos de negócio. Como não é viável a realização de testes exaustivos, esta pesquisa faz uso de teste estatístico para que se tenha uma confiabilidade aceitável com um conjunto de testes menor. Sabe-se que as redes de Petri, além de possibilitarem a representação de informações estatísticas, são um formalismo muito indicado para descrever processos de negócio. As informações relevantes ao teste estatístico de *workflow* são representadas utilizando redes de Petri, para então gerar casos de teste a partir de uma rede.

Este trabalho tem o objetivo de apresentar uma solução para a geração de casos de teste para sistemas de gerenciamento de *workflows*. Estes casos de teste visam a interação com a interface de usuário. Assim, neste trabalho, é apresentada uma descrição referente à automação de *workflows* e ao teste destes sistemas. A partir desta descrição, são determinados os requisitos para a formalização de um dialeto de redes de Petri, para então ser apresentada a formalização do dialeto utilizado na pesquisa. Além disso, é ilustrada a aplicação desta solução. Devido às similares necessidades verificadas no teste de desempenho em relação ao teste de sistemas de gerenciamento de *workflows*, um estudo da aplicação da solução desta pesquisa ao teste de desempenho é realizado.

Palavras-chave: Teste Estatístico de *Software*; Sistemas de Gerenciamento de *Workflow*; Redes de Petri; Teste de Desempenho.

ABSTRACT

Testing of a workflow management system is necessary when there is a need for error detection in the execution of systems which support business processes. Since it is not possible to carry out exhausting tests, this research uses statistical tests in order to have an acceptable reliability with a smaller testing set. Besides making the representation of statistical information possible, Petri nets provide a formalism very suitable to describe business processes. The information relevant to the workflow statistical test is represented by using Petri nets in order to generate test cases from a net.

This dissertation aims at presenting a solution for test case generation of workflow management systems. The object of these test cases is to interact with the user interface. Thus, in this work, a description of workflow automation and the testing of these systems is provided. The requirements for the formalization of a Petri net dialect are determined from this description. The formalization of the dialect used in the research is presented soon afterward. Furthermore, the application of this solution is illustrated. Due to the similar needs verified in the performance test in relation to the workflow management system test, a study of the application of this research solution to the performance test is conducted.

Key words: Software statistical test; workflow management systems; Petri nets; Performance test.

LISTA DE FIGURAS

Figura 1	Relacionamento entre termos provenientes de um processo de negócio (WfMC, 1999).	16
Figura 2	<i>Workitems</i> em <i>worklists</i>	16
Figura 3	Camadas de uma arquitetura de <i>workflow</i> (LEYMANN; ROLLER, 2000). . .	17
Figura 4	Modelo de referência (HOLLINGSWORTH, 1995).	18
Figura 5	Conjunto de casos de teste para um teste estrutural (JORGENSEN, 2002). .	25
Figura 6	Teste de caixa preta (JORGENSEN, 2002).	25
Figura 7	Conjunto de casos de teste para um teste funcional (JORGENSEN, 2002). .	26
Figura 8	Representação de uma atividade genérica para teste de <i>SGWf</i> utilizando <i>WfGSPN</i>	40
Figura 9	Representação de <i>AND-Join</i> para teste de <i>SGWf</i> utilizando <i>WfGSPN</i> . . .	41
Figura 10	Representação de <i>XOR-Split</i> para teste de <i>SGWf</i> utilizando <i>WfGSPN</i> . . .	41
Figura 11	Representação de uma transição de <i>workflow</i> (transição_WF) para teste de <i>SGWf</i> utilizando <i>WfGSPN</i>	42
Figura 12	Representação de um executor (<i>Performer</i>) para teste de <i>SGWf</i> utilizando <i>WfGSPN</i>	42
Figura 13	Representação de construções de <i>deadlines</i> para teste de <i>SGWf</i> utilizando <i>WfGSPN</i>	44
Figura 14	Exemplo 1 de tempo de espera.	45
Figura 15	Exemplo de caso de teste para seqüência de atividades do exemplo 1. . .	45
Figura 16	Exemplo 2 de tempo de espera.	45
Figura 17	Exemplo de caso de teste para seqüência de atividades do exemplo 2. . .	45
Figura 18	Exemplo de <i>deadline</i> síncrono.	46
Figura 19	Exemplo de casos de teste para <i>deadlines</i>	46

Figura 20	Representação do processo festa.	48
Figura 21	Rede gerada para a geração dos casos de teste.	49
Figura 22	Exemplo 1 de caso de teste gerado (<i>Test Case 6</i>).	51
Figura 23	Caminho percorrido pelo exemplo 1 (<i>Test Case 6</i>).	52
Figura 24	Exemplo 2 de caso de teste gerado (<i>Test Case 7</i>).	52
Figura 25	Caminho percorrido pelo exemplo 2 (<i>Test Case 7</i>).	53
Figura 26	Representação de início do fluxo com o PAoccurrence para teste de desempenho utilizando <i>WfGSPN</i>	56
Figura 27	Representação de uma atividade genérica para teste de desempenho utilizando <i>WfGSPN</i>	56
Figura 28	Representação de uma atividade com <i>think time</i> para teste de desempenho utilizando <i>WfGSPN</i>	56
Figura 29	Representação de uma atividade com tempo de resposta para teste de desempenho utilizando <i>WfGSPN</i>	57
Figura 30	Representação 1 para alocação de recurso para teste de desempenho utilizando <i>WfGSPN</i>	57
Figura 31	Representação 2 para alocação de recurso para teste de desempenho utilizando <i>WfGSPN</i>	57
Figura 32	Representação de <i>XOR Split</i> para teste de desempenho utilizando <i>WfGSPN</i>	58
Figura 33	Representação de <i>XOR Join</i> para teste de desempenho utilizando <i>WfGSPN</i>	58
Figura 34	Representação de <i>AND Split</i> para teste de desempenho utilizando <i>WfGSPN</i>	59
Figura 35	Representação de <i>AND Join</i> para teste de desempenho utilizando <i>WfGSPN</i>	59
Figura 36	Representação de um caso de uso para teste de desempenho utilizando <i>WfGSPN</i>	60
Figura 37	Arquitetura UpperT.	61
Figura 38	Diagrama de casos de uso de um servidor de FTP.	61
Figura 39	Diagramas de atividades de um servidor de FTP.	62
Figura 40	Rede de Petri gerada pelo UpperT.	63

LISTA DE TABELAS

Tabela 1	Possibilidades de teste para as interfaces do modelo de referência.	30
Tabela 2	Taxas das transições da rede.	50
Tabela 3	Relacionamentos com a pesquisa.	67

LISTA DE SIGLAS

API	<i>Application Programming Interface</i>
CPTS	<i>Centro de Pesquisas em Teste de Software</i>
DBMS	<i>Database Management Systems</i>
FTP	<i>File Transfer Protocol</i>
GSPN	<i>Generalized Stochastic Petri Nets</i>
HP	<i>Hewlett-Packard</i>
PN	<i>Petri Nets</i>
PUCRS	<i>Pontifícia Universidade Católica do Rio Grande do Sul</i>
SGWF	<i>Sistemas de Gerência de Workflow</i>
SPN	<i>Stochastic Petri Nets - Redes de Petri Estocásticas</i>
UML	<i>Unified Modeling Language</i>
UML SPT Profile	<i>UML Profile for Schedulability, Performance and Time</i>
WAPI	<i>Workflow Application Programming Interface</i>
WfGSPN	<i>Workflow - Generalized Stochastic Petri Nets</i>
WfMC	<i>Workflow Management Coalition</i>
WfMS	<i>Workflow Management Systems</i>
XPDL	<i>XML Processing Description Language</i>

SUMÁRIO

1	INTRODUÇÃO	13
2	AUTOMAÇÃO DE WORKFLOW	15
2.1	ARQUITETURA	16
2.2	MODELAGEM DE WORKFLOWS	20
2.3	PADRÕES DE FLUXO	21
2.4	PRESENÇA DE PARTICIPANTES	22
2.5	TRATAMENTO DE TEMPO	22
2.6	CONSIDERAÇÕES	22
3	TESTE DE SOFTWARE APLICADO A SISTEMAS DE GERENCIAMENTO DE WORKFLOWS	24
3.1	TESTE ESTRUTURAL	24
3.2	TESTE FUNCIONAL	25
3.3	TESTE ESTATÍSTICO	26
3.4	CONSIDERAÇÕES	28
4	TESTE ESTATÍSTICO DE WORKFLOW	29
4.1	CENÁRIO	29
4.2	CARACTERIZAÇÃO DO PROBLEMA E REQUISITOS DA SOLUÇÃO . . .	30
4.3	OBJETIVO GERAL E ESPECÍFICOS	31
4.4	QUESTÃO DE PESQUISA	32
4.5	CONSIDERAÇÕES	32

5	UMA METODOLOGIA PARA TESTE ESTATÍSTICO DE WORKFLOW	33
5.1	REQUISITOS DA FORMALIZAÇÃO PARA REDES DE PETRI A SER UTILIZADA	33
5.2	DEFINIÇÃO DA WfGSPN	34
5.2.1	REDES DE PETRI ESTOCÁSTICAS	35
5.2.2	REDES DE PETRI ESTOCÁSTICAS GENERALIZADAS	36
5.2.3	REDES DE PETRI ESTOCÁSTICAS GENERALIZADAS MODIFICADAS	36
5.3	CONSIDERAÇÕES	39
6	APLICAÇÃO DA SOLUÇÃO	40
6.1	GERAÇÃO DA REDE	40
6.2	APLICAÇÃO	44
6.3	EXEMPLO	47
6.4	CONSIDERAÇÕES	50
7	SOLUÇÃO APLICADA AO TESTE DE DESEMPENHO	54
7.1	UML SPT PROFILE	55
7.2	TRADUÇÃO DE UML SPT PROFILE PARA REDES DE PETRI	55
7.3	UPPERT	60
7.4	EXEMPLO DE UMA REDE GERADA PARA O TESTE DE DESEMPENHO	61
7.5	CONSIDERAÇÕES	61
8	TRABALHOS RELACIONADOS	64
8.1	WORKFLOWS REPRESENTADOS ATRAVÉS DE REDES DE PETRI	64
8.2	REDES DE PETRI COM PROBABILIDADE	65
8.3	TESTE DE SOFTWARE A PARTIR DE REDES DE PETRI	65
8.4	TRABALHOS RELEVANTES À PESQUISA	66
8.5	CONSIDERAÇÕES	66

9 CONCLUSÕES E TRABALHOS FUTUROS

68

REFERÊNCIAS

70

1 INTRODUÇÃO

Um processo de negócio é um conjunto de um ou mais procedimentos ou atividades relacionadas que realizam coletivamente um objetivo de negócio. Estes procedimentos ou atividades estão normalmente inseridos dentro do contexto de uma estrutura organizacional, que define papéis e relacionamentos funcionais. *Workflow* é a automatização de um processo de negócio, por inteiro ou em parte, onde documentos, informações ou atividades são passadas de um participante a outro, de acordo com um conjunto de regras. Um sistema que define, cria e controla a execução dos *workflows* com o uso de *software* é definido como um sistema de gerenciamento de *workflow* (*SGWf*), *Workflow Management System* (*WfMS*). Estes conceitos são definidos por Hollingsworth (1995).

Um sistema de *workflow* possibilita o gerenciamento do que se passa em um processo organizacional, através da gerência de atividades e da distribuição apropriada de recursos humanos e/ou associadas com as várias etapas do processo (HOLLINGSWORTH, 1995). Assim sendo, diversos benefícios podem ser obtidos com a utilização de sistemas de gerenciamento de *workflow* nas organizações: redução de tempo nas transferências de informações e execução de atividades; maior controle e segurança do processo; simplificação da supervisão humana; maior grau de integração entre os participantes dos processos; e melhor identificação de gargalos no processo.

Como em qualquer *software*, a execução de *workflows* pode conter falhas. Para isso, a comparação do que a definição do processo expressa e o que realmente está acontecendo durante a execução do *workflow* possibilita que se determine falhas na execução. Com isso, é objetivo desta pesquisa a geração de casos de teste para testes estatísticos em sistemas de gerência de *workflows*. Como a intenção é validar a execução de *workflows*, este teste está focado no ponto de vista do usuário responsável por esta execução. Assim, deve-se verificar ocorrências em que a execução não está compatível com o que a definição do processo representa, ou seja, se a gerência das atividades está se dando de maneira correta.

Devido à diversidade de possíveis execuções do fluxo de controle de *workflows*, a verificação de todas as possibilidades de execução é se torna impossível. Assim, é importante pri-

orizar os pontos mais utilizados do *software*. Desta forma, ao basear a geração de casos de teste em informações referentes à utilização do fluxo de controle, tem-se testes focados em submeter o *SGWf* a condições reais de uso. Com isso, é possível estimar a qualidade do teste realizado do ponto de vista do usuário.

Este documento apresenta a solução encontrada em relação ao objetivo desta pesquisa. Para isso, são discutidas as características relevantes ao teste de *SGWf* e formalizado um dialeto de redes de Petri que permita a representação destas características e a geração de casos de teste com base nelas. Então, é demonstrado um experimento com base na solução desta pesquisa. Por fim, é apresentado um estudo para a aplicação da solução desta pesquisa ao teste de desempenho, pois muitas das necessidades desta pesquisa também estão presentes no teste de desempenho.

Assim sendo, esta pesquisa apresenta a formalização de um modelo de uso para a geração de casos de teste para *SGWf*. Esta formalização contempla as informações estatísticas necessárias ao teste estatístico, informações temporais, participantes, além de permitir a representação dos padrões básicos encontrados em *workflows*.

Este documento está organizado como segue: o Capítulo 2 apresenta à base teórica referente a automação de *workflows*, descrevendo a arquitetura destes sistemas, a modelagem de *workflows*, juntamente com a descrição das principais características em *workflows* (padrões de fluxo, presença de participantes e tempo). No Capítulo 3 são apresentados conceitos referentes ao teste de software, relacionando-os ao teste de um *SGWf*. O Capítulo 4 apresenta o cenário da pesquisa, a caracterização do problema e os requisitos da solução. Estes itens servem de base para a apresentação dos objetivos gerais e específicos da pesquisa, além da questão de pesquisa que é solucionada neste trabalho. No Capítulo 5, é apresentada a contribuição desta pesquisa, a formalização de um dialeto de redes de Petri capaz gerar casos de teste estatísticos para um *SGWf*. No Capítulo 6, uma experimentação da solução é descrita, para isso um exemplo é ilustrado. No Capítulo 7, é descrita uma abordagem desta solução para o teste de desempenho, para isso é apresentada uma metodologia para geração de uma rede baseada no dialeto desta pesquisa. Feito isto, descreve-se uma ferramenta desenvolvida para a geração de scripts de teste, juntamente com a apresentação de um exemplo. No Capítulo 8, são citados os trabalhos que se relacionam com esta pesquisa. Por fim, no Capítulo 9, encontram-se as considerações finais e a descrição dos trabalhos futuros.

2 AUTOMAÇÃO DE WORKFLOW

Modelos de processo descrevem a estrutura de um processo de negócio do mundo real. Eles definem todos os trajetos possíveis do processo de negócio e as atividades que necessitam ser executadas (LEYMANN; ROLLER, 2000). Um modelo de *workflow* é a parte automatizada do modelo do processo, devendo ser modelado de tal maneira que a estrutura do processo de negócio se reflita claramente (AALST; HEE, 2002). O modelo de *workflow* é o molde para cada um dos processos instanciados (LEYMANN; ROLLER, 2000). Com isso, um sistema que gerencia *workflows* deve administrar distintas instâncias de processo. Uma instância de processo é composta por conjunto de instâncias de atividades. Cada instância de atividade pode ser definida como a chamada de uma aplicação ou como um item de trabalho a ser executado por participantes do *workflow*, podendo ser composta por ambos os casos. A Figura 1 demonstra o relacionamento entre os conceitos apresentados (WfMC, 1999). Pode-se verificar que o lado esquerdo da figura representa a definição do processo; já o direito representa a automatização e gerência do processo.

De acordo com Hollingsworth (1995), a tecnologia de *workflow* está concentrada na automação de procedimentos, nos quais as atividades ou os dados são passados entre os participantes de acordo com um conjunto de regras definidas para que se alcance, ou contribua, para um objetivo de negócio. Cada participante do *workflow* possui uma lista (*worklist*) composta pelas atividades que ele deve executar (*workitems*). A *worklist* de um usuário pode conter *workitems* provenientes de diversas instâncias de processo: uma *worklist* pode, por exemplo, conter duas instâncias de atividade, com a mesma definição, provenientes de diferentes processos. Isto é representado pela Figura 2, através das atividades *Atv_2* na *worklist 1* e das atividades *Atv_4* na *worklist 2*. Além disso, o mesmo *workitem* pode fazer parte de diversas *worklists* (LEYMANN; ROLLER, 2000). Neste caso, quando o primeiro participante selecionar a atividade para executá-la, ela sairá das demais *worklists* onde estava presente.

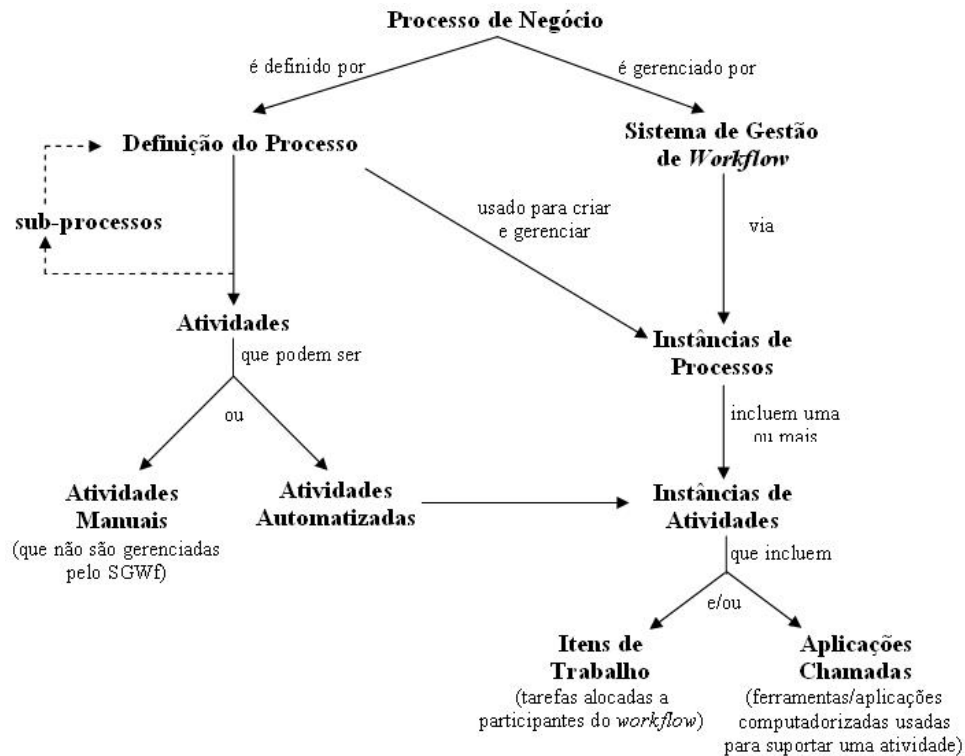


Figura 1: Relacionamento entre termos provenientes de um processo de negócio (WfMC, 1999).

