Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Exploratória de Análise de Dependabilidade no Contexto do Observatório da Web

Túlio César de Araújo Porto

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientadora
Prof.ª Dr.ª Genaína Nunes Rodrigues

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Flávio de Barros Vidal

Banca examinadora composta por:

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora) — CIC/UnB
Prof.ª Dr.ª Gisele Lobo Pappa — DCC/UFMG
Prof. Dr. Eduardo Yoshio Nakano — EST/UnB

CIP — Catalogação Internacional na Publicação

Porto, Túlio César de Araújo.

Uma Abordagem Exploratória de Análise de Dependabilidade no Contexto do Observatório da Web / Túlio César de Araújo Porto. Brasília : UnB, 2013.
123 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. dependabilidade, 2. confiabilidade, 3. Observatório da Web

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Exploratória de Análise de Dependabilidade no Contexto do Observatório da Web

Túlio César de Araújo Porto

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora)
CIC/UnB

Prof.ª Dr.ª Gisele Lobo Pappa
DCC/UFMG

Prof. Dr. Eduardo Yoshio Nakano
EST/UnB

Prof. Dr. Flávio de Barros Vidal
Coordenador do Curso de Computação — Licenciatura

Brasília, 25 de setembro de 2013

Dedicatória

Dedico esse trabalho a todos aqueles que de alguma forma estiveram presentas durante essa jornada. A minha mãe, Alice Porto, e meu irmão Caio Porto que sempre acreditaram e confiaram em mim. À minha avó, Luzia Alice, que se foi durante essa caminhada. Àqueles amigos que sempre estiveram ao meu lado quando precisei de ajuda e àqueles que entenderam meus momentos de ausência.

Agradecimentos

Agradeço primeiramente Deus por possibilitar todo e qualquer momento de aprendizado e confraternização, dentro e fora de nossas salas de aula.

À Universidade de Brasília, pela dedicação dos funcionários em manter a universidade funcionando diante de tantos problemas.

À Raquel Araújo por ter me acompanhado durante parte desse trabalho.

À minha orientadora, Drª Genaína Nunes Rodrigues, pela paciência durante todo esse tempo em que fui seu aludo e orientando, além de incentivar a pesquisa a universidade com palestras e apresentações de trabalhos, sempre acompanhadas de bolo e café.

A todos os professores que se esforçaram e dedicaram tempo e dedicação em prol do aprendizado.

À equipe do Observatório da Web que sempre esteve à disposição para nos auxiliar.

A Bruno Souza e Bruno Silva por terem iniciado os estudos nesse trabalho.

A Leonardo Melo por contribuir substancialmente para a conclusão desse trabalho, assim como do meu curso. Agradeço também pela companhia nas longas noites no laboratório, pelas madrugadas mal dormidas, pelas grandes referências bibliográficas sugeridas com plena competência assim como pelos profundos debates sobre os mais variados temas.

A Felipe Gules também pela contribuição nesse trabalho assim como pelas discussões e críticas ao processo educacional brasileiro e de aprendizagem na universidade.

Aos amigos que conheci durante o curso e sempre manterei contato.

A você que está lendo esse trabalho.

Resumo

A tarefa de extrair informação de grandes bases de dados não é trivial, demanda uma arquitetura complexa e constante monitoramento a fim de garantir que o serviço seja entregue de forma exemplar. O Observatório da Web foi idealizado para tal fim, buscando temas relevantes para a sociedade e extraindo informações das redes sociais. Para analisar a arquitetura utilizada pelo sistema do Observatório da Web e identificar quais etapas são mais significativas na disponibilidade do serviço, foi feita uma análise quantitativa de dados reais do Observatório além da prévia modelagem do sistema na linguagem PRISM para a verificação de propriedades PCTL viabilizadas pelo ProProST. O resultado obtido foi relevante e a utilização de dados reais durante a simulação fez total diferença nos resultados finais.

Palavras-chave: dependabilidade, confiabilidade, Observatório da Web

Abstract

The task of extracting information from large databases is not trivial, requires a complex architecture and constant monitoring to ensure that the service is delivered in an exemplary manner. The Observatório da Web was designed for this purpose, seeking relevant topics to society and extracting information from social networks. To analyze the system architecture used by the Observatório da Web and identify which steps are most significant in service availability, we performed a quantitative analysis of actual data from the Observatório beyond the previous system modeling language PRISM for checking PCTL properties enabled by ProProST. The results were relevant and the use of real data during the simulation made all the difference in the final results.

Keywords: dependability, reliability

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As diversas redes sociais existentes atualmente possibilitam contato entre as mais diversas culturas e pessoas ao redor do mundo. A todo momento são publicadas informações dos mais diversos temas nessas redes sociais. Com o objetivo de extrair informações relevantes dessas redes sociais acerca de um tema definido, foi criado o Observatório da Web. O desafio de se trabalhar com grandes bases de dados e a possibilidade de se extrair estatísticas muito próximas de um cenário real é a grande motivação dos pesquisadores que se dedicam a esse fim. Atualmente o Observatório da Dengue mostrou-se um dos mais bem sucedidos observatórios já implementados. Suas estatísticas são de fácil entendimento e ajudam a elaborar políticas públicas de controle da dengue pelo governo federal. Todo software está sujeito a falhas, assim como o sistema o Observatório. O objetivo desse trabalho é realizar uma previsão de dependabilidade a partir de dados reais.

1.1 Motivações

A partir do trabalho de Bruno Alves e Bruno Silva [8], foram percebidas partes que mereciam serem revistas ou revisitadas a fim de obter dados mais precisos além de uma falta de simulação com dados reais. O Observatório da Web atualmente é utilizado como um dos indicadores de dengue pelo Ministério da Saúde a nível nacional, o que mostra a significativa contribuição social do Observatório. A grande motivação para esse trabalho é contribuir para a criação de um *dashboard* mostrando estatísticas em tempo real da execução e qualidade dos dados analisados e assim auxiliar a equipe desenvolvedora na constante melhoria do sistema.

1.2 Objetivos

O objetivo desse trabalho é realizar uma previsão de dependabilidade a partir de dados reais do sistema, seguida de uma análise de sensibilidade dos componentes envolvidos na execução do *software* do observatório, e assim definir a importância de tais componentes para o observatório.

1.3 Metodologia

O trabalho consiste em uma avaliação quantitativa de de mensagens e defeitos resultantes do monitoramento do Observatório da Dengue e simulados em um modelo próximo a real execução do sistema resultante do estudo do arcabouço e arquitetura disponibilizados para estudo.

1.4 Organização do Trabalho

O trabalho foi dividido em 6 capítulos:

1. Capítulo 2 - Conceitos Básicos: são abordados os principais conceitos necessários para a compreensão desse trabalho: PRISM, ProProST, Dependabilidade e RSO (Redes Sociais Online);
2. Capítulo 3 - Metodologia de Previsão de Dependabilidade: nesse capítulo é descrito o funcionamento do Observatório da Web, a modelagem realizada assim como as propriedades analisadas.
3. Capítulo 4 - Análise: esse capitulo conta com a análise realizada, os resultados obtidos.
4. Capítulo 5 - Conclusão: nesse capítulo são discutidos os resultados finais e propostos trabalhos futuros.
5. Apêndice: Código da modelagem PRISM e classificação dos possíveis defeitos.

Capítulo 2

Conceitos Básicos

2.1 Dependabilidade

Dependabilidade pode ser definida como a capacidade de um sistema de entregar um serviço justificadamente confiável [3]. É uma propriedade de sistemas focada em praticamente todas as áreas de Ciência da Computação. Devido à sua importância ao usuário e ao ambiente de execução do software, o interesse em dependabilidade tem sido alvo de muitas pesquisas que envolvem todos os ciclos de desenvolvimento do software: desde a análise que antecede a implementação até o teste do software. Apesar da sua abrangência, importância e crescente relevância na geração de software com qualidade, poucos centros acadêmicos no mundo oferecem oportunidade ao aluno de Ciência da Computação de conhecer melhor como entender e analisar o software sob a perspectiva de dependabilidade.

Segundo Kaâniche [15] Todo sistema computacional é caracterizado a partir de 5 perspectivas diferentes, são elas:

1. **Funcionalidade**: O que o software se dispõe a realizar.
2. **Usabilidade**: Se a interface do *software* é fácil de ser utilizada.
3. **Desempenho**: Como o software executará tal função.
4. **Custo**: O custo desprendido para o funcionamento do software, custo de hardware, desenvolvimento, energia, etc.
5. **Dependabilidade**: Capacidade de oferecer um serviço que pode ser justificadamente confiável.

Nesse capítulo serão definidos alguns conceitos básicos de dependabilidade. Em seguida será apresentada a metodologia utilizada para monitorar as falhas do Observatório da Web e então serão caracterizadas as possíveis falhas e seus decorrentes defeitos.

2.1.1 Escopo

O conceito de dependabilidade aparece pela primeira vez na década de 1830 no conceito da máquina de calcular de Babbage. Desde a primeira geração de computadores componentes mais confiáveis são desenvolvidos assim como técnicas para melhorar os

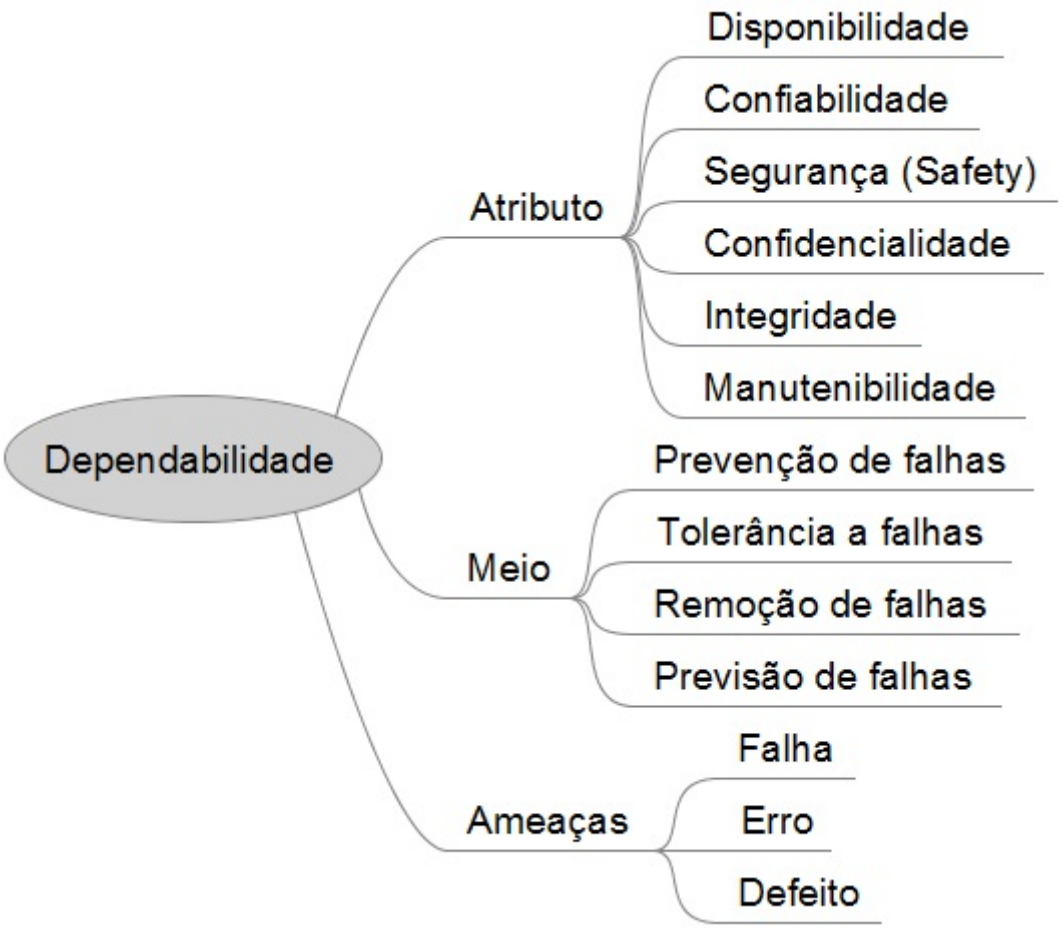


Figura 2.1: Árvore de Dependabilidade

atributos que envolvem a dependabilidade, como códigos de controle de erros, duplicação com comparação, triplicação com votação e diagnóstico de componentes defeituosos [3].

Desde então muitas pesquisas são desenvolvidas afim de garantir um serviço justificadamente confiável tanto para o usuário quanto para o ambiente do software para diminuir consideravelmente possíveis falhas, erros e defeitos.

2.1.2 Definição

Dependabilidade é uma tradução literal do termo inglês *dependability* que para ser entendido necessita ser dividido em 3 partes: o atributo afetado, os meios pelos quais a dependabilidade é atingida e a ameaça que isso representa conforme a Figura 2.1[3].

Os Atributos de Dependabilidade

Dependabilidade então é um conceito que engloba os seguintes atributos [3]:

1. Disponibilidade (*Availability*): probabilidade do sistema estar operacional num instante de tempo determinado; alternância de períodos de funcionamento e reparo [3].

2. Confiabilidade (*Reliability*): capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período [3].

3. Segurança (*Safety*): probabilidade do sistema ou estar operacional e executar sua função corretamente ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam [3].

4. Confidencialidade (*Confidentiality*): ausência de divulgação não autorizada de informação [3].

5. Integridade(*Integrity*): ausência de alterações impróprias do estado do sistema [3].

6. Manutenibilidade(*Maintainability*): significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo físico e a colocação em operação[20][3].

Os Meios

No desenvolvimento de um sistema, existem quatro técnicas, que combinadas, contribuem na dependabilidade do sistema. São elas [3]:

1. Prevenção de Falhas (*Fault Prevention*): trata de como prevenir possíveis falhas.

2. Tolerância a Falhas (*Fault Tolerance*): trata de como garantir que o serviço seja entregue corretamente, mesmo na presença de falhas.

3. Remoção de Falhas (*Fault Removal*): consiste na remoção de falhas. Pode ser dividida em três etapas: Verificação, diagnóstico e remoção da falha.

4. Previsão de Falhas (*Fault Forecasting*): consiste em sanar as falhas afim de estimar a quantidade e quando novas falhas ocorrerão. Dar-se por dois aspectos: análise qualitativa ou ordinal das falhas, em que é construído um *rankink* das falhas ou da combinação de eventos, ou pela análise quantitativa ou probabilística das falhas. Nessa caso, analiza-se probabilisticamente o impácto aos atributos de dependabilidade. Mais tarde esses atributos são vistos como medidas de confiabilidade. Em particular, esse trabalho contextualiza-se nesse âmbito.

As ameaças: Falhas, Erros e Defeitos

Serviço confiável é quando o serviço implementa a função desejada. Quando o serviço entregue é diferente do serviço desejado, temos um defeito seja porque a especificação da função não coincide com o resultado ou a especificação não descreve o resultado obtido.

Define-se que um sistema está em estado errôneo ou em erro se o processamento posterior a partir desse estado pode levar a defeito. Define-se falha como a causa física ou algorítmica do erro [20].

Como visto na Figura 2.2 falhas são associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário.

Em uma das máquinas qualquer, por exemplo, pode haver um chip de memória com um setor defeituoso (universo físico) que pode danificar um dado (universo da informação) e

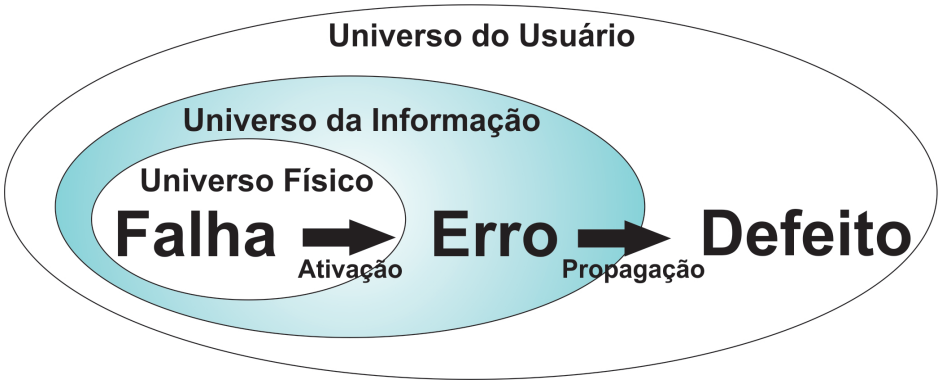


Figura 2.2: Modelo de Universos [20](com adaptações)

Figura 2.3: Modos de Defeitos. Tradução de Avizienis et. al. [4]

que por consequência pode resultar em uma informação divergente daquela desejada para o usuário final. No entanto essa zona defeituosa pode nunca ser usada, assim essa falta não provoca erro que não acarreta em falha.

2.1.3 Classificação dos Defeitos

Nem sempre um sistema apresenta defeito da mesma forma, os modos que um sistema pode falhar são chamados **Modos de Defeito**(*Failure Modes*) e se dão a partir de quatro pontos de vista diferentes como descritos na Figura 2.3:

Quanto ao domínio os defeitos podem ser:

1. De conteúdo: quando o conteúdo entregue difere do especificado.
2. De tempo: Quando o tempo de entrega do serviço difere do especificado. Pode ser tanto quando o serviço é entregue antes do previsto (*early timing failure*) ou quando ocorre depois do previsto (late timing lailure).

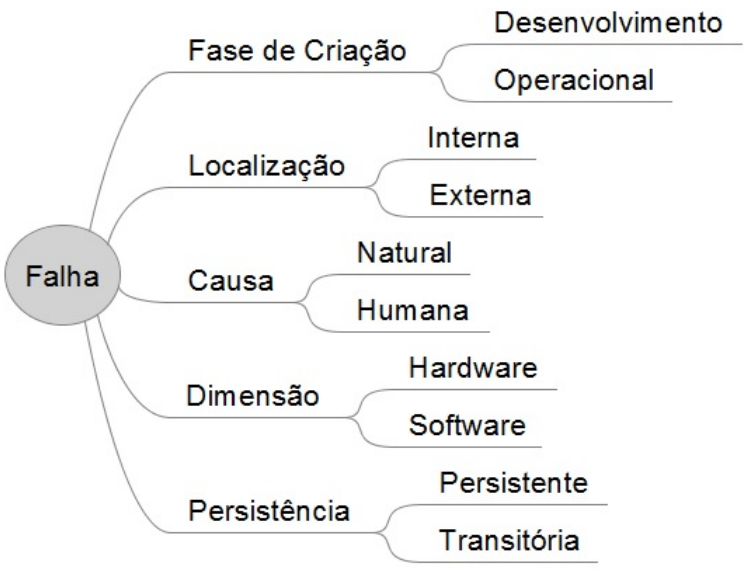


Figura 2.4: Modos de Defeitos [4]

Quando ocorre defeito tanto de conteúdo quanto de tempo, os defeitos classificam-se em duas classes:

1. Defeito de parada (*halt failure*): quando o sistema não apresenta nenhuma atividade. Um caso especial desse tipo de defeito é chamado de Defeito Silencioso quando nenhum serviço é entregue, nem mesmo mensagens de defeito.
2. Defeito instável (*erratic failure*): quando o conteúdo é entregue, porém com defeito, ou seja, diferente do especificado.

