PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Leonardo Cardoso Dias
Romulo da Silva Menna

TESTE DE DESEMPENHO A PARTIR DE MODELOS UML PARA COMPONENTES DE SOFTWARE

Porto Alegre
2008

Leonardo Cardoso Dias

Romulo da Silva Menna

TESTE DE DESEMPENHO A PARTIR DE MODELOS UML PARA COMPONENTES DE SOFTWARE

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Informática Pontifícia Universidade Católica do Rio Grande do Sul, como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Flávio Moreira de Oliveira

Porto Alegre

2008

Resumo

Atualmente na indústria de software, não existem muitos aplicativos voltados para o teste de desempenho em componentes de software. A reutilização de componentes de software no processo de desenvolvimento de uma aplicação hoje em dia é muito grande, isto devido ao baixo custo e ao pouco tempo para deixá-lo operável.

Neste contexto, a proposta desse trabalho é construir uma ferramenta capaz de gerar drivers de teste para testar o desempenho de componentes de software. A ferramenta será capaz de gerar os testes em fase de projeto de software, ou seja, os testes estarão prontos antes mesmo do sistema em construção estar implementado.

Os casos de teste serão gerados com base nas especificações de requisitos de desempenho inseridas no modelo *UML*. Para isto utilizaremos o *UML SPT Profile*, que permitirá coletar as informações de desempenho esperado para o software.

Para gerar os casos testes será gerada e simulada uma rede de *Petri*, os resultados da simulação serão convertidos para os casos de testes, logo após, estes serão convertidos em um driver para o *Apache JMeter*, na qual irá executar os testes sobre sistema em teste, resultando assim uma avaliação de desempenho dos componentes, de acordo com os requisitos de desempenho especificados.

Abstract

Currently in the industry of software, there are many applications focused on the test performance in components of software. The reuse of software components in the process of developing an application today is too big, that due to the low cost and little time to let it operating.

In this context, the proposal of this work is to build a tool capable of generating test drivers to test the performance of software components. The tool will be able to generate tests in the design phase of software, the tests will be ready even before of the system under construction be implemented.

The test cases will be generated based on the user performance requirements, entered into the UML model. For this, we will use the UML SPT Pro ? le, which will collect details of performance expected for the software.

Will be generated and simulated a network of Petri to build the tests, the results of simulantion will be converted to test cases, soon after, these will be converted into a ApacheJMeter driver, in which will perform the tests on system in trial, thereby resulting an evaluation of performance of the components, according to the user performance requirements.

Lista de Figuras

Sumário

1 Introdução

Atualmente, programas de computadores desempenham um papel fundamental, tanto no aspecto social quanto no econômico. Cada vez mais pessoas e empresas os utilizam para executar as mais diversas tarefas no cotidiano, e em determinados casos uma falha no sistema pode ocasionar uma perda catastrófica.

Devido a isso as empresas produtoras de softwares estão investido cada vez mais na qualidade dos seus produtos. O teste de software é um dos processos que auxiliam na qualificação de sistemas de computação; seu objetivo é identificar possíveis erros ou situações em que o sistema possa entrar em colapso.

Um tipo de teste conhecido é o teste de desempenho, onde seu objetivo principal é quantificar e avaliar a eficácia do sistema em determinados cenários a partir da análise de algumas características do sistema, tais como: tempo de resposta de uma requisição, o número de conexões ou usuários, o consumo excessivo de recursos e possíveis gargalos. Na maioria dos sistemas multi-usuário, o teste de desempenho é muito importante para garantir a qualidade de utilização do sistema no quesito de rapidez.

Com a globalização da sociedade através da internet, celular e outros meios de comunicação, os softwares estão se tornando mais complexos, tendo a necessidade de serem construídos em muito pouco tempo e para um grande número de usuários. A troca de informação e serviços com outros softwares e o reaproveitamento de código acabam se tornando essenciais.

Neste contexto, uma das técnicas utilizadas é a construção de software baseada em componentes, que permite uma alta modularidade dos processos e serviços do software, possibilitando uma reutilização de componentes em outros softwares e uma comunicação mais clara entre outros sistemas. Um componente utilizado em softwares é *DAO* (*Data Access Objec*t, objeto de acesso aos dados). O *DAO* é um padrão de projeto desenvolvido pela Java 2 Platform, Enterprise Edition (Aiur John Crupi 2002) que permite extrair e encapsular todos os acessos à origem de dados. O DAO gerencia a conexão com a origem de dados para obter e armazenar dados.

Garantir a qualidade do componente *DAO* é muito importante em qualquer sistema que o

utilize, pois qualquer erro pode corromper o banco de dados, o sistema de arquivos ou outro sistema de armazenamento. Propiciar que os dados serão processados rapidamente nessa camada pode ser crucial para o desempenho geral do sistema.

No mercado o teste de desempenho em componentes é ainda uma área pouco explorada. Com base nisso, a proposta deste trabalho é desenvolver uma solução de teste de desempenho em componentes focando no componente *DAO*, tendo conhecimento que a avaliação de desempenho deste é um dos pontos críticos para o desempenho geral de todo o sistema.

No capítulo de fundamentação teórica serão apresentados conceitos, técnicas e níveis de teste, juntamente com o teste de desempenho que é o enfoque desse trabalho. A ferramenta *Apache JMeter* (JMeter 2007) tambem será descrita pois é o robo executor dos testes de desempenho gerados pela proposta desse trabalho.

Logo após será apresentado a fundamentação para modelagem de desempenho de sistemas, na qual será nossa base para a geração de teste baseado em simulação. Em seguida será apresentado a concepção de *UML* (Group 2004) e UML SPT (Group 2002)profile, que mostrará como métricas de desempenho podem ser inseridas na *UML*, de onde será coletado as informações do sistema. Posteriormente, será apresentado o conceito do padrão *DAO*, que será exemplo base de utilização do trabalho proposto.

No capítulo de especificação do sistema será apresentado os objetivos da proposta e como será o funcionamento geral do sistema, com uma visão em alto nível. No capítulo de projeto é descrito a arquitetura interna do sistema, quais as atividades deste e como este procederá para atingir seu objetivo.

2 *Fundamentação Teórica*

2.1 Teste

"Porque testamos há duas principais razões: fazer julgamento sobre a qualidade ou acessibilidade e descobrir problemas" (Burnstein 2002) . .

O teste pode ser descrito como um processo usado para revelar defeitos em um software utilizando para isto uma série de métodos ao longo ou no final dos processos de desenvolvimento do software.

Testar o software é o mecanismo usado para ajudar a identificar a exatidão, a integridade, a segurança e a qualidade do sistema. Testar é um processo da investigação técnica, executado em nome das partes interessadas, que pretende revelar informação de qualidade relacionada sobre o produto com respeito ao contexto em que se pretende se operar.

A essência do teste de software é determinar um conjunto de **casos de teste** para um item a ser testado; antes de continuar precisa-se ter uma clara informação do que deverá estar no caso de teste. Um dos primeiro passos para se realizar um teste é identificar as informações de entrada na qual são de dois tipos: pré-condicionais (circunstâncias que impedem a execução do caso de teste) e as entradas propriamente ditas que são identificadas por algum método de teste.

A próxima etapa do caso de teste é a expectativa das respostas; novamente, podem ser de dois tipos: pós-condicionais e as saídas propriamente ditas. O ato de testar está em definir as pré-condições necessárias, providenciando as entradas do caso de teste e observando a saída a fim de encontrar algum ponto crítico não esperado pelo sistema.

2.1.1 Técnicas de Teste

As principais técnicas utilizadas para identificar casos de teste são conhecidas como testes funcionais, estruturais e não funcionais. Cada uma destas técnicas possuí um conjunto distinto de métodos de identificação de casos de teste (Jorgensen 1995).

Funcional

O teste funcional ou teste de caixa-preta desconhece a estrutura adotada na implementação do software, apenas tem o conhecimento do que o software faz. A descrição do comportamento ou funcionalidade para o software sobre teste pode vir de uma descrição formal de especificação, assim como as entradas e saídas, as pré e pós condições, para então o teste ser realizado e determinar se as saídas produzidas estão equivalentes às especificadas.

Estrutural

O teste estrutural ou teste de caixa-branca foca na estrutura interna do software a ser testado. Para criar os casos de teste utilizando esta estratégia o testador precisa ter o conhecimento da composição interna do software. O testador seleciona casos de teste para exercitar elementos da estrutura interna específica, para determinar se estes estão trabalhando devidamente corretos.

Aspecos não funcionais

Os testes de aspectos não funcionais são testes que avaliam exigências de qualidade do sistema tais como:

- Segurança: Avalia a segurança do sistema em relação a ataques maliciosos, segurança dos dados armazenados, acesso autorizado no sistema, para garantir que o cliente/usuário possa utilizar o sistema com confiança e que seus dados estejam protegidos.
- Desempenho: Avalia a eficiência do sistema para garantir as exigências de desempenho do sistema especificado pelo cliente; o foco desse trabalho está especificamente nesse tipo de teste.
- Compatibilidade: Avalia se o sistema opera com todas as potenciais configurações de usuários, por exemplo, a compatibilidade de diferentes tipos de browsers caso o sistema seja web, ou a compatibilidade com vários sistemas operacionais.
- Usabilidade: Analisa o *design* e facilidade de uso das interfaces de comunicação entre o sistema e o usuário.

2.1.2 Níveis de teste

O teste de software, especialmente para grandes sistemas, é usualmente realizado por diferentes níveis. Na maioria dos processos existem de 3 a 4 níveis, ou na maioria das fases do processo de desenvolvimento de software. Os níveis de teste são os seguintes: teste unitário, teste de integração, teste de sistema e em alguns tipos teste de aceitação.

Cada um destes consiste em um ou mais sub-níveis ou fases. Cada nível possui objetivos de teste específicos; por exemplo, em um teste unitário um simples componente é testado. O objetivo principal é detectar defeitos funcionais e estruturais na unidade. No nível de integração uma série de componentes são testados como um grupo, onde o testador investiga a interação entre estes componentes. No nível de sistema o principal objetivo é avaliar atributos como usabilidade, confiabilidade e desempenho. Por fim existe o teste de aceitação, nesse nível verifica se o sistema está de acordo com o que o usuário solicitou utilizando para isto o teste de sistema juntamente com os requisitos do usuário.

2.2 Teste de Desempenho

No processo de desenvolvimento de software os principais documentos de requisitos são:

- Requisito funcional: Usuário descreve quais são as funções que o software deve possuir.
- Requisito de qualidade: São características não funcionais do software descrevendo o nível de qualidade esperada para este. Um exemplo de requisito de qualidade é o nível de desempenho; o usuário deve ter objetivos para o sistema em termos de uso de memória, tempo de resposta e atrasos.

O objetivo do teste de desempenho é identificar possíveis gargalos ou insuficiências de desempenho e verificar se o desempenho do software satisfaz os requisitos de desempenho especificados.

O teste de desempenho pode servir para várias finalidades. Pode demonstrar que o sistema encontra-se com critérios de desempenho. Pode comparar dois sistemas com a finalidade de encontrar o mais eficiente, ou medir que parte do sistema ou um *workload* está comprometendo a desempenho deste. O teste de desempenho permite também otimizar a alocação de recursos do sistema, por exemplo, o testador pode achar que é necessário realocar o espaço de memória,

ou modificar o nível de prioridade de determinada operação do sistema. Os testadores ainda são capazes de projetar os futuros níveis de desempenho do sistema, o que é muito útil para planejar as versões subseqüentes do sistema.

Os objetivos de desempenho devem ser articulados claramente pelos usuários/clientes no documento de requisitos e devem ser indicados no plano de teste do sistema. Os objetivos devem ser quantificados, por exemplo, em uma exigência de que o sistema responda a uma consulta em uma quantidade de tempo aceitável, o tempo exigido deve ser especificado de uma maneira quantitativa. No final do teste os testadores saberão, por exemplo, o número de ciclos que a CPU utilizou, o tempo de resposta real em segundos (minutos, horas, etc) e o número de transações reais processadas por um período de tempo. Esses resultados podem ser avaliados com os respectivos objetivos exigidos (Dustin Jeff Rashka 2002).

2.2.1 Medidas de Desempenho

Desempenho pode ser definido como a maneira como um sistema se comporta. Isto é, o desempenho de um sistema é determinado por suas características de execução. Portanto, avaliar o desempenho de um sistema demanda definir quais características comportamentais devem ser consideradas. Por exemplo, se quisermos avaliar o desempenho de um automóvel, iremos considerar fatores tais como velocidade máxima, capacidade de aceleração (tempo necessário para ir de 0 Km/h a 100 Km/h), espaço de frenagem a uma dada velocidade, consumo médio de combustível, etc. Para sistemas computacionais, em geral consideram-se quatro fatores para medida de desempenho:

- Vazão (throughput): taxa de atendimento de pedidos pelo sistema. Ex.:
    1. Sistemas em lotes: jobs por segundo; Sistemas interativos: requisições por segundo; CPU: MIPS ou MFLOPS;
    2. Redes: pacotes por segundo (pps) ou bits por segundo (bps); Sistemas de processamento de transações: transações por segundo (TPS);
- Utilização: fatia de tempo em que o sistema permanece ocupado, atendendo a pedidos;
- População: quantidade de clientes a serem atendidos em um determinado instante;
- Tempo de resposta: intervalo de tempo entre o pedido e o início/conclusão do serviço.

2.2.2 Apache JMeter

O JMeter é uma ferramenta de código aberto do grupo Apache Software Foundation, desenvolvida totalmente com tecnologia Java e utilizada para testes de carga em serviços oferecidos por sistemas computacionais. Também é possível realizar testes de desempenho e de caixa-preta. Um de seus atrativos é o fato de permitir a execução de plano de testes que podem ser configurados graficamente (JMeter 2007).

Para a realização de testes, a ferramenta JMeter disponibiliza diversos tipos de requisições e *assertions* (para validar o resultado dessas requisições), além de controladores lógicos como *loops*(ciclos) e controles condicionais para serem utilizados na construção de planos de teste, que correspondem aos testes funcionais.

O JMeter disponibiliza também um controle de *threads*, chamado *Thread Group*, no qual é possível configurar o número de *threads*, a quantidade de vezes que cada *thread* será executada e o intervalo entre cada execução, que ajuda a realizar os testes de stress. E por fim, existem diversos *listeners*, que se baseando nos resultados das requisições ou dos *assertions*, podem ser usados para gerar gráficos e tabelas. Componentes no JMeter são recursos que podem ser utilizados para criar rotinas de testes para aplicações.

Test Plan

Para qualquer teste que venha a ser feito utilizando o JMeter, é necessário criar um Test Plan incluindo os elementos do teste. Estes elementos podem ser:

- **Thread Group**: este é ponto de começo, todos os outros elementos do Test Plan devem estar sob este. Como o próprio nome ressalta, este controla as threads que serão executadas pelo teste;
- **Controllers**: estes são divididos em dois grupos Samplers e Logic Controllers.
  - **Samplers**: são controladores pré-definidos para requisições especificas. Podendo ser customizada com a inserção de configurações (Configurations), Assertions e etc;
  - **Logic Controllers** : são controladores mais genéricos. Podendo ser customizada com a inserção de outros controllers, configuration elements, assertions, etc.
- **Listeners**: estes são os elementos que fornecem acesso as informações obtidas pelo JMeter durante os testes.

- **Timers**: por padrão, o JMeter faz requisições sem pausas entre elas. Os timers são utilizados para incluir pausas entre as requisições.
- **Assertions**: usado para verificar se a resposta obtida na requisição é a esperada. Podendo ser usado expressões regulares (Perl-style regular expression) na comparação.
- **Configuration Elements**: embora não faça requisições (exceto para HTTP Proxy Server), este elemento pode adicionar ou modificar as requisições.
- **Pre-Processor Elements**: executa alguma ação antes de fazer a requisição. Mais usado para pré-configurações das requisições.
- **Post-Processor Elements**: executa alguma ação depois de fazer a requisição. Mais usado para processar as respostas da requisição.