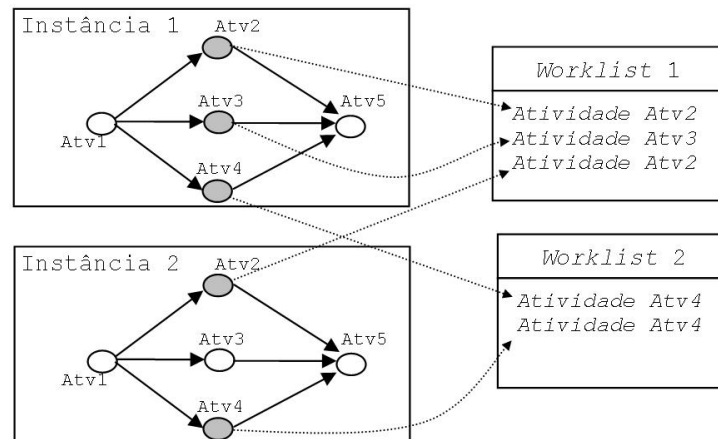


Figura 2: Workitems em worklists.

2.1 ARQUITETURA

Em *SGWfs*, as notificações de trabalho são enviadas aos participantes de forma automática. Estes sistemas possuem tipicamente uma camada responsável pelo gerenciamento destas notificações (LEYMANN; ROLLER, 2000). Com isso, a estrutura de um sistema de gerenciamento de *workflow* é tipicamente estruturada em três camadas (LEYMANN; ROLLER, 2000). Assim, a primeira camada funciona como um cliente do *WfMS*; da mesma forma que a segunda camada (*WfMS*) é o cliente do *DBMS* (*Database management systems*), Figura 3.

A primeira camada é tipicamente a interface na qual o usuário final trabalha. A segunda

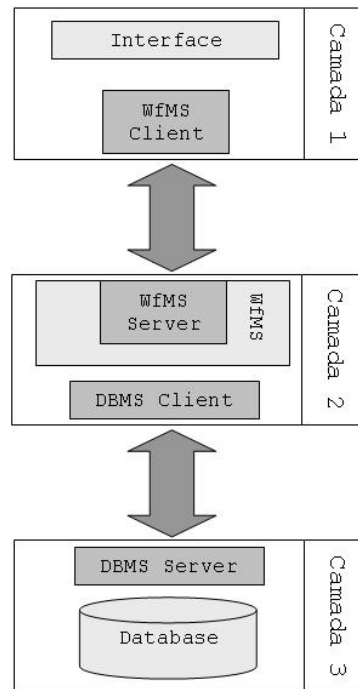


Figura 3: Camadas de uma arquitetura de *workflow* (LEYMANN; ROLLER, 2000).

camada é a responsável pelo gerenciamento e execução de *workflows*. Esta camada está associada com a execução do processo, com a navegação entre as atividades, gerenciamento dos participantes, e disparo de *workitems*. A terceira camada é responsável pelo armazenamento e manutenção da consistência dos dados. Pode-se optar por uma visão que trate a segunda e a terceira camada como uma única (LEYMANN; ROLLER, 2000).

Segundo Leymann e Roller (2000), fica claro que entre o usuário final e a gerência de *workflows* há uma camada responsável pela interação com o usuário final. Desta forma, a interface de um sistema de *workflow* é encarregada do encaminhamento de todas as execuções realizadas por seus participantes ao sistema de gerenciamento de *workflow*.

A *Workflow Management Coalition (WfMC)*, instituição que tem por objetivo promover e desenvolver o uso da tecnologia de *workflow* através de sua padronização, propõe um modelo de referência para a interoperabilidade de sistemas de *workflow* (HOLLINGSWORTH, 1995). Assim, os *SGWf* exibem determinadas características comuns no intuito de possibilitar a integração e a interoperabilidade entre diferentes produtos. Este modelo de referência é apresentado na Figura 4.

De acordo com Hollingsworth (1995), os módulos que compõem o modelo de referência são:

Serviço de execução de *workflow* é quem fornece um ambiente de execução em que a instanciação e a ativação dos processos utilizam um ou mais motores de *workflow*, responsáveis

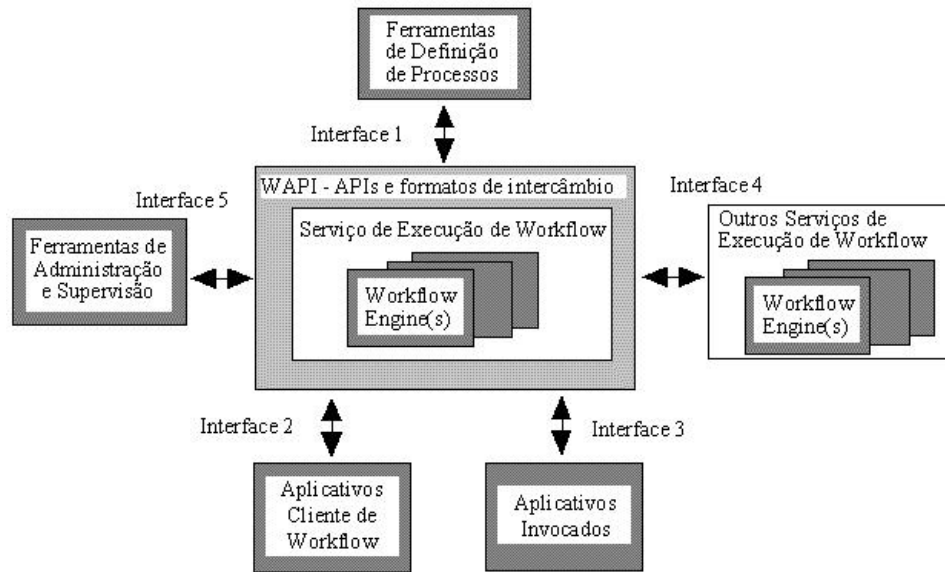


Figura 4: Modelo de referência (HOLLINGSWORTH, 1995).

pela interação com os recursos externos necessários para processar as várias atividades, além de interpretar e ativar uma parte ou toda a definição do processo.

Ferramentas de definição de processos usadas para analisar, modelar, descrever e documentar um processo do negócio.

Aplicativos clientes de *workflow* são módulos do *software* que interagem com as atividades que requerem envolvimento de usuários.

Aplicativos invocados são aplicações que podem ser chamadas para que determinada atividade seja executada.

Outros serviços de execução de *workflow* são diferentes ambientes de execução que interagem para que o objetivo do negócio seja alcançado.

Ferramentas de administração e supervisão têm como objetivo permitir uma visão completa do *status* das atividades da organização.

A comunicação entre os diferentes módulos se dá através de interfaces. Cada um dos módulos possui uma interface de comunicação com o serviço de execução de *workflow*. Estas interfaces definem o formato da comunicação. Esta comunicação se dá através de chamadas *API* (*Application Programming Interface*). De acordo com Hollingsworth (1995), as interfaces que ligam os diferentes módulos ao serviço de execução do *workflow* são:

Interface de definição do *Workflow* (Interface 1) permite uma ligação entre as ferramentas de definição de processos e os ambientes onde são executados (Serviço de execução de *workflow*). Fornece o acesso às seguintes propriedades:

- Estruturação do processo;
- Atividades e navegação;
- Regras e participantes;
- Condições dos gatilhos;
- Chamadas de aplicações.

Interface da aplicação do cliente do *workflow* (Interface 2) possibilita a comunicação entre o serviço de execução do *workflow* e os aplicativos clientes de *workflow*. Possibilita o acesso as seguintes funções:

- Funções de estado do processo;
- Comandos de manipulação de *worklist*;
- Funções de controle do processo e atividade;
- Conexão/desconexão;
- Configuração de comando.

Interface de invocação de aplicações (Interface 3) tem como foco estabelecer a ligação com as aplicações que foram projetadas para interagirem com o serviço de execução do *workflow*. As *APIs* devem ter funções para:

- Estabelecimento de sessões;
- Funções de gerenciamento de atividades;
- Funções de manipulação de dados.

Funções de interoperabilidade (Interface 4) define uma forma padrão para a interação entre dois serviços de execução de *workflow*. Esta interface deve suportar a transferência de dados de controle, informações relativas ao estado do processo de *workflow*. Para isso, é necessária uma interpretação comum da definição dos processos. As funções que podem ser usadas para a interoperabilidade entre os serviços de *workflow* são:

- Chamadas de atividades ou de sub-processos;
- Status dos processos/atividades;
- Status de controle;
- Transferência de dados relevantes entre a aplicação e o *workflow*;
- Coordenação de pontos de sincronização;

- Leitura e escrita na definição dos processos.

Interface de administração e monitoramento (Interface 5) trata da administração e auditoria/monitorização dos sistemas de *workflow*. Têm-se como funções acessadas através das chamadas de *API* os seguintes itens:

- Gerência de usuários;
- Gerência de papéis;
- Funções de supervisionamento de processos;
- Controle de recursos.

Assim sendo, a comunicação entre os diferentes módulos com o serviço de execução de *workflow* se dá através das chamadas de *API* (ou *WAPI*, *Workflow Application Programming Interface*) e dos documentos padrão de troca de informações (HOLLINGSWORTH, 1995).

2.2 MODELAGEM DE WORKFLOWS

Os diagramas de atividades são bastante utilizados para realizar a modelagem dos aspectos dinâmicos do sistema, mostrando o fluxo de uma atividade para outra, dando ênfase ao fluxo de controle entre objetos (BOOCH; RUMBAUGH; JACOBSON, 2000). O diagrama de atividades é uma ótima notação para a representação de seqüência de atividades, sendo especialmente utilizado para a representação de fluxo de trabalho (LARMAN, 2004).

São encontradas, também, representações de *workflows* utilizando redes de Petri, visto que as redes de Petri são uma classe de formalismos utilizados para a representação de processos. Em (AALST, 1998), apresenta-se diversas vantagens em utilizar Redes de Petri para a representação de fluxo de processos:

1. possuem uma definição precisa do fluxo, pois se tratam de um formalismo.
2. é uma linguagem gráfica, o que favorece muito a compreensão.
3. suportam todas as construções básicas encontradas em processos de *workflow*; nas últimas décadas diversas pessoas pesquisaram as propriedades das Redes de Petri, o que possibilita uma firmeza na conceituação e modelagem das redes.
4. são marcadas pela disponibilidade de muitas técnicas da análise.
5. são independentes de versões ou distribuições.

2.3 PADRÕES DE FLUXO

O fluxo de um *workflow* pode possuir diversas construções. De acordo com WfMC (1999), o fluxo em sistemas de *workflow* pode apresentar sete diferentes padrões: *Parallel Routing*, *Sequential Routing*, *AND-Split*, *AND-Join*, *OR-Split*, *OR-Join* e *Iteration*. Segundo Russell et al. (2006), podem ser encontrados 43 padrões de fluxos para *workflows*. Estes padrões estão classificados em diversas categorias: **(1) Basic Control-Flow Patterns**; **(2) Advanced Branching and Synchronization Patterns**; **(3) Structural Patterns**; **(4) Multiple Instance Patterns**; **(5) State-based Patterns**; **(6) Cancellation Patterns**. Os padrões básicos (*Basic Control-Flow Patterns*) encontrados em (RUSSELL et al., 2006) correspondem aos seguintes padrões, encontrados em (WfMC, 1999): *Sequential Routing*, *AND-Split*, *AND-Join*, *OR-Split* e *OR-Join*. Estes padrões são caracterizados da seguinte forma (WfMC, 1999):

- *Sequential Routing* é caracterizado pelo fato de uma instância de atividade ter o início de sua execução apenas após o término da execução da instância de atividade anterior.
- *AND-Split* consiste em um ponto no processo onde um único fluxo se divide em múltiplos fluxos do controle que podem ser executadas paralelamente, permitindo, assim, que instâncias de atividades sejam executadas simultaneamente.
- *AND-Join* é um ponto no processo onde diferentes fluxos paralelos convergem em um único fluxo de controle. Para que a execução do fluxo prossiga, é necessário que todos os fluxos paralelos que convergem para a sincronização. (*AND-Join*) tenham sido completados.
- *OR-Split* caracteriza-se pelo fato de em um ponto no processo onde, baseado em dados de decisão ou de controle do *workflow*, é escolhido um caminho entre os possíveis a serem seguidos.
- *OR-Join* um ponto no *workflow* onde dois ou mais fluxos de execução unem-se sem que haja uma sincronização.

Ainda em (AALST et al., 2003), realizou-se um levantamento com quinze ferramentas distintas de gerência de *workflow* para determinar a quais padrões estas davam suporte. Apenas seis, dos vinte padrões, tinham suporte de todas as ferramentas investigadas. Destes, cinco são classificados como padrões básicos e o outro padrão é chamado de *Multiple Instances With a Priori Design Time Knowledge* e classificado como padrões com múltiplas instâncias.

2.4 PRESENÇA DE PARTICIPANTES

A presença de participantes atrelados às atividades faz com que a autenticação de usuários seja algo relevante ao teste de *workflows*. Isto se deve ao fato de que cada participante do *workflow* possui uma lista (*worklist*) composta pelas atividades que ele deve executar (*workitems*). Desta forma, a autenticação de cada usuário se faz importante pois, dependendo do usuário, o conjunto de atividades que devem ser executadas será distinto. Isto pode ser verificado na Figura 2, na qual a *worklist* do Participante 1 (*Worklist 1*) possui duas instâncias da atividade *Atv_2* e uma da atividade *Atv_3* para serem executadas ao passo que a *worklist* do Participante 2 (*Worklist 2*) possui duas instâncias da atividade *Atv_4*.

2.5 TRATAMENTO DE TEMPO

O aspecto tempo, em *SGWf*, possui uma importância muito grande, pois o fluxo pode ter seu trajeto alterado simplesmente em função dele. Sistemas de *workflow* possibilitam que o tempo de realização de cada atividade seja controlado (LEYMANN; ROLLER, 2000) e procedimentos extras possam ser invocados se o tempo para a execução de determinada atividade for ultrapassado (WfMC, 1999).

Atividades em *workflows* podem estar atreladas a exceções temporais, as quais podem ou não alterar o fluxo (WfMC, 2005). Estas exceções são chamadas de *deadlines*. Um *deadline* pode ser síncrono ou assíncrono. Um *deadline* síncrono encerra a atividade caso ela não tenha sido executada até o momento e segue o fluxo de exceção. Um *deadline* assíncrono consiste na ativação de um ramo de execução do fluxo de controle em paralelo com a atividade em questão. Pode-se encontrar ocorrências em que ambos os tipos de *deadlines* estejam presentes na mesma atividade (WfMC, 2005). Desta forma, o tempo do condicional do *deadline* síncrono deve ser superior ao do assíncrono caso ambos existam na mesma atividade, pois não faz sentido abrir um fluxo em paralelo a uma atividade que já foi encerrada.

2.6 CONSIDERAÇÕES

Neste capítulo, verificou-se que *SGWf* possuem, claramente, uma separação entre a camada de apresentação ao usuário e a camada responsável pelo gerenciamento das atividades. Além disso, há a definição de uma interface para esta comunicação. Estas características sugerem a possibilidade de realização de testes em *SGWf* independentemente da forma como é desenvolvida e/ou estruturada a camada de apresentação ao usuário. Assim, os casos de teste gerados

devem verificar se atividades estão sendo entregues aos participantes corretos. Desta forma, a geração de casos de teste deve conter, além da listagem das atividades a serem executadas, a informação de qual usuário deve executar cada atividade.

Além disso, sistemas de *workflows* possuem características temporais. Estas características devem ser levadas em consideração no momento do teste. O teste deve simular o tempo gasto para a execução das atividades do *workflow*, a fim de validar também as exceções temporais presentes nestes sistemas. As atividades não devem ser executadas de forma instantânea. O atraso de atividades faz com que seja verificada a validade das ocorrências temporais presentes nos *workflows*.

No capítulo seguinte, são apresentados alguns conceitos de testes de software, relacionando-os ao teste de sistemas de gerenciamento de *workflows*.

3 **TESTE DE SOFTWARE APLICADO A SISTEMAS DE GERENCIAMENTO DE WORKFLOWS**

Sistemas devem ser testados para que se faça um julgamento a respeito da qualidade do *software*, além de descobrir problemas (JORGENSEN, 2002). Durante a geração de código de um *software*, pessoas erram. Estes erros ocasionam defeitos. Estes defeitos ocasionam falhas na execução do *software*. Em outras palavras, no momento em que um programador gera um erro na codificação, um defeito está sendo criado, que durante a execução do *software* ocasionará uma falha (BURNSTEIN, 2002) (JORGENSEN, 2002). Desta forma, o teste tem como objetivo encontrar falhas na execução do *software*.

Um teste é o ato de exercitar um *software* com casos de teste. Um caso de teste é um conjunto de entradas e uma lista de saídas esperadas (JORGENSEN, 2002). A essência do teste de *software* é determinar um conjunto de casos de teste para o que deve ser testado (JORGENSEN, 2002). Duas categorias fundamentais são utilizadas para identificar casos de teste: o teste estrutural e o teste funcional (JORGENSEN, 2002). O teste estatístico é outra técnica de teste (PROWELL et al., 1999), que tem como objetivo expor o *software* a condições reais de uso. Esta técnica baseia-se em informações referentes à utilização do *software* para determinar os casos de teste.

3.1 **TESTE ESTRUTURAL**

O teste estrutural tem como objetivo determinar se todos os elementos lógicos e de dados do *software* estão funcionando corretamente (BURNSTEIN, 2002). Os casos de teste para o sistema são obtidos a partir de sua implementação. Assim, o conjunto de casos de teste está baseado no conjunto de implementações do sistema, não estando totalmente voltado ao conjunto das especificações do *software*, ver Figura 5 (JORGENSEN, 2002).

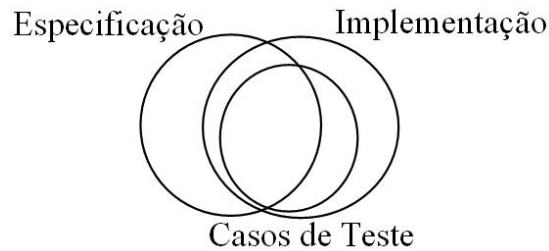


Figura 5: Conjunto de casos de teste para um teste estrutural (JORGENSEN, 2002).