Quanto à detectabilidade um defeito pode ser sinalizado ou não sinalizado. Essa característica depende de mecanismos de detecção como, por exemplo, ferramentas de inspeção utilizadas na coleta de falhas. Todo mecanismo de detecção de falhas também está sujeito a defeito, logo, algumas falhas podem deixar de ser percebidas (falso negativo) ou alguma ação não indevida do sistema pode ser percebida como defeito (falso negativo).

Quanto à consistência, os defeitos dividem-se em consistentes e inconsistentes. São consistentes os defeitos que são percebidos da mesma forma por todos os usuários daquele sistema, seja ele o usuário final ou outro sistema. São inconsistentes aqueles que são percebidos de forma diferenciada por cada usuário, podendo de acontecer até mesmo que algum usuário receba a informação corretamente.

Por fim, quanto à consequência, os defeitos dividem-se em uma escala defeito pequeno, médio e grave. Pequeno quando o usuário final não percebe o defeito no serviço. Defeito médio quando o usuário final percebe pequenos defeitos no serviço e defeito grave, quando o usuário percebe que o serviço está indisponível.

2.1.4 Classificação das falhas

Como podemos ver na Figura 2.4 as falhas são classificadas quanto a:

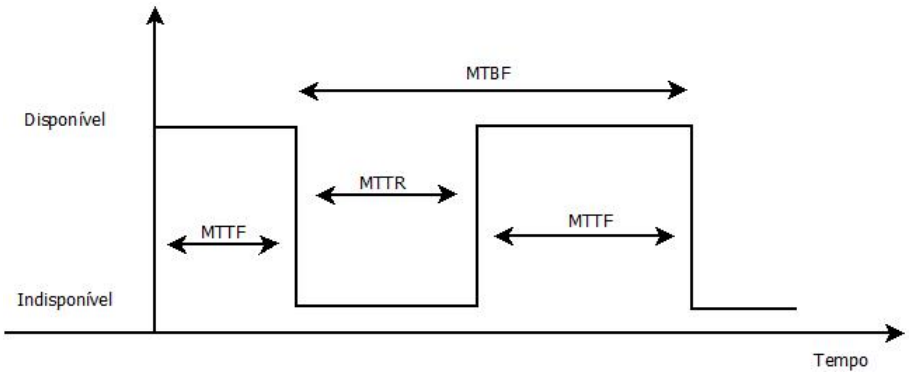


Figura 2.5: MTTF, MTTR e MTBF

1. Fase de criação:

   De desenvolvimento: ocorre durante a fase de desenvolvimento do sistema.

   Operacional: ocorre durante a entrega do serviço proposto.

2. Localização:

   Interna: ocorre dentro do sistema.

   Externa: ocorre fora dos limites do sistema e provoca danos durante a interação com o mesmo.

3. Causa:

   Natural: causada por fenômenos da natureza e sem intervenção humana.

   Humana: é resultado de intervenções impróprias do homem.

4. Dimensão:

   Hardware: se origina no hardware ou afeta o mesmo.

   Software: de alguma forma afeta o software do sistema.

5. Persistência:

   Persistente: persiste durante o tempo a partir da sua ativação.

   Transitória: ocorre durante um período limitado de tempo.

2.1.5 Métricas

Existem diversas métricas para se estimar a confiabilidade de um software. Dentre as mais usadas estão o MTTF (*Mean Time to Failure*), que consiste no tempo médio esperado para que ocorra um defeito; MTTR (*Mean Time to Repair*), é o tempo médio para que o sistema seja reparado; MTBF (*Mean Time Between Failure*), que é o tempo médio entre as falhas do sistema. A Figura 2.5 representa a relação entre as métricas citadas. O MTBF pode ser encontrado pela fórmula:

$$MTBF = MTTF + MTTR$$

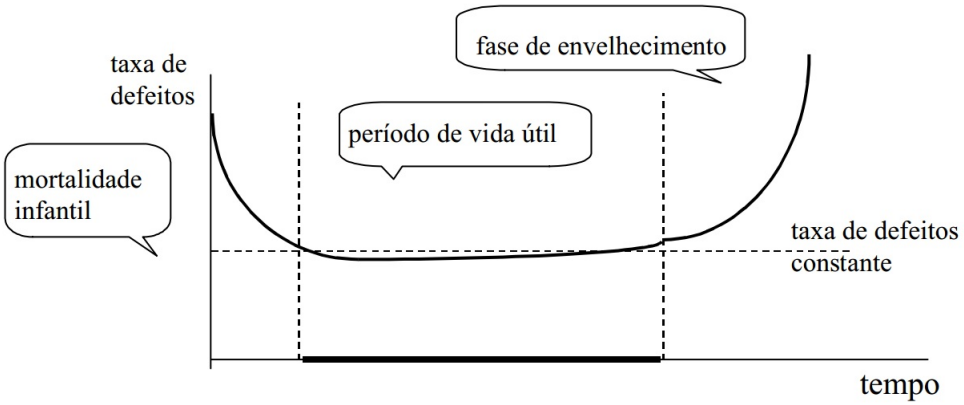


Figura 2.6: Curva de Defeitos [20]

Já a disponibilidade assintótica, Segundo Lopes [7], é a probabilidade de um sistema estar disponível em um determinado instante, está diretamente ligada à confiabilidade de um sistema e pode ser estimada pela função:

$$Disponibilidade_{Assintótica} = \frac{MTTF}{MTTF + MTTR}$$

A taxa de defeitos também pode ser usada para se medir a confiabilidade. A taxa de defeitos é o número de defeitos num determinado período de tempo:

$$Taxa\ de\ Defeitos = \frac{Total\ de\ Defeitos}{Tempo\ Total}$$

A taxa de defeitos é dada por falhas por unidade de tempo e pode ser representada pela Figura 2.6. Discute-se se o conceito da curva da banheira estende-se não só a componentes de hardware quanto de software. Na verdade, em sua fase de desenvolvimento (mortalidade infantil), o software passa por uma fase de teste e enfrenta alto índice de defeitos até a sua implantação definitiva quando a taxa de defeitos é constante. Por fim o software pode tornar-se obsoleto ou incompatível com as novas tecnologias e volta a apresentar taxa de defeitos crescente [20].

Na análise proposta nesse trabalho utilizaremos a taxa de defeitos por mensagens , calculada pela relação entre o total de defeitos detectados e o total de mensagens da RSO coletadas pelo Observatório da Web.

$$Taxa\ de\ Defeito\ por\ Mensagem = \frac{Total\ de\ Defeitos}{Total\ de\ Mensagens\ Coletadas}$$

2.2 Modelagem UML

Embora a linguagem UML (*Unifield Modeling Language*) não seja uma metodologia de desenvolvimento, auxilia muito no planejamento, documentação e visualização do funcionamento do Software orientado a objetos. Existem dois tipos básicos de diagramas UML [9]:

1. Diagramas estruturais - focam na estrutura ou arquitetura estática do sistema. Exemplos: diagramas de classes, diagrama de componentes e diagrama de objetos.

2. Diagramas comportamentais - descreve a parte dinâmica de cada componentes, como troca de mensagens e mudanças de estado. Exemplos: diagramas de atividades, sequência ou casos de uso.

Nesse trabalho serão utilizados diagramas UML de componentes, atividades e sequência para descrever o funcionamento do Observatório da Web. O diagrama de componentes descreve a divisão do software em componentes e suas dependências. O diagrama de atividades descreve o comportamento e os possíveis caminhos em que o dado pode tomar no sistema. Já o diagrama de sequência também é comportamental e descreve a comunicação entre os objetos através de mensagens.

2.3 PRISM

O PRISM[16] é um software *open source*, disponibilizado pela licença GNU GPL. O software permite uma modelagem formal e análise de sistemas com comportamento aleatório ou probabilístico. É amplamente utilizado desde em protocolos de comunicação a sistemas biológicos. Suporta diversos modelos probabilísticos como Cadeias de Markov de tempo discreto (DTMCs [1]), cadeias de Markov de tempo contínuo (CTMCs [2]), processos de decisão de Markov (MDPs [3]), autômatos probabilísticos (PAs [4]), autômatos probabilísticos temporizados (PTAs [5]). Possui três ambientes: *model* [Figura 2.7], para modelagem do software na linguagem PRISM; *properties* [Figura 2.8], para verificação de propriedades do modelo; e *simulation* [Figura 2.9], que permite executar o modelo.

O modelo PRISM é descrito na linguagem em alto nível de mesmo nome e baseia-se em estados. O modelo é composto por módulos que interagem entre si. Cada módulo possui variáveis locais e o valor da variável representa o estado em que o módulo se encontra. As variáveis estão acessíveis para qualquer módulo do modelo, mas só podem ser modificadas pelo módulo onde se encontra.

$$[action]guard -> prob_1 : update_1 + ... + prob_n : update_n; \tag{2.1}$$

A fórmula 2.1 apresenta a sintaxe que demonstra um comando no PRISM. *Guard* é uma condição que engloba todas as variáveis do modelo, cada *update* representa uma mudança de estado possível a partir do estado atual de acordo com a probabilidades (*prob*) associadas a cada *update*. As probabilidades (*prob*)variam de 0 a 1 e a soma de todas as probabilidades são iguais a 1 [Fórmula 2.2]. O campo *action* é usado para sincronizar

[1] Do inglês - Discrete Time Markov Chain
[2] Do inglês - Continuous Time Markov Chain
[3] Do inglês - Markov Decision Processes
[4] Do inglês - Probabilistic Automata
[5] Do inglês - Probabilistic Timed Automata

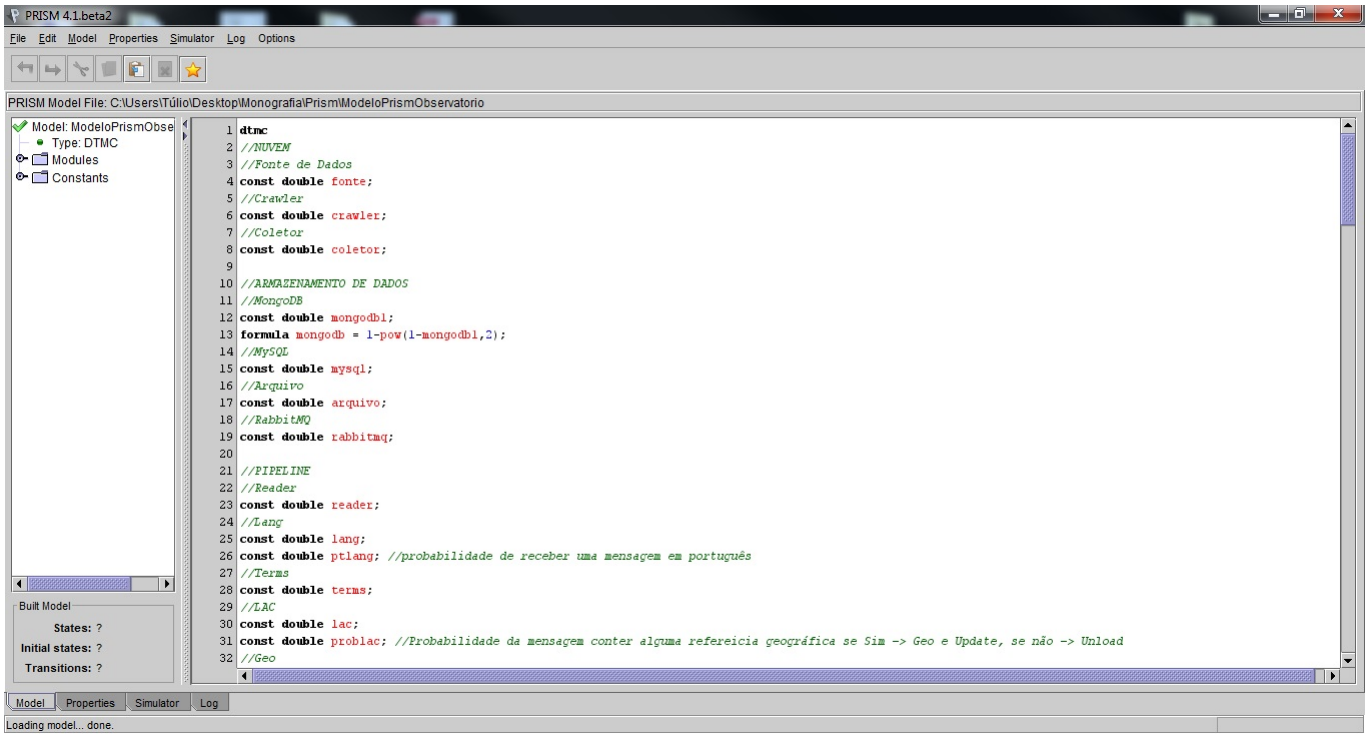


Figura 2.7: Captura de tela do PRISM, ambiente *model*

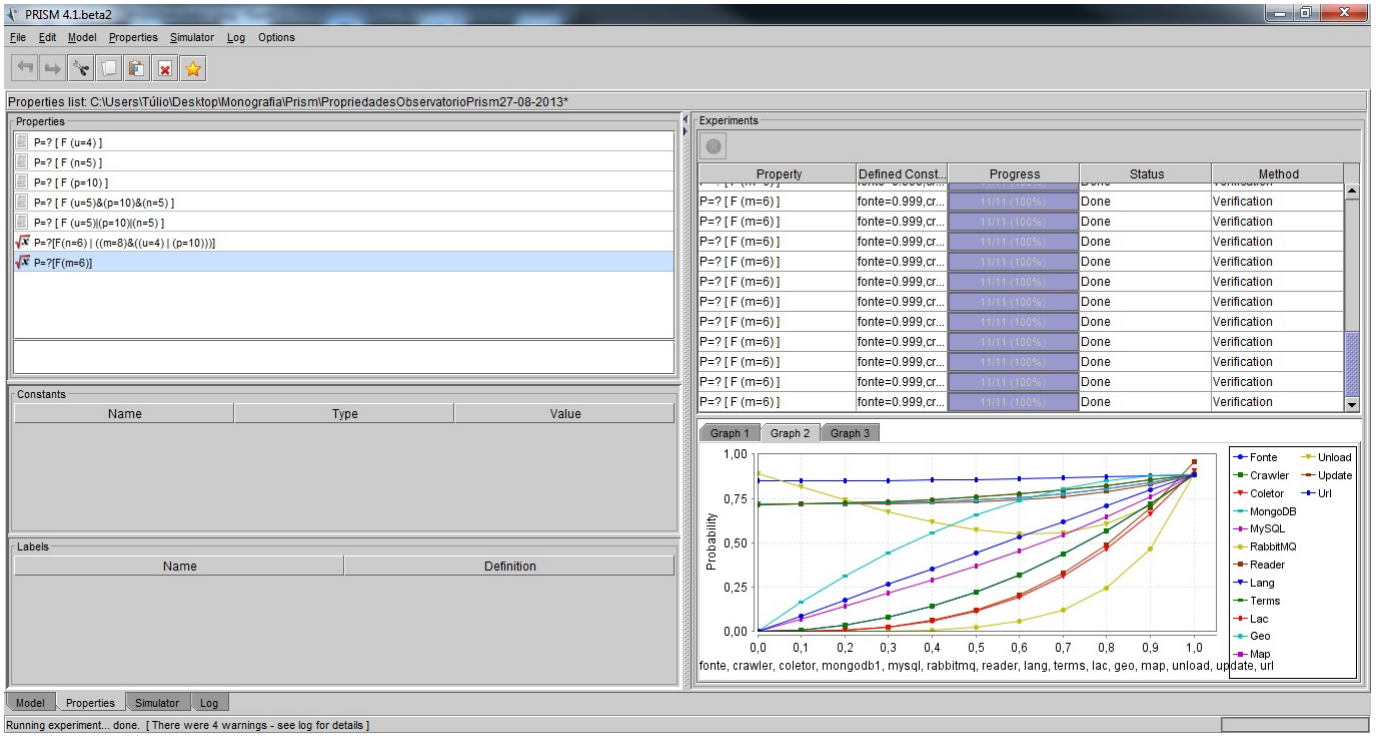


Figura 2.8: Captura de tela do PRISM, ambiente *properties*

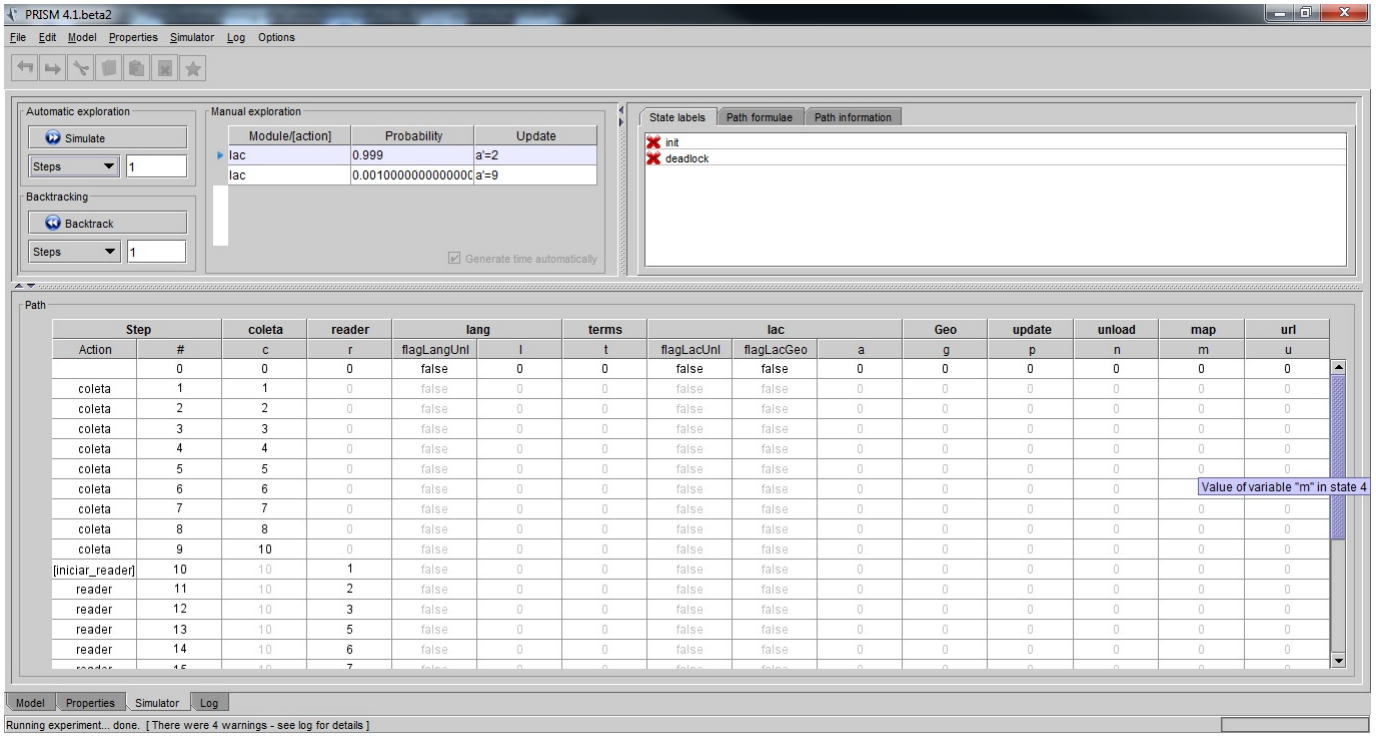


Figura 2.9: Captura de tela do PRISM, ambiente *simulation*

comandos. Caso não exista outro campo *action* com o mesmo nome, ou o campo for vazio, será executado o comando subsequente.

$$\sum_{i=1}^{n} prob_i = 1 \tag{2.2}$$

2.4 ProProST

Afim de validar as propriedades utilizadas durante esse trabalho foi utilizado o ProProST [14].