Tipos de serviços

O JMeter suporta requisições dos seguintes tipos de serviços em suas rotinas de testes:

- **FTP**: permitir criar requisições usando o protocolo FTP (com autenticação ou não) e executa o comando de retrieve em um arquivo especifico.
- **HTTP**: permitir criar requisições usando o protocolo HTTP ou HTTPS (com autenticação ou não), podendo incluir parâmetros ou arquivos a requisição, escolher o método usado (GET ou POST) e manipular Cockies. Este sampler possui dois tipos de implementação: Java HTTP ou Commons HTTPClient.
- **JDBC**: Com esta requisição é possível executar queries em um banco de dados específico.
- **Objeto java**: ajuda no teste de carga de classes Java, exigindo seja implementado uma classe do tipo JavaSamplerClient para executar o método a ser testado. A estrutura deste objeto é similar à usada pelo JUnit.
- **SOAP/XML-RPC**: permite enviar requisições SOAP para um WebService, ou enviar XML-RPC através do protocolo HTTP.
- **LDAP**: permite enviar requisições para um servidor LDAP. Possui uma implementação simplificada e outra estendida.
- **Testes JUnit**: usado para fazer teste de carga em testes de unidade que utilizam o framework JUnit.

Existem outros tipos de requisições que, até a atual versão do JMeter, estão em versão alpha, eles são: Web service (SOAP), Access Log, BeanShell, BSF, TCP, JMS Publisher, JMS Subscriber, JMS Point-to-Point. Neste trabalho, ele será utilizado para executar os testes gerados pela ferramenta proposta utilizando, para isso, o acesso aos objetos java a fim de mensurar seu desempenho.

2.3 Modelagem de desempenho de sistemas

Nesta seção será introduzido o conceito de modelagem de sistemas de uma maneira geral; os tipos de sistemas, quais as formas de representação e alguns modelos importantes como de Markov. Logo após será explicado às redes de Petri, modelo na qual será utilizado para a simulação dos casos de teste do trabalho proposto; junto a este uma breve descrição do JFern, robô de simulação das redes de Petri.

2.3.1 Linguagens de Modelagem de Sistemas

Uma linguagem de modelagem é o meio pelo qual se expressam modelos, tendo como principal objetivo a descrição de sistemas (o mesmo que deve possuir uma série de características orientadas a esta atividade). Abaixo seguem algumas destas características necessárias:

- Possuir uma base formal, visando obter uma interpretação exata e precisa;
- Clareza, visando facilitar a comunicação entre todos os envolvidos numa modelagem, e;
- Possibilitar a construção de modelos que preencham os requisitos de conceitualização (contendo apenas as propriedades desejadas do sistema modelado) e de totalidade (todas as propriedades desejadas do sistema modelado).

Porém é difícil que uma linguagem tenha todas as características, mesmo porque algumas delas são conflitantes. É comum nos sistemas encontrar componentes que apresentem atividades concorrentes ou paralelas. Neste sentido, por exemplo, as Redes de Petri são uma linguagem de modelagem que foi desenvolvida especificamente para modelar sistemas discretos que possuem componentes que interagem concorrentemente (Peterson 1981; Agervala 1979).

Depois de termos definido o conceito de sistema em forma geral, podemos agora dividi-los segundo sua natureza, os sistemas de modo geral podem ser classificados como discretos e contínuos. Na prática poucos sistemas são totalmente discretos ou contínuos, porém, depois

de fazer algum tipo de mudanças para a maioria dos sistemas, será possível então classificá-los como sendo discretos ou contínuos.

2.3.2 Sistemas Contínuos

Sistemas Contínuos são aqueles sistemas nos quais as variáveis de estado mudam continuamente no tempo. Na realidade, todos os sistemas são contínuos. São os modelos (representação aproximada dos sistemas) que são discretos.

Comportamento Dinâmico dos Sistemas de Variáveis Contínuas

As teorias de controle de sistemas evoluindo no tempo, satisfazendo basicamente leis físicas, são fundamentadas basicamente em modelos matemáticos descritos por equações diferenciais. A noção do tempo é uma variável independente, a qual é substituída por uma seqüência de eventos num SDED (sistemas de estados e eventos discretos). A trajetória de um sistema dinâmico de variável contínua (SDVC) está constantemente mudando com o estado, tomando valores em R^n , sendo capaz de representar o comportamento do sistema em um instante qualquer a partir de um instante inicial.

2.3.3 Sistemas Discretos

Sistemas discretos são sistemas nos quais as variáveis de estado mudam apenas num conjunto discreto de pontos no tempo. Por exemplo: O banco é um exemplo de um sistema discreto, pois a variável de estado, o número de clientes no banco, muda só quando um cliente chega ou quando o serviço prestado a um cliente é completado. Como vimos anteriormente, a maioria dos sistemas são contínuos, mas para casos de estudo, eles podem se tornar discretos.

Sistemas Discretizados

Sistemas Discretizados são sistemas estudados somente em instantes precisos. Trata-se, portanto, de sistemas contínuos observados em instantes discretos (sistemas amostrados), as variáveis de estado evoluem de maneira contínua, sem mudança brutal de comportamento, mas é somente a instantes discretos que há um interesse no seu valor.

Sistemas Discretos

Sistemas Discretos são sistemas para os quais as variáveis de estado, ou ao menos algumas delas, variam brutalmente a certos instantes. Entretanto, estes instantes podem necessariamente ser previstos, e o conhecimento do estado há um instante dado não permite que, sem cálculo se conheça o estado seguinte.

Sistemas a eventos discretos

Sistemas a eventos discretos são sistemas modelados de tal maneira que as variáveis de estado variam brutalmente em instantes determinados e que os valores das variáveis nos estados seguintes podem ser calculados diretamente a partir dos valores precedentes e sem ter que considerar o tempo entre estes dois instantes.

2.3.4 Cadeias de Markov

As cadeias de Markov, introduzidas pelo matemático Andrei Andreyevich Markov é um formalismo matemático para a modelagem de sistemas reais. Uma cadeia de Markov é um tipo de processo estocástico em que a probabilidade de estar em certo estado em um tempo futuro pode depender do estado atual do sistema, mas não dos estados em tempos passados. Em outras palavras, em uma cadeia de Markov, dado o estado atual do sistema, o próximo estado independe do passado (mas poderá depender do estado presente). Existem dois tipos: tempo discreto e tempo contínuo (Brémaud 1999).

2.3.5 Redes de Petri

A Rede de Petri introduzida por Carl Adam Petri em sua tese intitulada "Comunicação com autômatos" é uma ferramenta gráfica e algébrica que apresenta um bom nível de abstração em comparação com outros modelos gráficos. Uma rede de Petri é um modelo do tipo estado-evento, onde cada evento possui pré-condições que vão permitir sua ocorrência e pós-condições decorrentes desta, as quais são por sua vez pré-condições de outros eventos posteriores.

Uma Rede de Petri é vista também como um tipo particular de grafo orientado que permite modelar as propriedades estáticas de um sistema a eventos discretos, constituído de dois tipos de nós: as transições (que correspondem aos eventos que caracterizam as mudanças de estado do sistema), e os lugares (que correspondem às condições que devem ser certificadas para os eventos acontecerem) interligados por arcos direcionados ponderados.

Redes de Petri têm sido utilizadas principalmente para a modelagem de sistemas de eventos em que estes possam ocorrer concorrentemente, havendo obstáculos na concorrência, precedência ou freqüência desses eventos (Peterson 1977) .

As Redes de Petri Ordinárias, também chamadas de primitivas ou autônomas, possuem baixo poder de modelagem por representarem apenas relações de causa e efeito entre os eventos e as condições. A sua utilização é restrita, portanto, a diversos tipos de sistemas pertencentes a classe de sistemas (dinâmicos) de eventos discretos, onde sincronização externa e o tempo não intervêm. Um dos campos de aplicação mais freqüente são os protocolos de comunicação em sistemas de computador. Como concorrência, sincronização e compartilhamento de recursos podem ser achados na especificação de tais sistemas, as Redes de Petri são uma ferramenta muito apropriada para sua modelagem.

Rede de Petri Temporizada Determinística

Para suprir a necessidade de escolha determinística, foram desenvolvidas as redes de Petri temporizadas. Essas redes possibilitam a representação do comportamento dinâmico de sistemas que possuam atividades concorrentes, assíncronas e não-determinísticas, através da adição do conceito de tempo no modelo.

O tempo também pode ser usado de maneira probabilística, ou seja, o disparo de transições está associado a distribuições de probabilidade. Essas redes são denominadas redes de Petri estocásticas, pois seus comportamentos podem ser descritos por processos estocásticos A associação do tempo a componentes da rede pode se realizar de várias maneiras, as principais são (P.R.M. Lins R.D. 1996):

- O tempo associado aos lugares: nesse caso, os tokens (após o disparo de uma transição) só estarão disponíveis para disparar uma nova transição após um determinado tempo que está associado ao lugar.
- O tempo associado aos tokens: nesse caso o tempo indica quando o token estará disponível para disparar uma transição.
- O tempo associado às transições: o objetivo desta é enfocar as redes de Petri temporizadas determinísticas com tempos associados às transições. Por uma questão prática, quando se fizer referência à rede de Petri temporizada, estará subentendido que está-se referindo às redes de Petri temporizadas determinísticas com tempos associados às transições.

Redes de Petri Estocásticas Generalizadas(RPEG)

As redes estocásticas e generalizadas são obtidas permitindo-se que as transições possam ter um atraso associado a elas de valor nulo. Ou seja, sejá definidos dois tipos de transições as imediatas, que possuem atraso nulo e as temporizadas que possuem um atraso exponencialmente distribuíddo associado a elas (M. Balbo G. 1995).

Conceituação Formal

Uma Rede de Petri (simples ou Autônoma) é composta de quatro partes: um conjunto de lugares P, um conjunto de transições T, uma função de entrada I ou Pré, e uma função de saída O ou Pos. As funções de entrada e saída relacionam transições e posições. Sendo assim a estrutura das Redes de Petri é definida por suas posições, transições, a função de entrada I (ou Pré), e a função de saída O (ou Pós).

Definição Formal:

Uma Rede de Petri (RdP) é uma quádrupla $R = (P, T, Pr, Ps)$ onde:

- P = p1, p2, ...pn é um conjunto de lugares, n0.
- T = t1, t2, ... tmé um conjunto de transições, m0.
- Pré: $PxT \rightarrow N$ é a aplicação de arcos de entrada das transições (lugares precedentes).
- Pós: $TxP \rightarrow N$ é a aplicação de arcos de saída das transições (lugares posteriores).
- $P \cap T = \oslash$
- N : é o conjunto dos números naturais.

Rede de Petri Marcada :

Uma Rede de Petri marcada é uma dupla $N = (R, Mo)$ onde:

- R é uma Rede de Petri
- Mo é a marcação inicial dada pela aplicação
- $M : P \rightarrow N$

- Sendo M(p)0 equivalente ao número de marcas contidas em cada lugar. A marcação de todos os lugares é representada por um vetor n-dimencional:
- $M = [m1, m2, ...mj]^{T^0}$ onde:
- mj = é o número de marcas do lugar pj

A marcação num determinado instante representa o estado da Rede de Petri, ou mais precisamente o estado do sistema descrito pela RdP, assim a evolução do estado da RdP corresponde a uma evolução da marcação, a qual é causada pelo disparo de transições como veremos posteriormente.

Uma marcação M é, portanto, uma aplicação que associa a cada lugar na RdP um inteiro não negativo chamado de marca ou token. Pode-se dizer também que M é a distribuição das marcas nos lugares "p" ou o número de marcas nos lugares.

Representação Gráfica

Uma Rede de Petri pode ser vista como um multigrafo bipartito e orientado definido por R = (P,T,Pre,Pos,M0), onde P é um conjunto de lugares representado por círculos e T é um conjunto de transições representado por barras, Pré e Pós são as relações de precedência e poscedência aplicados sobre PxT e TxP respectivamente, e M representa a marcação dos lugares através de marcas (pontos pretos) no interior de cada lugar.

- Pré (pi, tj) > 0 Se existe um arco do lugar pi para a transição tj;
- Pré (pi, tj) = 0 Caso contrário;
- Pós (tm, pa) > 0 Se existe um arco da transição tm para o lugar pa;
- Pós (tm, pa) = 0 Caso contrário.

Um dos objetivos deste trabalho é fazer uma tradução dos diagramas UML (casos de uso, atividades e seqüência) para as Redes de Petri, pois as Redes de Petri permitem a representação gráfica e a simulação do comportamento dinâmico do sistema modelado, o que nos proporciona uma visão global do processo em funcionamento.
A figura 2.1 mostra um exemplo de uma rede de Petri e seus principais elementos.

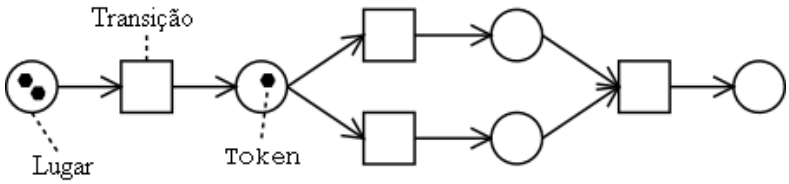

Figura 2.1: Exemplo de Rede de Petri

2.3.6 Rakiura JFern

Rakiura JFern é um pequeno framework opensource para redes de Petri (JFern 2006). Através dele é possível construir redes de Petri simples e/ou complexas e simulá-las. Infelizmente existe pouca documentação sobre ele, mas através de uma análise de seu código fonte e de sua API verificou-se que ele satisfaz os requisitos necessários para este trabalho. Através dele podemos manipular qualquer entidade de uma rede de Petri (Ex: lugares, transições, arcos de entrada, etc). Assim, podemos criar uma rede personalizada e adaptada para o propósito deste trabalho.

2.4 Desempenho em UML

Nesta seção será descrito o conceito da UML, com esta surgiu e quais as principais vantagens da versão 2.0. Logo após será descrito o *UML SPT Profile* juntamente com a sua concepção de escopo, na qual é importante entender para compreender quais as métricas de desempenho podem ser inseridas no projeto UML e como estas são incluídas.

2.4.1 OMG

Fundado em 1989, o OMG promove a teoria e a prática de tecnologia orientada a objeto no desenvolvimento do software. A carta patente da organização inclui estabelecimento das especificações da gerência diretrizes da guia e do objeto da indústria para fornecer uma estrutura comum para o desenvolvimento de aplicações. Os objetivos preliminares são a reusabilidade, a portabilidade, e a interoperabilidade do *software* baseado em objetos em ambientes distribuídos, heterogêneos. Conforme estas especificações tornarão possível desenvolver um ambiente heterogêneo das aplicações através de todas as principais plataformas de hardware e sistemas operacionais. Os objetivos da OMG é promover o crescimento da tecnologia orientada a objeto e influenciar seu sentido estabelecendo a arquitetura da gerência do objeto (OMA). O OMA fornece a infra-estrutura conceitual em que todas as especificações da OMG são baseadas (Group 2007).

2.4.2 UML 2.0

A UML (*Unified Modeling Language*) apareceu primeiramente nos anos 90 como um esforço para selecionar os melhores elementos de vários sistemas modelados e propostos naquele tempo a fim de combiná-los em uma única notação. A UML tem transformado o padrão da indústria para modelar projetos de software, assim como a modelagem de outros processos nos mundos científicos e dos negócios.

A UML possui uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos. Sintetiza os principais métodos existentes, sendo considerada uma das linguagens mais expressivas para modelagem de sistemas orientados a objetos. Utilizando seus diagramas é possível representar sistemas de software sob diversas perspectivas de visualização. A partir da versão 2.0 da UML é possível utilizar pacotes de perfis, que contém os mecanismos que permitem estender componentes a fim de se adaptar para finalidades diferentes.

Dentre as inovações inseridas nas últimas versões da UML está o *XML Metadata Interchange* (XMI), que é uma padronização criada pela OMG para troca de meta dados via XML (*Extensible Markup Language*). Um dos usos mais importantes do XMI é a representação de modelos UML no formato XML, possibilitando a exportação destes para diferentes aplicações (Group 2004).