Utilizando o teste estrutural, os casos de teste: garantem que todos os caminhos independentes dentro de um módulo tenham sido exercitados; verificam todas as decisões lógicas para valores falso ou verdadeiro; executam todos os laços em suas fronteiras e dentro de seus limites operacionais; e exercitam as estruturas de dados internas para garantir a sua validade (PRESSMAN, 1995).

O teste estrutural é utilizado no nível de teste unitário (BURNSTEIN, 2002) (JORGENSEN, 2002). Desta forma, esta categoria de teste é indicada ao teste de cada componente utilizado no sistema de gerenciamento de *workflows*. Para isto, é necessário o conhecimento do código fonte do sistema. Conhecendo o código do sistema, é possível realizar uma análise dos pontos mais críticos do *software*, o que qualificaria o teste. Entretanto, para isso, exigir-se-ia o acesso ao código fonte da aplicação. Considerando que cada módulo do modelo de referência pode ser desenvolvida por diferentes fabricantes, seria complicada uma análise geral do produto.

3.2 TESTE FUNCIONAL

O teste de funcional baseia-se na especificação do sistema. Testes funcionais não levam em consideração a implementação do sistema. Desta forma, sua realização se dá apenas em função das entradas e das saídas do sistema, por isso este tipo de teste também é chamado de teste de caixa preta, ver Figura 6 (JORGENSEN, 2002).

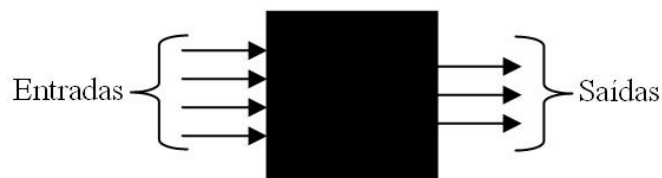


Figura 6: Teste de caixa preta (JORGENSEN, 2002).

Os casos de teste para o sistema são obtidos a partir de sua especificação. Assim, o conjunto de casos de teste está baseado no conjunto das especificações do sistema, não estando totalmente voltado ao conjunto de operações do *software*, ver Figura 7 (JORGENSEN, 2002).

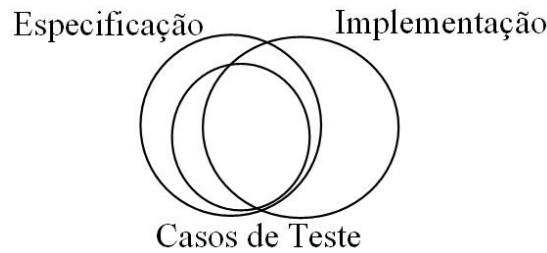


Figura 7: Conjunto de casos de teste para um teste funcional (JORGENSEN, 2002).

As categorias de erros mais evidenciadas pelo teste funcional são: erros de interface, funções incorretas ou ausentes, erros nas estruturas de dados ou no acesso a bancos de dados externos, erros de desempenho e erros de iniciação e término (PRESSMAN, 1995). O foco está nas entradas e nas saídas apropriadas para cada função testada (BURNSTEIN, 2002).

No teste funcional, todos os requisitos funcionais para o sistema devem ser alcançados pelo sistema (BURNSTEIN, 2002):

- Todos os tipos ou classes de entradas legais devem ser aceitos pelo *software*.
- Todas as classes de entradas ilegais devem ser rejeitadas.
- Todas as classes possíveis de saída do sistema devem ser executadas e examinadas.
- Todos os estados do sistema e suas transições devem ser executados e examinados.
- Todas as funções devem ser executadas.

Desta forma, em *SGWfs*, o teste funcional permite validar a interação entre os diferentes módulos do sistema, aplicando diferentes entradas e verificando suas saídas. Ferramentas de definição de processo, por exemplo, podem ser analisadas através da interação do usuário e da verificação se os requisitos da Interface 1 estão sendo atendidos em função das ações realizadas pelo usuário. Além disso, em sistemas de gerenciamento de *workflow*, pode-se verificar ocorrências em que a execução de um *workflow* não ocorre de acordo com a definição do seu respectivo modelo de *workflow*. Esta é carregada pelo *SGWf* e pode ser expressa através de um arquivo *XPDL* (*XML Processing Description Language*) (WfMC, 2005). Ela contém todas as informações necessárias para a criação e gerenciamento das instâncias de *workflow*.

3.3 TESTE ESTATÍSTICO

Em *workflows*, a variedade de possíveis execuções do fluxo de controle é muito grande, impossibilitando a realização de testes exaustivos para modelos de média e alta complexidade.

Com o objetivo de agregar qualidade do teste do ponto de vista do usuário, deve-se priorizar os pontos mais utilizados do fluxo de controle. Assim sendo, é indicada a utilização de teste estatístico. Esta técnica tem como objetivo expor o *software* a condições reais de uso. Desta forma, ao se analisar a correta execução do fluxo, é possível simular usuários reais, fazendo com que pontos mais utilizados do fluxo recebam uma maior atenção no momento do teste. Para isso, são necessárias informações referentes à execução do fluxo. Estas informações são mapeadas para um modelo de uso (PROWELL et al., 1999) (BURNSTEIN, 2002).

Os modelos de uso empregados no teste estatístico tratam da descrição dos possíveis usos de um sistema, e suas interações com o usuário, através do emprego de um modelo de estados discretos (KALLEPALLI; TIAN, 2001). Os tempos presentes no sistema também são mapeados de maneira discreta para o modelo. Modelo de uso é um grafo onde existem estados e transições entre esses estados. O conjunto dos estados e das transições entre os estados é chamado de estrutura do modelo. Esta estrutura é utilizada para descrever os possíveis usos do *software*. São atribuídas, à estrutura, as probabilidades que representam o uso previsto do *software* (TRAMMELL, 1995). Estas probabilidades descrevem o comportamento do usuário diante do sistema. Assim, em testes baseados em modelos, tem-se, como entrada, a execução de determinada atividade e, como saída, a ativação da próxima atividade a ser executada. Desta forma, o conjunto de casos de teste é dado pela seqüência de atividades a serem executadas.

Para o mapeamento de um modelo de uso, são muitas vezes empregadas cadeias de Markov, a fim de possibilitar o emprego de elementos estatísticos, para que as probabilidades de ocorrência de cada transição possam ser mapeadas (PROWELL et al., 1999). As cadeias de Markov têm como problema a explosão na quantidade de estados, pois algumas construções de fluxo (paralelismos) não são representadas com muita eficiência por elas.

Por outro lado, as redes de Petri são uma classe de formalismos bastante utilizadas para a representação de processos, permitindo que todas as construções básicas encontradas em processos de *workflow* sejam representadas (AALST, 1998). Além do mais, este formalismo permite a representação do tempo de forma contínua, tornando a geração de casos de teste mais realista. Desta forma, agrega-se bastante qualidade aos casos de teste gerados, devido ao fato que, em *workflows*, o fluxo de controle pode ser alterado apenas em função do tempo. Um simples atraso na execução de uma atividade pode fazer com que outra atividade seja encaminhada a outro participante do fluxo.

Nas redes de Petri, é possível atrelar taxas às transições, possibilitando a geração de casos de testes através da análise do modelo. Assim, um lugar corresponde a uma circunstância que pode ser usada como pré- e/ou pós-condição para tarefas (AALST; HEE, 2002). Além disso, este formalismo foi aplicado com sucesso à análise de desempenho de sistemas cujas características

principais incluem a simultaneidade e a sincronização (CHIOLA et al., 1993) (OLIVEIRA et al., 2007), presentes e importantes em *SGWfs*.

3.4 CONSIDERAÇÕES

Considerando a presença de interfaces bem definidas para a comunicação com o serviço de execução de *workflow*, torna-se possível a execução dos testes independentemente da implementação. Com isso, possibilita-se também o teste de cada um dos módulos do modelo de referência. Para tanto, a estratégia adotada deve ser o teste funcional, o qual não depende da implementação do *software*. Em se tratando do teste do ponto de vista dos executores do *workflow*, tem-se que o teste estatístico apresenta um retrato do comportamento do usuário em relação ao processo. O teste estatístico proporciona que se obtenha um conjunto de casos de teste que priorizam os pontos mais utilizados do sistema.

4 TESTE ESTATÍSTICO DE WORKFLOW

4.1 CENÁRIO

Na tentativa de obter uma maior qualidade em seus processos, cada vez mais as empresas têm procurado estruturá-los. Uma alternativa para isto é a utilização de *SGWf*. Desta forma, além de estruturar, é possível que se realize a gerência dos processos na organização, obtendo um maior controle sobre cada um.

Entretanto, em *SGWf*, como em qualquer outro *software*, pode haver falhas na execução. Isto implica em erros na gerência dos processos, ocasionando uma queda em sua qualidade. Assim sendo, há a necessidade de se testar sistema de gerenciamento de *workflows*, pois não adianta se utilizar um *SGWf* para aumentar a qualidade do processo, se os *workflows* contém falhas na execução. Um determinado *deadline* pode não estar sendo respeitado, gerando uma perda na pró-atividade do *SGWf* e, por exemplo, ocasionando uma ociosidade no processo.

Considerando os cinco módulos que interagem com este serviço de execução através de suas respectivas interfaces no modelo de referência, veja Figura 4, é possível acessar o serviço de execução do *workflow* através de qualquer uma de suas interfaces. Desta forma, cada módulo poderia ser simulado, podendo ser realizados testes funcionais para cada uma das funcionalidades de um sistema de *workflow*. Cada interface do modelo de referência é bidirecional, ou seja, permite o acesso tanto ao módulo como ao serviço de execução de *workflow*. Assim, cada interface fornece dois objetivos ao teste. Na Tabela 1, são apresentadas quais verificações são possíveis para cada interface.

Além disso, considerando que *SGWf* têm por objetivo a gerência das atividades e da distribuição apropriada de recursos humanos e/ou ferramentas ou aplicações associadas com as várias etapas do processo (HOLLINGSWORTH, 1995), pode-se testar se a gerência destas atividades e recursos está se dando de maneira correta. Para isto, deve-se verificar se a gerencia está condizente com o que foi definido. Com isso, tem-se que o *SGWf* é uma caixa preta (teste

Tabela 1: Possibilidades de teste para as interfaces do modelo de referência.

Interfaces	Módulo -> Serviço de Execução É possível verificar se:	Serviço de Execução -> Módulo É possível verificar se:
Interface 1	o motor de <i>workflow</i> está carregando corretamente o documento de definição de processo.	a geração de documento de definição de processo está sendo realizada de maneira correta; o módulo está respondendo de maneira correta a chamadas do motor de <i>workflow</i> .
Interface 2	o motor de <i>workflow</i> está se comportando corretamente frente às diversas intervenções dos executores das atividades; o controle da lista de trabalho (<i>worklist</i>) dos usuários, composta pelas atividades a serem executadas (<i>workitems</i>); o roteamento das mensagens está sendo realizado corretamente.	cada função enviada pelo motor de <i>workflow</i> está sendo realizada corretamente na interface de apresentação ao usuário.
Interface 3	o retorno de cada aplicação está sendo tratado corretamente pelo motor de <i>workflow</i> ; o chamado de outras aplicações está sendo realizado de maneira correta; as sessões estão sendo estabelecidas; se atividades estão sendo gerenciadas; e se os dados estão sendo manipulados corretamente.	a comunicação com outras aplicações está ocorrendo de maneira correta; as aplicações estão realizando as solicitações do motor de <i>workflow</i> .
Interface 4	o motor de <i>workflow</i> está se comportando corretamente mediante as respostas referentes às solicitações realizadas a outros motores de <i>workflow</i> .	outros motores de <i>workflow</i> estão trabalhando como o esperado a cada solicitação do motor de <i>workflow</i> .
Interface 5	o motor de <i>workflow</i> está fornecendo corretamente as informações relativas a status dos processos; ações administrativas estão sendo executadas corretamente	as informações relativas a supervisão e administração fornecidas pelo motor de <i>workflow</i> estão sendo apresentadas corretamente.

funcional), cuja entrada é a definição do processo e a saída é a gerencia do processo. Assim sendo, para uma dada entrada (documento de definição de processo, interface 1) temos apenas um conjunto de possibilidade de saídas válidas.

4.2 CARACTERIZAÇÃO DO PROBLEMA E REQUISITOS DA SOLUÇÃO

Usuários podem interagir com um sistema de *workflow* através de duas maneiras: como administradores ou como executores. Administradores, responsáveis pela gestão e acompanhamento dos *workflows*, interagem com o módulo de Ferramentas de administração e supervisão do modelo de referência, o qual se comunica com o serviço de execução de *workflow* através da interface 5. Usuários responsáveis pela execução dos *workflows* interagem com o módulo Aplicativos clientes de *workflow*, o qual se comunica com o serviço de execução do *workflow*

através da interface 2. A simulação da interação do usuário com o sistema para a realização de teste pode se dar através destas duas interfaces do modelo de referência. Desta forma, a interface 2 possibilita que seja testado todo o controle das *worklists*, enquanto a interface 5 proporciona que seja verificada se a gerência e a apresentação dos dados está sendo realizada de maneira correta. Assim sendo, para que o teste leve em consideração a presença de distintos usuários ao longo do fluxo de controle é indicado o teste a partir da interface 2, pois através desta interface as atividades são apresentadas do ponto de vista dos executores do *workflow*, sendo distribuídas através das *worklists* dos usuários.

A garantia de que o teste dê suporte a distintos padrões de fluxo encontrados em *workflows* é uma preocupação desta pesquisa. Considerando que apenas utilizando os padrões básicos é possível a representação dos demais padrões, esta pesquisa se preocupa com a realização de testes em *workflows* que utilizem apenas os padrões básicos (AALST et al., 2003).

No teste estatístico, os casos de teste são específicos, pois são gerados tendenciosamente, dando maior atenção aos pontos mais utilizados na execução do *software*. Além disso, em *workflows*, decisões podem ser tomadas apenas em função do tempo, sem que necessariamente tenha a interação de um usuário ou aplicação. O fluxo de controle pode ser alterado apenas em função do tempo. Um simples atraso na execução de uma atividade pode fazer com que outra atividade seja encaminhada a outro participante do fluxo. Sabendo que o teste estatístico pode utilizar modelos de uso, para a descrição dos possíveis usos de um sistema e suas interações com o usuário, este trabalho utiliza rede de Petri para esta descrição. Isto se deve ao fato de que as redes de Petri permitem a representação das informações necessárias ao teste estatístico baseado em modelos utilizando tempo de forma contínua, possibilitando a geração de casos de teste mais realistas. Além disso, este é um formalismo muito indicado para a representação de *workflows*.

4.3 OBJETIVO GERAL E ESPECÍFICOS

A utilização de um dialeto de redes de Petri que possibilite a geração de casos de teste para *SGWf* se mostra como um caminho a ser seguido para que se atinja o objetivo desta pesquisa. Este dialeto deve contemplar as definições necessárias ao teste estatístico e as características presentes em *workflows*, para então ser apresentada uma forma de tradução destas informações para uma rede definida por este dialeto.

Assim sendo, o presente trabalho tem como objetivo apresentar uma forma de geração de casos de testes para *SGWf*. Estes casos de testes consideram as particularidades presentes em *workflows*. Devido à complexidade de cenários e a preocupação com a confiabilidade do teste,

o presente trabalho utiliza o teste estatístico. Além disso, a geração destes casos de teste se dá de forma automatizada.

Para isto, são realizados os seguintes objetivos específicos:

1. definir uma forma de representação do modelo que contenha todas as informações necessárias ao teste, incluindo características temporais, participantes e padrões de fluxo;
2. agregar informações estatísticas ao modelo;
3. gerar um conjunto de casos de teste baseados no modelo, levando em consideração as diversas informações contidas nele.

4.4 QUESTÃO DE PESQUISA

Como gerar, de forma automatizada, casos de teste para *SGWf*, aplicando técnicas estatísticas e levando em consideração os distintos padrões de fluxo, as características temporais e a presença de participantes?

4.5 CONSIDERAÇÕES

Após a apresentação do cenário de pesquisa, o problema foi caracterizado e os requisitos da solução determinados. Baseado nestes, tem-se o objetivo e a questão de pesquisa.

No capítulo seguinte, é apresentada e demonstrada uma abordagem ao teste de sistemas de gerenciamento de *workflows*. Para isto, são descritos os requisitos da formalização para a rede de Petri. Para então, ser demonstrada a formalização da rede.

5 UMA METODOLOGIA PARA TESTE ESTATÍSTICO DE WORKFLOW

Para a realização de testes estatísticos em *SGWf*, faz-se necessário a utilização de um modelo que possibilite a representação das informações estatísticas. Para isto, esta pesquisa utiliza redes de Petri. As redes de Petri possibilitam a inserção de informações estatísticas, possibilitando a geração de casos de testes através da análise do modelo, além de serem um formalismo bastante utilizado para a representação de processos.

5.1 REQUISITOS DA FORMALIZAÇÃO PARA REDES DE PETRI A SER UTILIZADA

A seguir, são enumerados estes requisitos para a formalização da rede a ser utilizada, para então apresentar a formalização do dialeto de rede de Petri que possibilita a geração de casos de teste para *SGWf*:

1. **Início do fluxo:** deixar claro a representação da atividade inicial. Assim, é possível determinar por onde se deve iniciar a execução da rede. Além disso, *workflows* possuem apenas uma atividade inicial.
2. **Recursos para atividades:** possibilitar a representação e simulação dos recursos (*Performers*) presentes do processo.
3. **Seleção de caminhos:** basear as seleções de caminhos em informações estatísticas.
4. **Deadlines:** executar a rede considerando as restrições impostas pelos *deadlines*. Para que isto ocorra, é necessário que a rede contemple e simule os tempos de execução das atividades
5. **Fim do fluxo:** deixar clara a representação da atividade final. *Workflows* podem conter

mais de um fim de fluxo, porém adotou-se a representação de apenas um, para que a determinação de quando uma dada instância terminou sua execução na rede seja simplificada.

6. **Informações referentes ao processo:** representar as informações referentes ao processo, como identificador da instância de *workflow*, dados relevantes ao processo, lista de participantes, nome do processo, etc.
7. **Informações referentes aos recursos:** representar o tipo de cada recurso, por exemplo um usuário do Setor financeiro.
8. **Padrões básicos:** suportar a representação dos padrões básicos (*Sequence*, *AND-Split*, *AND-Join*, *OR-Split* e *OR-Join*)

5.2 DEFINIÇÃO DA WfGSPN

Uma rede de Petri compreende um conjunto de lugares P , um conjunto de transições T , e um conjunto de arcos dirigidos F . Na representação gráfica de redes de Petri os lugares são expressos como círculos e transições como barras. Os arcos conectam transições aos lugares e os lugares às transições. Os lugares podem conter *tokens*, que são expressos como pontos pretos. O estado de uma rede de Petri é definido pelo número de *tokens* contido em cada lugar e denotado por um vetor M , cujo i -ésimo componente m_i representa o número de *tokens* no i -ésimo lugar. O estado da rede é chamado, geralmente, de marcação da rede. A definição de uma rede de Petri requer a especificação da marcação inicial M' .