Com base em 152 propriedades retiradas de artigos acadêmicos e 48 retiradas de exemplos reais, foi criado o ProProST[6][14]. O projeto visa auxiliar na criação de propriedades formais a parir de uma gramática estruturada na língua inglesa.

Traduzir propriedades probabilísticas especificadas em linguagem comum para linguagens formais é sempre um desafio [14]. A intenção do ProProST é investigar e extrair padrões de especificação para linguagens probabilísticas.

Para tanto, os padrões foram divididos segundo a Tabela 2.1.

Utilizaremos apenas ocorrências probabilísticas de estado fixo nesse trabalho.

Um padrão de propriedade de especificação descreve uma propriedade generalizada recorrente e proporciona uma solução sob a forma de um especificação formal em diferentes lógicas temporais tradicionais [14], por exemplo PCTL. A ferramenta ProProST Wizard

[6]Do inglês - *Probabilistic Property Specification Templates*

Classe	Padrão	Descrição
Ocorrência probabilística	Probabilidade de transição de estado	Exatamente após t unidades de tempo o estado s é verdadeiro com probabilidade p
	Probabilidade de estado fixo	Ao longo da execução, a probabilidade do estado s ser alcançado é sempre p.
	Invariância probabilística	O estado s num período determinado será alcançado sempre com probabilidade p.
	Existência probabilística	O estado s eventualmente torna-se verdadeiro num período determinado com probabilidade p.
Ordem probabilística	Probabilidade de ocorrência	Um estado $s1$ eventualmente é verdadeiro para um periodo determinado depois que um estado $s2$ é atingido.
	Probabilidade de precedência	Um estado $s1$ precede um estado $s2$ durante a execução com probabilidade p.
	Probabilidade de resposta	Dado um período, depois que um estado $s1$ for atingido, o estado $s2$ será verdadeiro com probabilidade p.
	Probabilidade de resposta restrita	Depois que um estado $s1$ for atingido, o estado $s2$ será sempre verdadeiro com probabilidade p.

Tabela 2.1: Tabela: Classificação dos padrões do ProProST. Tradução de Grunske [14](com adaptações)

[Figura 2.10] possibilita que propriedades bem formadas em inglês sejam convertidas para a foma PCTL que podem ser verificada no PRISM.

2.5 Conversão UML para PRISM

A partir dos diagramas UML comportamentais e estruturais construídos a partir do entendimento do funcionamento do Observatório da Web podemos visualizar melhor como o sistema funciona, possíveis transições e estados finais. Assim, cada atividade do sistema, representada por um diagrama de sequência, torna-se um módulo (*module*) na linguagem PRISM e cada interação no diagrama de atividades. As transições entre módulos se dão através do sincronismo entre *labels* ou *flags* que sinalizam que um módulo deve ser iniciado quando o valor da *flag* é setado como verdadeiro (*true*). A simulação do PRISM deve seguir comportamento semelhante aos diagramas de sequência.

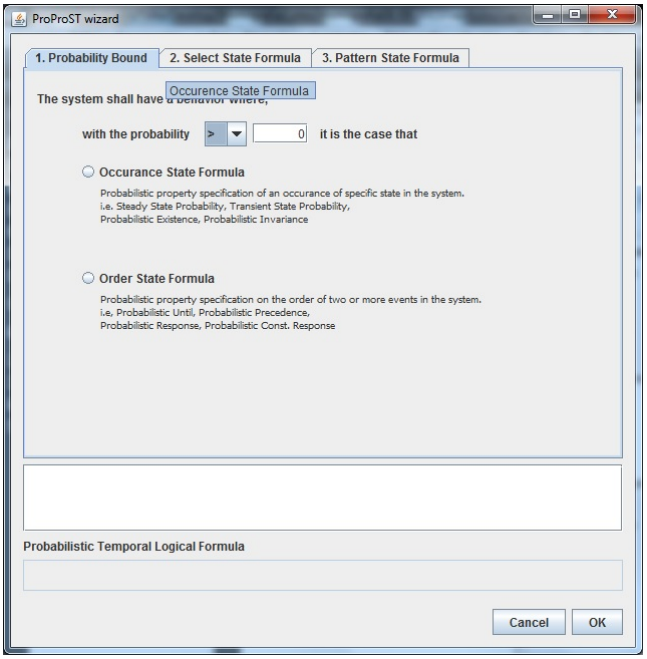


Figura 2.10: Captura de tela do ProProST

2.6 Validação do Modelo PRISM

O modelo PRISM precisa ser validado. Essa validação consiste em verificar se o modelo executa de acordo com o proposto e sem *deadlocks*. A validação é efetuada no ambiente *simulation* do PRISM em que pode ser verificado o valor de cada variável durante a execução, avançar e retroceder além de testar etapas onde ocorrem decisões no sistema.

2.7 Propriedade de Alcançabilidade (PCTL)

A propriedade de alcançabilidade trata da possibilidade de alcançar um determinado estado de sucesso ou falha a partir de um estado inicial. Essa propriedade pode ser verificada a partir da modelagem PRISM, no ambiente *properties* em que podem ser verificadas propriedades expressas em linguagens lógico temporais (PCTL[7], CSL[8], ou LTL[9]) para a verificação do modelo probabilístico criado [16].

Para tal verificação, O PRISM adota dois tipos de análises 2.7:

$$Pbound[pathprop]$$

$$P =?[pathprop]$$

No primeiro caso, a propriedade P é delimitada (exemplo: $P < 0,1[.]$). O resultado é verdadeiro se a propriedade P é satisfatível com requisito *bound*. No segundo caso, a

[7]Do inglês - *Probabilistic Computation Tree Logic*
[8]Do inglês - *Computation Structure Language*
[9]Do inglês - *Linear Temparal Logic* ou *Linear-Time Temporal Logic*

propriedade P é quantitativa(P =?[.]) e vem em forma de pergunta. O resultado é a probabilidade de se alcançar o estado *pathprop* a partir de um estado inicial.

2.8 Análise de Sensibilidade

A análise de sensibilidade consiste em uma análise que permite analisar o impacto que cada componente ou módulo do PRISM tem no sistema como um todo. Também são criados *experiments* no PRISM com os possíveis cenários de execução do sistema.

2.9 RSO (Redes Sociais Online)

Segundo Castells [6] o consumo de mídia é a segunda maior categoria de atividade depois do trabalho e, certamente, a atividade predominante nas casas. è provável que o jovem estudante atualmente gaste a maior parte de seu tempo consumindo algum tipo de mídia já a sociedade em rede está proporcionando grandes mudanças na sociedade contemporânea. Castells ainda afirma que o surgimento de um sistema eletrônico de comunicação de alcance global que possibilita a integração de todos os meios de comunicação e que possui interatividade potencial está mudando e mudará para sempre nossa cultura. Claramente observa-se que as redes sociais atuais como o Facebook possibilitam tal interação, outras atingem esses requisitos parcialmente. São notáveis as mudanças que a internet e as redes sociais causam na sociedade. A atividade de docência sofreu mudanças substanciais devido ao acesso que o aluno tem a informação. Como dito anteriormente, o jovem certamente passa a maior parte de seu tempo consumindo algum tipo de mídia, seja para fins lúdicos, estudo ou trabalho.

Figura 2.11: Exemplos de Redes Sociais Online

2.9.1 Definição

Segundo Benevenuto [5], o termo rede social online é utilizado para descrever um grupo de pessoas que interagem através de qualquer mídia de comunicação, logo existem redes sociais online desde o advento da internet. No entanto essa definição é muito abrangente. A definição que Benevenuto utiliza em seu trabalho [5] é melhor para o entendimento também desse trabalho. Segundo ele, RSO é um serviço Web que permite o usuário:

1. Construir perfis públicos ou semi-públicos dentro de um sistema;
2. Articular uma lista de outros usuários com os quais ele compartilha conexões;
3. Visualizar e percorrer suas listas de conexões assim como outras criadas por outros usuários do sistema.

Segundo Santana [11] redes sociais online podem ser classificadas em:

1. Genéricas - quando seu conteúdo é muito diverso e acabam por promover uma função lúdica, informal e recreativa. Exemplos: Facebook, Hi5, MySpace, Orkut, Sonico e Twitter 2.11.
2. Especializadas - fornecem ferramentas direcionadas a um público alvo, com temática específica. São tratadas de forma bem diferente das redes sociais genéricas pois possuem finalidade mais formal, de apoio a educação, pesquisa e trabalho. Exemplos: Lemill, LinkedIn, Ning e Xing.

Toda rede social possui elementos característicos da função que o sistema se dispõe a realizar. Benevenuto [5] trata de sete elementos presentes em uma RSO:

1. Perfis dos usuários - são usados para identificar os usuários. Contém dados demográficos (idade, sexo, localização, etc) assim como dados de interesse por tipos musicais, esporte, hobby, arte ou restaurantes. Os dados podem ser abertos ou privados.
2. Atualizações - são mensagens compartilhadas por amigos ou publicamente. São formas efetivas de disseminar conteúdo na rede.
3. Comentários - nem todo usuário de uma rede social publica conteúdo de fato, alguns apenas comentam conteúdo já publicado. Podem ser reclamações, elogios ou opiniões de usuário e podem ser de grande valor para ferramentas de mineração de dados.
4. Avaliações - podem vir na forma de "gostei"(*like*) como no Facebook ou binária (gostei ou não gostei) como em vídeos no YouTube. Algumas redes sociais também possibilitam que o conteúdo seja avaliado como na forma de estrelas onde o usuário pode escolher de uma a 5 estrelas.
5. Lista de Favoritos - o usuário tem a liberdade de separar conteúdo em uma lista de favoritos para que possa acessar futuramente. Geralmente a RSO analisa tais dados para sugerir conteúdo e propagandas dirigidas ao gosto do usuário.
6. Lista de Mais Populares - bem comum no Twitter na forma de *Trending Topics* ou *Top Lists*. Aquele conteúdo, seja música, vídeo ou mensagem mais comentado, compartilhado ou assistido na rede acaba sendo ranqueado nessa lista.

7. Metadados - é comum que usuários associem metadados, como *tags*, em conteúdos diversos nas redes sociais. Tais metadados podem ser usados para pesquisa de conteúdo relacionado. Muito comum no Twitter chamados de *hashtags* e geralmente são da forma de "*cerquilha*" + *tag* (#hashtag).

Capítulo 3

Metodologia de Previsão de Dependabilidade

Nesse Trabalho a metodologia para a previsão formal de dependabilidade do Observatório da Web dar-se-á da seguinte forma:

1. Compreensão do funcionamento do Observatório da Web através de reuniões com a equipe, documentação existente e códigos.
2. Modelagem no sistema padrão UML
3. Conversão da modelagem UML para a a linguagem PRISM
4. Validação do modelo PRISM
5. Verificação da alcançabilidade (Avaliação da confiabilidade prevista)
6. Definição das propriedades de qualidade no ProProST.
7. Análise de Sensibilidade

3.1 Compreensão do funcionamento do Observatório da Web

Para o entendimento do Observatório da Web, foi disponibilizado em uma máquina virtual uma instanciação do Observatório da Dengue. O acesso disponibilizado foi de praticamente todo o Observatório, com exceção do portal de publicação. Os bancos de dados MySQL e MongoDB também localizavam-se na mesma máquina virtual e o MongoDB não sofreu redundância como especificado em alguns documentos.

3.1.1 Observatório da Web

O Ibope Media o Brasil registrou 102,3 milhões de pessoas com acesso a internet no primeiro trimestre de 2013 e a internet vem contribuindo cada vez mais para a formação de opinião de cada um deles. Com o surgimento da Web 2.0, além de influenciados, os usuários puderam expor suas sugestões, reclamações ou simples comentários gerando

uma quantidade imensa de dados. Ferramentas como o Twitter, Facebook e YouTube possibilitam tal interação.

As redes sociais têm tido um aumento de uso exorbitante, as pessoas comentam sobre tudo, sua vida, vida dos outros, celebridades, eventos festivos, política, religião, tudo pode ser encontrado nas redes sociais. E nada mais natural do que usá-las para essas pesquisas de opiniões, já que o consta nas redes é a opinião do usuário.

Dessa forma a internet deixou de ser apenas um ambiente estático, ganhou dinamicidade e vem contribuindo inclusive na área política e econômica [10]. Tal impacto pode ser notado em organizações de manifestações públicas como a marcha contra a corrupção ou mesmo impactando em eventos políticos, como visto nas eleições americanas de 2008.

Projeto de pesquisa do Instituto Nacional de Ciência e Tecnologia para a Web (InWeb) e com o objetivo de monitorar importantes fatos a partir dos dados disponíveis na web, foi criado o Observatório da Web, uma ferramenta gratuita financiada pelo CNPq e pela Fapemig que conta com a contribuição de especialistas da Universidade Federal de Minas Gerais (UFMG), Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Universidade Federal do Amazonas (UFAM) e Universidade Federal do Rio Grande do Sul (UFRGS).

O projeto inicialmente visava a criação de um portal em que assuntos de grande repercussão fossem mostrados de forma sintetizada através de metáforas visuais e indicadores. Até hoje foram desenvolvidos seis observatórios: Observatório das Eleições Presidenciais de 2010, Observatório das Eleições Municipais de 2012, Observatório da Copa do Mundo de Futebol de 2010, Observatório das Olimpíadas de Londres e os ainda ativos Observatório do Brasileirão e o Observatório da Dengue, este que é foco nesse trabalho.

Funcionamento do Observatório

O observatório da Web foi criado a partir de um modelo conceitual envolvendo diferentes contextos. Para o seu devido funcionamento, o Observatório deve atender a quatro requisitos mínimos [8]

1. Necessidade de processamento em tempo real ou próximo a isso, a fim de absorver rapidamente o que ocorre nas redes sociais.
2. Diversificar os tipos de fontes como medida de tratar informações falsas, contraditórias ou complementares.
3. Produzir metáforas visuais que retratem de forma eficaz o que está sendo comentado nas redes sociais.
4. Ser escalável afim de armazenar e processar informações independente do volume de dados e sem haver perda de mensagens.

Arquitetura e Arcabouço

O Observatório da Web pode ser dividido em três partes que interagem entre si: o modelo conceitual, a arquitetura do sistema e o arcabouço do software, assim como na Figura 3.1.

O modelo conceitual é composto por sete definições básicas [10]:

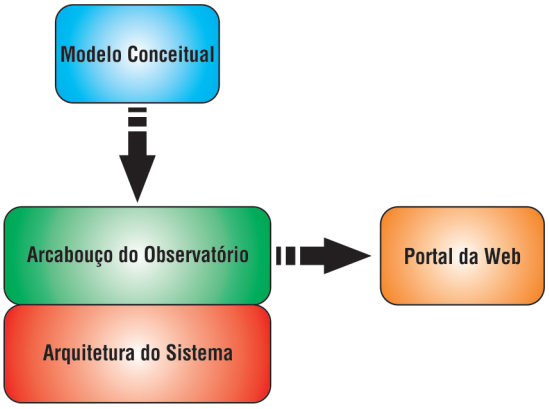


Figura 3.1: Dependências gerais do Observatório da Web. Adaptado de Almeida et. al. [10]

Figura 3.2: Fases de execução no arcabouço. Adaptado de Almeida et. al. [10]

1. Contexto - Assunto observado na Web. Exemplo: dengue.
2. Entidades - Algo que está sendo observado. Exemplo: dor no corpo, *Aedes Aegypti*.
3. Fontes - Local de onde serão retiradas as informações. Exemplo: Twitter, Facebook.
4. Temas - Assuntos específicos pertencentes ao contexto. Exemplo: saúde, saneamento básico.
5. Grupos - Organizações ou qualquer forma de agrupamento de entidades. Exemplo: grupos no Facebook.
6. Autores - É o responsável pelo conteúdo disponível na fonte. Exemplo: usuário da RSO.
7. Eventos - Acontecimentos importantes no contexto do observatório. Exemplo: surto de denque em uma determinada cidade.

O modelo conceitual então é instanciado e baseando-se nos parâmetros descritos, a entrada de dados é formalizada para o arcabouço. O arcabouço executa sobre um arquitetura complexa e que será descrita em seguida.