2.4.3 UML SPT Profile

O UML Profile for Schedulability,Performance, and Time é um perfil que define uma linguagem para projetar, visualizar, especificar, analisar, construir e documentar os artefatos de testes de um sistema. Através dela, é possível informar algumas características que descrevem o comportamento de carga e tempo do sistema, como a freqüência esperada de usuários em um dado tempo, qual a probabilidade de uma determinada funcionalidade ocorrer, quanto tempo esta funcionalidade deveria gastar, entre outras características (Group 2002). O perfil fornece facilidades para:

- Capturar exigências de desempenho dentro do contexto do projeto.
- Associar características de desempenho de QdS (Qualidade de Serviço) com os elementos selecionados de um modelo de UML.
- Especificar os parâmetros da execução que podem ser usados pelo modelo para computar características de desempenho.

- Apresentar os resultados do desempenho computados pelo modelo ou encontrados no testes.

A concepção geral do domínio do profile SPT é basicamente classificada nos seguintes conceitos:

Scenario

Um cenário é uma seqüência de uma ou mais etapas de cenário; as exigências de QdS são colocadas nos cenários. Cada cenário é executado por um *workload* com uma intensidade aplicada de carga. Um *workload* pode ser categorizado por um tipo de trabalho ou um tipo de usuário.

Workload

Workload ou cargas de trabalhos são classes de trabalho que definem a carga de usuários ou requisições que um cenário terá. Há dois tipos de *workload*, que são:

- *Workload* aberto: tem uma carga de requisições que chegam a uma determinada taxa em algum padrão pré-determinado (tal como Poisson).
- *Workload* fechado: tem um número fixo de usuários ou de trabalhos (ativos ou potenciais), entre os ciclos de execução dos cenários e que gastam um tempo de atraso (também chamado de *Think Time*) fora do sistema, entre o fim de uma resposta e o próximo pedido.

Step

Os steps ou as atividades são elementos do cenário e estão agrupados em uma seqüência, com relacionamentos predecessor/sucessor que podem incluir *forks, joins*, e *loops*. Representam um passo ou uma ação elementar onde pode ser especificado o tempo de resposta esperado, a probabilidade desta ao ocorrer, entre outros. Cada *Step* possui um contador que especifica o numero de vezes que este foi executado, e uma demanda de recursos utilizados na execução. Um *Step* pode opcionalmente possuir propriedades de QdS.

Resources

Resouces são recursos que são modelados por servidores. Os recursos ativos são os servidores usuais em modelos de desempenho e possui tempos de serviço. Os recursos passivos

são adquiridos e liberados durante o cenário e possuem tempos de apreensão. Os *resources-operations* ou as operações de recursos são os *Steps*/atividades ou uma seqüência de *Steps* que requerem o recurso. O recurso é obtido no inicio da execução de um *Step* e liberado no final.

Resource demands

Representam os recursos que estão sendo utilizados na execução dos *Steps*; também podem incluir demandas aos recursos através de operações externas de recurso (tais como a entradas de IO) que no são definidas no modelo do software de UML, mas não são compreendidos pela ferramenta de modelagem de desempenho. Estas demandas são dadas como um número médio das operações nomeadas e podem ser interpretadas apropriadamente pela ferramenta de modelagem.

Service time

Service time é o tempo de serviço de um recurso ativo, definido enquanto a demanda da execução de um *Steps* está ativa e hospedando o recurso.

2.4.4 Exemplo de modelagem utilizando a UML SPT Profile

Nessa seção será demonstrado um modelo de um sistema UML com as especificações de desempenho, utilizando para isto o *UML SPT Profile*. O sistema a ser modelado será um programa gerenciador de contratos, tento em vista que um contrato neste caso possui uma grande quantidade de dados, sendo necessário vários acessos ao banco de dados. A figura 2.2 apresenta o diagrama de caso de uso do sistema.

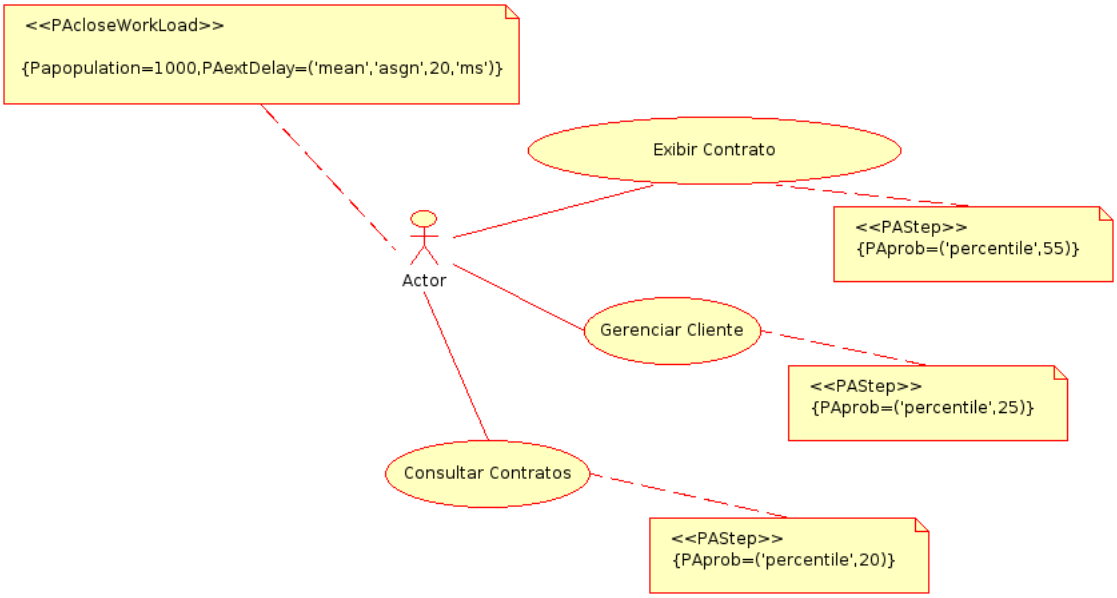

Figura 2.2: Diagrama de caso de uso exemplo

Foi definido que a carga de usuários é fechada com 100 usuários utilizando o sistema, para especificar essa característica foi inserido no ator o estereótipo *PAcloseWorkLoad*, onde a *tag Papopulation=100* indica que o sistema terá uma população fechada de 100 usuários e tag *PaextDelay=('mean','asgn',20,'ms')* indica que o tempo de utilização de cada usuário no sistema é em média 20 min.

Os casos de uso possuem o estereótipo *PAStep*, o qual contem a *tag* de *Paprob* que indica a probabilidade destes acorrerem. Nas métricas desse exemplo foi definido que o caso de uso *Exibir contrato* possui a maior probabilidade de ocorrer, um vez que este tem com valor de *Paprob* igual 0.55, o que representa que há 55 % de chance do cliente executar este caso de uso.

A figura 2.3 apresenta o diagrama de classes do sistema exemplo, para fins de simplicidades foi omitido algumas classes. Na classe *DAOContratos* foi especificado que método *pesquisaContrato(cliente cliente)*, deve responder em 85% de suas requisições em menos de 300 ms. Para informar essa restrição ao método foi utilizado o estereótipo *PAStep*, na qual possui na tag *ParespTime* que representa o tempo de resposta, onde especificado os valores

requeridos.

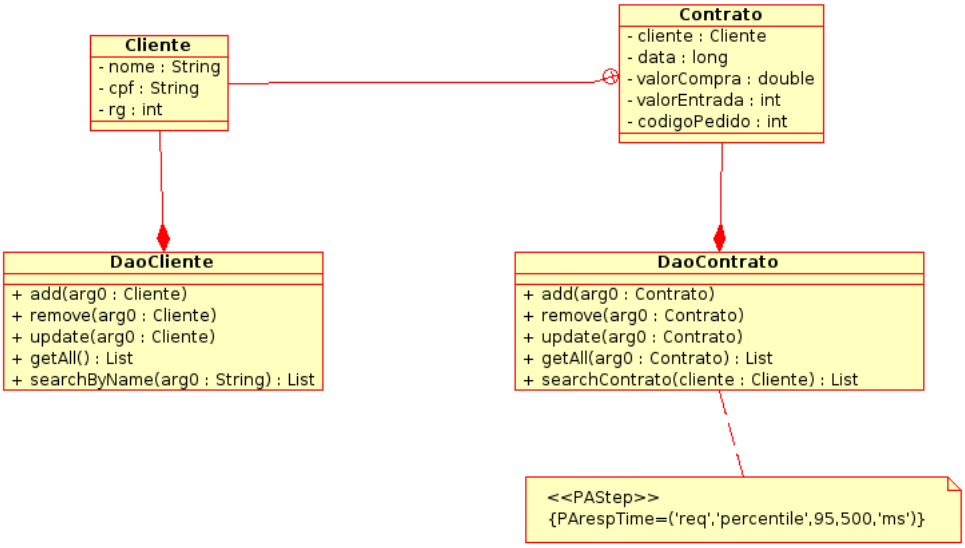

Figura 2.3: Diagrama de classes contendo as métricas de desempenho

2.5 Data Access Object

O DAO é considerado um padrão de projeto pela *Java 2 Platform, Enterprise Edition* (J2EE), que tem a finalidade de extrair encapsular todos os acessos à origem de dados. O DAO gerencia a conexão com a origem de dados para obter e armazenar dados. A origem de dados poderia ser um armazenamento persistente como uma conexão RDBMS, um repositório com um banco de dados LDAP ou um serviço de negócios, acessado através do CORBA Internet Inter-ORB Protocolo (IIOP) ou soquetes de baixo nível (Aiur John Crupi 2002).

O componente de negócio que conta com o DAO utiliza a interface mais simples exposta pelo DAO para seus clientes. Quando a implementação da origem de dados subjacente se altera, a interface de dados com o cliente exposto pelo DAO não se altera. Esse padrão permite que o DAO se adapte a esquemas diferentes de armazenamento sem afetar seus clientes ou componentes de negócios. Essencialmente, o DAO age como um adaptador entre os componentes e a origem de dados.

Muitas arquiteturas MVC em sua camada de negócio no modelo de dados utilizam o padrão DAO para separa sua tecnologia. Esse trabalho enfoca o padrão DAO como um componente para ser testado, levando em consideração a importância deste.

3 *Especificação do Sistema*

Atualmente na indústria de software, não existem muitos aplicativos voltados para o teste de desempenho em componentes de software. A reutilização de componentes de software no processo de desenvolvimento de uma aplicação hoje em dia é muito grande, isto devido ao baixo custo e ao pouco tempo para deixá-lo operável. Um dos principais componentes empregado nas aplicações é o componente DAO. Garantir um bom desempenho para o componente DAO é essencial para o desempenho geral do sistema que vai utilizá-lo.

Neste contexto, a proposta desse trabalho é construir uma ferramenta capaz de gerar drivers de teste para testar o desempenho de componentes de software, com o enfoque no componente DAO. A ferramenta será capaz de gerar os testes em fase de projeto de software, ou seja, os testes estarão prontos antes mesmo do sistema em construção estar implementado.

Os casos de teste serão gerados com base nas especificações de requisitos de desempenho inseridas no modelo UML. Para isto utilizaremos o UML SPT Profile, que permitirá coletar as informações de desempenho esperado para o software. Os drives de teste serão gerados automaticamente para Apache JMeter onde serão executados gerando uma saída que apresentará o desempenho do componente.

A figura 3.1 apresenta o funcionamento do sistema, mostrando o fluxo de entradas e saídas do sistema proposto. Os componentes de software são modelados utilizando um editor UML com suporte a versão UML 2.0 como, por exemplo, Rational Rose (IBM 2007), Argo UML (ArgoUML 2007), Omondo UML2 plug-in para Eclipse (Omondo 2007). Em nosso trabalho utilizamos o *Umbrello UML Modeller* (Modeller 2007).

Os requisitos de desempenho são especificados utilizando o UML SPT profile; os dados da modelagem são exportados em um arquivo XMI que os descrevem. A aplicação **ComponUp** (trabalho proposto) irá importar o arquivo XMI e colocar os dados necessarios em uma estrutura de dados. Utilizando um algoritmo de geração de rede de petri que será apresentado na seção 4.6 os dados selecionados serão convertidos para uma rede de petri na qual será executada gerando uma simulação de requisições ao sistema.

Os resultados da simulação serão convertidos para casos de teste, logo após, estes serão convertidos em um driver de teste para o *Apache JMeter* (apresentado na seção 2.2.2), na qual irá executar os testes sobre sistema em teste, resultando assim a avaliação de desempenho dos componentes, de acordo com os requisitos de desempenho especificados.

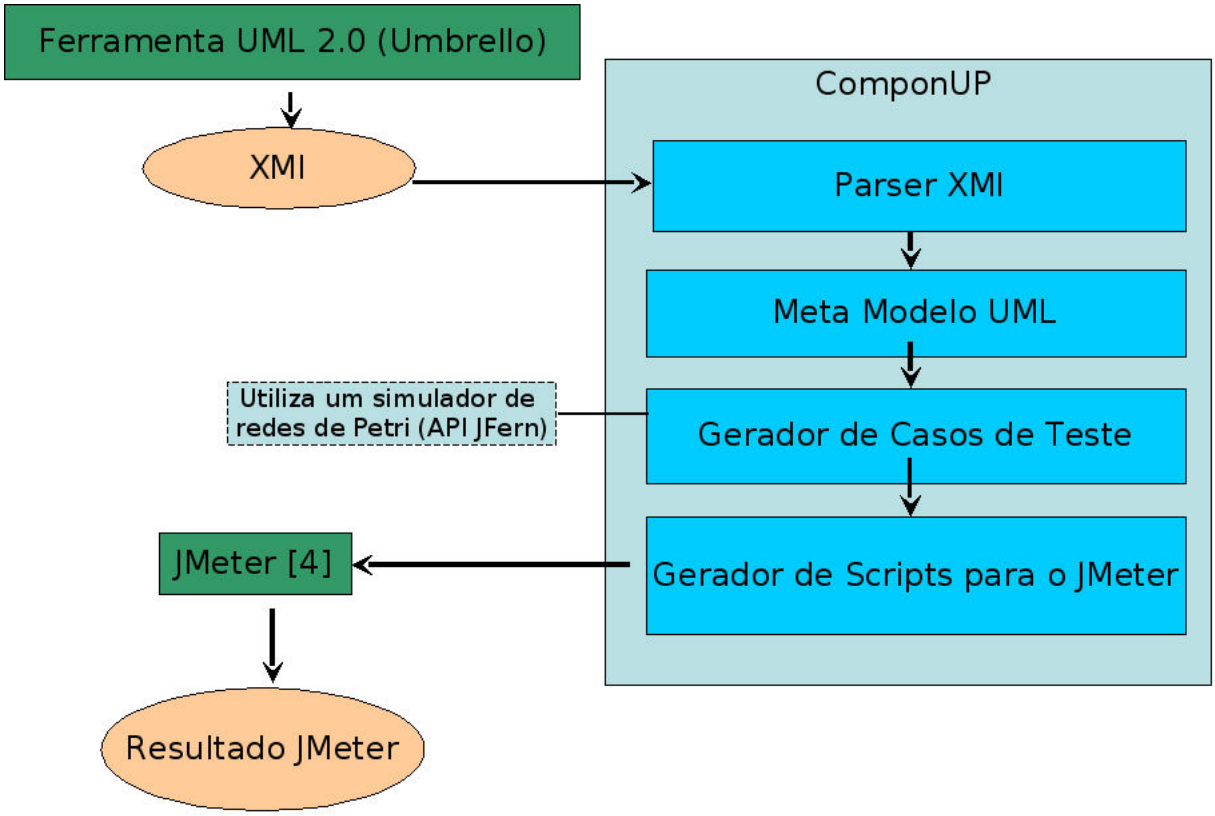

Figura 3.1: Funcionamento do sistema

3.1 Diagrama de caso de uso

Na figura 3.2 pode-se ver o diagrama de caso de uso, onde é apresentado somente um caso de uso que é *Testar desempenho do sistema*. O sistema possui um caso de uso apenas, pois, a interface com usuário é simples, onde o usuário só precisa especificar o arquivo XMI de entrada e o sistema se encarregara de construir os testes e criar o script para o JMeter.