A definição de formal de redes de Petri PN (*Petri Nets*) é dada por:

- $PN = (P, T, F, M')$
- $P = \{p_1, p_2, \dots, p_n\}$
- $T = \{t_1, t_2, \dots, t_m\}$
- $F \subseteq (P \times T) \cup (T \times P)$;
- $M' = \{m'_1, m'_2, \dots, m'_n\}$

Um lugar p é chamado de um lugar de entrada de uma transição t se e somente se existe um arco dirigido de p a t . Um lugar p é chamado de lugar de saída de uma transição t se existe um arco dirigido de t para p (MARSAN; CONTE; BALBO, 1984). É utilizado $\bullet t$ para se referenciar ao conjunto de lugares de entrada para uma transição t . Para denotar o conjunto de lugares de saída

da uma transição t utiliza-se $t\bullet$. Da mesma forma, é utilizada esta notação para se referenciar ao conjunto de transições de entrada e de saída de um lugar ($\bullet p$ e $p\bullet$) (AALST; HEE, 2002).

A execução de uma transição t consiste na retirada dos *tokens* dos lugares de entrada e o depósito de *tokens* nos lugares de saída. A quantidade de *tokens* que é retirada ou depositada depende do peso de cada arco. Peso corresponde ao número de marcações que são levadas de um lugar a uma transição, ou de uma transição a um lugar, por um dado arco. Quando for omitido o valor do peso de um dado arco, é considerado que o peso do arco é 1. Uma transição é dita habilitada em uma dada marcação M se, e somente se, para todo os seus lugares de entrada a quantidade de *tokens* for igual ou maior ao peso do respectivo arco que liga o lugar à transição.

5.2.1 REDES DE PETRI ESTOCÁSTICAS

Redes de Petri estocásticas *SPN* (*Stochastic Petri Nets*) são dadas por (MARSAN; CONTE; BALBO, 1984):

- $SPN = (P, T, F, M', R)$,

onde (P, T, F, M') é uma *PN*, e

- $R = \{r_1, r_2, \dots, r_m\}$

é um conjunto de taxas de ativação (execuções por unidade de tempo) associadas com as transições de *PN*.

O tempo de espera em uma marcação M é dado por uma variável aleatória distribuída exponencialmente com média dada por (MARSAN; CONTE; BALBO, 1984):

$$\left[\sum_{i \in H} r_i \right]^{-1}$$

onde H é o conjunto de transições ativas na marcação.

A taxa de uma transição, a qual altera o estado da rede de M_i para M_j é (MARSAN; CONTE; BALBO, 1984):

$$\sum_{k \in H_{ij}} r_k$$

Onde H_{ij} é o conjunto de transições que levam do estado M_i ao estado M_j . Este conjunto de transições é geralmente formado por apenas uma transição.

5.2.2 REDES DE PETRI ESTOCÁSTICAS GENERALIZADAS

Nas redes de Petri estocástica generalizada *GSPN* (*Generalized Stochastic Petri Nets*), as transições da rede podem ser imediatas ou temporizadas (MARSAN; CONTE; BALBO, 1984). Transições imediatas são executadas imediatamente ao serem habilitadas. As transições temporizadas aguardam um certo tempo após sua habilitação, dado por uma função distribuída exponencialmente. As transições imediatas são representadas por barras finas e as temporizadas por barras largas. Sendo H o conjunto de transições ativas, e tendo apenas transições temporizadas, a transição temporizada $t_i \in H$ tem sua probabilidade de execução dada por (MARSAN; CONTE; BALBO, 1984):

$$\frac{r_i}{\sum_{k \in H_{ij}} r_k}$$

Havendo apenas uma transição imediata dentre as transições ativas, esta deve ser executada antes das demais. Caso haja mais transições imediatas, deve-se utilizar uma função de probabilidade em cada seleção de caminhos. A função de probabilidade utilizada nos *deadlines* é apresentada mais adiante. A função de probabilidade das demais seleções de caminhos presentes no processo deve ser fornecida pelo conhecedor do comportamento do fluxo. Se a probabilidade associada a uma transição imediata for zero, esta transição não pode ser executada. Nas *GSPN*, a quantidade de elementos no conjunto R dado pelo número de transições temporizadas na rede.

5.2.3 REDES DE PETRI ESTOCÁSTICAS GENERALIZADAS MODIFICADAS

Nesta pesquisa é necessária a representação das características do fluxo, bem como os recursos (executores de atividades) presentes neste fluxo. Para isto, decidiu-se estender as redes de Petri estocásticas generalizadas, para que possibilitem a representação das características presentes em *workflows*. Assim, uma rede *WfGSPN* (*Workflow - Generalized Stochastic Petri Nets*) que representa as informações necessárias para a geração de casos de teste pra *SGWf* é uma extensão de *GSPN*.

Nas *WfGSPN*, são depositados objetos nos lugares ao invés de *tokens*. Objetos podem ser de dois tipos distintos: **Objetos de Execução** e **Objetos de Recursos**. Objetos de execução correspondem aos objetos que representam as instâncias de *workflow*. Os objetos de recursos representam os recursos disponíveis na rede. **Lugares de Recursos** correspondem aos lugares onde se encontram apenas objetos de recursos. Cada lugar corresponde a uma determinada

classe; esta classe é a responsável pela criação das instâncias de recursos (objetos). Desta forma, tem-se que Objeto de execução, Objeto de recurso e Lugar de recurso são dados por:

Objeto de execução: Um objeto de execução é uma tupla $o = (ID_{Obj}, Data_{Obj})$ onde:

- ID_{Obj} é o identificador do objeto de execução;
- $Data_{Obj}$ é o conjunto de todos dados relevantes à instância de execução;

Objeto de Recurso: Um objeto de recurso é um par $or = (ID_{Rec}, N_{Rec})$ onde:

- ID_{Rec} é o identificador do recurso;
- N_{Rec} é o nome do recurso.

A classe do recurso é dada pela classe do Lugar de Recurso no qual o objeto está depositado. Sendo assim, um objeto de recurso pode apenas ser replicado em lugares de recurso que possuam a mesma classe na qual ele estava depositado.

As definições de lugar de recurso e classe basearam-se nas definições apresentadas por Ruiz (1995). Em (RUIZ, 1995), estas definições fazem parte de uma formalização em redes de Petri para a representação de atividades em aplicações de escritório.

Lugar de Recurso: Lugar de Recurso são os lugares de rede responsáveis pelo depósito dos objetos que representam os recursos necessários à execução do processo. Lugar de recurso consiste em uma tupla (ID_{RL}, L_{OR}, c) :

- ID_{LR} é o identificador do lugar de recurso;
- L_{OR} consiste em uma lista de objetos de recurso presentes no lugar. Assim, tem-se que $or \in L_{OR}$;
- c é a classe a qual pertence este lugar, sendo $c \in Cl$ e Cl são o conjunto de todas as classes do processo.

Classe: Uma classe c denota o tipo de recurso presente do lugar. Assim, se uma transição possui um lugar de recurso da classe *Setor Financeiro* como entrada representa que para a execução da atividade, representada pela transição, é necessário um executor do *Setor Financeiro*.

Lugares de recurso devem receber apenas objetos de recurso. Objetos de recurso só podem ser depositados em lugares de recurso de mesma classe, ou seja, um objeto de recurso proveniente de um lugar de recurso cuja classe seja x pode ser depositado apenas em outro lugar de recurso cuja classe também seja x .

Habilitação: para que uma transição possa ser executada deve haver objetos com mesmo ID (ID_{Obj}) em todos os seus lugares de entrada que não sejam lugares de recurso e com a quantidade suficiente exigida pelo peso de cada arco de entrada da transição.

Arcos restauradores (HEUSER, 1988): Atividades necessitam de executores (*Performers*) para serem executadas, porém estes executores não são alterados. Nestas ocorrências, utiliza-se arcos restauradores. Os arcos restauradores representam necessidades para ocorrência de uma transição que não geram efeitos nas marcações dos lugares de entrada. Um arco restaurador é representado por uma seta dupla.

Dadas as definições apresentadas por Aalst e Hee (2002) referentes à *WF-net*, adaptou-se algumas destas definições para este trabalho. Uma *WfGSPN* deve possuir as seguintes características (AALST; HEE, 2002):

- Exista um lugar de origem $i \in P$ tal que $\bullet i = \emptyset$;
- exista um lugar de destino $o \in P$ tal que $o \bullet = \emptyset$ e;
- todo trajeto $x \in P \cup T$ é um caminho, ou parte de um caminho, de i para o .

O estado inicial da rede (estado i) é dado pelo estado no qual os objetos de execução da rede se encontram apenas no lugar i , e apenas os lugares de recursos podem conter objetos. O estado final da rede (estado o) é dado pelo estado em que os objetos de execução se encontram apenas no lugar o , os demais objetos existentes na rede se encontram apenas nos lugares de recurso.

Tempo de habilitação: Tempo de habilitação de uma transição t é dado pelo tempo que t está habilitada consecutivamente. Assim sendo, estando a rede em uma marcação M por um tempo x e estando uma transição t habilitada nesta marcação e após este tempo uma transição t' é executada e a rede passa para a marcação M' por um tempo y e permanecendo habilitada a transição t nesta marcação, diz-se que o tempo de habilitação da transição t é dado por $x + y$. Para esta pesquisa é importante que se possa ter acesso ao último tempo de habilitação tido pelas transições. Assim sendo, caso uma transição não esteja habilitada, seu tempo de habilitação é o mesmo da última vez em que esteve habilitada. Ao ocorrer uma nova habilitação, seu tempo de habilitação é reiniciado.

Denota-se $Time_Active(t)$ o tempo de habilitação da transição t .

5.3 CONSIDERAÇÕES

Este capítulo tratou de apresentar a formalização de um dialeto de redes de Petri. Este dialeto possibilita a representação das características relevantes ao teste de *SGWf*. Através deste dialeto, é possível a representação dos padrões básicos, deadlines, além de possibilitar a representação e simulação da alocação dos recursos necessários à execução de cada atividade. Além disso, a execução da rede é baseada em informações estatísticas, possibilitando a geração de casos de teste estatísticos.

No capítulo seguinte, são descritos os passos para a geração da rede com base neste formalismo. Além disso, é apresentada a forma de geração dos casos de teste. Por fim, é descrito um exemplo.

6 APLICAÇÃO DA SOLUÇÃO

Neste capítulo, é ilustrada a aplicação desta solução vista no capítulo anterior. Para isto, são ilustrados os passos para a geração da rede. Feito isto, é apresentada a aplicação desta rede, ou seja, a geração dos casos de teste a partir dela. Por fim, é ilustrado um exemplo.

6.1 GERAÇÃO DA REDE

Através das informações referentes ao processo (informações estatísticas, padrões de fluxo, participantes e questões temporais), um modelo de uso é gerado utilizando o dialeto de rede de Petri apresentado. Exceto informações estatísticas quanto à execução do fluxo de controle, que devem ser fornecidas, todas as demais informações estão presentes na definição do modelo de *workflow*. Assim, com base nestas informações, a rede é gerada através das seguintes regras:

1. Cada **atividade** do *workflow* é transcrita como uma transição, um lugar e um arco de entrada ligando o lugar à transição, esta ocorrência é ilustrada pela Figura 8. Para a transição que representa atividade final deve ser gerado um lugar (*o*) e um arco de saída ligando ambos. O lugar de entrada da transição que representa a atividade inicial é o lugar *i*.

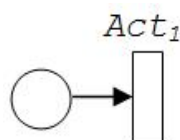


Figura 8: Representação de uma atividade genérica para teste de *SGWf* utilizando *WfGSPN*.

2. **Objeto de execução** é o objeto que representa a instância de execução de *workflow*. Este objeto é depositado no início do fluxo, ou seja, no lugar *i*. Neste objeto, devem estar as informações referentes ao processo assim como o identificador do objeto (instância de execução).
3. **Objeto de recurso** é uma instância da classe do lugar onde ele se encontra.

4. **AND-Split** é representado da mesma maneira que uma atividade genérica.
5. **AND-Join** é representado por uma transição e n lugares de entrada, sendo estes representantes dos n fluxos de controle que chegam até o **AND-Join**, a Figura 9 apresenta esta ocorrência.

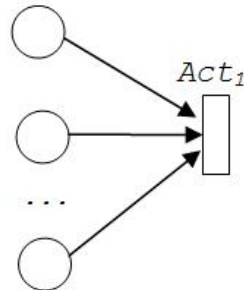


Figura 9: Representação de **AND-Join** para teste de *SGWf* utilizando *WfGSPN*.

6. **XOR-Split** é representado através de uma transição temporizada, responsável pela simulação do tempo gasto para a realização da atividade em questão. A seguir, ligadas ao lugar de saída desta transição, há n transições imediatas, responsáveis pela representação de cada fluxo a ser seguido, a Figura 10 apresenta esta ocorrência.

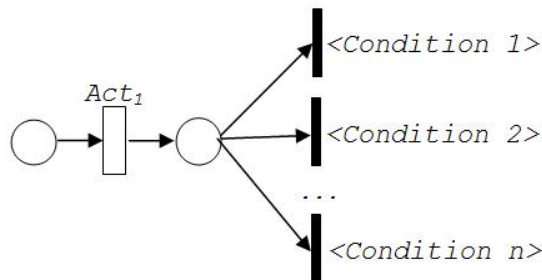


Figura 10: Representação de **XOR-Split** para teste de *SGWf* utilizando *WfGSPN*.

7. **XOR-Join** é representado da mesma forma que uma atividade genérica.
8. Cada **arco** que liga as atividades no documento de definição é chamado de transição. Para evitar confusão com as transições da rede de Petri, definiremos que as transições do documento de definição são chamadas de transições_WF. Desta forma, cada transição_WF é representada por um arco de saída da transição que representa a atividade de origem até o lugar de entrada da transição que representa a atividade de destino, esta representação pode ser vista na Figura 11.
9. O **executor** de cada atividade (*Performer*) deve ser representado com um objeto em um lugar de recurso, o qual deve ser ligado até a transição que representa a atividade por um arco restaurador. Isto se deve ao fato de que executores do *workflow* não são alterados

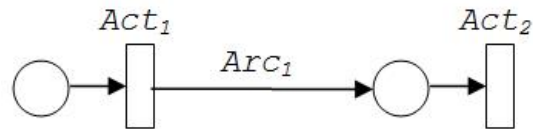


Figura 11: Representação de uma transição de *workflow* (transição_WF) para teste de *SGWf* utilizando *WfGSPN*.

durante a execução das atividades, apenas são necessários para que estas atividades sejam executadas. A classe deste lugar corresponde ao papel deste executor (por exemplo, setor financeiro em uma empresa de cobrança) e o objeto a uma instância desta classe. A Figura 12 apresenta esta representação.

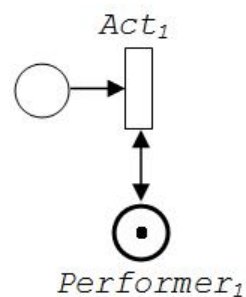


Figura 12: Representação de um executor (*Performer*) para teste de *SGWf* utilizando *WfGSPN*.

10. **Deadlines** são representados por uma transição que demanda tempo e uma seleção de caminhos entre os possíveis caminhos a serem seguidos. A transição que consome tempo ($Time_{Act_1}$) determina o tempo gasto para a execução desta ocorrência, ao passo que a seleção de caminhos determina o fluxo a ser seguido em função deste tempo. Assim, em:
- (A) atividades com *deadline* síncrono, a transição que representa a atividade em questão é executada (Act_1) ou então é seguido o fluxo correspondente ao *deadline* síncrono (DS);
 - (B) atividades com *deadline* assíncrono, apenas a transição que representa a atividade em questão é executada ou então esta transição é executada em paralelo com o fluxo correspondente ao *deadline* assíncrono (DA);
 - (C) atividades com *deadline* síncronos e assíncronos, apenas a transição que representa a atividade em questão é executada, ou esta transição é realizada em paralelo ao fluxo correspondente ao *deadline* assíncrono ou, então, apenas os fluxos correspondentes a ambos os *deadlines* são executados em paralelo (DA_{DS}). Considerando que T_{DS} e T_{DA} denotam, respectivamente, os tempos dos *deadlines* síncrono e assíncrono, a seguir são apresentadas as funções de probabilidade para as seleções de caminhos de cada tipo de *deadline*:

$$\begin{array}{l}
A \left\{ \begin{array}{l}
se(Time_Active(Time_Act_1) < T_DS), \\
Act_1 = 1, \\
DS = 0; \\
\\
se(Time_Active(Time_Act_1) \geq T_DS), \\
Act_1 = 0, \\
DS = 1.
\end{array} \right. \\
\\
B \left\{ \begin{array}{l}
se(Time_Active(Time_Act_1) < T_DA), \\
Act_1 = 1, \\
DA = 0; \\
\\
se(Time_Active(Time_Act_1) \geq T_DA), \\
Act_1 = 0, \\
DA = 1.
\end{array} \right. \\
\\
C \left\{ \begin{array}{l}
se(Time_Active(Time_Act_1) < T_DA), \\
Act_1 = 1, \\
DA = 0, \\
DA_DS = 0; \\
\\
se(Time_Active(Time_Act_1) \geq T_DA) \\
e(Time_Active(Time_Act_1) < T_DS), \\
Act_1 = 0, \\
DA = 1, \\
DA_DS = 0; \\
\\
se(Time_Active(Time_Act_1) \geq T_DS), \\
Act_1 = 0, \\
DA = 0, \\
DA_DS = 1.
\end{array} \right.
\end{array}$$

A Figura 13 apresenta atividades com as três ocorrências de *deadline* adotadas neste trabalho: **(A)** atividade com *deadline* síncrono, **(B)** atividade com *deadline* assíncrono e **(C)** atividade com ambos os *deadlines*.

Pode-se verificar que cada construção é iniciada com lugar(es) e terminada com transição(ões), possibilitando que seja simplificado o arranjo das construções. Isto se deve ao fato de que, basta a utilização de arcos, que correspondem diretamente às transições presentes em um modelo de *workflow*, entre as construções para a construção da rede.

Como esta pesquisa tem o objetivo de testar *workflows* do ponto de vista dos usuários, apenas as atividades executadas por usuários dentro do fluxo devem estar presentes nos casos de teste. As atividades realizadas por usuários correspondem às transições que possuem, dentre seus lugares de entrada, lugares de recurso.

O tempo em que cada atividade é executada está relacionado ao tempo de espera em cada estado da rede. Assim, quando uma atividade é concluída, a próxima atividade deve ser executada após o tempo de espera determinado pela rede.