O arcabouço é composto por cinco fases como mostrado na Figura 3.2. Inicialmente as mensagens são coletadas e armazenadas. Em seguida acontece uma seleção daquelas que correspondem aos requisitos previamente levantados. Depois disso ocorre a análise dos dados coletados e em posteriormente a publicação no portal.

As cinco fases apresentadas são suficientes para o funcionamento do observatório que é executado pelo intermédio do *middleware* RabbitMQ, responsável pelo gerenciamento da execução dos filtros que compões cada etapa e das filas de mensagens que serão alimentadas e consumidas.

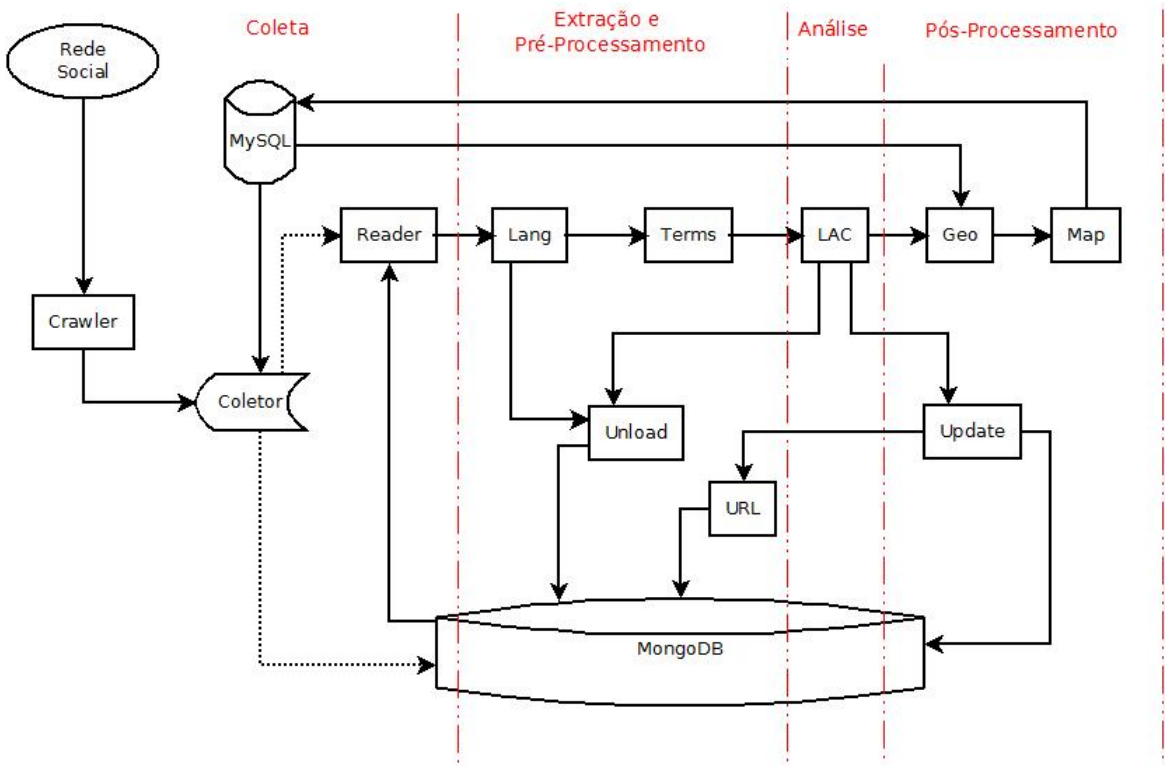


Figura 3.3: Arquitetura do Observatório da Web

3.2 Caracterização dos Componentes e Modelagem no sistema padrão UML

A partir da documentação fornecida [10], de reuniões com a equipe e com os responsáveis pelo Observatório da Web e da análise do código do Observatório da Dengue realizada por Leonardo Melo [17], sem a qual seria impossível garantir plena validade do modelo, foram encontrados os seguintes componentes interligados como representado na Figura 3.3.

Como só foi possível acessar os dados do Observatório da Dengue e ainda sem a parte de publicação, serão caracterizados apenas os componentes que tivemos acesso aos códigos. São eles: Coletor, Reader, Lang, Terms, LAC, Geo, Map, Unload, Update e URL.

São chamados de módulos ou filtros cada etapa da execução do Observatório, que serão descritas a seguir.

1. Rede Social – Componente externo, rede social online (RSO). Exemplos: Twitter, Facebook e fóruns.
2. Crawler - Componente responsável pela coleta na RSO. Exemplo: API do twitter.
3. Coletor – Executa uma rotina pré-definida a fim de buscar novos comentários acerca do tema escolhido através do Crawler. Armazena diretamente na fila Reader no RabbitMQ ou, caso o RabbitMQ não estiver acessível, no banco de dados MongoDB.

4. Mongo DB – Banco de dados responsável pela armazenagem da maior parte dos dados do sistema. Por isso, este encontra-se em uma máquina diferente da do sistema e é espelhado para uma maior segurança.

5. MySQL – Banco de dados responsável por armazenar arquivos de configuração e a parte de publicação na WEB.

6. RabbitMQ – Middleware responsável por gerenciar os dados do Observatório. Organiza os dados em uma fila do tipo FIFO (first in, first out) em que o primeiro dado que entra na fila é o primeira a sair. esse middleware gerencia todos os processos que compõem o pipeline do Observatório, e cada fila recebe o mesmo nome do processo que irá consumi-la. São eles: Reader, Lang, Terms, LAC, Geo, Map, URL, Unload e Update.

7. Reader - Responsável pela criação do workflow, ou seja, das filas dos processos no RabbitMQ, também recebe os dados do Coletor ou do MongoDB e alimenta a fila Lang.

8. Lang - Responsável pela detecção do idioma da mensagem coletada. Para mensagens do Twitter, este faz isso apenas verificando se o campo referente a localização do emissor da mensagem é o desejado. Ex: para português o campo de localização deve ser "PT". No caso do Facebook, apenas as mensagens consideradas pelo Facebook como a língua desejada são retornadas. Caso a mensagem seja na língua desejada, o processo Lang alimenta a fila Terms, caso contrário, alimenta-se a fila Unload.

9. Terms - A partir de uma lista pré-definida de termos relevantes ao tema do Observatório em questão realiza-se uma filtragem das mensagens buscando apenas aquelas que contenham uma ou mais palavras da lista. Nessa etapa também pode ser realizado o procedimento de Stemming que, segundo Felipe Nunes [12], é o processo de conflação[1] de formas diferentes de uma palavra em uma representação única. Essa representação é chamada stem. No caso de verbos, um stemmer reduziria todas as conjugações em seu radical, por exemplo: escrever e escreveu seriam reduzidos para escrev. Em um stemmer essa ideia se amplia para todas as formas linguísticas. Percebe-se que não há exigência de que o stem pertença ao vocabulário da língua nativa, basta que este guarde o significado original da palavra. Stemmers são largamente utilizados em sistemas de busca e recuperação de informações. Como o significado de certas palavras depende do contexto em qual estão inseridas, essa redução pode resultar em exceções.

10. LAC (*Lazy Associative Classification*)[19] - Classificador responsável por mapear mensagens a classes correspondentes. A classificação fica mais evidente no Observatório das Eleições em que a mensagem serve de voto a favor ou contra um dos candidatos. O LAC classifica uma variável a partir de um conjunto de regras previamente configurado que cresce com o tempo. Quanto maior a ocorrência de certa expressão ou palavra na coleta, maior a probabilidade dessa se tornar uma nova regra para o LAC. Nessa etapa é realizada a remoção de stopwords ou, em uma tradução livre: palavras de parada. São palavras irrelevantes para os resultados em

[1] Freund(1982)[13], conflação é uma tradução livre do inglês (*conflation*) que refere-se a utilização indistinta de dois ou mais termos para uma determinada finalidade.

máquinas de busca ou para mineração de dados como é o caso do observatório da web. A princípio, são aceitas como *stopwords* palavras como artigos, conjunções e proposições, no entanto, a escolha das stopwords não é tão simples assim. Em uma busca pelo filme "O Chamado" o artigo não pode ser considerado uma *stopword* já que modifica o contexto da palavra chamado, ao contrario de "comprei um carro novo" onde o artigo "um" pode ser subtraído pois não altera a busca ou a mineração. Sendo assim, a escolha das *stopwords* é delicada pois pode influenciar no resultado final da pesquisa. Uma técnica para determinar se uma palavra é uma *stopword* ou não é quebrar as palavras da pesquisa em *tokens* contendo e não contendo palavras de parada. Em seguida pesquisa-se com uma ferramenta de busca na internet e compara-se então o resultado da pesquisa. Se a *stopword* modifica significativamente o resultado a palavra não é considerada *stopword*[18]. Por fim, quando a mensagem é considerada spam, o filtro LAC alimenta a fila Unload, caso contrario, alimenta a fila Geo e Update com a mesma mensagem.

11. Geo - Procura determinar uma localização bem definida do local de onde foi enviada a mensagem. A partir do campo de coordenada geográfica e de uma lista pré-definida e armazenada no MySQL com cidades e suas respectivas coordenadas, o filtro Geo busca determinar a cidade de onde foi enviada a mensagem. Caso exista cidade correspondente, o filtro alimenta a fila Map.

12. Map - Prepara a mensagem para publicação no portal e a salva no MySQL.

13. Unload - Responsável por recolher as mensagens recusadas pelos filtros Lang e LAC e armazena-las no MongoDB para fins de estatística.

14. Update - Recebe as mensagens do filtro Lac e atualiza o MongoDB. Caso a mensagem possua URLs[2] comprimidas, o filtro Update passa esta para o filtro URL.

15. URL - Responsável por expandir URLs comprimidas.

Para a devida compreensão das ações de cada componente, foram levantados modelos de comportamento do sistema, em notação UML [3], Foram utilizados diagramas de atividades [Figura 3.5], componentes [Figura 3.4] e sequência.

Apesar de ser idealizado como uma sequência de etapas, arquitetura mediada pelo *middleware* possibilita relativa independência na execução de cada módulo. A Figura 3.5 apresenta o diagrama de atividades do Observatório da Dengue.

A Figura 3.6 apresenta um diagrama de sequência que mostra os componentes envolvidos durante a etapa de Coleta de dados na rede social. O coletor conecta-se ao banco de dados MySQL e solicita os parâmetros para iniciar a coleta de dados. Em seguida aciona o crawler que coleta os dados solicitados na rede social. Por fim, o coletor conecta-se ao RabbitMQ, se conectar com sucesso, armazena diretamente no RabbitMQ na fila Reader, caso contrario, o coletor conecta-se ao MongoDB e armazena os dados coletados no nesse banco de dados.

No módulo Reader representado na Figura 3.7 é onde ocorre toda a criação do *workflow* no RabbitMQ. É nessa etapa que são criadas as filas de cada processo. Primeiramente o

[2]Uma URL (*Uniform Resource Locator*), em português Localizador Padrão de Recursos, é o endereço de um recurso disponível em uma rede.