Figura 3.2: Diagrama de caso de uso

4 Projeto

Nesta seção será explicado o projeto do sistema **ComponUP**. Primeiramente será revisto o fluxograma de entradas e saídas da ferramenta, logo após será apresentado os diagramas de atividades contendo as atividades do sistema e do usuário. Em seguida será explicado o diagrama de pacotes com responsabilidade de cada pacote que a ferramenta possui. Junto a explicação das responsabilidades dos pacotes serão apresentados os diagramas de classes destes. Por fim, serão exibidos os diagramas de seqüencia que mostrarão interação entre as classes da ferramenta.

4.1 Fluxograma de entradas e saídas do sistema

As entradas e saídas da ferramenta podem ser visualizadas na figura 4.1. Basicamente o sistema possui como entrada um arquivo XMI que representa o modelo UML que contém os componentes a serem testados. A saída do sistema é um arquivo XML que representa um driver para a *engine* de execução dos testes JMeter, neste driver de teste estarão contidos os testes criados pelo **ComponUp**.

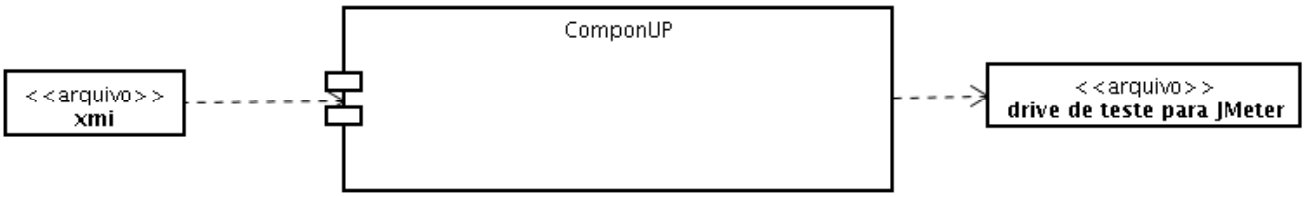

Figura 4.1: Entrada e saída do ComponUP

4.2 Diagramas de atividades

O diagrama de atividades do caso de uso *Testar desempenho dos componentes* é representado pela figura 4.2 e a figura 4.3, as quais mostram todas as atividades do sistema e do usuário. Na figura 4.2 as atividades do usuário estão em apenas informar o arquivo de entrada e o local do arquivo de saída; as principais atividades do sistema estão em:

- Fazer o *parsing* do arquivo de entrada; nesta atividade o sistema deve ler o arquivo XMI e selecionar as informações necessárias.
- Validar semanticamente as informações do modelo. Atividade na qual o sistema faz uma verificação dos dados coletados na atividade anterior; verificando se todos os casos possuem informações de taxa de chegada e probabilidade, se cada caso de uso possui diagramas de seqüencia; entre outras verificações que serão investigadas mais afundo na continuação deste trabalho.
- Gerar a rede de Petri; nesta atividade o sistema deve gerar a rede de Petri a partir do das informações coletadas e validadas.
- Executa a simulação da rede de Petri gerando os casos de teste com o resultado desta. Nesta atividade o sistema irá simular a rede criada, de acordo com a taxa de chegada e probabilidades especificadas; utilizando para isto o JFern. Com o resultado passo a passo da simulação os casos de teste serão gerados.
- Com os casos de teste prontos, o último passo é gerar o script para o JMeter. A atividade do sistema *Gera redes de Petri* pode ser vista na figura 4.3; nela é apresentado com mais detalhes como será gerado as redes de Petri.

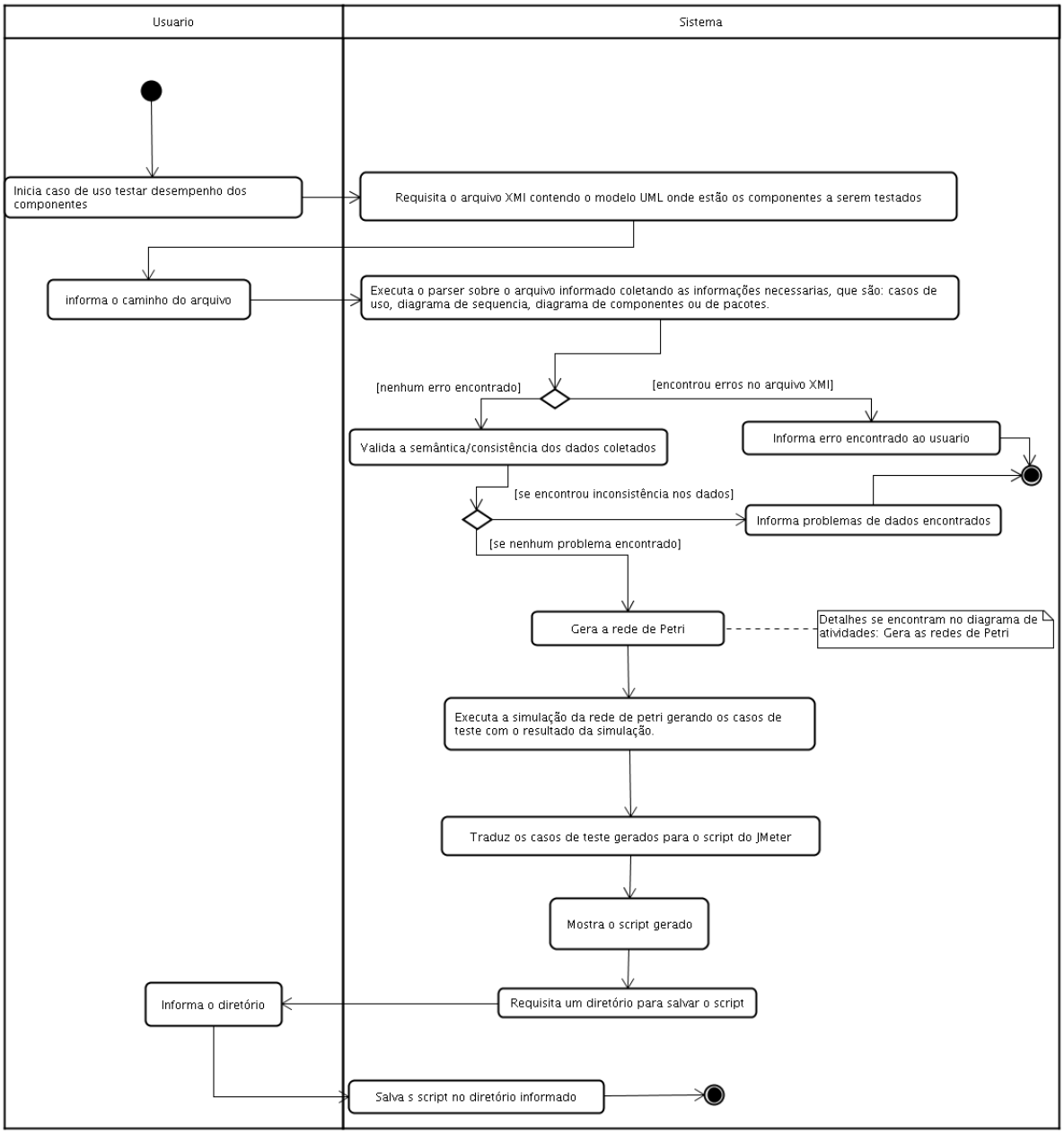

Figura 4.2: Diagrama de atividades

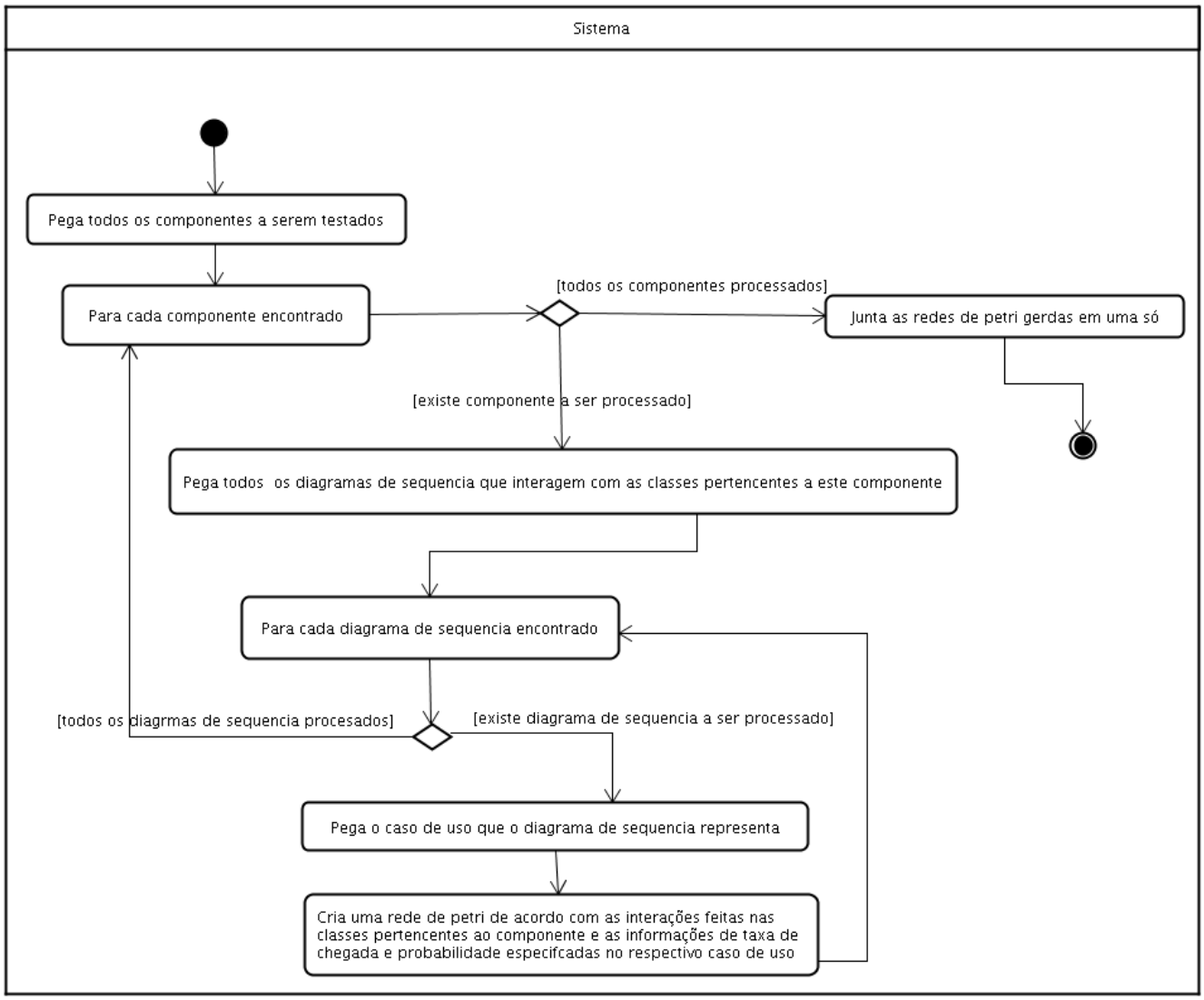

Figura 4.3: Diagrama de Atividades de *Gera redes de Petri*

4.3 Diagramas de pacotes e classes

Nessa seção serão apresentados os pacotes do sistema junto com a responsabilidade de cada um em relação a as atividades que o sistema deve cumprir. A figura 4.4 apresenta os pacotes que o sistema possui, assim como a dependência de cada pacote em relação aos outros.

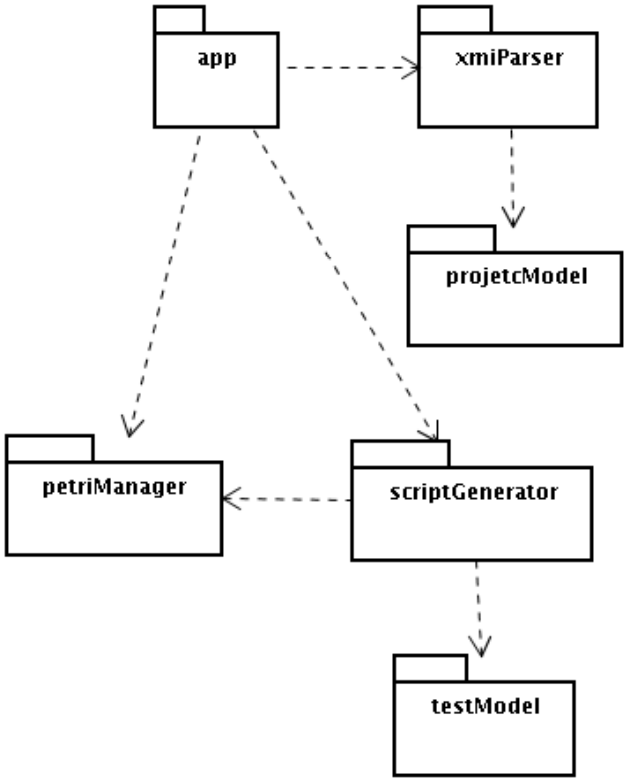

Figura 4.4: Diagrama de pacotes

4.3.1 Pacote *project model*

O pacote *project model* é a estrutura de dados que representa os dados do modelo UML. Como não são utilizados todos os elementos da UML, na estrutura de dados, só são representados os elementos necessários para a geração dos testes. A figura 4.5 mostra as classes que compõem o pacote *projectModel*. Alguns métodos e atributos menos importantes foram suprimidos do diagrama para fins de simplicidade.

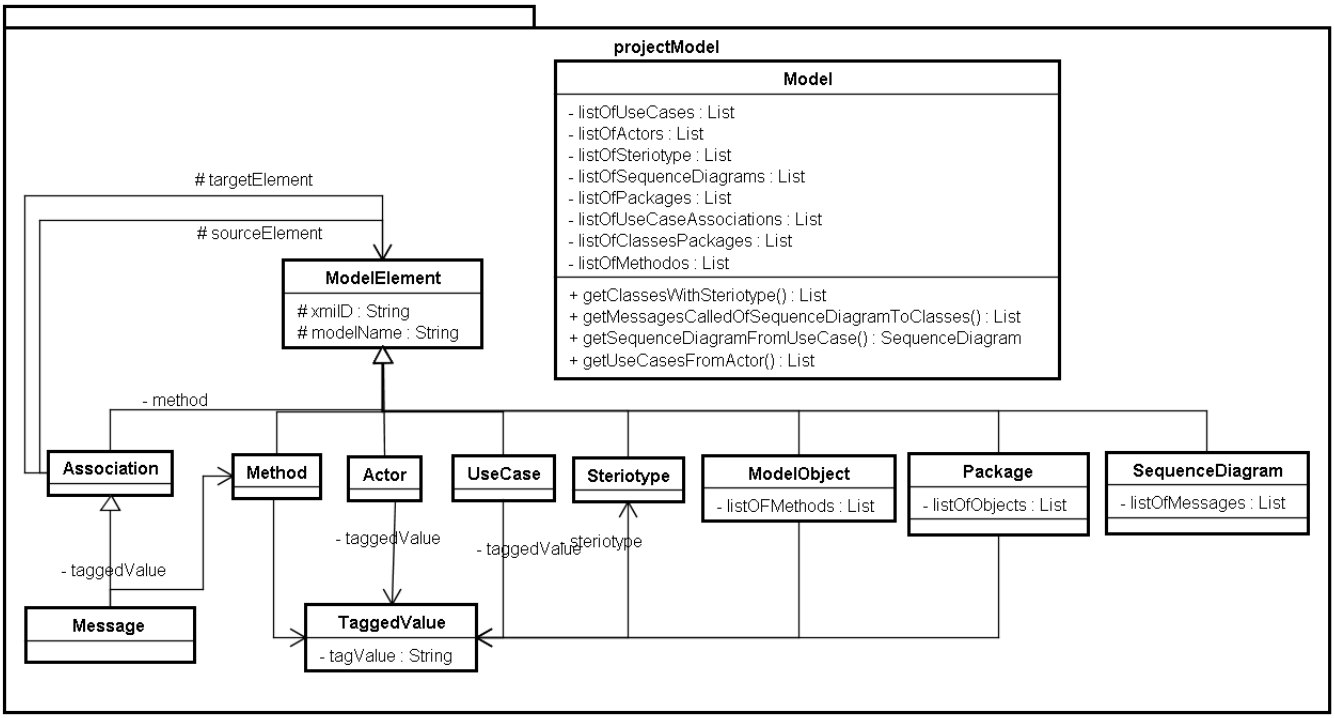

Figura 4.5: Diagrama de classes do pacote *projetcModel*

- **TaggedValue:** Classe que armazenará o valor de uma determinada tag, declarada em algum estereótipo.
- **ModelElement:** Super-classe responsável por armazenar informações genéricas de cada entidade do modelo.
- **UseCase:** Classe responsável por armazenar as informações referentes a cada caso de uso do modelo UML
- **Association:** Classe responsável por armazenar as ligações entre duas entidades do modelo. Exemplo: a associação entre um ator e um caso de uso.
- **Actor:** Classe que armazenar as informações referentes a cada ator do modelo UML.
- **SequenceDiagram:** Classe que representa um diagrama de seqüência do modelo UML e possui uma lista de mensagens.
- **Message:** Classe responsável por representar uma mensagem de interação no diagrama de seqüência, possui uma referência ao método que será acionado.
- **ModelObject:** Classe responsável por representar uma classe do modelo UML.
- **Method:** Classe que representa um método de uma classe.