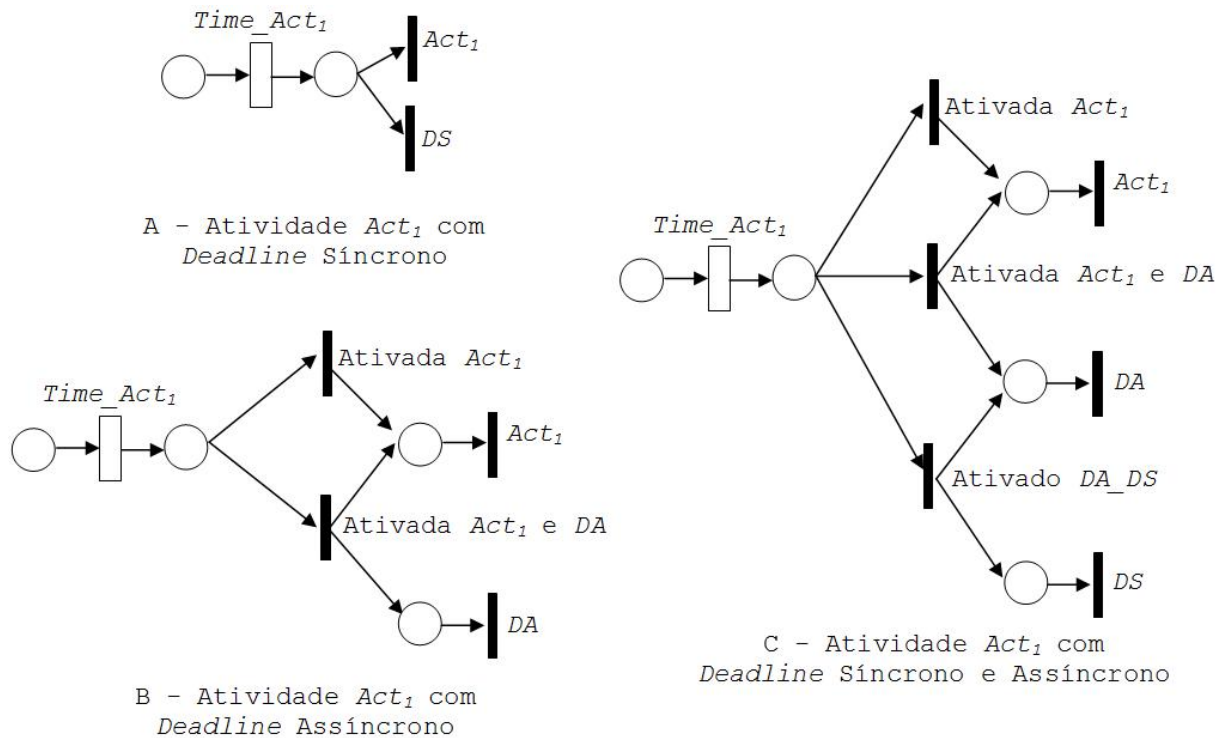


Figura 13: Representação de construções de *deadlines* para teste de *SGWf* utilizando *WfGSPN*.

6.2 APLICAÇÃO

Um caso de teste deve conter a seqüência de atividades a serem executadas, bem como em que tempo cada uma deve ser executada. Cada caso de teste está dividido em passos. Cada passo corresponde a uma atividade que um dado usuário deve executar. Além disso, para que um usuário possa executar uma dada atividade, ele deve realizar *login* no sistema.

O tempo de espera em cada marcação da rede é dado por uma variável aleatória distribuída exponencialmente. A cada estado da rede, o tempo de espera é determinado e então é definida a transição a ser executada. Isto se dá quando houver apenas transições temporizadas ativas. Assim, cada passo (*Step*) de um caso de teste é composto, respectivamente, pelo tempo de espera (*wait...*), pelo acesso do executor da atividade ao sistema (*Login with*), pela descrição de qual atividade deve ser executada (*Execute Activity*) e pelo encerramento do acesso ao sistema (*Logout with*). Na Figura 14, é apresentado um exemplo para esta ocorrência onde a atividade Act_1 deve aguardar um tempo de espera x , seguida à atividade Act_2 que deve aguardar um tempo de espera y . A Figura 15 ilustra um caso de teste para este exemplo.

Os casos de teste correspondem às atividades executadas pelo usuário. Desta forma, o tempo para a execução da atividade corresponde à soma dos tempos de espera de todas as transições executadas na rede desde a última atividade apresentada no caso de teste. Na Figura 16, é apresentado um exemplo onde a atividade Act_1 deve aguardar um tempo de espera x , em

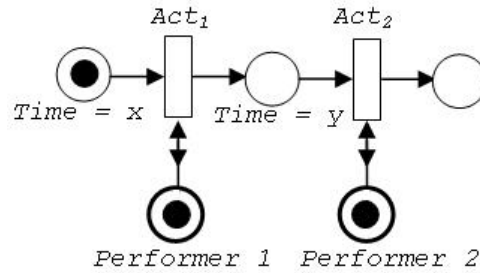


Figura 14: Exemplo 1 de tempo de espera.

```

=====
Test Case: Exemplo 1
Step: 1
Wait ... x
Login with: Performer 1 in Process Process1 (Package Package1)
Execute Activity: Act1
Logout with: Performer 1

Step: 2
Wait ... y
Login with: Performer 2 in Process Process1 (Package Package1)
Execute Activity: Act2
Logout with: Performer 2
=====

```

Figura 15: Exemplo de caso de teste para seqüência de atividades do exemplo 1.

seguida atividade Act_3 , próxima atividade do caso de teste, deve aguardar um tempo de espera $y + z$, pois a atividade Act_2 não requer a interação do usuário, entretanto consome tempo. A Figura 17 ilustra um caso de teste para este exemplo.

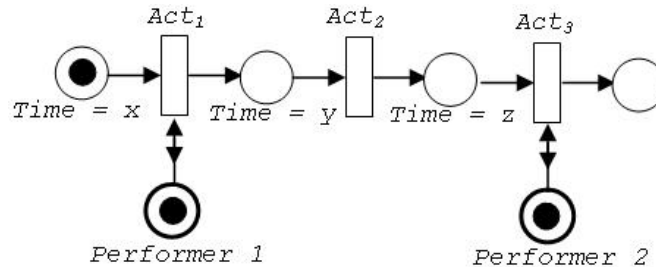


Figura 16: Exemplo 2 de tempo de espera.

```

=====
Test Case: Exemplo 2
Step: 1
Wait ... x
Login with: Performer 1 in Process Process1 (Package Package1)
Execute Activity: Act1
Logout with: Performer 1

Step: 2
Wait ... y + z
Login with: Performer 2 in Process Process1 (Package Package1)
Execute Activity: Act3
Logout with: Performer 2
=====

```

Figura 17: Exemplo de caso de teste para seqüência de atividades do exemplo 2.

Os *deadlines*, apesar de possuírem uma seleção de caminhos em sua representação, não são tratados como tal. Esta seleção de caminhos utiliza a função de probabilidade específica apresentada na formalização. Dada uma atividade Act_1 com um *deadline* síncrono de 4 unidades de

tempos: ocorrendo o *deadline*, é necessário que se execute a atividade *Act₂*; se não, ocorrer, o fluxo deve seguir para a atividade *Act₃*. Este exemplo é representado na Figura 18.

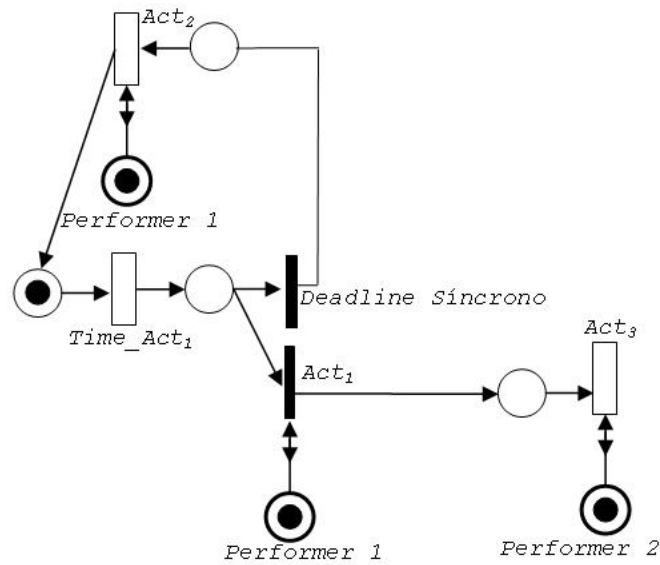


Figura 18: Exemplo de *deadline* síncrono.

Supondo que a transição *Time_Act₁*, que corresponde à demanda de tempo para a execução da atividade *Act₁*, levou 6 unidades de tempo para ser executada, deve ser executado o fluxo correspondente ao *deadline* desta atividade, ou seja, a atividade *Act₂*, visto que o *deadline* é de 4 unidades de tempo. Após a execução da transição *Act₂*, que demandou 2 unidades de tempo por exemplo, o fluxo retorna para a execução da transição *Time_Act₁*. Desta vez, supõe-se que a transição levou 3 unidades de tempo para ser executada. Com isso, é executada a transição correspondente ao fluxo padrão (*Act₁*), para então ser habilitada a próxima transição (*Act₃*), a qual demorou 3 unidades de tempo para ser executada. Se a atividade que possui o *deadline* necessita de um participante para sua execução este deve estar representado na atividade imediata que representa a tomada de decisão pela execução da atividade em questão. A Figura 19 apresenta um exemplo de caso de teste para este caso.

```

=====
Test Case: Exemplo Deadline
Step: 1
wait ... 8
Login with: Performer 1 in Process Process1 (Package Package1)
Execute Activity: Act2
Logout with: Performer 1

Step: 2
wait ... 3
Login with: Performer 1 in Process Process1 (Package Package1)
Execute Activity: Act1
Logout with: Performer 1

Step: 3
wait ... 3
Login with: Performer 2 in Process Process1 (Package Package1)
Execute Activity: Act3
Logout with: Performer 2
=====

```

Figura 19: Exemplo de casos de teste para *deadlines*.

6.3 EXEMPLO

Para a geração de casos de teste, utilizou-se o exemplo de uma organização de festa. Este exemplo foi extraído de Aalst e Hee (2002). Este exemplo foi simplificado para uma melhor compreensão. Foram removidos os fluxos referentes à rejeição ao convite e à avaliação da banda, além disso o *deadline* para a resposta da banda faz com o fluxo retorne para o envio de um novo convite. Neste exemplo, um cliente registra a solicitação para a organização de uma festa. A festa necessita de um local e de música. O local da festa pode ser *indoor*, com o aluguel de um salão, ou *outdoor*, com o aluguel de um terreno e uma tenda. Este terreno pode, ou não, necessitar de uma permissão para a execução da festa; caso necessite, deve-se obtê-la. A música da festa pode ser som mecânico ou uma banda. Para uma festa com som mecânico, faz-se necessária à contratação de um aparelho. Caso a decisão seja por banda, deve-se enviar o aviso de show à banda. A banda deve confirmar o interesse em realizar o *show*, para então ser contratada. Caso não seja confirmado o *show*, deve-se enviar um novo aviso. Após ser obtido o local e a música para a festa, deve-se conseguir a comida e a bebida. Por fim, é feita uma visita ao local da festa para verificar se está tudo como o planejado, para então emitir a nota fiscal. A Figura 20 apresenta a representação deste processo. Nesta modelagem, as atividades em verde correspondem às atividades executadas pelos usuários, enquanto que as atividades vermelhas correspondem às atividades de rota, executadas pelo Motor de *Workflow*. Esta representação foi realizada utilizando a ferramenta *open source* de modelagem de *workflow* JaWE (ENHYDRA, 2007).

Para a execução da rede utilizou-se o *framework* JFern (NOWOSTAWSKI, 2003). Para que a rede executasse como esperado foram necessárias algumas extensões, para: **(1)** que a seleção das transições a serem executadas fosse baseada nas taxas de cada uma delas; **(2)** que se pudesse simular o tempo de espera de cada transição para sua execução; **(3)** que fossem tratados os Lugares de Recurso e Objetos de Recurso, assim como Objetos de Execução; **(4)** que fosse respeitada a definição de habilitação; **(5)** que, nos *deadlines*, as funções de probabilidades fossem atualizadas em função do tempo de habilitação; e **(6)** que o conjunto de casos de teste fosse gerado. Na Figura 21, é apresentada a modelagem da rede.

O conjunto R , para este exemplo, corresponde à Tabela 2. Neste exemplo, utiliza-se o tempo de 2,5 para o *deadline* síncrono. O *deadline* e as seleções de caminhos possuem as seguintes funções de probabilidades:

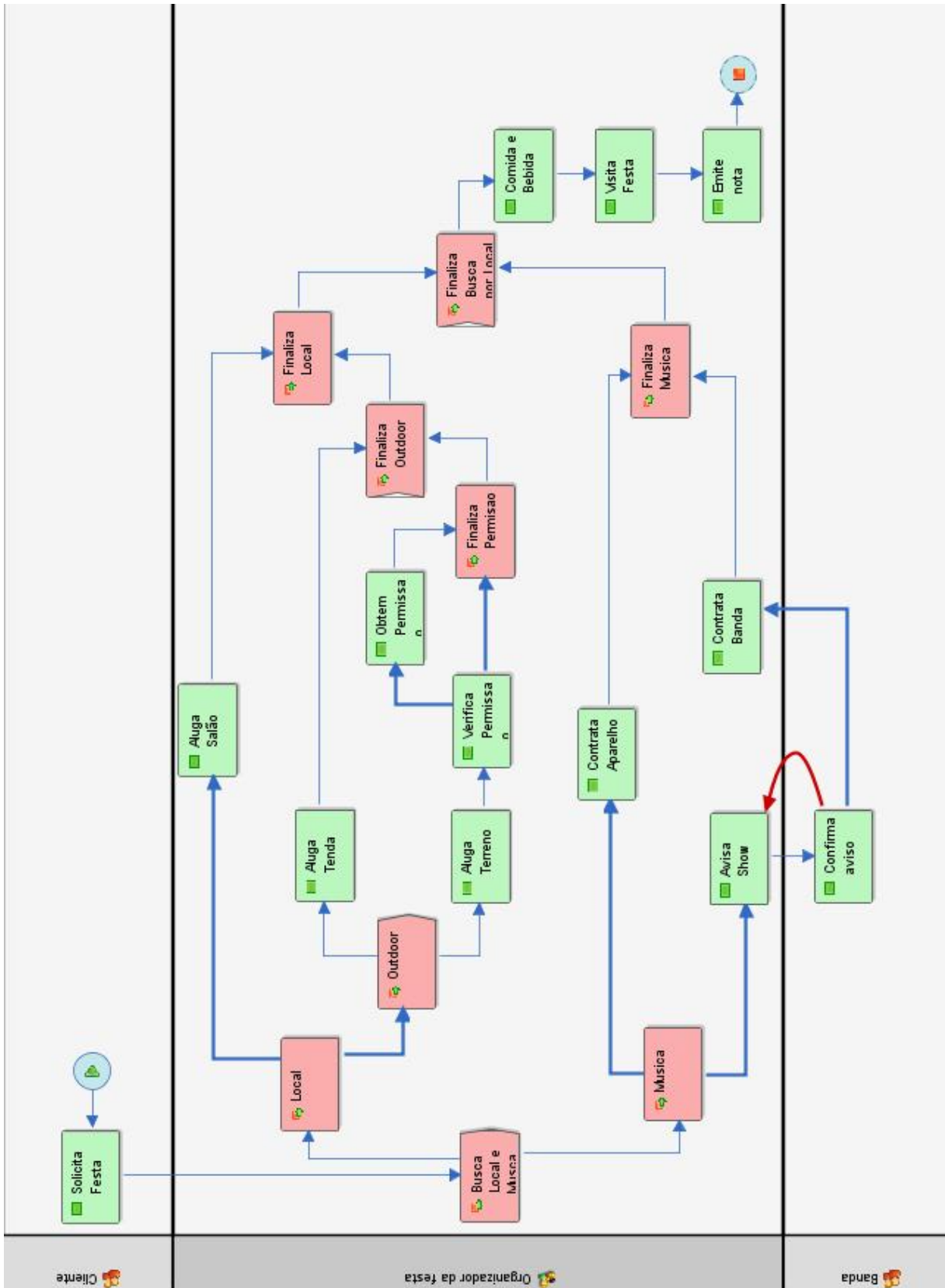


Figura 20: Representação do processo festa.