[3]Unified Modeling Language http://www.uml.org

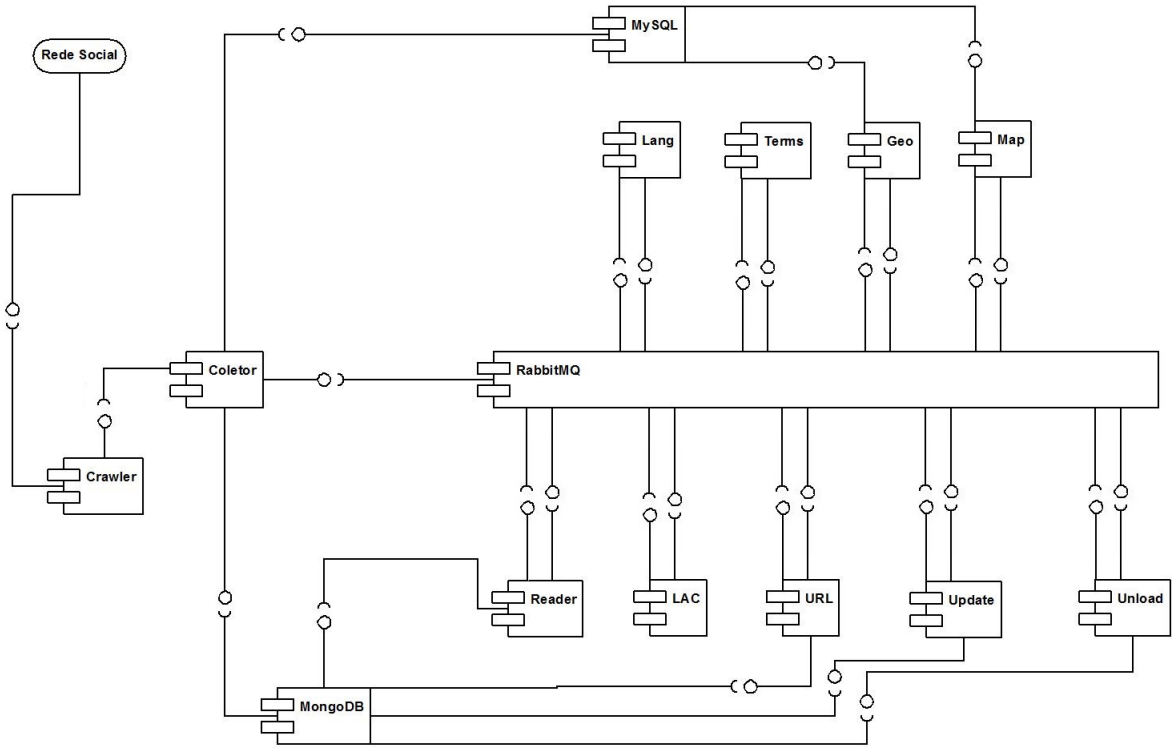


Figura 3.4: Diagrama de Componentes do Observatório da Web

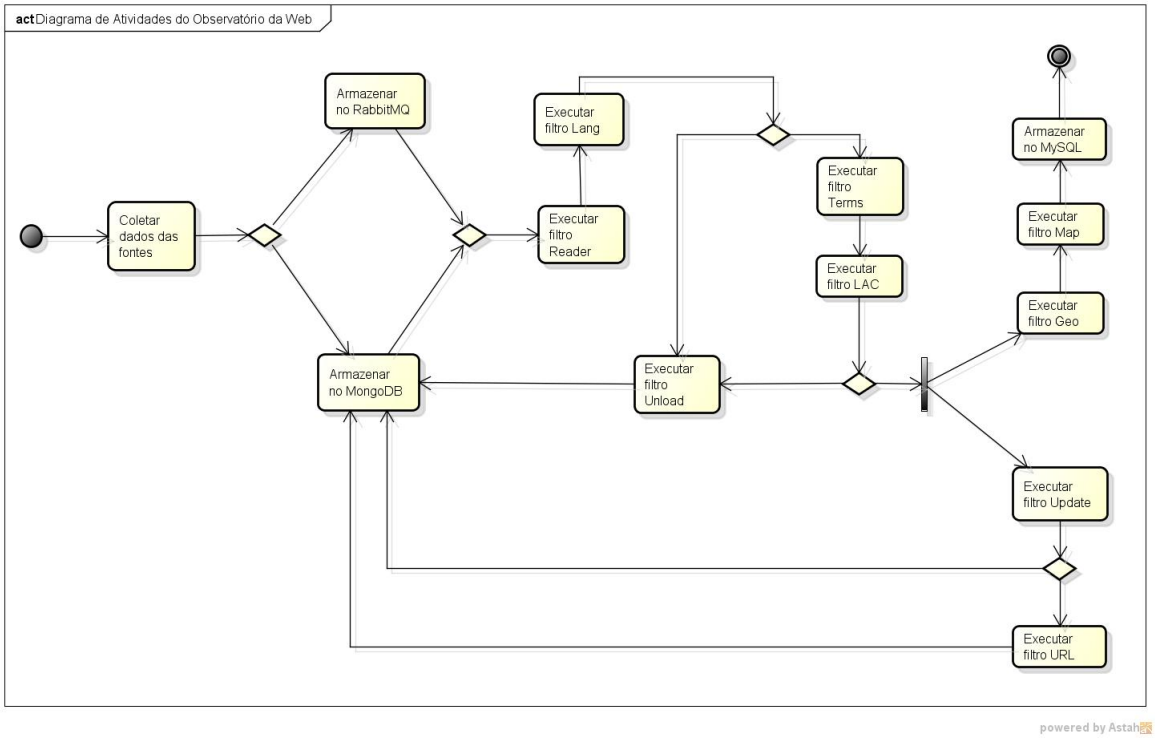


Figura 3.5: Diagrama de Atividades do Observatório da Web

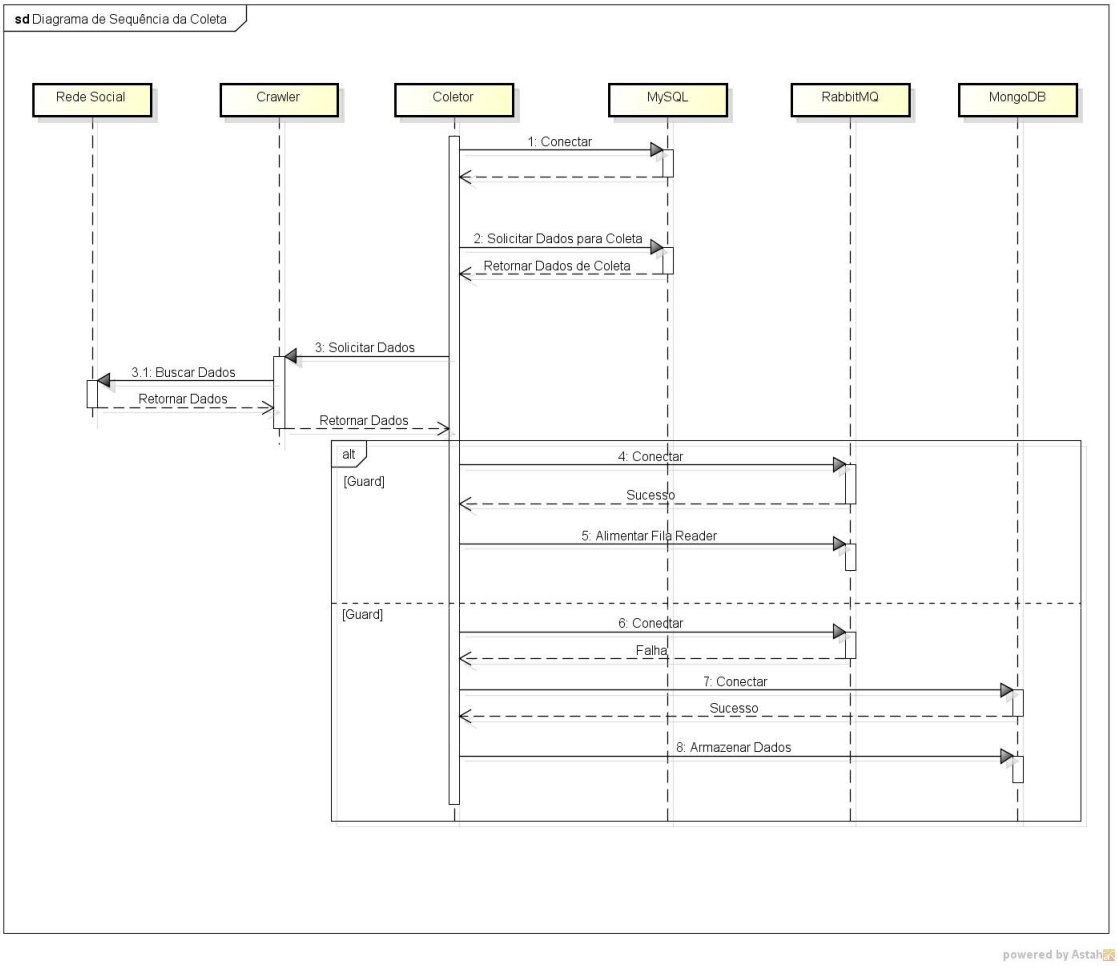


Figura 3.6: Diagrama de Sequência do Módulo de Coleta

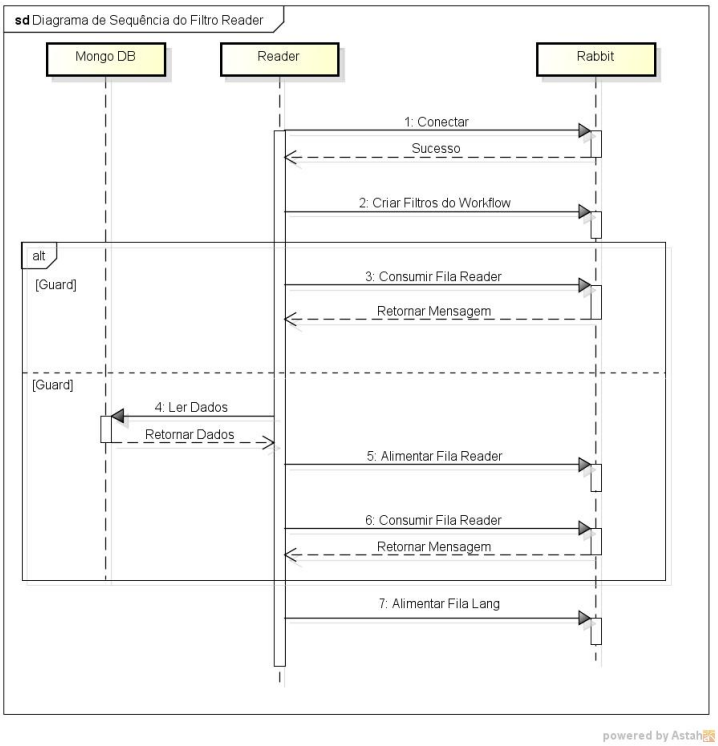


Figura 3.7: Diagrama de Sequência do Módulo Reader

processo Reader conecta-se com o RabbitMQ, em seguida cria todo o *workflow* e tenta consumir a fila Reader. Caso não existam dados na fila Reader, isso se dá caso o Observatório tenha acabado de ter sido instanciado ou o RabbitMQ tenha saído do ar, então o Processo Reader tenta consumir diretamente do MongoDB. Por fim a fila Lang é alimentada, seja por dados vindos da fila RabbitMQ ou do MongoDB.

O processo Lang [Figura3.8] é executado em seguida. Nessa etapa o processo conecta-se com o Rabbit e consome a fila Lang no RabbitMQ. A mensagem provinda do twitter passa por verificação e localidade, que consiste na checagem do campo de origem da mensagem. No cado do Observatório da Dengue, são aproveitadas apenas as mensagens na língua portuguesa e espanhola, então as mensagens que contém dados diferentes de PT ou ES no campo de localidade são adicionadas na fila Unload pois não serão aproveitadas. As demais São adicionadas na fila Terms. Caso a coleta seja do Facebook, o próprio define a língua da mensagem durante a coleta.

A fila Terms é consumida pelo processo Terms como a Figura 3.9 mostra. Nessa etapa são selecionadas apenas as mensagens que contenham uma ou mais dos termos citados em uma lista que compõe o filtro assim que o Observatório é criado. Nessa etapa pode ser realizado o processo de Stemming [Figura 9] descrito anteriormente. Por fim, o processo alimenta a fila LAC.

A etapa seguinte é a do processo LAC [Figura3.10]. Nesse filtro executa-se a remoção de *stopwords*10. Após consumir a mensagem da fila com mesmo nome, executa-se a remoção de *stopwords*10. Em seguida ocorre a classificação e posteriormente é verificado se no campo referente a posição geográfica. Caso exista alguma referência geográfica, alimenta-se as filas Geo e Update com a mesma mensagem. Caso contrário, alimenta-se

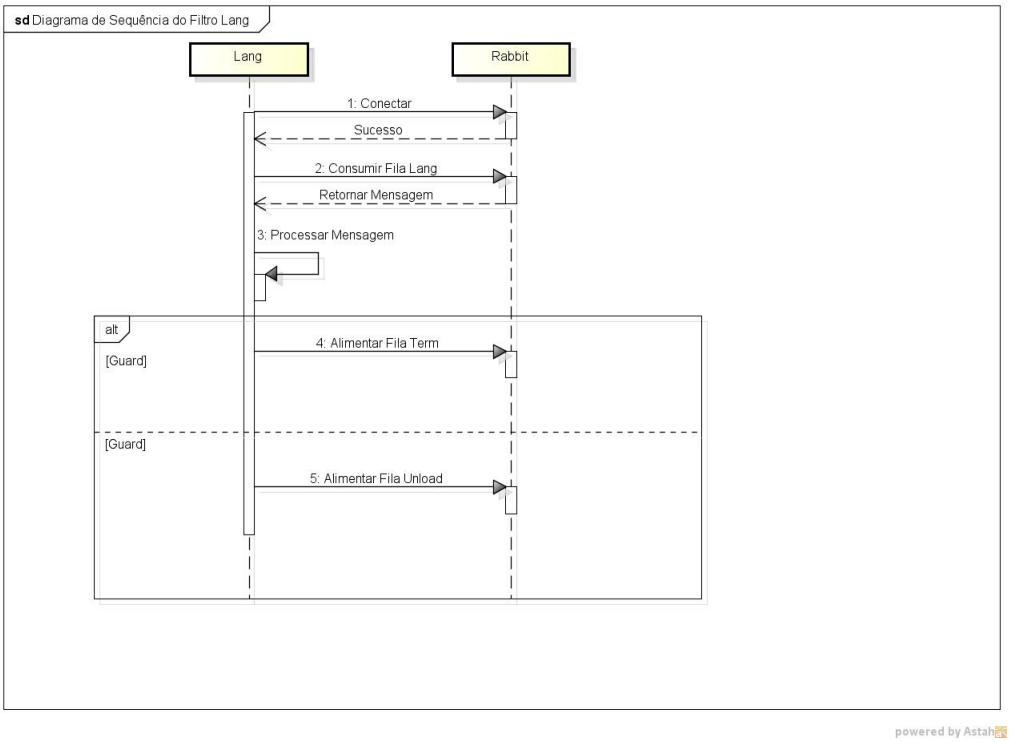


Figura 3.8: Diagrama de Sequência do Módulo Lang

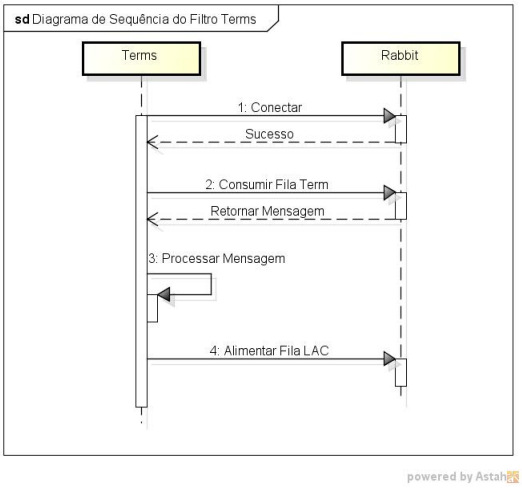


Figura 3.9: Diagrama de Sequência do Módulo Terms

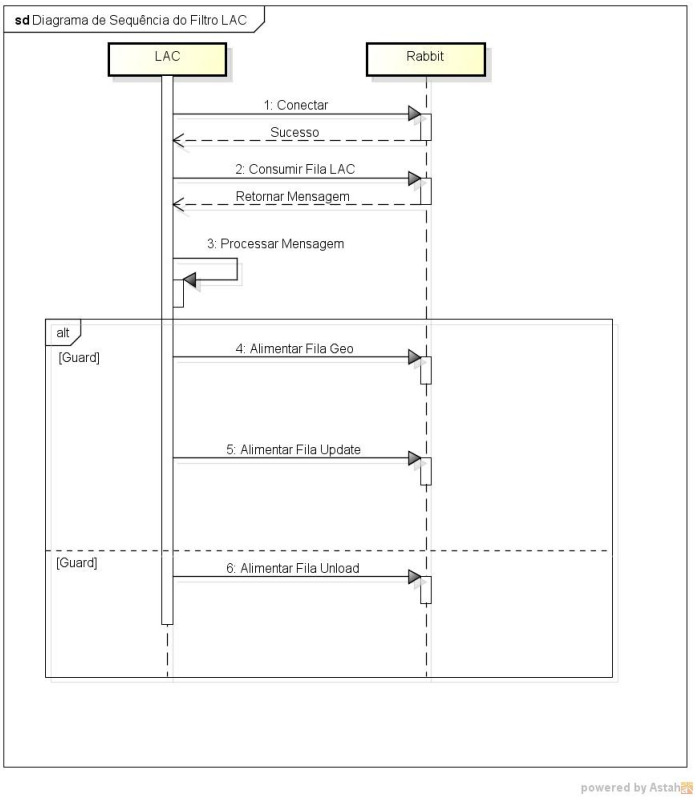


Figura 3.10: Diagrama de Sequência do Módulo LAC

a fila Unload.

A Figura 3.11 a seguir montra o funcionamento do filtro Geo. Nessa etapa ocorre a conexão padrão de todo filtro com o RabbitMQ, mas também com o MySQL a fim de buscar uma lista de referências para as coordenadas geográficas contidas na mensagem. Consome-se então a fila Geo e verifica-se se há referência com a lista, caso exista, a fila Map é alimentada com a mensagem.

Depois do filtro Geo executa-se então o filtro Map [Figura 3.12]. Nesse ponto as mensagens são consumidas da fila Map e salvas de forma apropriada para posterior publicação no portal.

Já o filtro Update [Figura 3.13] consome a fila Update e verifica se existe URL comprimida na mensagem. Caso exista, a fila URL é alimentada com a mensagem, caso não exista, o filtro verifica se a mensagem existe no banco de dados MongoDB. Caso exista ele atualiza o banco de dados com informações de data e hora da filtragem, caso não exista, ele insere a mensagem na coleção Padrão do MongoDB.

No módulo URL [Figura 3.14] ocorre a tentativa de expansão de URLs comprimidas seguida na atualização do MongoDB da mesma forma que no filtro Update.

Verifica se a mensagem recebida existe na coleção padrão no MongoDB. Caso exista, Exclui-se o dado da coleção Padrão e insere-se a mensagem na coleção Unload. Caso a mensagem não exista na coleção Padrão, atualiza-se a coleção Padrão de forma semelhante a realizada no filtro Update [Figura 3.2].

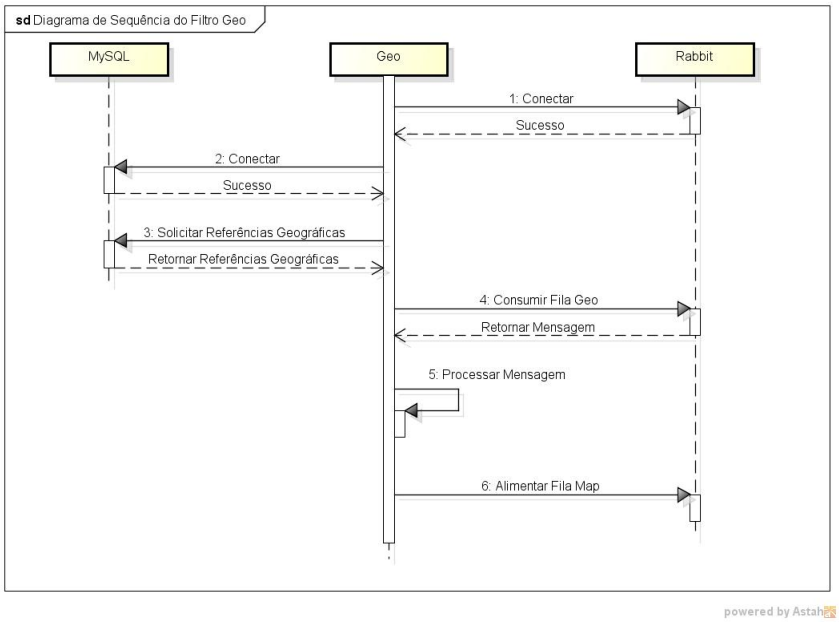


Figura 3.11: Diagrama de Sequência do Módulo Geo

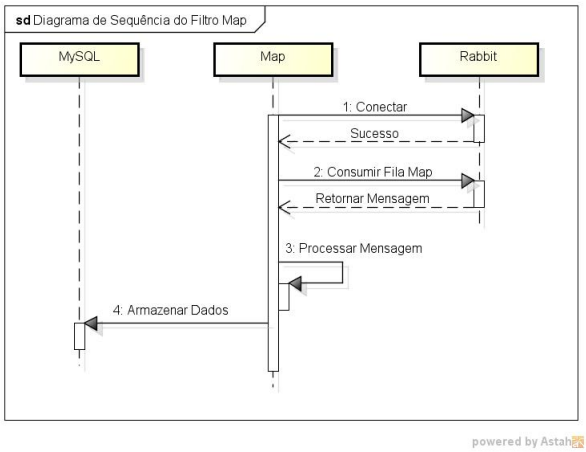


Figura 3.12: Diagrama de Sequência do Módulo Map

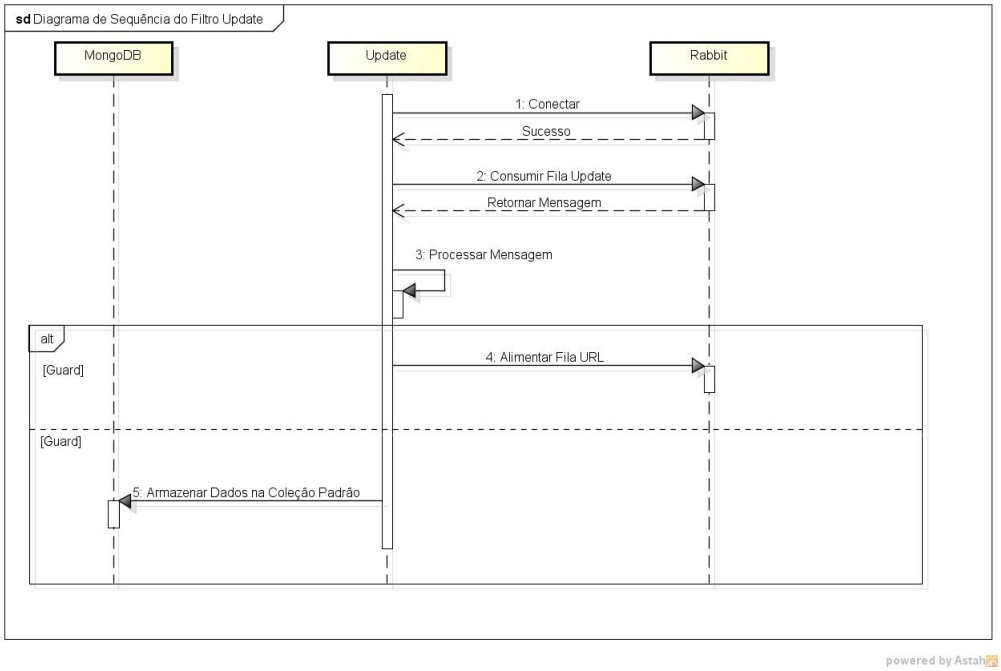


Figura 3.13: Diagrama de Sequência do Módulo Update

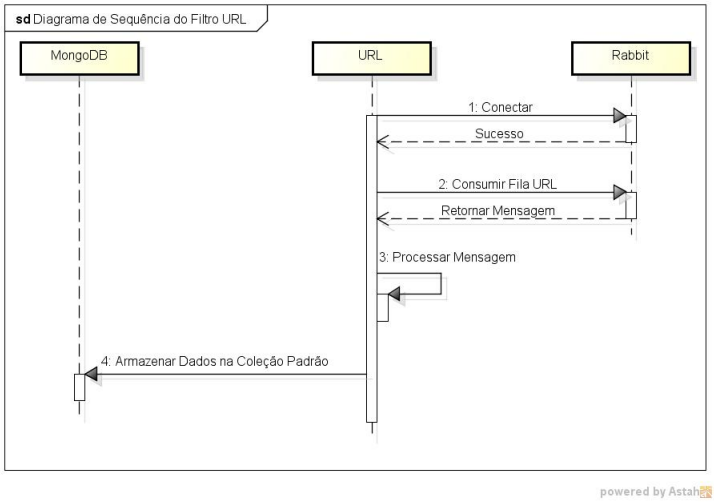


Figura 3.14: Diagrama de Sequência do Módulo URL

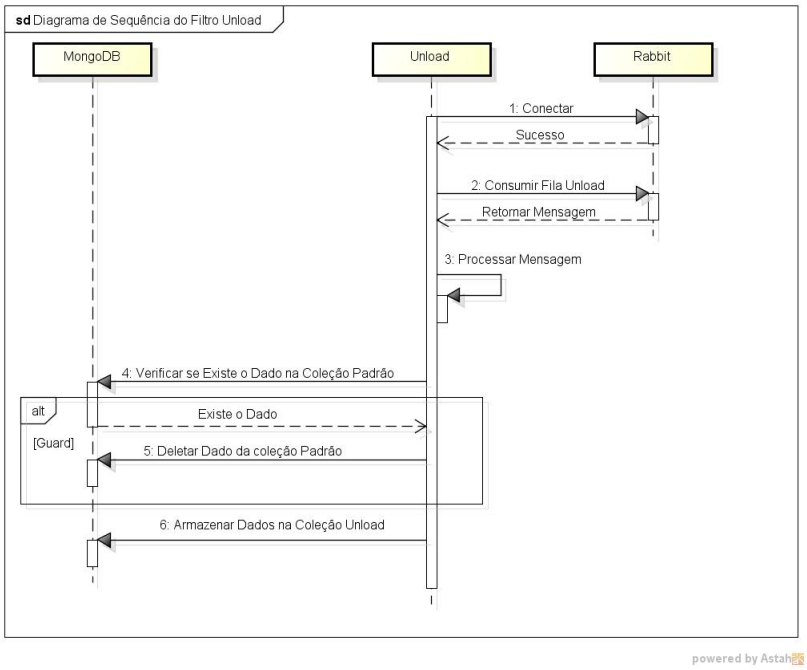


Figura 3.15: Diagrama de Sequência do Módulo Unload

3.3 Conversão da modelagem UML para a a linguagem PRISM

Na análise proposta, foi utilizado o modelo DTMC [4] para modelar o Observatório. A modelagem consiste primeiramente num preâmbulo com as variáveis correspondentes a todos os componentes envolvidos nos diagramas de sequência, além de 3 variáveis de controle: ptlang, problac e proburl, responsáveis por armazenar as probabilidades de conter mensagem na língua configurada (no caso do Observatório da Dengue, português ou espanhol); conter referência geográfica; conter url comprimida; respectivamente. Seus valores Constam na Tabela 4.3.