- **Stereotype:** Classe que armazenará informações sobre cada estereótipo declarado no modelo.

- **Package:** Classe que representa um pacote de classes, possui uma lista de objetos e um estereótipo.

- **Model:** Representa o modelo do projeto, que guardará todas as listas de informações do projeto, que são: uma lista de casos de uso,uma lista de atores,uma lista de estereótipos, uma lista de diagramas de seqüência, uma lista de pacotes, uma lista de associações de casos de uso, uma lista de classes dos pacotes e por fim uma lista de métodos das classes.

4.3.2 Pacote *xmiParser*

O pacote *XMIParser* é responsável por traduzir os dados necessários do arquivo XMI para a estrutura de dados do pacote *projectModel*. A figura 4.6 mostra as classes que compõem o pacote *xmiParser*. Alguns métodos e atributos menos importantes foram suprimidos do diagrama para fins de simplicidade.

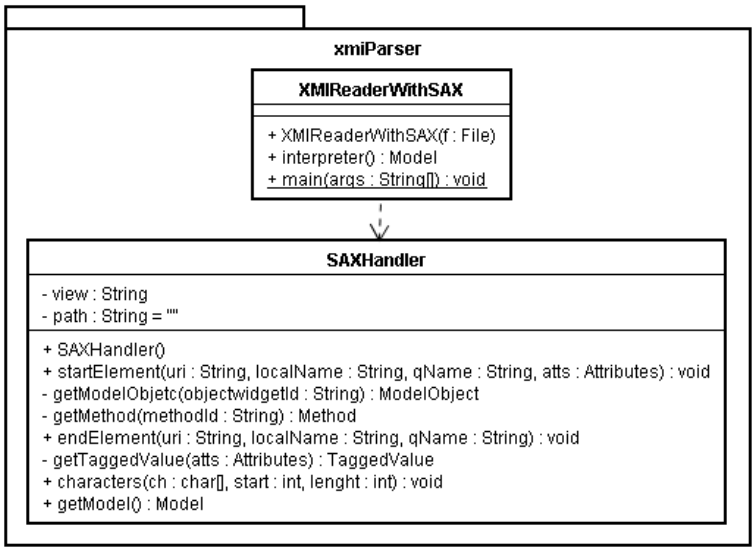

Figura 4.6: Diagrama de classes do pacote *xmiParser*

- **SAXHandler:** Classe que faz o parser do arquivo XMI de entrada.

- **XMIReaderWithSAX:** Classe que manipula a criação do *SAXHandler* com as informações necessárias.

4.3.3 Pacote *petri manager*

Petri manager é o pacote que contém todas as classes para a criação, manipulação e execução dos modelos de simulação de redes de Petri. A figura 4.7 mostra o diagrama de classes desse pacote.

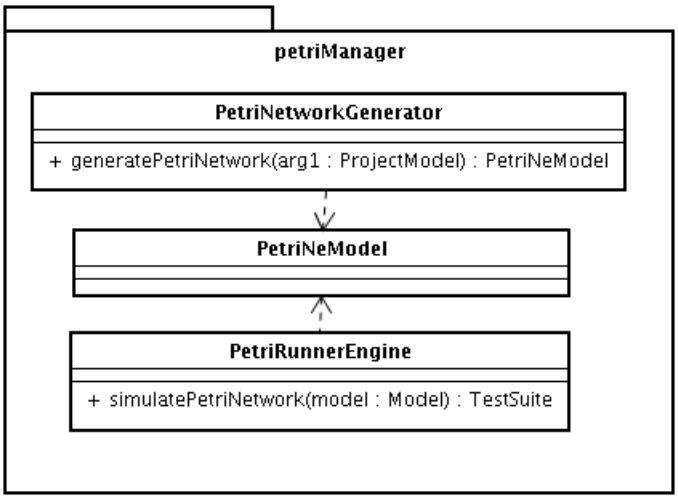

Figura 4.7: Diagram de classes do pacote *petriManager*

- **PetriNetModel:** Representa o modelo da rede de Petri, manipula as estruturas de dados do *JFern*, esta classe ainda esta em fase de análise podendo sofrer modificações.
- **PetriNetworkGenerator:** Estrutura responsável por criar uma *PetriNetModel* a partir das informações contidas no modelo do projeto.
- **PetriRunnerEngine:** Classe que executa a simulação a partir de uma *PetriNetModel*, utilizando o *JFern*.

4.3.4 Pacote *testModel*

O pacote *fig:testModel* é a estrutura que representara os teste a ser executado no sistema em teste. Esta estrutura intermediaria foi desenvolvida para se obter uma maior flexibilidade no futuro; uma vez que a partir desses testes genéricos podem-se gerar drives (scripts) de teste para outros *engines* executores e não só para o JMeter. A figura 4.8 mostra o diagrama de classes desse pacote, esta estrutura é bem básica podendo sofrer modificações na implementação do sistema. As seguintes classes estão presentes no diagrama:

- **TestCase:** Representa um caso de teste, com as informações para o acesso a classe a ser testada.
- **TestSuite:** Classe que contem uma composição de TesteCases ordenados. Esta classe representa teste com um todo.

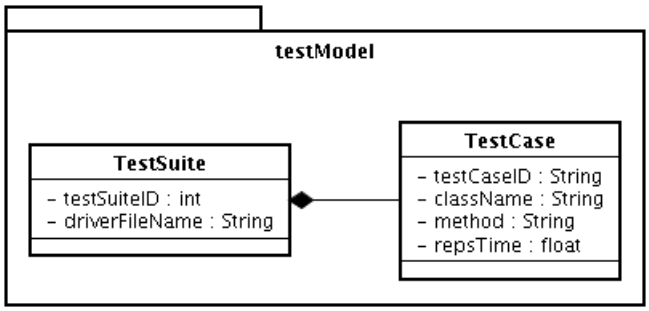

Figura 4.8: Diagrama de classes do pacote *testModel*

4.3.5 Pacote *scriptGenerator*

O pacote *script generator* contém apenas uma classe que é a JMeterScriptGenerator. Esta classe é responsável em traduzir os testes gerados e representados pela classe *TestSuite* para um arquivo com extensão JMX que é o XML de entrada aceito pela *engine* de teste *JMeter*.

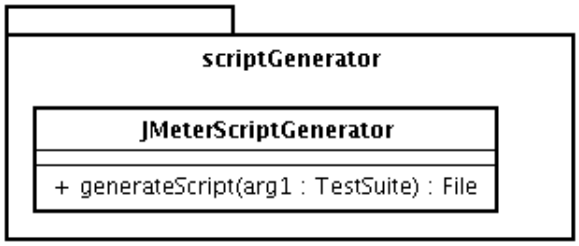

Figura 4.9: Diagrama de classe do pacote *scriptGenerator*

4.4 Diagramas de Seqüência

A figura 4.10 apresenta o diagrama de seqüencia do caso de uso *Testar desempenho do componentes*, neste diagrama pode-se visualizar a interação entre as classes do sistema. Basicamente o processo do sistema está em:

- **(iteração 1.1)** receber o caminho do arquivo XMI de entrada.
- **(iteração 2.1)** ler o arquivo XMI informado e converter os dados necessários encontrados neste para a estrutura *projetcModel*.
- **(iteração 3)** validar o contexto do modelo de projeto, verificando se os dados estão coerentes ; caso os dados não estejam coerentes é lançado uma exceção e é finalizada a execução do sistema.
- **(iteração 4)** pegar o modelo de projeto gerado.
- **(iteração 5)** gerar o modelo da rede de Petri a partir do modelo de projeto.
- **(iteração 6)** com o modelo da rede de Petri gerado o próximo passo é executar a simulação deste a fim de construir os testes.
- **(iteração 7)** depois do modelo de teste gerado este é convertido para um XML da engine JMeter.
- **(iteração 8)** depois de gerado o script este é apresentado em uma tela e é requisitado o local para salvar o arquivo contendo o script.
- **(iteração 9)** o usuário informa o diretório destino onde o script será salvo.
- **(iteração 10)** é salvo o arquivo contendo o script de teste no diretório informado.

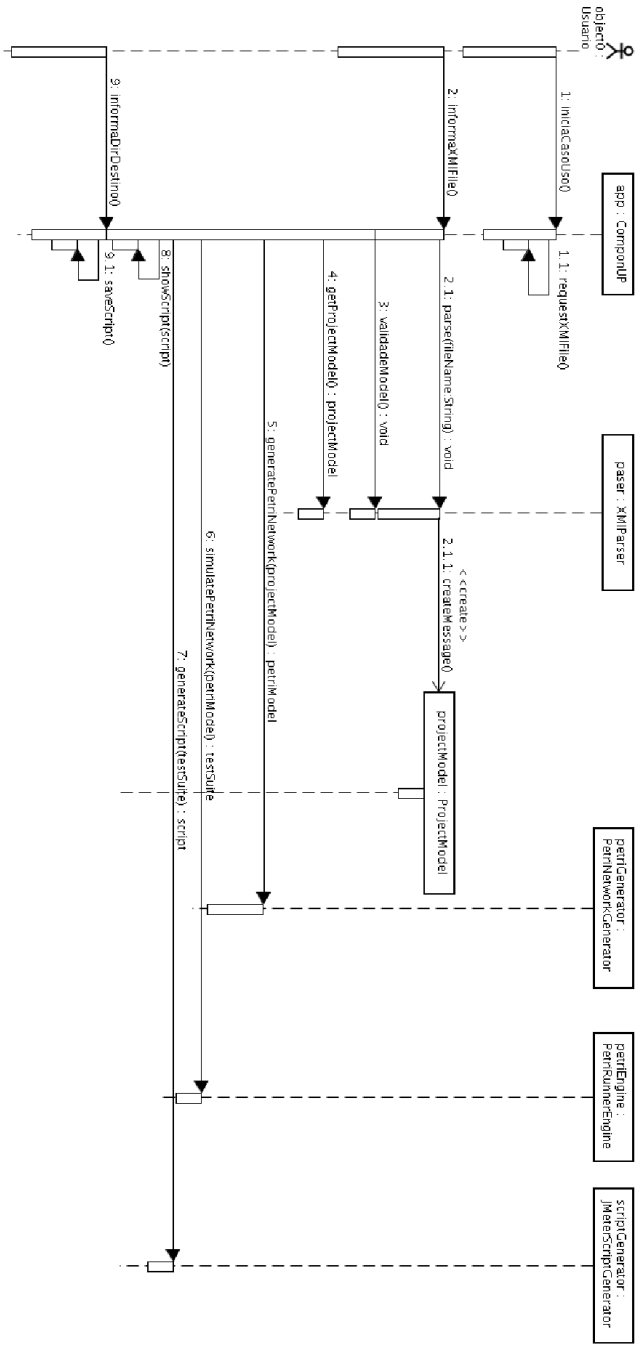

Figura 4.10: Diagrama de seqüência do caso de uso *Testar desempenho do componentes*

4.5 Caso de estudo

Nesta seção será apresentado um sistema simples na qual servirá como caso de estudo para demonstrar características da implementação assim como apresentação dos resultados obtidos.

O sistema que será demonstrado é um programa gerenciador de e-mails, para fins de simplicidade será suposto que os usuários do sistema já estão cadastrados e logados no sistema. Serão apresentados os seguintes diagramas: diagrama de caso de uso, diagrama de classes, diagramas de seqüência.

Antes de começar a especificar os estereótipos e tags no modelo UML é necessário possuir as especificações de desempenho do sistema, as seguintes especificações foram escolhidas para serem colocadas no diagrama de caso de uso.

- O sistema terá um carga fechada de 1000 usuários
- Cada usuário permanecerá no sistema em média 20 min
- O sistema terá três casos de usos com as seguintes probabilidades de execução

  Caso de uso: *Verificar E-mail* com 60%

  Caso de uso: *Redigir E-mail* com 25%

  Caso de uso: *Deletar E-mail* com 15%

A figura 4.11 apresenta o diagrama de caso de uso, neste caso o ator possui o estereótipo *PAcloseWorkLoad*, onde a *tag* de *Papopulation=1000* indica que o sistema terá uma população fechada de 1000 usuários e tag *PaextDelay=('mean','asgn',20,'ms')*indica que o tempo de utilização de cada usuário no sistema é em média 20 min. Os casos de uso possuem o estereótipo *PAStep*, o qual contem a *tag Paprob* que representa a probabilidade destes acorrerem.

Nas métricas desse exemplo foi definido que o caso de uso *Verificar e-mail* possui a maior probabilidade de ocorrer, uma vez que este tem valor de *Paprob* igual 0.60, o que representa que há 60 % de chance do cliente executar este caso de uso.

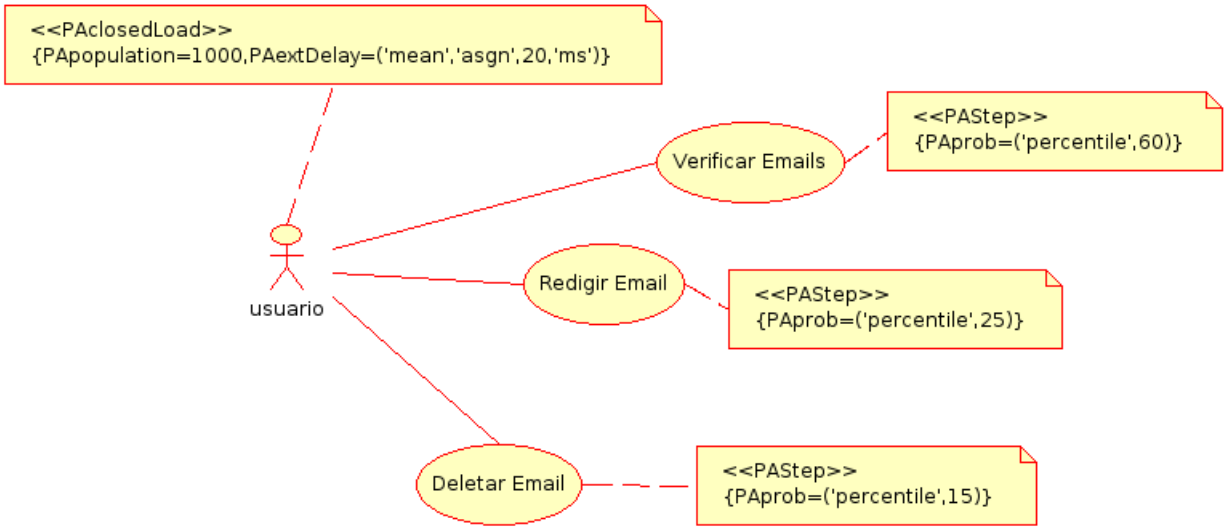

Figura 4.11: Diagrama de caso de uso exemplo

Após indicado as métricas de demanda de usuário, as probabilidades dos casos de uso e outras citadas acima, o próximo passo é especificar os requisitos de desempenho nos principais métodos das classes escolhidas. Os requisitos são valores supostos, estimados ou medidos. Neste exemplo a principal classe a ser testada é a classe *DAOUsuarioEmail*, devido a sua importância já discutida na seção 2.5. Nos métodos da classe *DAOUsuarioEmail* os requisitos são:

- No método *getAllEmails(...)* é requerido que em 95 % das requisições e o sistema responda com uma média de tempo de 500 ms.
- No método *addEmail(...)* é estimado que o sistema responda com uma média de tempo de 10 ms.
- No método *delEmail(...)* é estimado que o sistema responda com uma média de tempo de 5 ms.