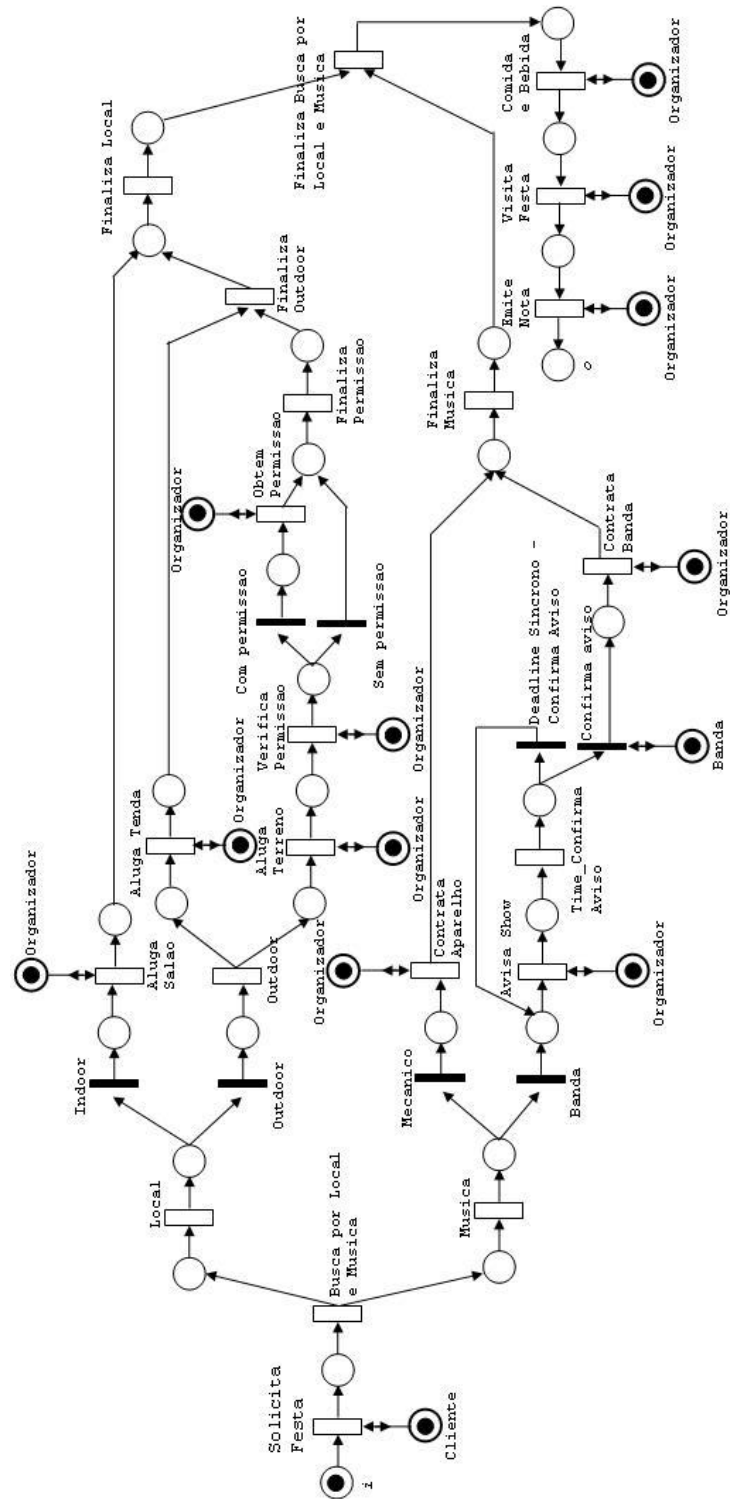


Figura 21: Rede gerada para a geração dos casos de teste.

$$\begin{aligned}
\text{ConfirmaAviso} & \left\{ \begin{array}{l} \text{se}(\text{Time_Active}(\text{Time_ConfirmaAviso}) < 2,5), \\ \text{ConfirmaAviso} = 1, \\ \text{deadline_sincrono} = 0; \\ \\ \text{se}(\text{Time_Active}(\text{Time_ConfirmaAviso}) \geq 2,5), \\ \text{ConfirmaAviso} = 0, \\ \text{deadline_sincrono} = 1. \end{array} \right. \\
\text{Local} & \left\{ \begin{array}{l} \text{Indoor} = 0, 1 \\ \text{Outdoor} = 0, 9 \end{array} \right. \\
\text{Musica} & \left\{ \begin{array}{l} \text{Mecanico} = 0, 3 \\ \text{Banda} = 0, 7 \end{array} \right. \\
\text{VerificaPermissao} & \left\{ \begin{array}{l} \text{ComPermissao} = 0, 6 \\ \text{SemPermissao} = 0, 4 \end{array} \right.
\end{aligned}$$

Tabela 2: Taxas das transições da rede.

T	R	T	R
Solicita Festa	0.3	Verifica Permissao	0.4
Busca por Local e Musica	0.4	Obtem Permissao	0.4
Local	0.3	Contrata Banda	0.4
Música	0.3	Finaliza Permissao	0.3
Aluga Salão	0.8	Finaliza Outdoor	0.3
Outdoor	0.4	Finaliza Musica	0.3
Contrata Aparelho	0.8	Finaliza Local	0.3
Avisa Show	0.4	Finaliza Busca por Local e Musica	0.4
Aluga Tenda	0.4	Comida e Bebida	0.3
Aluga Terreno	0.4	Visita Festa	0.3
Time_Confirma Aviso	0.4	Emite Nota	0.3

As figuras 22, 23, 24 e 25 apresentam amostras de casos de teste gerados e ilustração dos caminhos percorridos por estes casos de teste. Nesta geração de casos de teste, são descritas a seqüência de atividades que cada tipo de usuário (*Performer*) deve executar, ou seja, é apresentada toda a seqüência de interações dos usuários com o *workflow*. Além disso, a intenção, neste exemplo, é apresentar diversos casos de teste que representam execuções independentes. Desta forma, foi executada a mesma rede n vezes com apenas um objeto de execução no *place* i . Na identificação de qual atividade deve ser executada, apresenta-se o ID da atividade seguido do seu nome entre parênteses. Além disso, é fornecido o tempo de espera para a execução de cada atividade.

6.4 CONSIDERAÇÕES

Neste capítulo, foram apresentados os passos para a criação da rede a partir de uma definição de fluxo. Por fim, foi ilustrado um exemplo. Neste exemplo, foram apresentados os casos de teste gerados a partir de uma definição de processo.

```

=====
Test Case: 6
Step: 1
Wait ... 18,36
Login with: Cliente in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id11 (Solicita Festa).
Logout with: Cliente

Step: 2
Wait ... 30,86
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id41 (Aluga Salao).
Logout with: Organizador

Step: 3
Wait ... 27,75
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id119 (Avisa Show).
Logout with: Organizador

Step: 4
Wait ... 22,67
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id119 (Avisa Show).
Logout with: Organizador

Step: 5
Wait ... 0,02
Login with: Banda in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id235 (Confirma Aviso).
Logout with: Banda

Step: 6
Wait ... 0,02
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id136 (Contrata Banda).
Logout with: Organizador

Step: 7
Wait ... 0,03
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id150 (Comida e Bebida).
Logout with: Organizador

Step: 8
Wait ... 8,81
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id152 (Visita Festa).
Logout with: Organizador

Step: 9
Wait ... 8,81
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id159 (Emite Nota).
Logout with: Organizador
=====

```

Figura 22: Exemplo 1 de caso de teste gerado (*Test Case 6*).

Com este capítulo, verifica-se que a formalização apresentada possibilita a representação das características relevantes ao teste de *SGWf*. As representações mostradas neste capítulo exploram os padrões básicos, além de definir a representação de executores das atividades, *deadlines*, dados relevantes do *workflow*, participantes presentes no *workflow*, identificadores de pacotes e nomes dos processos.

No âmbito do *CPTS* (Centro de Pesquisas em Teste de *Software*), que é uma parceria entre a *HP* (*Hewlett-Packard*) e a *PUCRS* (Pontifícia Universidade Católica do Rio Grande do Sul) com o objetivo de desenvolver inovações tecnológicas e formar pesquisadores no campo de Testes de *Software*, analisou-se a possibilidade de adaptar esta técnica para a geração de casos de teste para o teste de desempenho. No próximo capítulo, este estudo é apresentado.

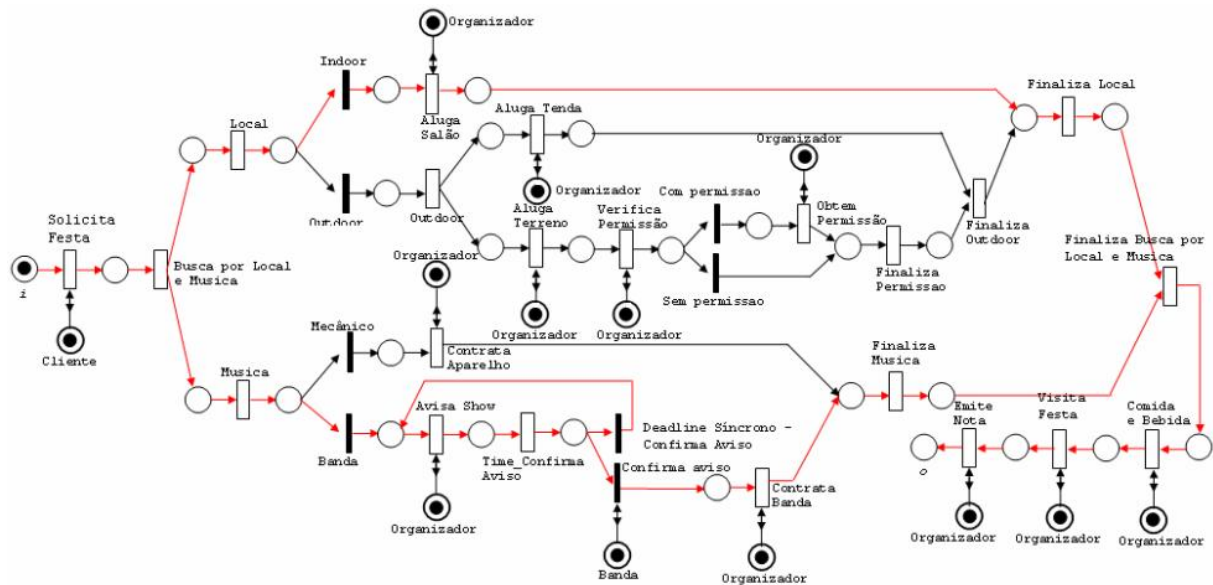


Figura 23: Caminho percorrido pelo exemplo 1 (*Test Case 6*).

```

=====
Test Case: 7
Step: 1
wait ... 8,87
Login with: Cliente in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id11 (Solicita Festa).
Logout with: cliente

Step: 2
wait ... 15,57
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id41 (Aluga Salao).
Logout with: organizador

Step: 3
wait ... 11,94
Login with: organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id112 (Contrata Aparelho).
Logout with: organizador

Step: 4
wait ... 33,90
Login with: organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id150 (Comida e Bebida).
Logout with: organizador

Step: 5
wait ... 9,42
Login with: Organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id152 (Visita Festa).
Logout with: organizador

Step: 6
wait ... 9,50
Login with: organizador in ProcessovamosTerFesta (Package Pacote1)
Execute Activity: id159 (Emite Nota).
Logout with: organizador
=====

```

Figura 24: Exemplo 2 de caso de teste gerado (*Test Case 7*).

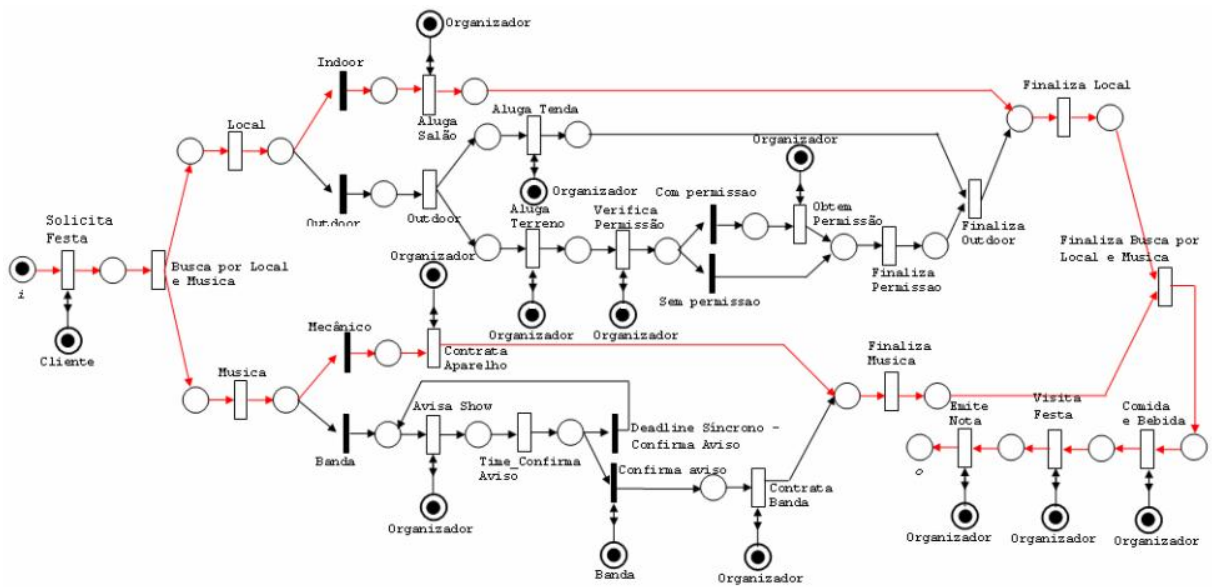


Figura 25: Caminho percorrido pelo exemplo 2 (Test Case 7).

7 SOLUÇÃO APLICADA AO TESTE DE DESEMPENHO

Neste capítulo, é apresentada uma estratégia para modelagem de desempenho adaptada à geração de cenários do teste de desempenho. Este estudo foi estimulado pela necessidade de se gerar cenários de testes realistas para o teste de desempenho, para isso sugere-se testes estatísticos. Além disso, neste estudo, há a necessidade de que a alocação de recursos seja tratada, para que não haja equívocos caso determinado recurso não esteja disponível e o tempo de resposta se torne maior.

Com isso, verificou-se os requisitos para a geração de um modelo de uso para a criação de cenários de teste de desempenho eram semelhantes aos requisitos para o teste de sistemas de gerenciamento de *workflows*. Como em *SGWf*, o teste de desempenho tem a preocupação com a validação temporal e com a utilização de informações estatísticas para a determinação dos casos de teste. Além disso, a simulação da concorrência pelos recursos disponíveis ao teste, que em *workflows* correspondem aos executores atrelados a cada atividade, também é relevante ao teste de desempenho. Assim, o estudo apresentado neste capítulo trata de apresentar a metodologia utilizada para, a partir de informações referentes ao sistema, gerar um modelo de uso baseado no dialeto apresentado neste trabalho, ver Sessão 5.2. Em (OLIVEIRA et al., 2007), são apresentadas mais informações sobre este estudo.

Como o objetivo é simular as atividades executadas pelo usuário, a fim de verificar o comportamento apresentado pela aplicação, este estudo utiliza diagramas *UML* (*Unified Modeling Language*). Os diagramas de atividades possibilitam a modelagem das atividades executadas pelos usuários e pela própria aplicação. Desta forma, é possível simular a interação dos usuários e verificar se o sistema executou a ação esperada. Utilizou-se, também, diagramas de casos de uso. Assim sendo, cada caso de uso possui um diagrama de atividade atrelado, que descreve seu comportamento esperado. Através da análise do diagrama, é possível determinar seqüências de atividades que devem ser executadas para que se tenham casos de teste.

7.1 UML SPT PROFILE

O *UML SPT Profile* (*UML Profile for Schedulability, Performance and Time*) é uma extensão da *UML* para modelar exigências de desempenho das aplicações. Sua finalidade é definir padrões de paradigmas de uso para modelagem de sistemas de recursos, tempo, escalabilidade, e desempenho relacionados a aspectos de sistemas de tempo real (OMG, 2005).

Através do *UML SPT Profile*, é possível informar algumas características que descrevem o comportamento de carga e tempo do sistema, a probabilidade de uma determinada funcionalidade ocorrer, quanto tempo esta funcionalidade deveria gastar, além dos recursos presentes no sistema (OMG, 2005).

7.2 TRADUÇÃO DE UML SPT PROFILE PARA REDES DE PETRI

Algumas regras para a realização da tradução de um diagrama *UML* em uma rede Petri baseada no dialeto formalizado, ver Sessão 5.2, devem ser seguidas. Em algumas representações, utilizou-se um quadrantes pontilhados para dar destaque ao que se deseja ilustrar, estes quadrantes não fazem parte da rede. As representações são descritas a seguir:

- **Objeto de execução:** são depositados no início do fluxo, no lugar i . Estes objetos contêm as seguintes informações:
 - Identificador da instância que representa cada usuário que acessará o sistema;
 - Conjunto com os dados a serem utilizados durante o teste (IP, argumentos, etc.).
- **PApopulation:** número de usuários que devem acessar o sistema corresponde a quantos objetos existirão no lugar i . Cada objeto presente no lugar i corresponde a um usuário que acessará o sistema. Ao final do fluxo, a última transição tem seu arco de saída com entrada do lugar o .
- **Recursos:** os recursos disponíveis para o teste devem ser representados na rede através de objetos de recursos nos respectivos lugares de recurso. Desta forma, a marcação inicial da rede M' é dada pelos objetos de recursos nos seus respectivos lugares de recurso e pelos objetos que representam as instâncias de execução no lugar i .
- **PAoccurrence:** é representado através de uma transição temporizada cujo lugar de entrada é o lugar inicial (lugar i). Como lugar de saída, tem-se o lugar que representa o

início do fluxo descrito no diagrama. Assim, esta transição tem como objetivo lançar as diversas instâncias de execução (casos de teste) na rede. A taxa da transição corresponde ao $PA_{occurrence}$. A representação desta construção é apresentada na Figura 26.

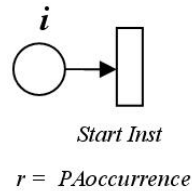


Figura 26: Representação de início do fluxo com o $PA_{occurrence}$ para teste de desempenho utilizando *WfGSPN*.

- **Atividade:** cada atividade é representada por três transições imediatas e lugares entre estas transições: uma transição representa a disponibilidade de execução da atividade (transição *Start*); outra transição imediata representa o início da execução da atividade (transição *Run*); e a outra transição representa o final da execução da atividade (transição *End*). Esta construção é apresentada pela Figura 27.

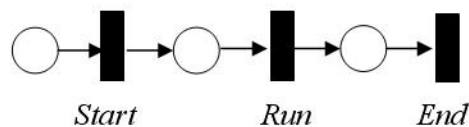


Figura 27: Representação de uma atividade genérica para teste de desempenho utilizando *WfGSPN*.

- **$PA_{extDelay}$:** é o tempo que um usuário espera para realizar uma nova requisição, também chamada *think time*, ou seja, é o tempo esperado entre a disponibilidade de uma execução e o real início da execução desta atividade. É representado por uma transição temporizada ($r = 1/PA_{extDelay}$) antes da transição imediata *Run*, a qual representa o início da execução da atividade em questão, Figura 28.

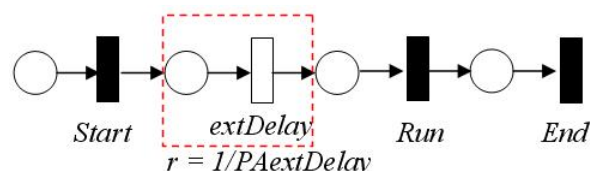


Figura 28: Representação de uma atividade com *think time* para teste de desempenho utilizando *WfGSPN*.

- **$PA_{respTime}$:** indica o tempo de resposta esperado, ou seja, o tempo que determinada atividade leva para responder. Desta forma, corresponde a uma transição temporizada ($r = 1/PA_{respTime}$) após a execução da transição imediata *Run*, Figura 29.

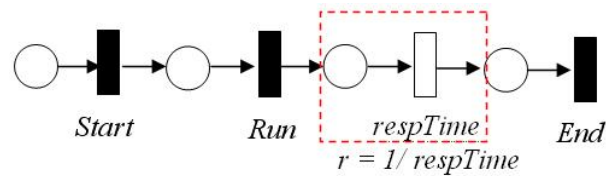


Figura 29: Representação de uma atividade com tempo de resposta para teste de desempenho utilizando *WfGSPN*.

- **Recursos:** representam os recursos necessários a execução de uma dada atividade. Desta forma, tem-se que um recurso é alocado quando a atividade que o necessita inicia sua execução (o início da execução é dado pela transição *Run*). Assim sendo, o recurso passa a estar disponível após o término da atividade que o alocou (o final da execução é dado pela transição *End*), Figura 30. Há a possibilidade de uma dada atividade alocar um recurso que será disponibilizado novamente apenas após a execução da próxima atividade, esta construção é apresentada na Figura 31.