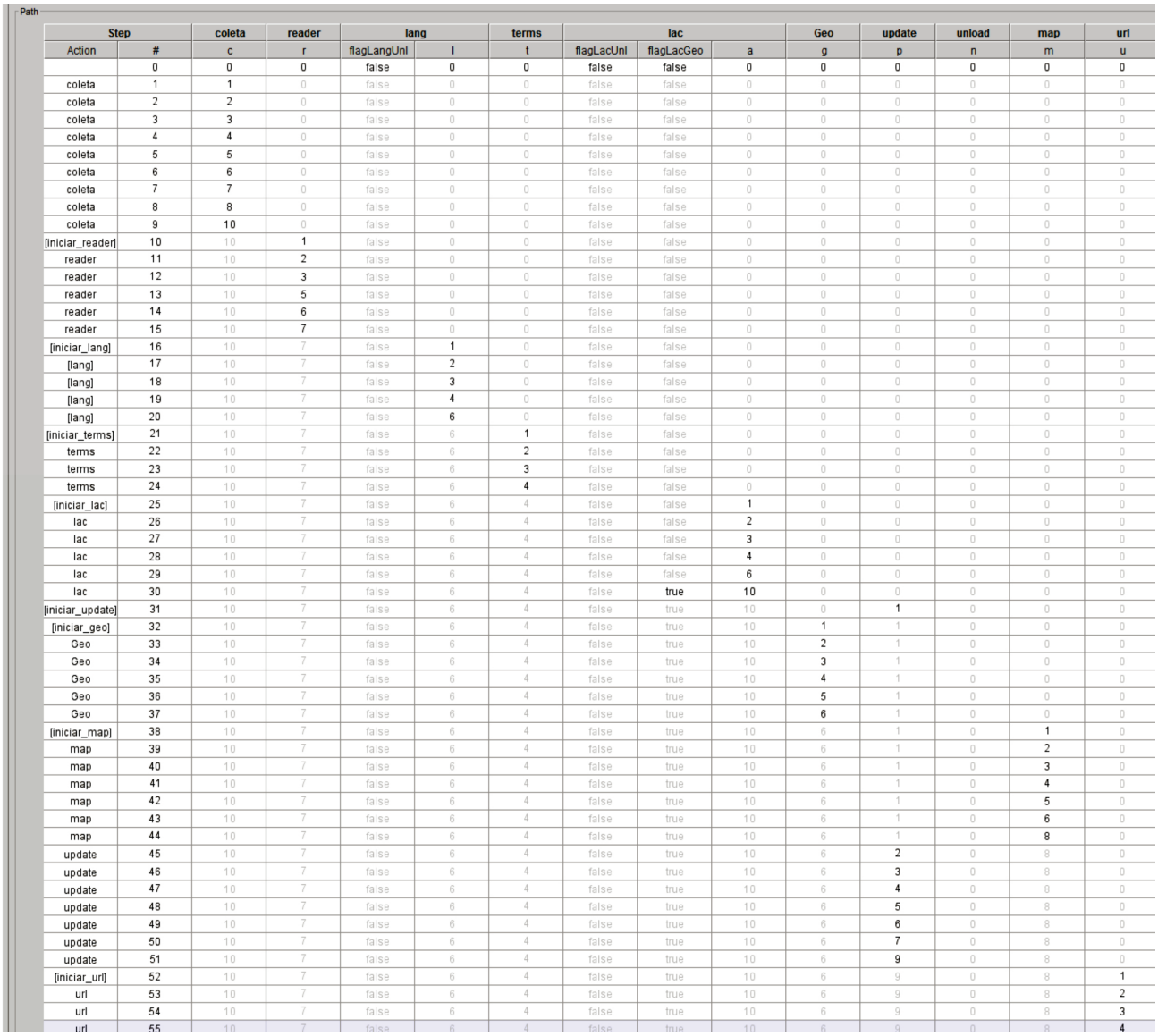
Pra cada etapa do Observatório há um módulo representando a execução daquela etapa. Dentro de cada módulo há uma variável única cujo valor representa onde a execução encontra-se. A transição entre módulos dá-se pelo sincronismo de *labels* ou por *flags* como descrito anteriormente.

Após a construção do modelo, o mesmo foi validado no ambiente de simulação (*simulation*) no PRISM. A validação consiste em buscar *deadlocks* durante qualquer possível execução do observatório e confirmar se os caminhos modelados no PRISM correspondem aos caminhos corretos como se espera. Como esperado, nenhum *deadlocks* foi encontrado e todos os estados finais possíveis foram alcançados.

A modelagem completa encontra-se no Apêndice A.

A Figura 3.16 mostra uma simulação no ambiente *simulation* do PRISM relativa a execução do Observatório. A imagem simula uma mensagem coletada pelo Coletor, que passou pelos módulos, Reader, Lang, Terms, LAC, Geo e Map até a publicação no portal

[4] Do inglês - Discrete Time Markov Chain



Path

Step		coleta	reader	lang		terms	lac			Geo	update	unload	map	url
Action	#	c	r	flagLangUnl	l	t	flagLacUnl	flagLacGeo	a	g	p	n	m	u
	0	0	0	false	0	0	false	false	0	0	0	0	0	0
coleta	1	1	0	false	0	0	false	false	0	0	0	0	0	0
coleta	2	2	0	false	0	0	false	false	0	0	0	0	0	0
coleta	3	3	0	false	0	0	false	false	0	0	0	0	0	0
coleta	4	4	0	false	0	0	false	false	0	0	0	0	0	0
coleta	5	5	0	false	0	0	false	false	0	0	0	0	0	0
coleta	6	6	0	false	0	0	false	false	0	0	0	0	0	0
coleta	7	7	0	false	0	0	false	false	0	0	0	0	0	0
coleta	8	8	0	false	0	0	false	false	0	0	0	0	0	0
coleta	9	10	0	false	0	0	false	false	0	0	0	0	0	0
[iniciar_reader]	10	10	1	false	0	0	false	false	0	0	0	0	0	0
reader	11	10	2	false	0	0	false	false	0	0	0	0	0	0
reader	12	10	3	false	0	0	false	false	0	0	0	0	0	0
reader	13	10	5	false	0	0	false	false	0	0	0	0	0	0
reader	14	10	6	false	0	0	false	false	0	0	0	0	0	0
reader	15	10	7	false	0	0	false	false	0	0	0	0	0	0
[iniciar_lang]	16	10	7	false	1	0	false	false	0	0	0	0	0	0
[lang]	17	10	7	false	2	0	false	false	0	0	0	0	0	0
[lang]	18	10	7	false	3	0	false	false	0	0	0	0	0	0
[lang]	19	10	7	false	4	0	false	false	0	0	0	0	0	0
[lang]	20	10	7	false	6	0	false	false	0	0	0	0	0	0
[iniciar_terms]	21	10	7	false	6	1	false	false	0	0	0	0	0	0
terms	22	10	7	false	6	2	false	false	0	0	0	0	0	0
terms	23	10	7	false	6	3	false	false	0	0	0	0	0	0
terms	24	10	7	false	6	4	false	false	0	0	0	0	0	0
[iniciar_lac]	25	10	7	false	6	4	false	false	1	0	0	0	0	0
lac	26	10	7	false	6	4	false	false	2	0	0	0	0	0
lac	27	10	7	false	6	4	false	false	3	0	0	0	0	0
lac	28	10	7	false	6	4	false	false	4	0	0	0	0	0
lac	29	10	7	false	6	4	false	false	6	0	0	0	0	0
lac	30	10	7	false	6	4	false	true	10	0	0	0	0	0
[iniciar_update]	31	10	7	false	6	4	false	true	10	0	1	0	0	0
[iniciar_geo]	32	10	7	false	6	4	false	true	10	1	1	0	0	0
Geo	33	10	7	false	6	4	false	true	10	2	1	0	0	0
Geo	34	10	7	false	6	4	false	true	10	3	1	0	0	0
Geo	35	10	7	false	6	4	false	true	10	4	1	0	0	0
Geo	36	10	7	false	6	4	false	true	10	5	1	0	0	0
Geo	37	10	7	false	6	4	false	true	10	6	1	0	0	0
[iniciar_map]	38	10	7	false	6	4	false	true	10	6	1	0	1	0
map	39	10	7	false	6	4	false	true	10	6	1	0	2	0
map	40	10	7	false	6	4	false	true	10	6	1	0	3	0
map	41	10	7	false	6	4	false	true	10	6	1	0	4	0
map	42	10	7	false	6	4	false	true	10	6	1	0	5	0
map	43	10	7	false	6	4	false	true	10	6	1	0	6	0
map	44	10	7	false	6	4	false	true	10	6	1	0	8	0
update	45	10	7	false	6	4	false	true	10	6	2	0	8	0
update	46	10	7	false	6	4	false	true	10	6	3	0	8	0
update	47	10	7	false	6	4	false	true	10	6	4	0	8	0
update	48	10	7	false	6	4	false	true	10	6	5	0	8	0
update	49	10	7	false	6	4	false	true	10	6	6	0	8	0
update	50	10	7	false	6	4	false	true	10	6	7	0	8	0
update	51	10	7	false	6	4	false	true	10	6	9	0	8	0
[iniciar_url]	52	10	7	false	6	4	false	true	10	6	9	0	8	1
url	53	10	7	false	6	4	false	true	10	6	9	0	8	2
url	54	10	7	false	6	4	false	true	10	6	9	0	8	3
url	55	10	7	false	6	4	false	true	10	6	9	0	8	4

Figura 3.16: Simulação da execução do modelo no PRISM

e também pelos filtros Update e URL pois contia URL comprimida que precisou ser expandida antes do armazenamento no MongoDB.

3.3.1 Propriedades - ProProST

Para obter uma descrição das propriedades PCTL próximas às que serão utilizadas no PRISM foram selecionados alguns módulos do Observatório e possíveis indicadores. Utilizando a ferramenta *ProProST Wizard*, tais indicadores foram especificados utilizando o ProProST como veremos na Tabela 3.1.

Já na Tabela 3.2, tais indicadores foram especificados em PCTL utilizando o ProProST.

Tabela 3.1: Possíveis Indicadores dos Módulos

Módulo	Indicador	Definição	Forma de Obtenção
Coletor	Cobertura	Avalia o quão completa a coleta de dados está de acordo com o universo que deveria ter sido coletado.	Depende da comparação de duas coletas da mesma fonte de dados de maneira diferente.
Reader	Cobertura	Avalia se a leitura de dados da fila e do banco de dados foi completa.	Compara-se o total de mensagens que foram para o próximo filtro e a quantidade anteriormente armazenada na fila Reader e no MongoDB.
Lang	Precisão	Avalia se as mensagens tiveram sua linguagem classificada adequadamente.	Depende de uma verificação manual para determinar se a língua da mensagem corresponde a determinada.
Geo	Cobertura	Avalia a abrangência da base de dados das cidades do Observatório.	Depende de um processo manual de verificação das coordenadas para determinar se a localização realmente é inadequada.
URL	Cobertura	Avalia a qualidade na expansão de URLs.	Também depende de um processo manual para determinar se a URL foi devidamente expandida.

3.4 Verificação da alcançabilidade

A verificação da alcançabilidade do Observatório será verificada a pela Propriedade 3.1 que verifica a probabilidade de se alcançar qualquer estado final do Observatório.

$$P =?[F(n = 6)|((m = 8)\&((u = 4)|(p = 10)))] \tag{3.1}$$

Outras Propriedades também serão analisadas durante a análise dos dados. São elas:

1. Probabilidade de se alcançar o estado de publicação no portal [Equação 3.2];
2. Probabilidade de se alcançar o estado de publicação e armazenamento no banco de dados [Equação 3.3];

$$P =?[F(m = 6)] \tag{3.2}$$

$$P =?[F(m = 8)\&((u = 4)|(p = 10))] \tag{3.3}$$

Tabela 3.2: Propriedades Especificadas pelo ProProST e em PCTL

Módulo	Indicador	ProProST Wizard	Propriedade
Coletor	Cobertura	The system shall have a behavior where with a probability >0.99 it is the case that coleta1=coleta2 holds in run.	S>0.99[(coletor1=coleta2)]
Reader	Cobertura	The system shall have a behavior where with a probability >=0.99 it is the case that (readerFila+ReaderMongoDB)=ReaderTotal holds in run.	S>=0.99[((readerFila+ReaderMongoDB)=ReaderTotal)]
Lang	Precisão	The system shall have a behavior where with a probability >=0.98 it is the case that (linguaMensagem=TRUE) holds in run	S>=0.98[(linguaMensagem=TRUE)]
Geo	Cobertura	The system shall have a behavior where with a probability >=0.98 it is the case that (coordenada=TRUE) holds in run	S>=0.98[(coodenada=TRUE)]
URL	Cobertura	The system shall have a behavior where with a probability >=0.98 it is the case that (url=TRUE) holds in run	S>=0.98[(url=TRUE)]

Capítulo 4

Análise dos Dados

Nesse capítulo será avaliada a confiabilidade do Observatório conforme metodologia apresentada. Os dados para avaliação foram coletados no período de 17 de agosto de 2013 até 09 de setembro de 2013 na execução do Observatório da Dengue. Foram utilizadas a Ferramenta de Inspeção [2] para o monitoramento de falhas no software e o Zabbix [1] para monitoramento de hardware.

Todos os dados utilizados nessa avaliação foram coletados por Leonardo Melo e Felipe Gules [17].

4.1 Ferramentas

4.1.1 Ferramenta de Inspeção

A Ferramenta de Inspeção [2] é um software baseado em registro de logs que permite a filtragem pelo contexto que operador considerar importante. Utiliza uma técnica intrusiva, ou seja, o código é injetado diretamente no código do software que se deseja monitorar. A ferramenta possibilita muita liberdade para o desenvolvedor e o deixa livre para escolher a melhor forma de monitorar o software, podendo o programador monitorar apenas o que deseja, facilitando na disponibilização de relatórios de eventos ocorridos ou apenas falhas. A ferramenta disponibiliza um campo para filtragem, viabilizando que só uma parte da coleta seja mostrada em tela. A filtragem pode ser feita por exclusão ou restrição de um termo qualquer. Dessa forma, o esforço na detecção de falhas no sistema é consideravelmente reduzido.

4.1.2 Zabbix

O Zabbix [1] foi a solução adotada para o monitoramento de falhas em nível de hardware e protocolos de comunicação a fim de monitorar o Observatório da Web. O intuito do monitoramento com o Zabbix foi o de monitorar e diagnosticar falhas em nível físico.

4.1.3 Monitoramento

Durante o monitoramento foram coletadas 23875 mensagens do Twitter, única rede social fonte de dados do Observatório analisado. A quantidade recebida por cada filtro

Tabela 4.1: Total de mensagens do Twitter por módulo

Filtro	Mensagens Recebidas	Destino	Mensagens Enviadas
Coletor	23875	RabbitMQ MongoDB	17680 6195
Reader	17681	Lang	17681
Lang	17681	Terms Unload	3452 14229
Terms	3452	LAC	3452
LAC	3452	Geo e Update Unload	3405 47
Geo	3428	Map	2030
Update	3408	URL MongoDB	754 2654
Map	2030	MySQL	2030
URL	218791	MongoDB	755
Unload	14283	MongoDB	14283

pode ser vista na Tabela 4.1: Durante o monitoramento pela ferramenta de inspeção foram registrados diversos defeitos.A Tabela4.2 apresenta a quantidade total de erros e defeitos em cada módulo e a confiabilidade de cada um.

A confiabilidade do sistema foi calculada utilizando-se a métrica de taxa de defeitos por mensagens recebidas como descrito anteriormente [Seção 2.1.5] segundo a equação.

$$Confiabilidade = 1 - \frac{Total\ de\ Defeitos}{Total\ de\ Mensagens\ Coletadas}$$

Para o módulo Reader, foram consideradas apenas as mensagens enviadas para o RabbitMQ já que as enviadas para o MongoBD, que teoricamente deveriam ser consumidas pelo Reader e alimentadas na fila Lang, não foram.