A figura 4.12apresenta as especificações dos requisitos, onde foi atribuído ao método *getAllEmails(...)* o estereótipo *PAStep* com valor de tag que representa a sua métrica exigida.

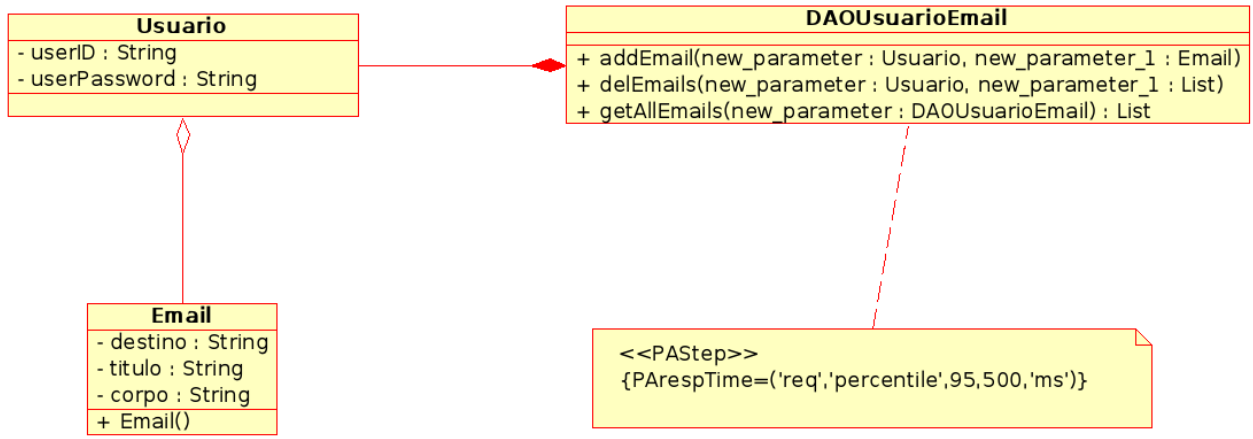

Figura 4.12: Diagrama de classes contendo as métricas de desempenho

O diagrama de seqüência para o caso de uso *Verificar Email* é demonstrado pela figura 4.12, onde é possível verificar as iterações com a classe *DAOUsuarioEmail* na qual foi especificado os requisitos.

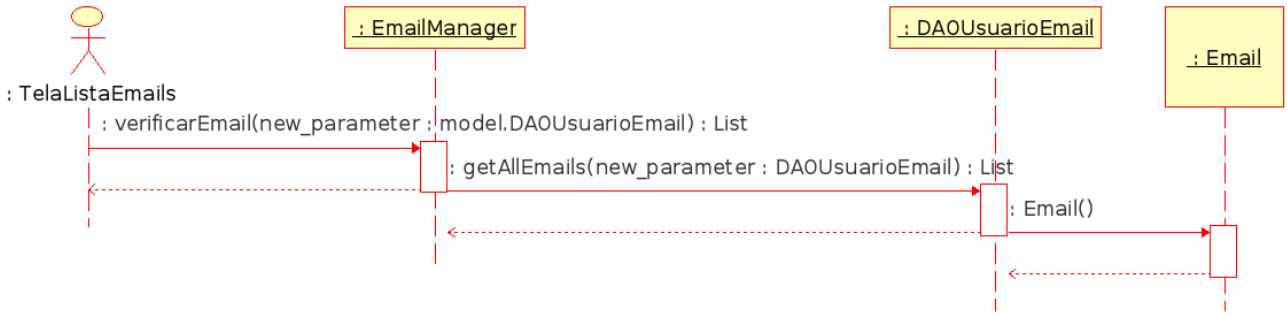

Figura 4.13: Diagrama de seqüência do caso de uso *Verificar e-mail*

4.6 Algoritmo de geração da rede de Petri

Nessa seção será descrito em forma algorítmica o processo automático de criação da rede de Petri a partir do modelo UML lido na etapa de parser. A figura 4.14 demonstra a rede criada pelo algoritmo para o modelo exemplo apresentado na seção 4.5, os passos para criação podem ser visualizados na figura 4.15:

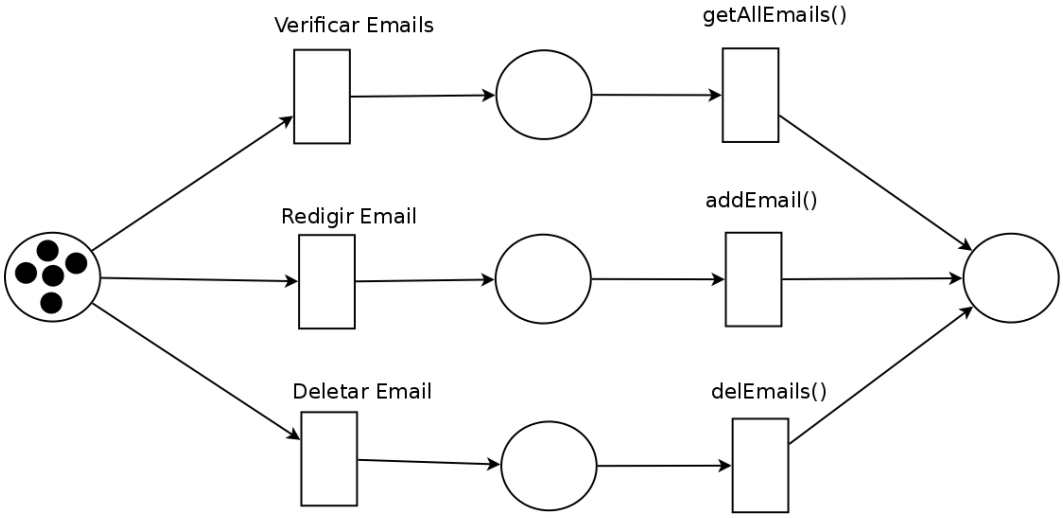

Figura 4.14: Rede de Petri gerada

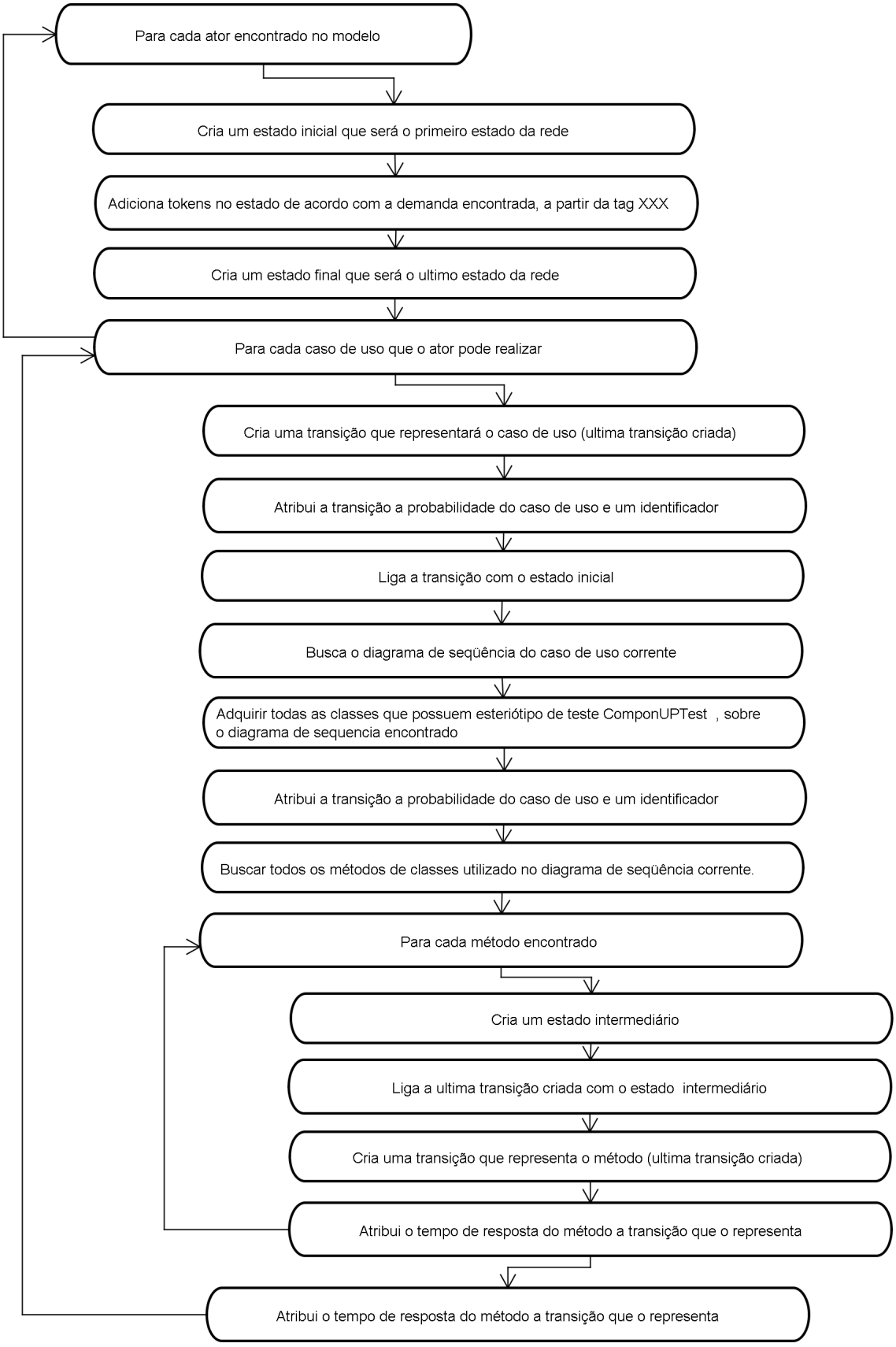

Figura 4.15: Algoritmo de geração de testes

5 *Implementação*

5.1 Editor UML utilizado

Antes do começar a modelar o sistema é necessário escolher um editor UML, o editor escolhido para esse trabalho foi o *Umbrello UML Modeller* (Modeller 2007). Este editor foi escolhido depois de realizada uma pesquisa sobre outros editores livres existentes; no final da pesquisa o *Umbrello* foi o que atendeu a todos os requisitos necessários, que foram:

- suportar UML 2.0;
- possibilidade de exportar XMI;
- permitir especificar estereótipos nos casos de uso, pacotes, classes e métodos de classes;
- permitir colocar valores de tags para os estereótipos;

5.2 Implementação do parser

O parser foi desenvolvido utilizando a biblioteca *SAX* (XML 2007), o qual vem junto com o Java 1.4, o SAX é um conjunto de interfaces que devem ser implementadas para efetuar o *parsing* de arquivos *XML*. No *ComponUP* o arquivo de entrada é um *XMI*, que como já mencionado é um *XML* para representar diagramas e objetos de um modelo *UML*.

Inicialmente antes de codificar o parser foi preciso identificar onde as informações necessárias para execução do sistema localizavam se no XMI. Como já mencionado para esse trabalho é preciso dos seguintes diagramas com seus respectivos elementos:

- **Diagrama de Caso de Uso:** No diagrama de caso de uso é importante obter os casos de uso junto com suas probabilidades de execução e a demanda de usuários no sistema na qual é atrelada ao ator.

- **Diagrama de Seqüência:** Para cada caso de uso encontrado no diagrama de seqüência deve-se obter o seu respectivo diagrama de seqüência, onde será possível identificar quais classes que estão sendo utilizadas por cada caso de uso.
- **Classes do model:** As classes do modelo são necessárias para identificar quais as classes que serão testadas e seus respectivos métodos.

Após identificar as informações no *XMI*, o próximo passo foi programar a estrutura de dados que comportará os elementos lidos pelo parser, citada na seção 4.3.1. Com a estrutura implementada o parser já pode ser desenvolvido e testado.

5.3 Implementação da geração de rede de Petri

A geração da rede de Petri foi implementada seguindo o algoritmo definido na seção 4.6, foi uma das implementações mais complicadas, para ajudar o desenvolvimento do algoritmo foram criados métodos auxiliares, tais como: buscar diagrama de seqüência de um determinado caso de uso, buscar classes que estão sendo utilizadas em um determinando diagrama de seqüência, verificar quais classes que possuem um determinado estereótipo, entre outros.

5.4 Implementação da simulação da rede de Petri e geração dos casos de teste

Para realizar o desenvolvimento da simulação foi necessário um estudo sobre a API de simulação JFern citada na seção 2.3.6. Esta etapa do trabalho foi uma que mais consumiu tempo de desenvolvimento, devido a dificuldades de encontrar documentação e exemplos de utilização desta.

Na criação do núcleo de simulação houve a necessidade de adicionar suporte a probabilidades nas transições da rede, pois esta funcionalidade não está implementada na API do JFern.

Depois de criado o núcleo de simulação seguindo a API do JFern, foi implementado a estrutura de teste demonstrada na seção 4.3.4, para seja possível a conversão da simulação em casos de teste.

5.5 Implementação da geração de script para o JMeter

Após a simulação da rede de Petri, que definirá os casos de testes para a aplicação alvo na estrutura de dados que os representa, é feita a geração do script (.jmx) a ser utilizado pelo JMeter para que possa-se colher os resultados dos testes.

Inicialmente, optou-se por utilizar o *sampler Java Request* (que pode ser visualizado na figura 5.1) disponível no JMeter para carregar e executar um determinado método de uma classe java. Na prática isto não foi possível, pois se percebeu que o *sampler Java Request* era apenas uma interface entre a *engine* do JMeter e uma aplicação externa, possibilitando uma comunicação de baixo nível entre eles. Tendo percebido que não seria possível utilizar o sampler citado, e não encontrando outro *sampler* na ferramenta que atendesse as necessidades deste projeto, foi imperativo que se criasse um novo sampler para que a proposta deste trabalho fosse atingida.