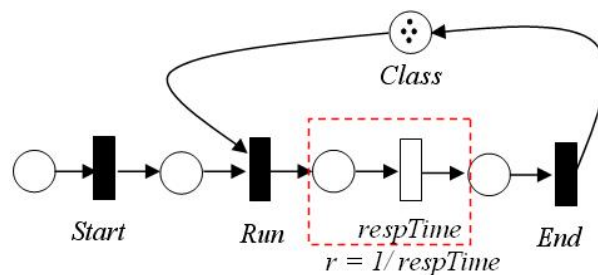


Figura 30: Representação 1 para alocação de recurso para teste de desempenho utilizando *WfGSPN*.

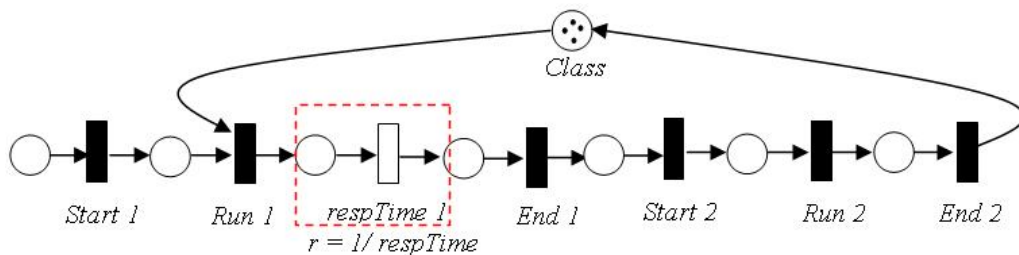


Figura 31: Representação 2 para alocação de recurso para teste de desempenho utilizando *WfGSPN*.

- **P_{Ap}rob:** indica a probabilidade da requisição ocorrer. É representado por uma transição imediata contendo a probabilidade do fluxo ser seguido. As taxas presentes nas transições que representam as atividades possuem o valor 1, pois não há possibilidade de seguir mais de um caminho dentro de uma única atividade.
- **Seleção de Caminhos (OR Split):** corresponde aos pontos em que o fluxo deve seguir por um dos n caminhos possíveis. Como estas ocorrências consistem apenas na seleção

de qual caminho a seguir, utilizou-se apenas transições que representam a disponibilidade de seleção (transição *Start*) e escolha do caminho (transição *End*). A taxa da transição (r) que representa a escolha do caminho corresponde ao $PAprob$. Esta construção é apresentada pela Figura 32.

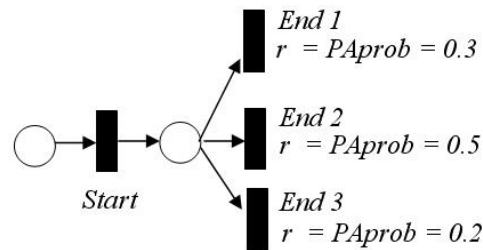


Figura 32: Representação de *XOR Split* para teste de desempenho utilizando *WfGSPN*.

- **União Simples (*OR Join*):** corresponde aos pontos do fluxo onde dois ou mais sub-fluxos se unem sem que haja nenhum tipo de sincronização. Como estas ocorrências tem apenas o objetivo de fazer com que sub-fluxos passem a seguir por um único sub-fluxo, utilizou-se apenas transições que representam a chegada de um sub-fluxo (transição *Start*) e que representam a união em um único fluxo (transição *End*). Esta construção é apresentada pela Figura 33.

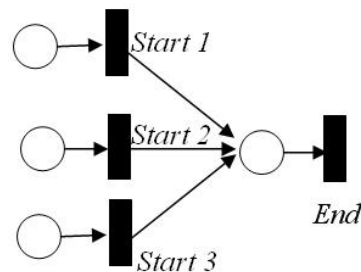


Figura 33: Representação de *XOR Join* para teste de desempenho utilizando *WfGSPN*.

- **Separação Paralela (*AND Split*):** consiste em um ponto onde um único fluxo se divide em múltiplos fluxos de controle que podem ser executadas paralelamente. Para a representação desta ocorrência, utilizou-se uma transição imediata que simboliza o início da separação paralela (transição *Start*), esta transição ativa os fluxos que devem ser executados paralelamente. Cada fluxo paralelo possui uma transição imediata (transição *End*) determinando que a separação foi realizada. Esta construção é apresentada pela Figura 34.
- **Sincronização (*AND Join*):** é um ponto no processo onde diferentes ramos de atividades paralelas convergem em um único fluxo de controle. Para que a execução do fluxo prossiga, é necessário que todos os fluxos paralelos que convergem para a sincronização

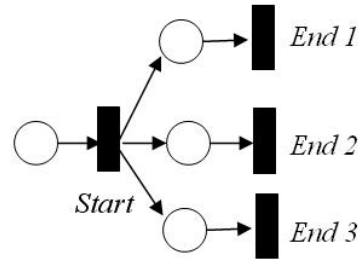


Figura 34: Representação de *AND Split* para teste de desempenho utilizando *WfGSPN*.

tenham sido completados. Uma transição imediata (transição *Start*) representa a possibilidade de sincronização, ou seja, quando esta transição está ativa, significa que é possível a sincronização dos fluxos. Outra transição imediata (transição *End*) corresponde ao final da sincronização. Esta construção é apresentada pela Figura 35.

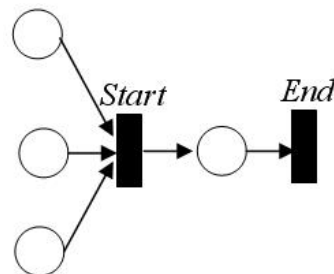


Figura 35: Representação de *AND Join* para teste de desempenho utilizando *WfGSPN*.

- Caso de uso:** a rede simula que um dado usuário ao acessar o sistema pode escolher, de acordo com a probabilidade, qual caso de uso ele irá executar. Com isso, após o objeto de execução ter sua execução iniciada através da transição *Start Inst*, é realizada uma seleção de caminhos, baseada nas probabilidades representadas nos casos de uso. Esta seleção de caminhos representa a escolha por um dos casos de uso presentes no sistema. Após a escolha pelos casos de uso, cada uma das transições *End* da seleção de caminhos deve ser ligada ao respectivo lugar de entrada da transição *Start* relativo à primeira atividade do diagrama de atividades que descreve o caso de uso em questão. No final do fluxo, é utilizada uma construção de união simples para que os sub-fluxos abertos pela escolha do caso de uso convertam para um único lugar final (*o*). Esta construção é apresentada pela Figura 36.

Pode-se verificar que cada construção é iniciada por lugar(es) e terminada por transição(ões), possibilitando que seja simplificado o arranjo das construções. Isto se deve ao fato de que, basta a utilização de arcos entre as construções para a construção da rede, sem que seja necessário tratamentos especiais para ligar as construções. Desta forma, a solução encontrada nesta pesquisa apresenta uma tradução de *UML SPT Profile* em redes de Petri que pode facilmente ser automa-



Figura 36: Representação de um caso de uso para teste de desempenho utilizando *WfGSPN*.

tizada, pois a forma de construção da rede se dá de maneira modular, para a construção da rede basta a inserção dos módulos (construções), ligando-os através de arcos.

7.3 UPPERT

O objetivo do estudo descrito neste capítulo é, a partir de um arquivo que descreve um diagrama *UML*, gerar casos de teste para teste de desempenho, se preocupando com as diversas características existentes no teste de desempenho: simulação do *delay* que o usuário tem ao executar uma dada atividade e verificação do tempo de resposta do sistema.

As redes de Petri proporcionam um bom trabalho com informações estatísticas, possibilitando a modelagem das atividades do usuário do sistema. Desta forma, as redes de Petri passam a ser o mecanismo de geração casos de teste, cada execução da rede simula a execução de um caso de teste.

O *UML SPT Profile* funciona como uma interface ao usuário interessado em modelar um sistema para a execução de testes de desempenho para aplicações WEB. Esta solução possibilita que o usuário utilize uma linguagem bastante usual para a representação da aplicação a ser testada. As informações representadas em um diagrama *UML* são modeladas em uma rede de Petri, ficando a cargo da execução da rede a geração de casos de teste. Assim, tem-se que o esquema desta solução consiste na entrada de um diagrama de casos de uso e atividades, estes diagramas são modelados em uma rede de Petri. A partir desta rede, são gerados casos de teste, para então ser gerado o *script* para a execução automática do teste. Este esquema é representado pela Figura 37. Esta solução possibilita que a geração de casos de teste se dê de forma independente da geração do *script* para a ferramenta alvo.

Este trabalho buscou o auxílio apenas de ferramentas *open source*, que possibilitem o teste de desempenho em aplicações *Web*. Desta forma, escolheu-se a ferramenta JMeter para a execução dos casos de teste. A ferramenta JMeter possibilita que seja configurada toda a realização dos testes a partir de um arquivo de entrada.

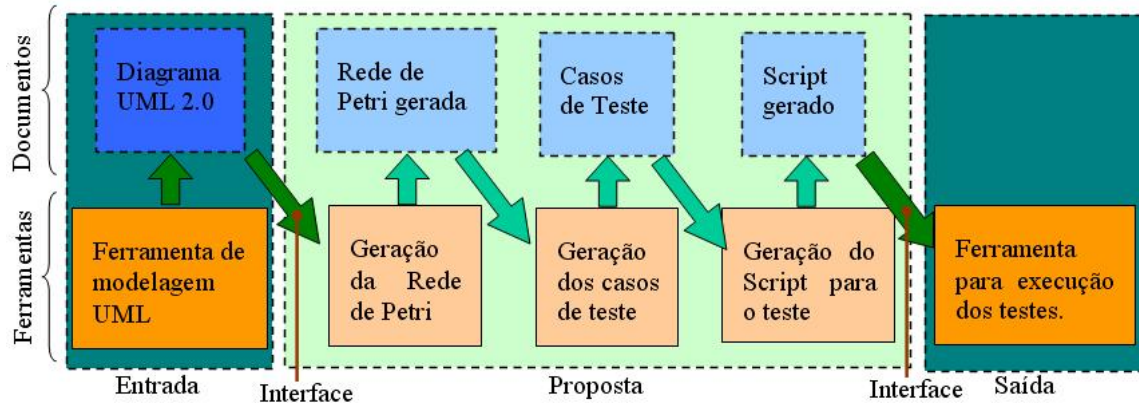


Figura 37: Arquitetura UpperT.

7.4 EXEMPLO DE UMA REDE GERADA PARA O TESTE DE DESEMPENHO

As figuras 38 e 39 apresentam um exemplo simples de modelos de entrada um usuário de *FTP* (*File Transfer Protocol*). Neste exemplo, definiu-se que somente 3 usuários podem acessar o servidor de FTP simultaneamente, a fim de simular a disputa por recursos. Os casos de uso apresentados são "List FTP Files" e "Get File", ambos com probabilidade 0.5. A Figura 39 mostra os diagramas correspondentes da atividade. Considerando que neste exemplo o objetivo é submeter o sistema a diversas requisições, gerou-se uma rede com n objetos de execução no *place i*. Com isso, esta rede gera um caso de teste que simula diversas execuções em paralelo do fluxo modelado. A Figura 40 apresenta a *WfGSPN* extraída do modelo.

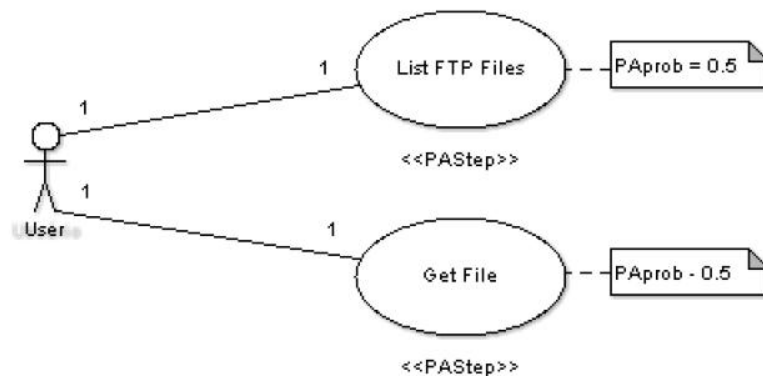


Figura 38: Diagrama de casos de uso de um servidor de FTP.

7.5 CONSIDERAÇÕES

Este capítulo apresenta uma metodologia para a geração de cenários para o teste de desempenho. Estes cenários são gerados com base em informações estatísticas e consideram o tempo gasto pelo usuário e pelo sistema para responder às suas solicitações. Além disso, tratam da

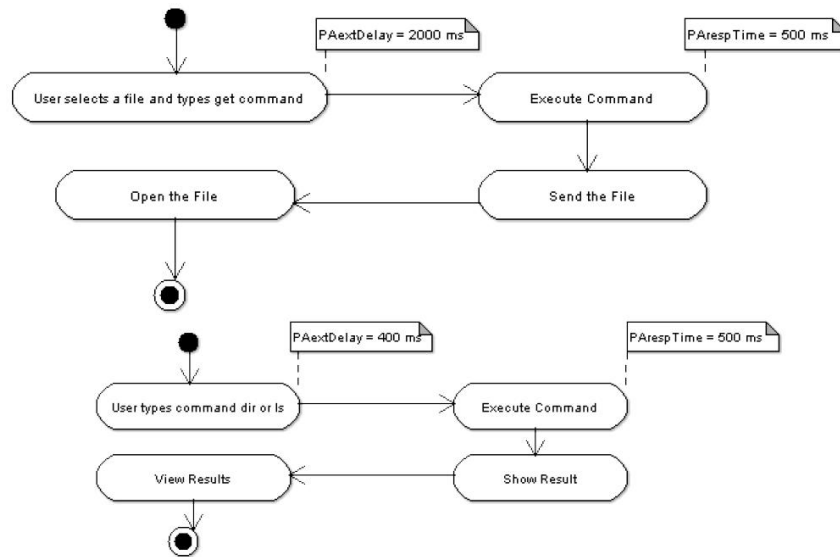


Figura 39: Diagramas de atividades de um servidor de FTP.

alocações dos recursos necessários à execução do sistema, evitando que falsos problemas sejam evidenciados. Além do mais, este dialeto possibilita a representação e simulação de diversos usuários acessando o sistema simultaneamente. Com isso, tem-se conjuntos de cenários que condizem com a real utilização do *software*.

Para isso, diagramas *UML* são utilizados para a descrição das atividades a serem executadas no sistema. A partir destes diagramas, é modelada uma rede baseada no dialeto *WfGSPN*. A execução desta rede gera casos de teste. A partir destes casos de teste é desenvolvido o *script* de teste para ser carregado pela ferramenta *JMeter*.

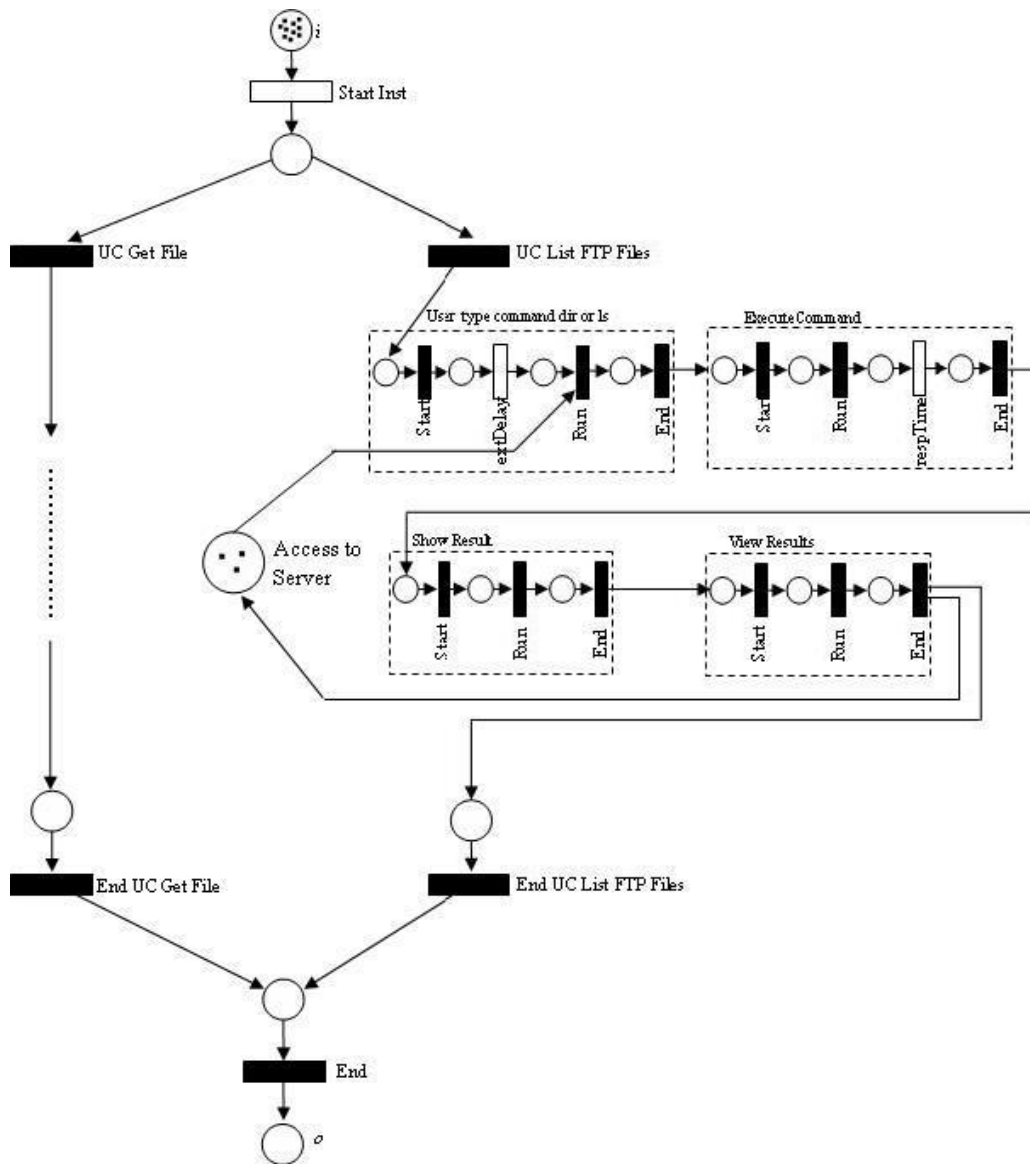


Figura 40: Rede de Petri gerada pelo UpperT.

8 TRABALHOS RELACIONADOS

A seleção de alguns artigos originados de uma pesquisa referente a assuntos que tratam de *workflows*, redes de Petri e/ou teste de *software*, servindo de base para um estudo dos caminhos já trilhados em torno do objetivo deste trabalho, é apresentada neste capítulo.