Para o módulo URL que registrou um grande número de mensagens recebidas do Twitter (218791), foi considerado apenas a quantidade armazenada no banco de dados MongoDB (755) para o calculo de confiabilidade. Essa grande quantidade deu-se pelo número de tentativas de expansão sem sucesso registradas como defeitos, que também foram desconsiderados no cálculo.

4.1.4 Alcançabilidade e Análise de Sensibilidade

Serão realizados 4 experimentos para análise de sensibilidade do Observatório da Dengue. O primeiro é referente alcançabilidade do sistema, o segundo e o terceiro referentes apenas aos estados finais desejáveis e o último para analisar o banco de dados MongoDB, replicado e sem replicação em relação ao RabbitMQ e ao MySQL.

O primeiro experimento foi verificar a alcançabilidade geral do sistema, ou seja, verificar a probabilidade do sistema alcançar um de seus estados finais. Para tanto, a Propriedade PCTL 4.1 foi verificada:

$$P =?[F(n = 6)|((m = 8)\&((u = 4)|(p = 10)))] \tag{4.1}$$

Tabela 4.2: Totais de Mensagens, Erros, Defeitos e Confiabilidade [17]

	Total de Mensagens	Total de Erros	Total de Defeitos	Confiabilidade
Coletor	23875	234	174	0,990
Reader	17681	450	450	0,974
Lang	17681	0	0	1
Terms	3452	0	0	1
LAC	3452	0	0	1
Geo	3428	284	5	0,998
Map	2030	0	0	1
Update	3408	0	0	1
URL	755	13	13	0,982
Unload	14283	7	7	0,999

Tabela 4.3: Probabilidades das variáveis de controle [17]

Variável	Probabilidade	Definição
ptlang	19,52%	Probabilidade da mensagem estar na língua desejada.
problac	98,63%	Probabilidade da mensagem conter referência geográfica.
proburl	22,12%	Probabilidade da mensagem conter url comprimida.

A Propriedade 4.1 tem como objetivo saber a probabilidade do sistema alcançar o estado final do módulo Unload ou do módulo Map juntamente com os estados finais do módulo Update ou Url. O resultado alcançado foi de 100% de probabilidade de sucesso.

A partir dessa propriedade foi gerado o gráfico de análise de sensibilidade [Figura 4.2]. Para tanto, foram gerados experimentos (*experiments*) variando a confiabilidade de cada componente de 0 a 1 em intervalos de 0.01, e as demais confiabilidades fixadas em 1. Apenas a variáveis de controle [Tabela 4.3] foram mantidas com os valores reais, relativos ao monitoramento e mostrados na Tabela 4.3. O gráfico final obtido está na Figura 4.1

Ainda para essa análise, foi gerado também o gráfico de sensibilidade utilizando os valores de confiabilidade reais, resultantes da coleta [Figura 4.2]. A Tabela 4.2 mostra os valores de confiabilidade dos módulos monitorados. Todo módulo, como podemos verificar na modelagem PRISM A, possui estados finais de sucesso e falha, logo, aqueles módulos na Tabela 4.2 com confiabilidade igual a 1, serão simulados com valor de 0.999 que seria o valor máximo de confiabilidade pois, ainda que não detectado erro algum durante o monitoramento, estes também estão passíveis de entrarem em estado de erro. Para os valores como confiabilidade da fonte de dados, crawler, dos bancos de dados e do RabbitMQ, componentes que não foram monitorados também foi adotado o valor máximo de $0,999$ pelo mesmo motivo. Estes dados serão considerados os valores reais, resultantes do perfil de uso observado no Observatório da Dengue. Estes dados serão utilizados nos demais experimentos.

O resultado alcançado utilizando os valores reais foi de $87,85\%$ de probabilidade de sucesso.

Por essa análise 4.2 nota-se que o *middleware* RabbitMQ é o componentes mais sensível do observatório devido a sua natureza de centralização de componentes, o que era esperado. Nota-se que para alguns componentes, a confiabilidade já inicia-se maior que

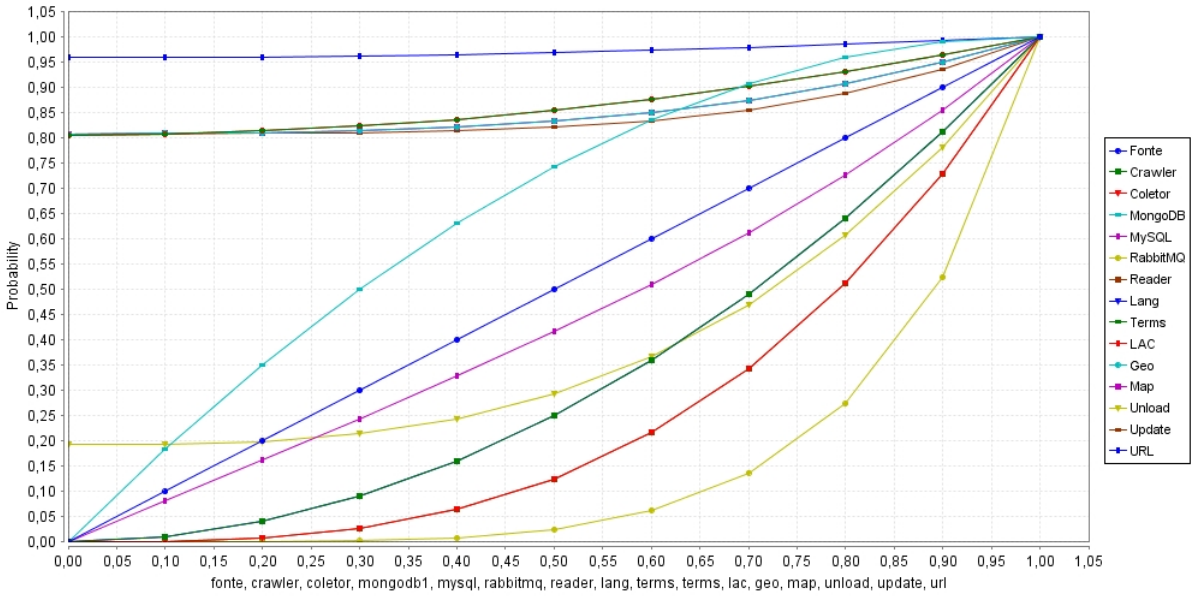


Figura 4.1: Gráfico de Sensibilidade do Observatório da Dengue - Confiabilidade 1

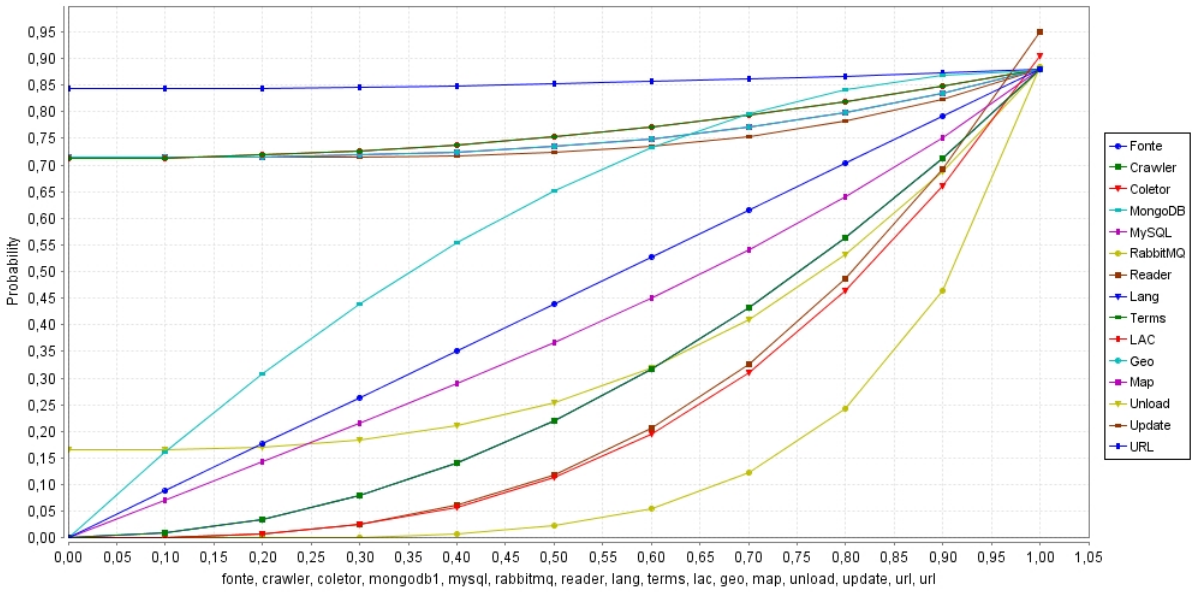


Figura 4.2: Gráfico de Sensibilidade do Observatório da Dengue - Confiabilidade real

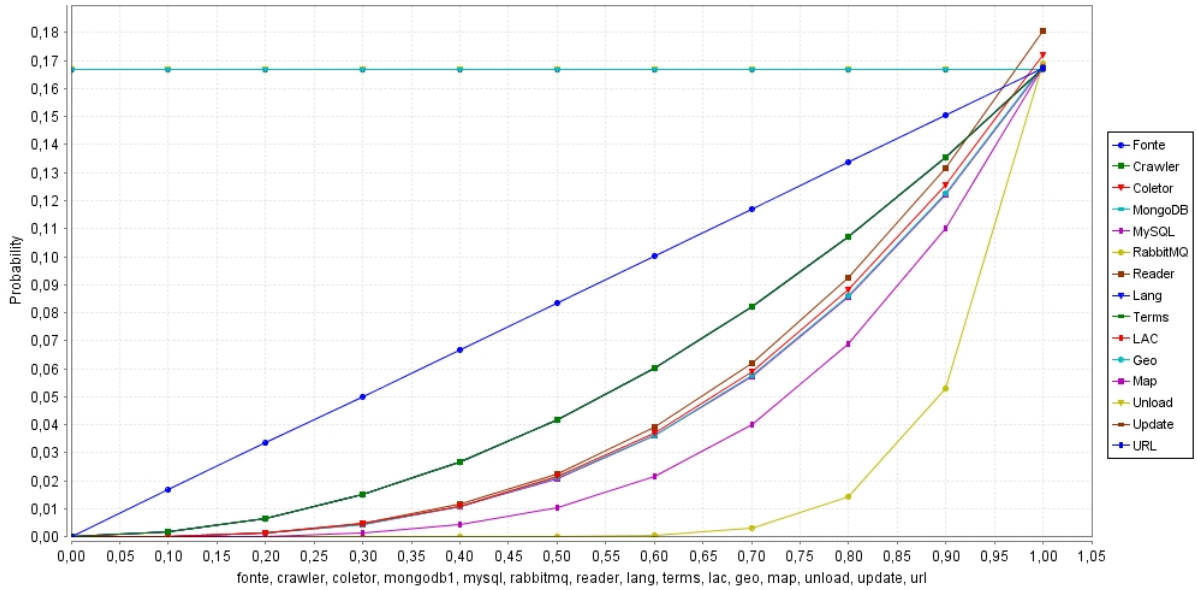


Figura 4.3: Gráfico de Sensibilidade do Observatório da Dengue - Componente Map

zero. Isso acontece devido a arquitetura do observatório com mais de um estados finais. O gráfico do componente Unload, que inicia com valor entre $0,1$ e $0,2$ tem esse comportamento porque algumas mensagens coletadas, chegam ao estado final do Observatório sem passar por esse componente. No entanto, no sistema em estudo, a maioria das mensagens coletadas não são publicadas, acabam filtradas pelo componente Lang e chegam ao componente Unload. Por esse motivo notamos que a confiabilidade do sistema é fracamente afetada por componentes como URL, Geo e Map já que poucas mensagens chegam a esse destino.

Sabendo que o objetivo do Observatório da Web são as metáforas visuais para o usuário, desejou-se saber a alcançabilidade apenas no filtro Map, já que este é o responsável pela publicação no portal. A Propriedade 4.2 então foi a utilizada:

$$P =?[F(m = 6)] \tag{4.2}$$

O resultado obtido foi bastante baixo, apenas $16,70\%$ de probabilidade de alcançar esse estágio visto que durante a filtragem do Observatório, observou-se que poucas mensagens estavam na língua desejada e não chegaram ao filtro Map.

A seguir (Figura 4.3)temos no gráfico de sensibilidade para a propriedade relativa ao filtro Map.

Nota-se que, como esperando, a confiabilidade de alguns componentes, entre eles o banco de dados MongoDB, nesse caso é irrelevante pois para alcançar o componente Map, as mensagens devem seguir um caminho bem definido no *workflow*.

Ainda em relação ao componente final Map, foi verificada a Propriedade 4.3 :

$$P =?[F(m = 8)\&((u = 4)|(p = 10))] \tag{4.3}$$

Essa propriedade tem o objetivo de calcular a probabilidade do sistema alcançar o componente Map, assim como os componentes Update ou URL pois além de publicar a

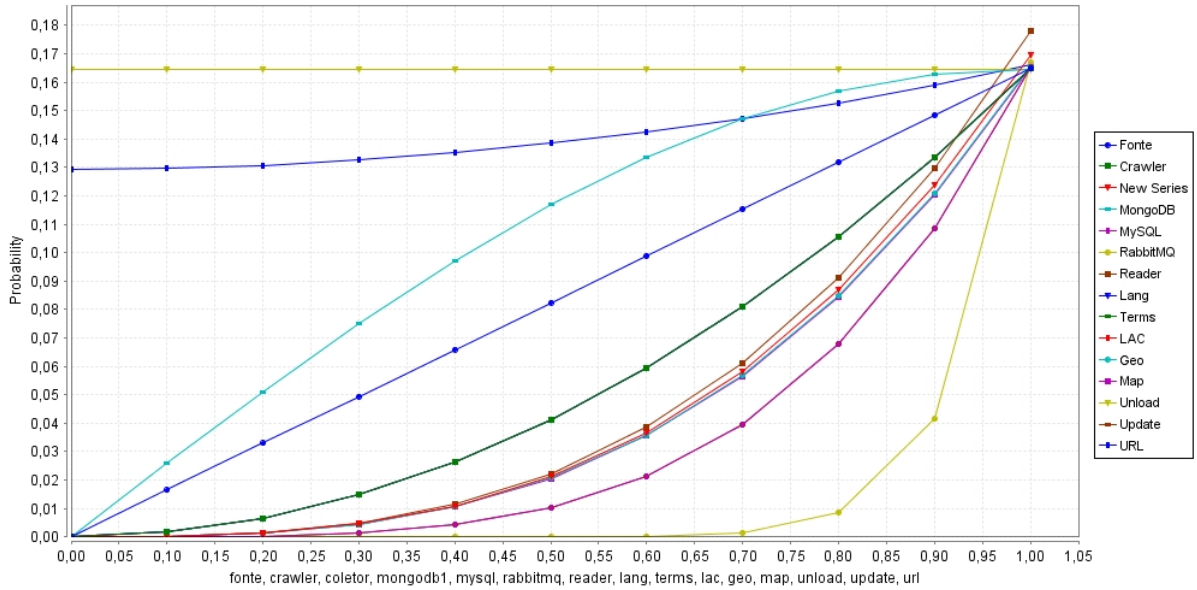


Figura 4.4: Gráfico de Sensibilidade do Observatório da Dengue - Componentes Map, Update e URL

mensagem no portal, deseja-se que a mesma seja armazenada no banco de dados MongoDB, e se houver URL comprimida, ela deve ser expandida antes da gravação no banco. O valor obtido foi de apenas $16,46\%$ e o gráfico de sensibilidade é mostrado na figura 4.4.

Como esperado, o componente Unload é irrelevante para essa propriedade. O gráfico também aponta baixa relevância do componente URL visto que, das mensagens que são publicadas no portal, apenas $22,12\%$ apresentam URL comprimida.

Para a última análise foi utilizada a primeira propriedade analisada [Propriedade 4.4], ou seja, a propriedade de alcançabilidade geral do sistema:

$$P = ?[F(n = 5)|((m = 8)\&((u = 4)|(p = 10)))] \tag{4.4}$$

Nessa análise desejou-se saber a sensibilidade do banco de dados MongoDB com replicação e sem replicação em relação ao outro banco de dados utilizado no sistema (MySQL) assim como em relação do *middleware* RabbitMQ. A figura 4.5 apresenta o gráfico resultante da análise.

Percebe-se que o MongoDB replicado melhora consideravelmente a alcançabilidade do Observatório, no entanto, o banco de dados MySQL é ainda mais sensível que o MongoDB sem replicação. Ainda assim, é indiscutível que o RabbitMQ é o componente mais sensível para o sistema do Observatório da Dengue.

4.2 Caracterização do Defeitos

A confiabilidade de um sistema difere de 100% devido a defeitos provenientes de falhas dormentes no sistema. A seguir serão caracterizados os principais defeitos, classificados como defeitos graves por Leonardo Melo e Felipe Gules [17]. Os demais defeitos podem ser encontrados no Apêndice [B].

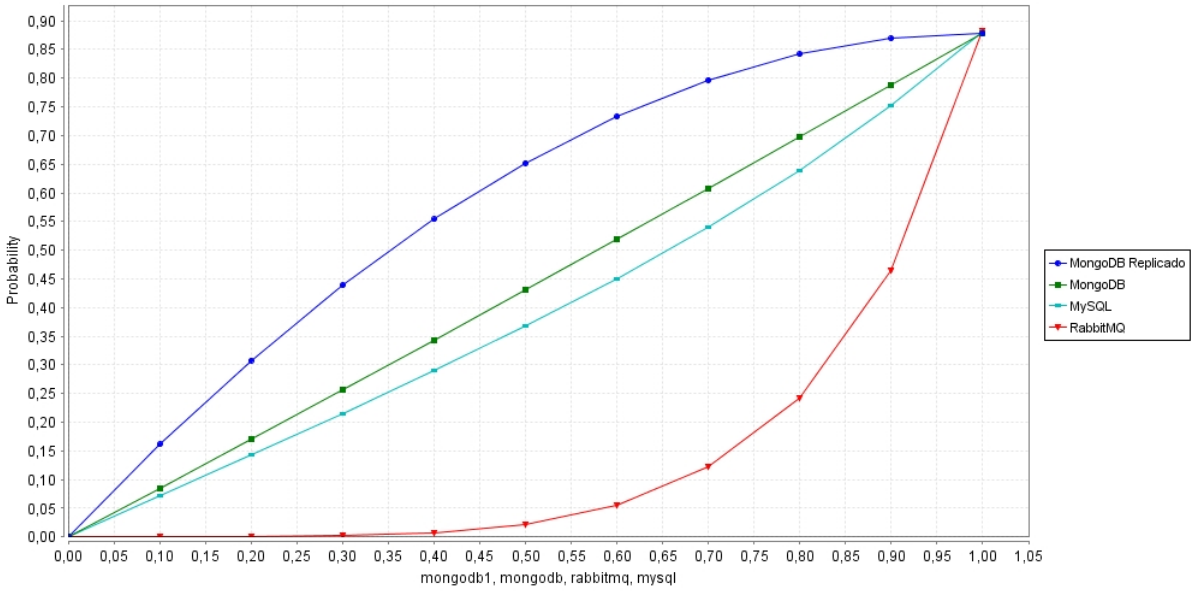


Figura 4.5: Gráfico de Sensibilidade do Observatório da Dengue - MongoDB, MySQL e RabbitMQ

1. Interrupção de serviço do processo:
    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Consistente.
    - **Consequência:** Grave.
2. Não entrega o serviço para o banco de dados MongoDB:
    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Grave.
3. Não entrega o serviço para o RabbitMQ:
    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Consistente.
    - **Consequência:** Grave.