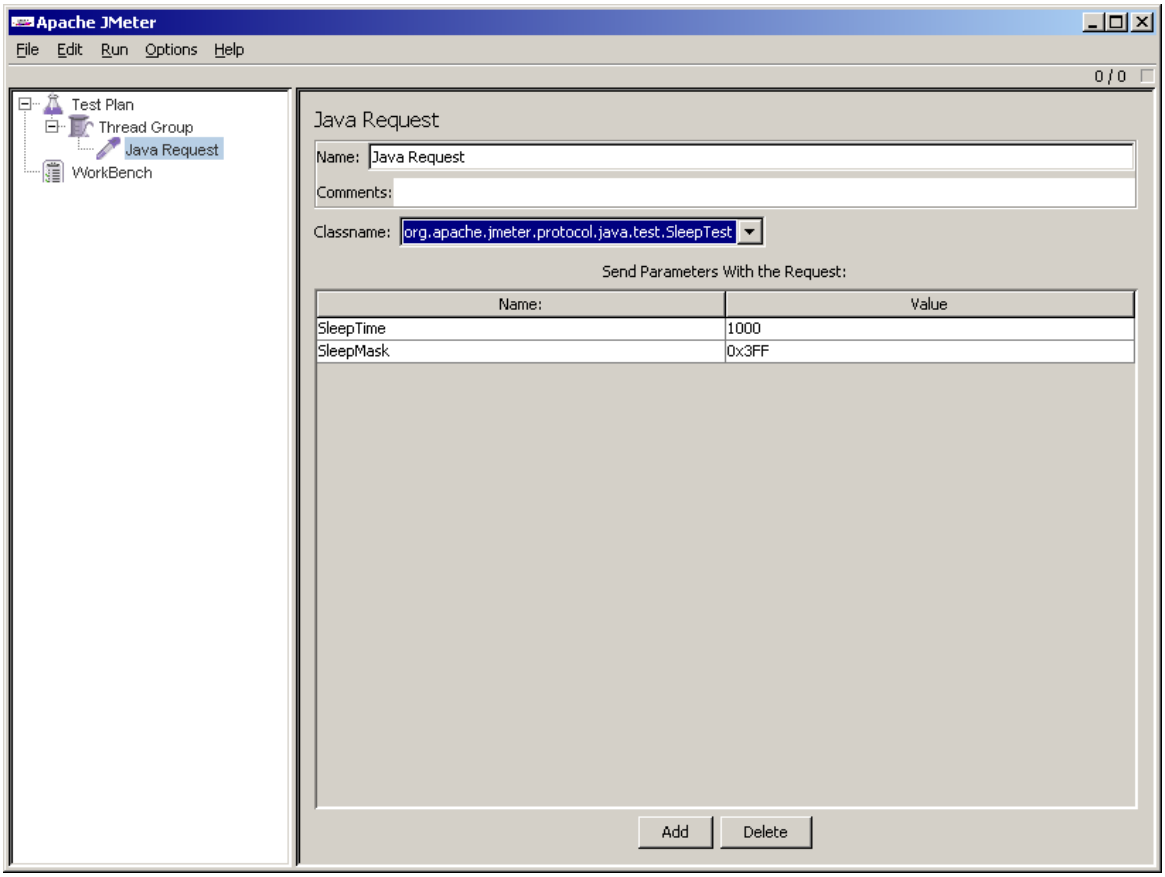

Figura 5.1: *JMeter Sampler Java Request*

5.5.1 ComponUP Sampler

Para atender as necessidades deste projeto, foi criado um novo sampler que é responsável por carregar as classes java e permitir que sejam executados os seus métodos. A proposta do *ComponUP Sampler* é simples e objetiva, ele permite carregar uma classe java e executar um determinado método desta, verificando se o tempo de execução deste método foi menor ou maior que o esperado. Para isso, deve ser informado como parâmetro para o *ComponUP Sampler*, uma tupla na seguinte forma: *[classe, método, tempo de resposta máximo]*, como pode ser visto na figura 5.2. No momento da execução do teste, será verificado se o método chamado ultrapassou o tempo máximo esperado para a sua execução, caso isso ocorra, o teste para este método falha.

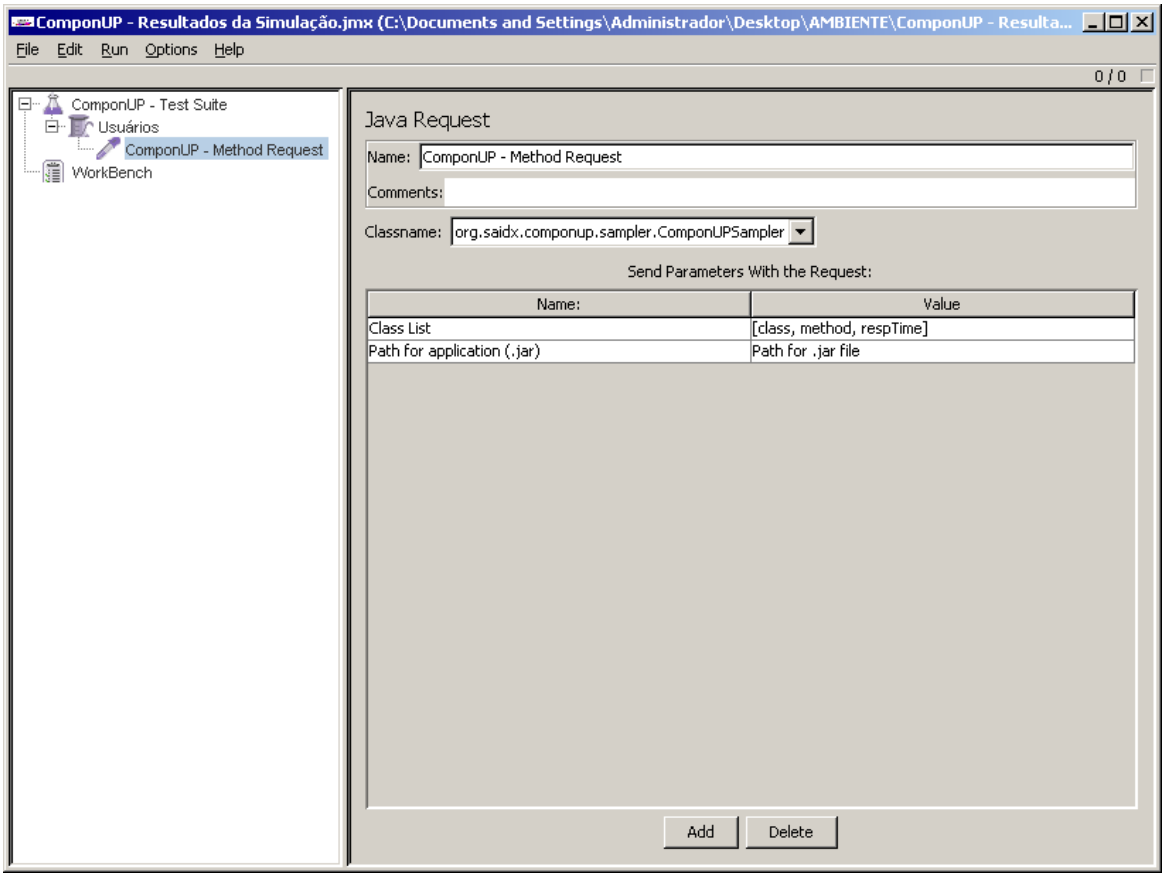

Figura 5.2: *ComponUP Sampler*

No script gerado para o JMeter, foi adicionado um *listener* do tipo *View Results in Table* que informa se um determinado teste passou ou falhou. A representação de um teste neste listener é feito através de uma linha na sua tabela, onde na coluna *Success* é informado o resultado do teste. Para visualizar esta funcionalidade, foi criado um exemplo onde uma *classe Test*, que contém apenas o método *methodTest*, será testada. Este método é executado em exatos dois segundos, logo, para forçar que o teste falhe informamos ao JMeter através do *ComponUP*

Sampler, que ele deve ter um tempo de respota de no máximo 100 ms, como mostrado na figura 5.3. A figura 5.4 mostra que o resultado do teste falhou como já era esperado. Caso aumentássemos o tempo de resposta, no *ComponUP Sampler*, deste método para qualquer valor acima de dois segundos, o resultado do teste seria aprovado.

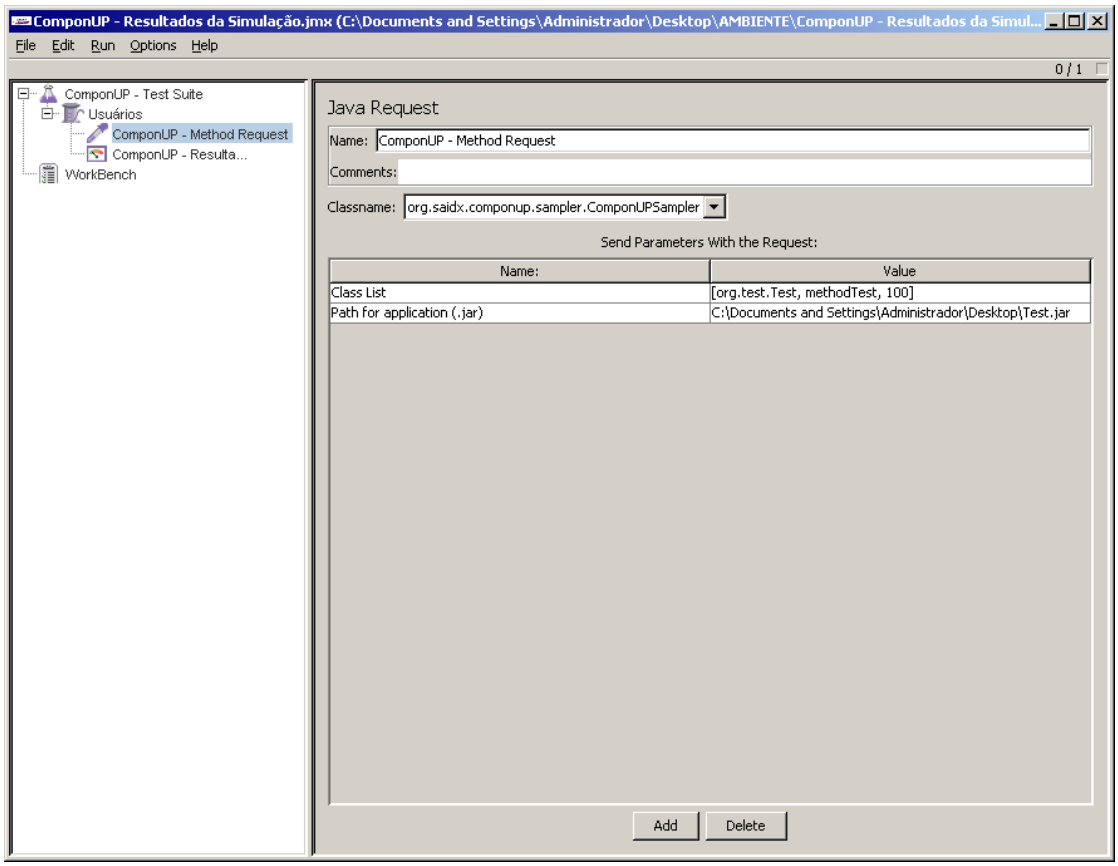

Figura 5.3: Exemplo de teste do *ComponUP Sampler*

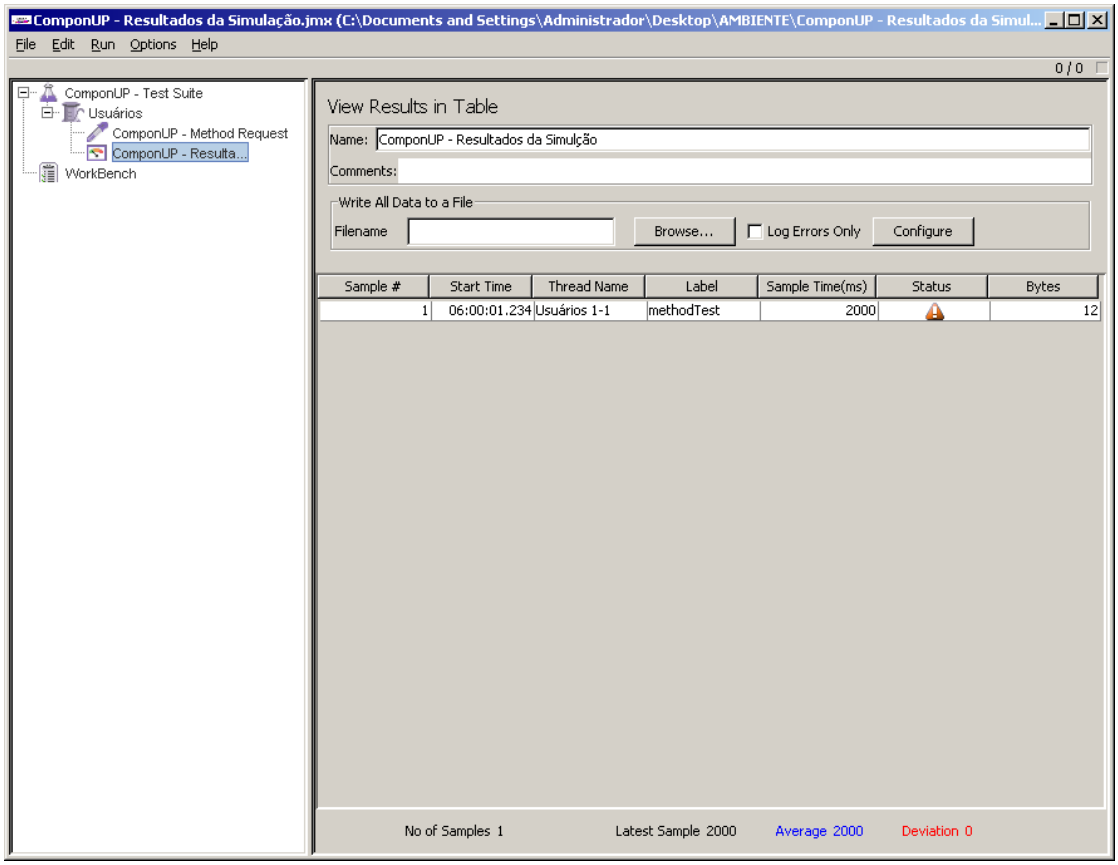

Figura 5.4: Resultado do exemplo de teste do *ComponUP Sampler*

5.6 Exemplo de Execução

Nesta seção será mostrado um exemplo da execução do sistema com o objetivo de demonstrar todos os passos e artefatos que estão presentes na geração do teste de desempenho do ComponUP.

Primeiramente será mostrado o XMI de entrada gerado após a construção do modelo UML do caso de estudo apresentado na seção 4.5, logo após será apresentado um log de simulação, o qual irá demonstrar o comportamento do algoritmo de geração de testes, explicado na seção 4.6, a rede de Petri gerada pela simulação pode ser visualizada na figura 4.14.

Em seguida serão mostrados os testes gerados após a simulação e o Script do JMeter que irá executar os testes. Ao final serão apresentados os resultados da execução estes teste, onde será possível avaliar a desempenho do sistema verificando se o sistema está suprindo os requisitos especificados.

5.6.1 XMI de entrada

Na figura 5.5 pode-se ver um trecho do XMI de entrada que representa as informações da classe *DAOUsuarioEmail* , onde esta possui o estereótipo *ComponUPTest*, a classe possui três operações : *addEmail*, *delEmails* e *getAllEmails*. Nas operações foram atribuídos o estereótipo *PAStep* e no comentário foi adicionado seus valores que serão as especificações de desempenho.

```
<UML:Class stereotype="ComponUPTest" isSpecification="false" isLeaf="false" visibility="public" namespace="LosmtcFTvArl"
xmi.id="NVD3Ao5QJPAO" isRoot="false" isAbstract="false" name="DAOUsuarioEmail" >
  <UML:Classifier.feature>
   <UML:Operation stereotype="PAStep" comment="{PArespTime=('est','mean',10,'ms')}" isSpecification="false" isLeaf="false"
visibility="public" xmi.id="cuasIbtd45eF" isRoot="false" isAbstract="false" isQuery="false" name="addEmail" >
   <UML:Operation stereotype="PAStep" comment="{PArespTime=('est','mean',5,'ms')}" isSpecification="false" isLeaf="false"
visibility="public" xmi.id="6nUNQ7m0zMrH" isRoot="false" isAbstract="false" isQuery="false" name="delEmails" >
   <UML:Operation stereotype="PAStep" comment="{PArespTime=('req','percentile',95,500,'ms')}" isSpecification="false" isLeaf="false"
visibility="public" xmi.id="VJ7WVIv6Z6dq" isRoot="false" isAbstract="false" isQuery="false" name="getAllEmails" >
  </UML:Classifier.feature>
 </UML:Class>
```

Figura 5.5: Trecho do arquivo XMI de entrada

5.6.2 Log de geração da rede de Petri

Na figura 5.6 é possível verificar o log de execução para a geração da rede de Petri, o log é gerado seguindo o algoritmo apresentado na seção 4.6..

```
* Gerando a rede de petri...
 Ator: [usuario] Demanda=[10]
 - Criando estado inicial com os Tokens
 - Adicicionando tokens
 - Criando estado final
        Caso de uso: [Verificar Emails]
        - Criando transicao do caso de uso, probabilidade=0.33
                Diagrama de sequencia: Verificar Emails
                        Menssagem: [getAllEmails]
                        - Criando estado intermediario do metodo
                        - Criando transicao que representara o metodo
                - Lingando ultima transicao criada com o estado final

        Caso de uso: [Redigir Email]
        - Criando transicao do caso de uso, probabilidade=0.33
                Diagrama de sequencia: Redigir Email
                        Menssagem: [addEmail]
                        - Criando estado intermediario do metodo
                        - Criando transicao que representara o metodo
                - Lingando ultima transicao criada com o estado final

        Caso de uso: [Deletar Email]
        - Criando transicao do caso de uso, probabilidade=0.33
                Diagrama de sequencia: Deletar Email
                        Menssagem: [delEmails]
                        - Criando estado intermediario do metodo
                        - Criando transicao que representara o metodo
                - Lingando ultima transicao criada com o estado final
```