8.1 WORKFLOWS REPRESENTADOS ATRAVÉS DE REDES DE PETRI

Primeiramente, foi verificada a existência de diversas pesquisas baseadas em redes de Petri. Marsan et al. (1987) utiliza as redes para a modelagem da arquitetura de um multiprocessador. Neste estudo, são apresentadas redes que possibilitam a representação de características temporais, juntamente com informações estatísticas referentes ao fluxo de controle. Yamaguchi et al. (2000) trabalha com uma extensão desta rede em uma especificação distribuída com alocação de recursos. Em (HAMADI; BENATALLAH, 2003), é apresentado um modelo baseado em redes de Petri para composição de serviços *web*.

Aalst (1998) apresenta diversas vantagens em utilizar redes de Petri para a representação de fluxo de processos. Com isso, alguns trabalhos relacionados a *workflows* e redes de Petri foram selecionados. Em (AALST; HEE; HOUBEN, 1994) e (PADUA et al., 2004), verificou-se representação de *workflows* através de redes de Petri. Raposo, Magalhães e Ricarte (2000) trata da representação de fluxos de processos, preocupando-se também com a alocação de recursos. Em (FREYTAG; LANDES, 2003), é ilustrada uma ferramenta (PWFtool) para a modelagem de fluxos de processos utilizando a notação de redes de Petri. Esta ferramenta, baseia-se na formalização apresentada por Aalst (1998). Lopez-Grao, Merseguer e Campos (2004) apresenta uma representação em redes de Petri para diagramas de atividades UML, preocupando-se com a representação de informações referentes ao desempenho da aplicação. Esta representação baseia-se nas redes de Petri estocásticas generalizadas, dialeto demonstrado por Marsan et al. (1995). Este dialeto trata da representação de transições de temporizadas (consomem tempo em sua execução) e de transições imediatas (executadas no instante em que estão habilitadas), além disso este dialeto tem sua execução baseada em informações estatísticas. Ruiz (1995) apresenta

um dialeto de redes de Petri para a modelagem das atividades em aplicação de escritórios. Esta abordagem trata da alocação dos recursos necessários a execução de atividade.

8.2 REDES DE PETRI COM PROBABILIDADE

Em um dado estado do fluxo, um determinado processo pode seguir sua execução por segmentos distintos do processo. Isto sugere que cada caminho possível está atrelado a uma determinada probabilidade. Por isso, é interessante relacionar propriedades estatísticas aos processos para que se possa ter uma idéia definida de quais pontos do fluxo requerem uma maior atenção na geração dos casos de teste. Esta constatação leva a uma busca por ocorrências na literatura referente a redes de Petri estocásticas.

Em (CHIOLA et al., 1993), utiliza-se redes de Petri estocásticas para notação de modelos estocásticos. Esta notação baseia-se nas redes de Petri estocásticas generalizadas (MARSAN; CONTE; BALBO, 1984). As *GSPN* também são base para a avaliação de desempenho de sistemas de paralelos/distribuídos (KORIEEM; PATNAIK, 1997).

8.3 TESTE DE SOFTWARE A PARTIR DE REDES DE PETRI

Ramaswamy (2000) explora o teste de *software* baseando-se, para isso, em modelos descritos através de redes de Petri. Ho e Lin (1999), também explora o teste de *software* através da geração de casos de teste a partir de modelos de redes de Petri. Para isto, alguns segmentos presentes em modelos de *software* são identificados:

- Caso 1: uma transição com dois locais diferentes na entrada mas o mesmo local na saída;
- Caso 2: uma transição com o mesmo local na entrada mas dois locais diferentes na saída;
- Caso 3: transições em seqüência;
- Caso 4: duas diferentes transições com o mesmo local de entrada mas diferentes locais de saída cada uma;
- Caso 5: duas diferentes transições com o mesmo local de entrada e o mesmo local de saída;
- Caso 6: duas transições diferentes com locais de entrada diferentes e com o mesmo local de saída;

- Caso 7: duas diferentes transições com locais de entrada e de saída diferentes;
- Caso 8: uma transição cujo local de entrada é o mesmo local de saída.

Estes segmentos são semelhantes aos padrões de construção de fluxo encontrados em *workflows*.

8.4 TRABALHOS RELEVANTES À PESQUISA

O teste de *workflow* deve, além de considerar informações estatísticas, contemplar a validação das características presentes em *workflows* (usuários, distintos padrões de fluxo e tempo) de maneira conjunta. Entretanto, não foram encontrados, na literatura, testes que abordassem a validação do fluxo de controle juntamente com questões temporais e tratando dos distintos participantes presentes no fluxo de controle.

A seguir, na Tabela 3, são apresentados alguns dos trabalhos citados que se relacionam com a pesquisa. Aalst (1998) apresenta um dialeto de redes de Petri para a representação de *workflows*. Ho e Lin (1999) trata da geração de casos de teste se preocupando com tempos discretos, entretanto não se preocupa com a validação do fluxo em função deste tempo. Marsan, Conte e Balbo (1984) define uma classe de redes de Petri que permite a representação de informações estatísticas, juntamente com a representação de transições na rede que consomem tempo para a execução. Ruiz (1995) apresenta um dialeto de redes de Petri que contempla a definição dos recursos necessários a execução de uma transição. Além destes trabalhos, também faz parte da comparação a ferramenta STAGE 6.2 para a geração de casos de teste de forma automatizada. Esta ferramenta, desenvolvida pelo CPTS, foi utilizada para a geração de casos de teste para *SGWf* (VIEIRA, 2005).

8.5 CONSIDERAÇÕES

Constatou-se que há diversas pesquisas circundando o objetivo deste trabalho. Diversos trabalhos utilizam as redes de Petri como uma ferramenta de modelagem de sistemas. Além disso, encontrou-se vantagens em utilizar este formalismo para a modelagem de sistemas de *workflows*. Há também pesquisas na utilização de redes de Petri para a representação de questões temporais e estatísticas. Verificou-se, também, trabalhos que utilizam Redes de Petri para a geração de casos de teste.

Assim sendo, esta pesquisa uniu diversas características presentes nestes trabalhos, com o objetivo de apresentar uma formalização precisa e que atendesse os requisitos desta pesquisa.

Tabela 3: Relacionamentos com a pesquisa.

Trabalhos	Informações Estatísticas	Propriedades Temporais	Padrões de Fluxo	Distintos Participantes
Aalst (1998)	Não	Sim	Sim	Sim, mas não considera como um recurso necessário à execução da atividade
STAGE 6.2	Sim	Não	Sim, mas pelo fato de utilizar Redes de Autômatos Estocásticos, a modelagem de todos os padrões básicos não é trivial	Não
Ho e Lin (1999)	Não	Sim, mas utiliza tempos discretos	Sim	Não
Marsan, Conte e Balbo (1984)	Sim	Sim	Sim	Não
Ruiz (1995)	Não	Não	Sim	Sim
WfGSPN	Sim	Sim	Sim	Sim

9 CONCLUSÕES E TRABALHOS FUTUROS

Inicialmente, mostrou-se os pontos a que um trabalho que tem por objetivo o teste de sistemas gerenciamento de *workflow* deve se ater. Apresentou-se a arquitetura de um sistema de *workflow*, além das características temporais presentes na execução de *workflows*, presentes constantemente no fluxo do processo. A validação destas questões temporais torna os testes mais realistas, possibilitando que *deadlines* sejam validados. Considerando a geração de casos de teste com base na visão do usuário responsável pela execução dos *workflows*, verifica-se que o indicado é a realização de testes funcionais. Com o teste estatístico, possibilita-se validar os pontos mais percorridos no fluxo.

Para isso, apresentou-se a formalização de um dialeto de redes de Petri capaz de representar estas informações. A seguir, demonstrou-se a validade deste dialeto a partir da realização de um exemplo. Para este exemplo, casos de teste foram gerados. Estes casos de teste validam os *deadlines* presentes em *workflows*, tratam dos recursos presentes no fluxo e foram gerados com base em informações estatísticas referentes ao processo.

Sistemas de gerenciamento de *workflows* possuem claramente uma arquitetura bem definida, na qual o gerenciamento das atividades ocorre uma camada abaixo da interface de interação com o usuário. De forma que, ao gerar casos de teste para *SGWfs* é possível submetê-los a esta interface de interação com o usuário ou diretamente à camada responsável pelo gerenciamento das atividades.

Considerando a similaridade dos requisitos do teste de desempenho e do teste de sistemas de gerenciamento de *workflows*, adaptou-se a formalização apresentada para o teste de *SGWf* ao teste de desempenho. Este estudo foi validado através de uma ferramenta (UpperT), que através de diagramas de atividades gera uma rede de Petri baseada no dialeto formalizado. A partir desta rede, casos de teste são gerados, para então serem lidos e executados pela ferramenta JMeter. Como fruto deste trabalho, tem-se a aprovação de um artigo (OLIVEIRA et al., 2007).

Por fim, os trabalhos que circundam esta pesquisa foram apresentados. Com isso, pos-

sibilitou-se uma comparação da solução apresentada e dos trabalhos relacionados frente aos requisitos levantados. O dialeto formalizado nesta pesquisa é a única solução que satisfaz todos estes requisitos.

Futuramente, como forma de analisar os casos de teste gerados, deve-se realizar um estudo sobre a cobertura dos casos de teste. Assim, será possível determinar a quantidade mínima de casos de teste para que se tenha uma cobertura total do modelo. Além disso, a partir da experimentação, pôde-se verificar tópicos que podem dar continuidade à pesquisa. A geração de um modelo de dados a partir de um documento de definição de processo possibilitará que a geração de casos de teste seja realizada de forma simples. Somando-se a isso, a criação de um módulo que gere *scripts* para serem submetidos à Interface 2 do modelo de referência possibilitará que o teste de *SGWf* seja realizado de forma automatizada. Além disso, a aplicação deste formalismo ao teste de desempenho sugere que sejam analisados outros problemas com requisitos semelhantes.

REFERÊNCIAS

- AALST, W. M. P. van der. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, v. 8, n. 1, p. 21–66, Feb. 1998.
- AALST, W. M. P. van der; HEE, K. M. van; HOUBEN, G. J. Modelling and analysing workflow using a petri-net based approach. In: MICHELIS, G.; ELLIS, C.; MEMMI, G. (Ed.). *CSCW '94: Second Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms*. [S.l.: s.n.], 1994. p. 31–50.
- AALST, W. M. P. van der et al. Workflow patterns. *Journal of Distributed and Parallel Databases*, v. 14, n. 1, p. 5–51, July 2003.
- AALST, W. van der; HEE, K. v. van. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. Cambridge, USA: The MIT Press, 2002. 384 p. Hardcover. ISBN 0-262-01189-1.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML, guia do usuário*. Rio de Janeiro, Brasil: Campus, 2000. 472 p. ISBN 8-535-20562-4.
- BURNSTEIN, I. *Practical Software Testing: A Process-oriented Approach*. New York, USA: Springer-Verlag, 2002. 699 p. ISBN 0-387-95131-8.
- CHIOLA, G. et al. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, USA, v. 19, n. 2, p. 89–107, Feb. 1993. ISSN 0098-5589.
- ENHYDRA. *JaWE: Java Workflow Editor*. 2007. Disponível em: <<http://www.enhydra.org/workflow/jawe/index.html>>. Acesso em: 25 de março de 2008.
- FREYTAG, T.; LANDES, S. I. Pwftool - a Petri net workflow modeling environment. In: *AWPN '03: 10th Workshop Algorithms and Tools for Petri Nets*. [s.n.], 2003. Disponível em: <<http://woped.ba-karlsruhe.de/woped/Download/AWPN2003.pdf>>. Acesso em: 26 de março de 2008.
- HAMADI, R.; BENATALLAH, B. A Petri net-based model for web service composition. In: *ADC '03: 14th Australasian Database Conference*. Darlinghurst, Australia: Australian Computer Society, Inc., 2003. (CRPIT, v. 17), p. 191–200. ISBN 0-909-92595-X. Disponível em: <<http://crpit.com/confpapers/CRPITV17Hamadi.pdf>>. Acesso em: 25 de março de 2008.
- HEUSER, C. A. *Modelagem Conceitual de Sistemas*. Edição preliminar. Buenos Aires, Argentina: III Escola Brasileiro-argentina de Informática, 1988. 94 p.
- HO, I.; LIN, J.-C. Generating test cases for a real-time software by time Petri nets model. In: *ATS '99: 8th Asian Test Symposium*. Washington, USA: IEEE Computer Society, 1999. p. 295–300. ISBN 0-7695-0315-2.

- HOLLINGSWORTH, D. *The Workflow Reference Model*. Winchester, UK, Jan. 1995. Disponível em: <<http://www.wfmc.org/standards/docs/tc003v11.pdf>>. Acesso em: 27 de março de 2008.
- JORGENSEN, P. *Software Testing: A Craftman's Approach*. 2. ed. Boca Raton, USA: CRC Press, Inc., 2002. 359 p. ISBN 0-8493-0809-7.
- KALLEPALLI, C.; TIAN, J. Measuring and modeling usage and reliability for statistical web testing. *IEEE Transactions on Software Engineering*, IEEE Press, London, UK, v. 27, n. 11, p. 1023–1036, Nov. 2001. ISSN 0098-5589.
- KORIEEM, S. M.; PATNAIK, L. M. A generalized stochastic high-level Petri net model for performance analysis. *Journal of Systems and Software*, Elsevier Science Inc., New York, USA, v. 36, n. 3, p. 247–265, Mar. 1997. ISSN 0164-1212.
- LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado*. 2. ed. Porto Alegre, Brasil: Bookman, 2004. 607 p.
- LEYMANN, F.; ROLLER, D. *Production workflow: concepts and techniques*. Second edition. Upper Saddle River, USA: Prentice Hall PTR, 2000. 479 p. Paperback. ISBN 0-13-021753-0.
- LOPEZ-GRAO, J. P.; MERSEGUER, J.; CAMPOS, J. From uml activity diagrams to stochastic Petri nets: application to software performance engineering. In: *WOSP '04: 4th International Workshop on Software and Performance*. ACM Press, 2004. v. 29, n. 1, p. 25–36. ISSN 0163-5948. Disponível em: <http://www.utdallas.edu/~htj041000/phd/From_UML_AD.pdf>. Acesso em: 26 de março de 2008.
- MARSAN, M. A. et al. Modelling the software architecture of a prototype parallel machine. In: *ACM SIGMETRICS '87: Conference on Measurement and Modeling of Computer Systems*. New York, USA: ACM, 1987. p. 175–185. ISBN 0-89791-225-X.
- MARSAN, M. A. et al. *Modelling with Generalized Stochastic Petri Nets*. 1. ed. New York, USA: John Wiley & Sons Ltd (Import), 1995. 324 p. Hardcover. ISBN 0471930598. Disponível em: <<http://www.di.unito.it/~greatspn/bookdownloadform.html>>. Acesso em: 26 de março de 2008.
- MARSAN, M. A.; CONTE, G.; BALBO, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, ACM Press, New York, USA, v. 2, n. 2, p. 93–122, May 1984. ISSN 0734-2071.
- NOWOSTAWSKI, M. *JFern: Java-based Petri net framework*. 2003. Disponível em: <<http://sourceforge.net/projects/jfern/>>. Acesso em: 26 de março de 2008.
- OLIVEIRA, F. et al. Performance testing from uml models with resource descriptions. In: *SAST '07: 1st Brazilian Workshop on Systematic and Automated Software Testing*. João Pessoa, Brazil: [s.n.], 2007. v. 1, p. 1–8. ISBN 978857669140-2.
- OMG. *UML Profile for Schedulability, Performance, and Time Specification*. [S.l.], Mar. 2005. Release 1.1. Disponível em: <<http://www.omg.org/cgi-bin/doc?ptc/02-03-02>>. Acesso em: 26 de março de 2008.

PADUA, S. et al. O potencial das redes de Petri em modelagem e análise de processos de negócio. *Gestão & Produção*, Scielo, São Carlos, Brasil, v. 11, n. 1, p. 109–119, Abr. 2004. ISSN 0104-530X. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-530X2004000100010&nrm=iso>. Acesso em: 26 de março de 2008.

PRESSMAN, R. S. *Engenharia de Software*. São Paulo, SP, Brasil: Markron Books, 1995.

PROWELL, S. et al. *Cleanroom software engineering: technology and process*. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0-201-85480-5.

RAMASWAMY, S. A Petri net based approach for establishing necessary software design and testing requirements. In: *IEEE SMC '00: International Conference on Systems, Man, and Cybernetics*. Nashville, USA: [s.n.], 2000. v. 4, p. 3087–3092. ISBN 0-7803-6583-6. ISSN 1062-922X.

RAPOSO, A. B.; MAGALHÃES, L. P.; RICARTE, I. L. M. Petri nets based coordination mechanisms for multi-workflow environments. *International Journal of Computer Systems Science & Engineering*, CRL Publishing, Leics, U.K., v. 15, n. 5, p. 315–326, Sept. 2000. ISSN 0267-6192. Disponível em: <<http://www.tecgraf.puc-rio.br/abraposo/pubs/IJCSSE/ijcsse.pdf>>. Acesso em: 26 de março de 2008.

RUIZ, D. D. A. *Um Modelo para Representação de Atividades em Aplicações de Escritórios*. 195 p. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, Mar. 1995.

RUSSELL, N. et al. *Workflow Control-Flow Patterns: A Revised View*. [S.l.], 2006. 134 p. Disponível em: <<http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf>>. Acesso em: 27 de março de 2008.

TRAMMELL, C. J. Quantifying the reliability of software: statistical testing based on a usage model. In: *ISESS '95: 2nd IEEE Software Engineering Standards Symposium*. Washington, USA: IEEE Computer Society, 1995. p. 208–218. ISBN 0-8186-7137-8.

VIEIRA, H. V. *Tratamento de Aspectos Temporais em Testes de Processos de Negócio*. Trabalho Individual II. Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brasil, Nov. 2005. 46 p.

WfMC, W. M. C. *Workflow Management Coalition: Terminology & Glossary*. Winchester, UK, Feb. 1999. Disponível em: <http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf>. Acesso em: 27 de março de 2008.

WfMC, W. M. C. *Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface - XML Process Definition Language*. Lighthouse Point, USA, Oct. 2005. 164 p. Disponível em: <http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf>. Acesso em: 26 de março de 2008.

YAMAGUCHI, H. et al. A Petri net based method for deriving distributed specifications with optimal allocation of resources. In: *SNPD '00: 1st Int. Conf. on Software Engineering Applied to Networking e Parallel/Distributed Computing*. [s.n.], 2000. p. 19–26. Disponível em: <<http://www.site.uottawa.ca/bochmann/dsrg/PublicDocuments/Publications/Yama00a.pdf>>. Acesso em: 27 de março de 2008.