Figura 5.6: Log do sistema para geração da rede de Petri

5.6.3 Simulação da rede e geração de testes

A figura 5.7 demonstra o log de simulação da rede de Petri criada na etapa anterior, os resultados da simulação são um conjunto de ações que determinam o acesso aos casos de uso do sistema e o acesso aos métodos que estão presentes em cada caso de uso. Neste caso pode-se perceber que foram acionados dez casos de uso, sendo que para cada caso de uso foi chamado os métodos especificados para teste.

```
* Iniciando Simulacao....
- novo chamada de caso de uso:[Acao --> Redigir Email]
- novo chamada de caso de uso:[Acao --> Deletar Email]
- novo chamada de caso de uso:[Acao --> Verificar Emails]
- novo chamada de caso de uso:[Acao --> Redigir Email]
- nova chamada de método[Acao --> addEmail]
- novo chamada de caso de uso:[Acao --> Verificar Emails]
- novo chamada de caso de uso:[Acao --> Redigir Email]
- novo chamada de caso de uso:[Acao --> Redigir Email]
- novo chamada de caso de uso:[Acao --> Deletar Email]
- novo chamada de caso de uso:[Acao --> Redigir Email]
- novo chamada de caso de uso:[Acao --> Redigir Email]
- nova chamada de método[Acao --> getAllEmails]
- nova chamada de método[Acao --> addEmail]
- nova chamada de método[Acao --> delEmails]
- nova chamada de método[Acao --> getAllEmails]
- nova chamada de método[Acao --> addEmail]
- nova chamada de método[Acao --> delEmails]
- nova chamada de método[Acao --> addEmail]
- nova chamada de método[Acao --> addEmail]
- nova chamada de método[Acao --> addEmail]
```

Figura 5.7: Log do sistema para simulação da rede de Petri

5.6.4 Script JMeter

Após a geração dos testes a partir da simulação da rede de Petri na etapa anterior, foi gerado o script para o JMeter para execução deste, na qual pode ser visto na figura 5.8. Foi gerado dez requisições na qual irá analisar o tempo de acesso aos métodos, as especificações de requisitos já foram citadas na seção 4.5.

5.6.5 Resultados Encontrados

Com o script pronto, o próximo passo foi a criação do sistema base para teste, neste caso foi implementado apenas o esqueleto do sistema do caso de estudo detalhado na seção 4.5, ao método *delEmail(...)* foi adicionado um tempo de espera maior do que o requerido para este, forçando assim a falha para todos os acessos a este método. O resultado da análise de desempenho dos métodos pode ser visualizado na figura 5.9.

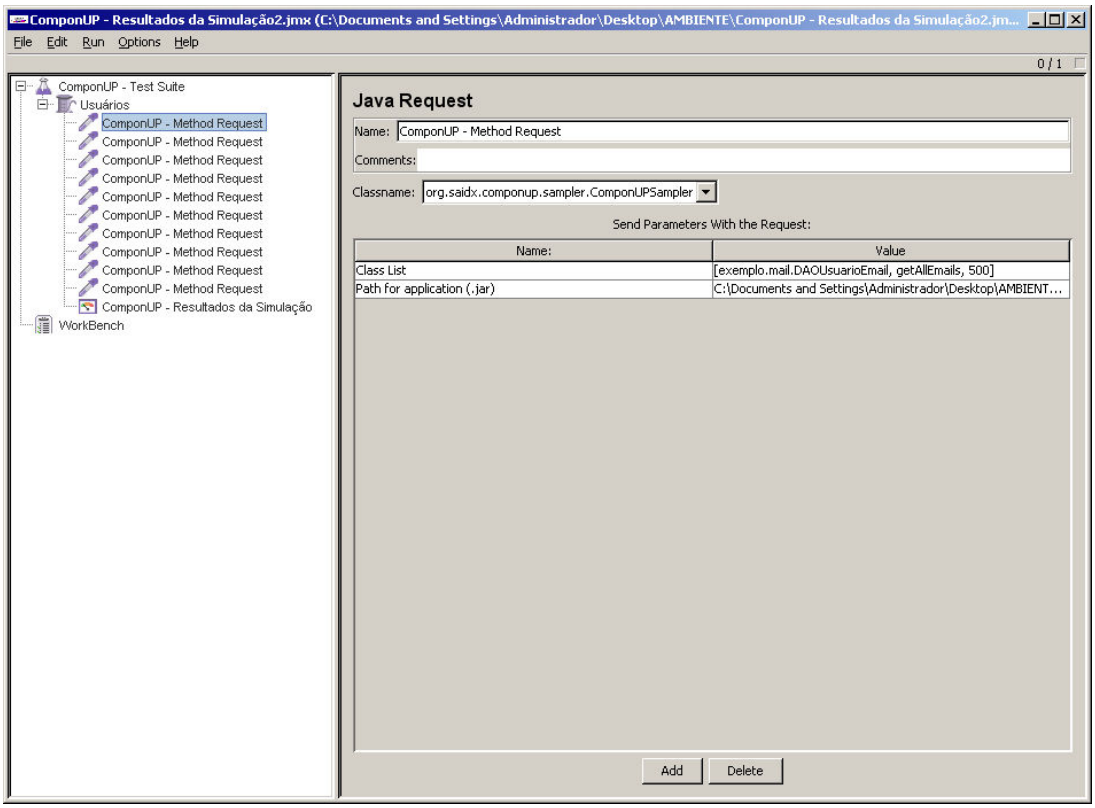

Figura 5.8: Script do JMeter contendo as requisições aos métodos

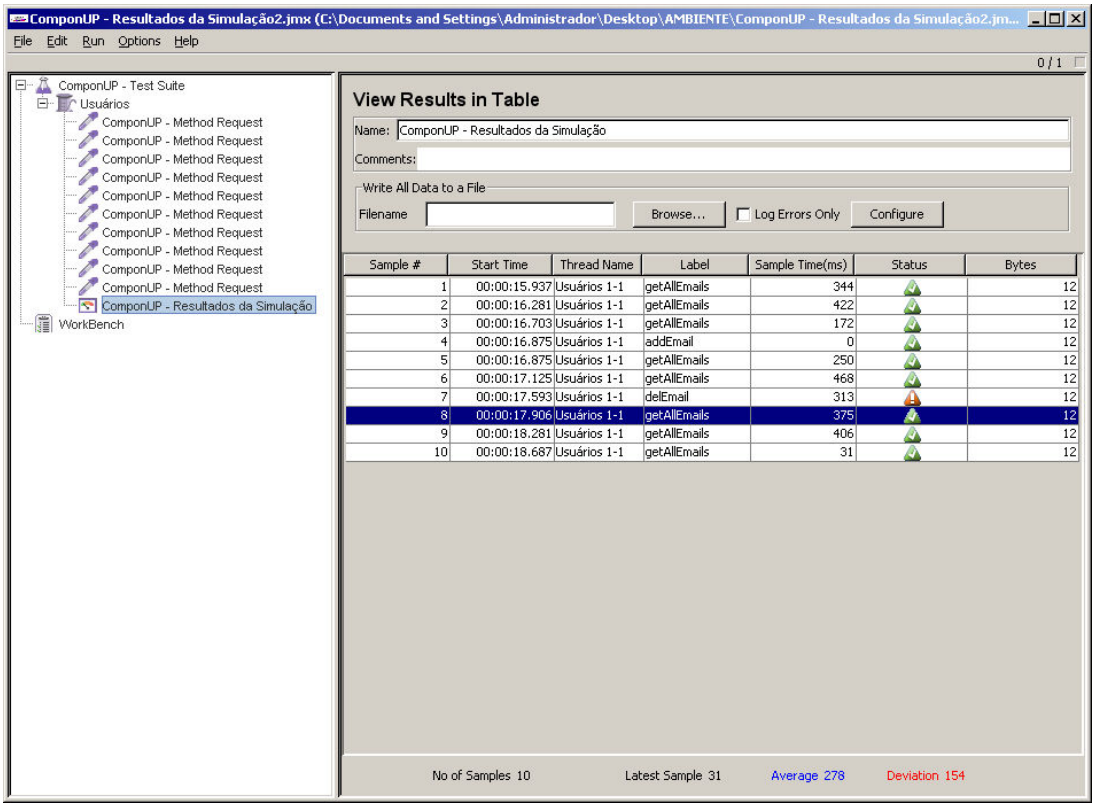

Figura 5.9: Resultados das requisições aos métodos e análise de desempenho requerido

6 *Conclusão e trabalhos futuros*

6.1 Objetivos Alcançados

Através deste trabalho, atingimos o objetivo proposto inicialmente, que era criar uma ferramenta para gerar casos testes de desempenho em componentes, especificamente componentes pertence a camada DAO dos mais variados tipos de projetos que utilizam esse padrão de projeto, levando em consideração sua importância já discutida na seção 2.5.

Mostrou-se viável a geração de casos de teste de software a partir de modelos/diagramas UML 2.0, utilizando para isso, modelos formais (rede de Petri) para a criação de um modelo de simulação que representasse com fidelidade a realidade descrita no projeto a ser testado.

Esse trabalho demonstrou que é possível a construção de teste que avaliam o desempenho do sistema em desenvolvimento, antes mesmo deste estar pronto, pois se baseia em modelos UML, podendo assim detectar possíveis gargalos no sistema e garantindo uma melhor qualidade do mesmo.

6.2 Limitações e Trabalhos Futuros

Durante a fase de implementação foram encontrados os mais diversos tipos de obstáculos que são inerentes a fase de codificação de um sistema. Vários destes obstáculos foram superados com sucesso, entretanto, não foi obtido o mesmo sucesso com alguns destes. Listados abaixo, estão os problemas encontrados durante a implementação do ComponUP que não foram superados ou que sofreram alguma adaptação que diverge da proposta prevista inicialmente neste trabalho.

- Inicialmente previámos que um pacote receberia um estereótipo de teste (*ComponUPTest*) para que todas as classes pertencentes a este pacote fossem testadas, mas para que isto acontecesse, o usuário da ferramenta de modelagem UML deveria especificar os tempos

de resposta para cada método de cada classe existente no pacote, tornando assim trabalhoso a utilização da ferramenta para pacotes com um grande número de classes.

Outro aspecto é que como a camada DAO abrange uma vasta lógica, principalmente por se tratar de um padrão de projeto já consagrado, ela possui várias classes (código fonte) que são requeridos para que ela seja implementada em sua plenitude, sendo que algumas destas classes são apenas interfaces e classes abstratas auxiliares.

Para que não houvesse a necessidade de gerar casos de teste que incluíssem essas classes, optou-se por especificar diretamente na classe a ser testada o estereótipo de teste *ComponUPTest*, não sendo necessário especificar este estereótipo no diagrama de pacotes como previsto anteriormente.

- Na versão atual do ComponUP, é contemplado apenas o parser sobre um subconjunto de tags que são utilizadas para descrever os requisitos de desempenho em um modelo/diagrama *UML 2.0*. Como trabalho futuro, poderia ser criado um *parser* com 100% de compatibilidade com a especificação da *UML SPT Profile*.

- Outro aspecto que sofreu alterações da proposta inicial foi maneira de geração do script para o *JMeter*. Inicialmente pensou-se ser possível gerar o script para carregar e testar uma classe java apenas utilizando o sampler *Java Request* disponível no *JMeter*.

  Na prática isto se mostrou não ser possível, pois este sampler é apenas uma interface que o JMeter utiliza para comunicar-se com outros programas externos. Para solucionar este problema, foi criado um novo sampler e através dele, executar toda a lógica esperada para atender os requisitos requeridos por este trabalho. Para que isto se torne possível, foi gasto uma parte do tempo para estudar a estrutura e o funcionamento interno do JMeter. Felizmente existe uma vasta documentação sobre o assunto, o que nos permitiu ter sucesso nesse quesito, sendo o tempo gasto no aprendizado o nosso maior problema.

  Esta característica incorporada ao ComponUP, poderia ser estendida e aperfeiçoada, criando-se um sampler (plugin) específico para carregar classes java para teste de desempenho em métodos, visto que isso não existe atualmente na ferramenta JMeter. Por ser um programa totalmente modular e que possibilita a "incorporação" de plugins, poderia ser viável a submissão e/ou aceitação deste plugin/sampler pela equipe de desenvolvimento do JMeter.

- Levando em consideração o tempo de construção de um modelo UML com as especificações de desempenho e de que nem todas as empresas possuem modelos UML de seus sistemas atualizados, identificamos que como trabalho futuro poderia ser pesquisado uma

forma alternativa para mapear as informações de entrada sem a necessidade de uma modelagem UML prévia.

Referências Bibliográficas

Aiur John Crupi 2002 AIUR JOHN CRUPI, D. M. D. *Core J2EE Patterns*. [S.l.]: Addison-Wesley, 2002.

ArgoUML 2007 ARGOUML. *ArgoUML - A UML design tool with cognitive support*. 2007. Disponível em:http://argouml.tigris.org/ Acessado em Junho de 2007.

Brémaud 1999 BRéMAUD, P. *Markov Chains Gibbs Fields, Monte Carlo Simulation, and Queues*. [S.l.]: Springer, 1999.

Burnstein 2002 BURNSTEIN, I. *Pratical Software Testing*. Chicago USA: Springer-Verlag, 2002.

Dustin Jeff Rashka 2002 DUSTIN JEFF RASHKA, D. M. E. *Quality Web Systems, Performance, Securyty, and Usability*. [S.l.]: Addison-Wesley, 2002.

Group 2002 GROUP, T. O. M. *UML Profile for Schedulability,Performance, and Time Specification*. 2002. Disponível em: www.omg.org/docs/ptc/02-03-02.pdf Acessado em Março de 2007.

Group 2004 GROUP, T. O. M. *Unified Modeling Language*. 2004. Disponível em: www.omg.org/docs/formal/05-07-04.pdf Acessado em Março de 2007.

Group 2007 GROUP, T. O. M. *OMG*. 2007. Disponível em: http://www.omg.org Acessado em Março de 2007.

IBM 2007 IBM. *IBM Rational Software*. 2007. Disponível em: http://www-306.ibm.com/software/rational/ Acessado em Junho de 2007.

JFern 2006 JFERN, R. *Framework para Redes de Petri*. 2006. Disponível em: http://sourceforge.net/projects/jfern Acessado em Março de 2007.

JMeter 2007 JMETER, A. *JMeter*. 2007. Disponível em:http://jakarta.apache.org/jmeter/ Acessado em Março de 2007.

Jorgensen 1995 JORGENSEN, P. C. *Software Testing, A Craftsmanś Approach*. [S.l.]: CRC Press, 1995.

M. Balbo G. 1995 M. BALBO G., C. G. M. *Modeling with Generalized Stochastic Petri Nets*. [S.l.: s.n.], 1995.

Modeller 2007 MODELLER, U. U. *UML Modeller Umbrello*. 2007. Disponível em: http://uml.sourceforge.net/index.php em agosto de 2007.

Omondo 2007 OMONDO, E. plugin. *Omondo plugin for Eclipse*. 2007. Disponível em: http://www.omondo.com/ Acessado em Junho de 2007.

Peterson 1977 PETERSON, J. L. *ACM Computing Surveys*. New York, NY, USA: ACM Press, 1977.

P.R.M. Lins R.D. 1996 P.R.M. LINS R.D., C. P. M. *Introdução às Redes de Petri e Aplicações*. Campinas: 10a Escola de Computação, 1996.

XML 2007 XML, S. A. for. *Simple API for XML*. 2007. Disponível em: http://www.saxproject.org/ em outubro de 2007.