Defeitos previstos no **Reader_filter**:

1. Não entrega o serviço para o RabbitMQ:
    - **Domínio:** Parada.

- **Detectabilidade:** Sinalizada.
- **Consistência:** Consistente.
- **Consequência:** Grave.

Defeitos previstos no **Lang_filter**:

1. Não entrega o serviço para o filtro seguinte:
   - **Domínio:** Conteúdo.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Defeitos previstos no **Terms_filter**:

1. Não entrega o serviço para o filtro seguinte:
   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Defeitos previstos no **Lac_filter**:

1. Não entrega o serviço para o filtro seguinte:
   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Defeitos previstos no **Geo_filter**:

1. Não entrega o serviço para o filtro seguinte (*RabbitMQ Fila Consumo*):
   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Defeitos previstos no **Map_filter**:

1. Não consegue salvar, no banco de dados MySQL, a localização descrita na mensagem:
   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Capítulo 5

Conclusão

O Observatório da Web foi uma iniciativa de grande importância para a sociedade atual e mostra o potencial que dados compartilhados nas redes sociais podem ter na sociedade. Para o estudo em questão foi disponibilizado apenas parte do Observatório da Denque, no entanto, foram alcançados resultados significativos mesmo com essa limitação. O sistema apresenta uma arquitetura com basicamente quatro estados/módulos finais, são eles: Unload, Map, Update e URL, sendo os dois últimos dependentes da execução do Map. Outra limitação que o estudo sofreu foi na obtenção de mensagens da fonte de dados, no caso, o *Twitter*. Foi coletada uma mensagem a cada um minuto e três segundos. Mesmo assim, o número de mensagens coletadas foi expressivo.

5.1 Resultados Alcançados

A modelagem do modelo utilizando o conceito de cadeias de Markov possibilitou que, dada a arquitetura do sistema, um modelo bastante fiel a execução fosse criado e analisado. A análise de dependabilidade mostrou que, ao contrário de estudos anteriores, o banco de dados MongoDB não é o componentes mais sensível do Observatório e sim o middleware RabbitMQ. O banco de dados MySQL mostrou-se ainda mais sensível que o MongoDB na análise, justificando também uma replicação para esse banco de dados. Verificou-se também que a maior parte das mensagens coletadas não chega a etapa de publicação pois a maior parte das mensagens coletadas foi com língua diferente da desejada, comprometendo assim grande parte do desempenho do Observatório. A quantidade de mensagens do twitter e de falhas pode variar de acordo com a instanciação do Observatório assim como do período em que é realizada. Em algumas regiões do Brasil verifica-se períodos do ano com mais incidência de Dengue por exemplo, o que pode incidir na mudança dos resultados.

5.2 Trabalhos Futuros

Esse trabalho concentrou-se na modelagem da arquitetura seguida da análise dos dados. A arquitetura encontrava-se incompleta e carece de complementação e validação por parte da equipe desenvolvedora. A coleta de dados foi possível por uma instanciação do Observatório da Dengue instalada em uma máquina virtual em um arcabouço restrito e

diferente do real além de ter sido realizado em um curto período de tempo. Nesse caso, a metodologia carece de um ambiente mais próximo do real e com todas as possíveis variáveis envolvidas. Como sugestão, a coleta deveria ser direcionada ao objetivo final do Observatório e a língua da mensagem poderia já ser filtrada durante a coleta pois o *crawler* disponibilizado pelo Twitter possibilita tal restrição, potencializando assim a ferramenta do Observatório. Por fim, esse trabalho permite que características do observatório sejam visualizadas e que possíveis melhorias possam ser implementadas futuramente.

Apêndice A

Modelagem PRISM

Segue o código da modelagem PRISM para o Observatório da Dengue :

```
dtmc
//NUVEM
//Fonte de Dados
const double fonte;
//Crawler
const double crawler;
//Coletor
const double coletor;

//ARMAZENAMENTO DE DADOS
//MongoDB
const double mongodb1;
formula mongodb = 1-pow(1-mongodb1,2);
//MySQL
const double mysql;
//RabbitMQ
const double rabbitmq;

//PIPELINE
//Reader
const double reader;
//Lang
const double lang;
const double ptlang; //probabilidade de receber uma mensagem em portugues
//Terms
const double terms;
//LAC
const double lac;
const double problac; //Probabilidade da mensagem conter alguma refereicia geografica se Sim -> Geo e Update, se nao ->
      Unload
//Geo
const double geo;
//Map
const double map;
//Unload
const double unload;
//Update
const double update;
const double proburl; //probalilidade de ir para o filtro url
//Url
const double url;


module coleta
    c : [0..11] init 0;
    []c=0 -> coletor        : (c'=1) + 1-coletor    :(c'=11);   //Coletor solicita parametros de coleta
    []c=1 -> mysql          : (c'=2) + 1-mysql  :(c'=11);   //MySQL retorna parametros de coleta
    []c=2 -> coletor        : (c'=3) + 1-coletor    :(c'=11);   //Coletor solicita dados ao crawler
    []c=3 -> crawler        : (c'=4) + 1-crawler    :(c'=11);   //Crawler solicita dados da fonte
    []c=4 -> fonte          : (c'=5) + 1-fonte  :(c'=11);   //Fonte de dados retorna os dados
    []c=5 -> crawler        : (c'=6) + 1-crawler    :(c'=11);   //Crawler retorna dados ao coletor
    []c=6 -> coletor        : (c'=7) + 1-coletor    :(c'=11);   //Coletor alimenta a fila Reader ou no MongoDB
//Probabilidade de Salvar no MongoDB =  1 -  Probabilidade do RabbitMQ ja que nesse ponto o MongoDB so e usado se o
     RabbitMQ estiver indisponivel. Assim:
    []c=7 -> rabbitmq       : (c'=8) + 1-rabbitmq   :(c'=9);    //Coletor alimenta a fila Reader ou no MongoDB
    []c=8 -> (c'=10);                           //Arquivos sao armazenados Diretamente no RabbitMQ
    []c=9 -> (c'=10);                           //Arquivos sao armazenados no MongoDB
    [iniciar_reader]c=10 -> (c'=10);                    //sucesso
    []c=11 -> (c'=11);                          //falha

endmodule

module reader
    r: [0..8] init 0;
    [iniciar_reader]r=0 -> reader   : (r'=1) + 1-reader :(r'=8);    //Reader conecta com rabbitmq
    []r=1 -> rabbitmq       : (r'=2) + 1-rabbitmq   :(r'=8);    //Rabbit confirma conexao
    []r=2 -> reader         : (r'=3) + 1-reader :(r'=8);    //Reader cria workflow no rabbitmq
```

```
//Probabilidade de obter dados do MongoDB =  1 -  Probabilidade do RabbitMQ ja que nesse ponto o MongoDB so e usado se
     o RabbitMQ estiver indisponivel. Assim:
    []r=3 -> rabbitmq       : (r'=5) + 1-rabbitmq    :(r'=4);    //Reader cria workflow no rabbitmq
    []r=4 -> mongodb        : (r'=5) + 1-mongodb     :(r'=8);    //Mensagens sao lidas do mongodb
    []r=5 -> reader         : (r'=6) + 1-reader :(r'=8);    //Reader envia para o rabbitmq
    []r=6 -> rabbitmq       : (r'=7) + 1-rabbitmq    :(r'=8);    //Rabbit salva na fila lang
    [iniciar_lang]r=7 -> (r'=7);                        //sucesso
    []r=8 -> (r'=8);                            //falha

endmodule

module lang
flagLangUnl: bool init false;
    l: [0..9] init 0;
    [iniciar_lang]l=0 -> lang           : (l'=1) + 1-lang    :(l'=8);    //Lang conecta com rabbitmq
    [lang]l=1 -> rabbitmq               : (l'=2) + 1-rabbitmq    :(l'=8);    //Rabbit confirma conexao
    [lang]l=2 -> lang               : (l'=3) + 1-lang    :(l'=8);    //Lang consome fila lang e salva na fila Terms ou
         Unload
    [lang]l=3 -> ptlang             : (l'=4) + 1-ptlang :(l'=5);    //Probabilidade de armazenar na fila term ou unload
    [lang]l=4 -> rabbitmq               : (l'=6) + 1-rabbitmq    :(l'=8);    //Rabbitmq salva dados na fila Terms
    [lang]l=5 -> rabbitmq               : (l'=7) + 1-rabbitmq    :(l'=8);    //Rabbitmq salva dados na fila Unload
    [iniciar_terms]l=6  -> (l'=6);                      //sucesso
    [lang]l=7       -> (l'=9) & (flagLangUnl'=true);                //sucesso
    [lang]l=8       -> (l'=8);                          //falha

endmodule

module terms
    t: [0..5] init 0;
    [iniciar_terms]t=0 -> terms : (t'=1) + 1-terms  :(t'=5);    //Terms conecta com rabbitmq
    []t=1 -> rabbitmq       : (t'=2) + 1-rabbitmq    :(t'=5);    //Rabbit confirma conexao
    []t=2 -> terms          : (t'=3) + 1-terms  :(t'=5);    //Terms consome fila terms e envia para o RabbitMQ
    []t=3 -> rabbitmq       : (t'=4) + 1-rabbitmq    :(t'=5);    //Rabbitmq salva dados na fila LAC
    [iniciar_lac]t=4 -> (t'=4);                     //sucesso
    [falha_terms]t=5 -> (t'=5);                     //falha
endmodule

module lac
flagLacUnl: bool init false;
flagLacGeo: bool init false;
    a: [0..10] init 0;
    [iniciar_lac]a=0 -> lac     : (a'=1) + 1-lac     :(a'=9);    //Lac conecta com rabbit
    []a=1 -> rabbitmq       : (a'=2) + 1-rabbitmq    :(a'=9);    //RabbitMQ confirma conexao
    []a=2 -> lac            : (a'=3) + 1-lac     :(a'=9);    //Lac consome fila lac e envia para o RabbitMQ
    []a=3 -> problac        : (a'=4) + 1-problac     :(a'=5);    //Probabilidade de armazenar na fila Unload ou Geo e
         Update
    []a=4 -> rabbitmq       : (a'=6) + 1-rabbitmq    :(a'=9);    //Rabbitmq salva dados na fila Geo e Update
    []a=5 -> rabbitmq       : (a'=8) + 1-rabbitmq    :(a'=9);    //Rabbitmq salva dados na fila Unload
    []a=6   -> (a'=10) & (flagLacGeo'=true);                    //sucesso - Salvo na Fila Geo
    //[iniciar_update]a=7   -> (a'=7);                  //sucesso - Salvo na fila Update
    []a=8   -> (a'=10) & (flagLacUnl'=true);                //sucesso - Salvo na fila Unload
    []a=9   -> (a'=9);                          //falha

endmodule

module Geo
    g: [0..7] init 0;
    [iniciar_geo](g=0) & (flagLacGeo) -> geo        : (g'=1) + 1-geo    :(g'=7);    //Geo conecta com rabbitmq
    []g=1 -> rabbitmq       : (g'=2) + 1-rabbitmq    :(g'=7);            //RabbitMQ confirma conexao
    []g=2 -> geo            : (g'=3) + 1-geo     :(g'=7);               //Geo conecta com MySQL
    []g=3 -> mysql          : (g'=4) + 1-mysql  :(g'=7);                //MySQL confirma conexao
    []g=4 -> geo            : (g'=5) + 1-geo     :(g'=7);               //Geo consome fila geo e envia para o RabbitMQ
    []g=5 -> rabbitmq       : (g'=6) + 1-rabbitmq    :(g'=7);           //Rabbitmq salva dados na fila map
    [iniciar_map]g=6 -> (g'=6);                             //sucesso
    []g=7 -> (g'=7);                                    //falha
endmodule

module update
    p: [0..11] init 0;
    [iniciar_update](p=0) & (flagLacGeo)-> update   : (p'=1) + 1-update :(p'=11);   //Geo conecta com rabbitmq
    []p=1 -> rabbitmq       : (p'=2) + 1-rabbitmq    :(p'=11);          //RabbitMQ confirma conexao
    []p=2 -> update         : (p'=3) + 1-update :(p'=11);           //Geo conecta com MongoDB
    []p=3 -> mongodb        : (p'=4) + 1-mongodb     :(p'=11);          //MongoDB confirma conexao
    []p=4 -> update         : (p'=5) + 1-update :(p'=11);           //Update consome fila update
    []p=5 -> update         : (p'=6) + 1-update :(p'=11);           //Geo envia dados para o rabbitmq
    []p=6 -> proburl        : (p'=7) + 1-proburl     :(p'=8);           //Probabilidade de conter ou nao url comprimida
    []p=7 -> rabbitmq       : (p'=9) + 1-rabbitmq    :(p'=11);          //Rabbitmq salva dados na fila URL
    []p=8 -> rabbitmq       : (p'=10) + 1-rabbitmq  :(p'=11);           //Rabbitmq salva dados no MongoDB
    [iniciar_url]p=9    -> (p'=9);                          //sucesso - Salvo na Fila URL
    []p=10          -> (p'=10);                         //sucesso - Salvo na fila MongoDB
    []p=11          -> (p'=11);                         //falha
endmodule

module unload

    n: [0..7] init 0;
    [iniciar_unload]n=0 & (flagLangUnl | flagLacUnl) -> unload  :(n'=1) + 1-unload  :(n'=6);    //Unload conecta com
         rabbitmq
    []n=1 -> rabbitmq           : (n'=2) + 1-rabbitmq    :(n'=6);           //RabbitMQ confirma conexao
    []n=2 -> unload             : (n'=3) + 1-unload :(n'=6);            //Unload consome fila unload
    []n=3 -> unload             : (n'=4) + 1-unload :(n'=6);            //Unload envia dados para o MongoBD
    []n=4 -> mongodb            : (n'=5) + 1-mongodb     :(n'=6);           //MongoBD salva dados no banco de dados
    []n=5 -> (n'=5);                                    //sucesso
    []n=6 -> (n'=6);                                    //falha
endmodule
```

```
module map
    m: [0..8] init 0;
    [iniciar_map]m=0 -> map     : (m'=1) + 1-map    :(m'=7);    //Map conecta com rabbitmq
    []m=1 -> rabbitmq       : (m'=2) + 1-rabbitmq   :(m'=7);    //RabbitMQ confirma conexao
    []m=2 -> map            : (m'=3) + 1-map    :(m'=7);    //Map conecta com MySQL
    []m=3 -> mysql          : (m'=4) + 1-mysql  :(m'=7);    //MySQL confirma conexao
    []m=4 -> map            : (m'=5) + 1-map    :(m'=7);    //Map consome fila map e envia para o MySQL
    []m=5 -> mysql          : (m'=6) + 1-mysql  :(m'=7);    //MySQL salva dados no banco de dados para publicacao
    []m=6 -> (m'=8);                            //sucesso
    []m=7 -> (m'=7);                            //falha
endmodule

module url
    u: [0..5] init 0;
    [iniciar_url]u=0 -> url     : (u'=1) + 1-url    :(u'=5);    //URL conecta com rabbitmq
    []u=1 -> rabbitmq       : (u'=2) + 1-rabbitmq   :(u'=5);    //RabbitMQ confirma conexao
    []u=2 -> url            : (u'=3) + 1-url    :(u'=5);    //Url consome fila url e envia para o MongoDB
    []u=3 -> mongodb        : (u'=4) + 1-mongodb    :(u'=5);    //MongoBD salva dados no banco de dados
    []u=4 -> (u'=4);                            //sucesso
    []u=5 -> (u'=5);                            //falha
endmodule
```

Apêndice B

Classificação dos defeitos previstos [17]

Defeitos previstos no **Coletor_twitter**:

1. Interrupção de serviço do processo:
    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Consistente.
    - **Consequência:** Grave.
2. Não entrega o serviço para o banco de dados MongoDB:
    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Grave.
3. Não entrega o serviço para o RabbitMQ:
    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Consistente.
    - **Consequência:** Grave.
4. Não entrega o serviço para o arquivo de backup:
    - **Domínio:** Instável.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

Defeitos previstos no **Reader_filter**:

1. Não entrega o serviço para o filtro seguinte (*limite*):

- **Domínio:** Instável.
- **Detectabilidade:** Sinalizada.
- **Consistência:** Consistente.
- **Consequência:** Média.

2. Não entrega o serviço para o filtro seguinte (*desconectado*):

   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Consistente.
   - **Consequência:** Média.

3. Não entrega o serviço para o filtro seguinte (*EncodingErrors*):

   - **Domínio:** Conteúdo.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Consistente.
   - **Consequência:** Média.

4. Não entrega o serviço para o RabbitMQ:

   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Consistente.
   - **Consequência:** Grave.

Defeitos previstos no **Lang_filter**:

1. Não entrega o serviço para o filtro seguinte:

   - **Domínio:** Conteúdo.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Defeitos previstos no **Terms_filter**:

1. Não entrega o serviço para o filtro seguinte:

   - **Domínio:** Parada.
   - **Detectabilidade:** Sinalizada.
   - **Consistência:** Inconsistente.
   - **Consequência:** Grave.

Defeitos previstos no **Lac_filter**:

1. Não entrega o serviço para o filtro seguinte:

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Grave.

Defeitos previstos no **Geo_filter**:

1. Não entrega o serviço para o filtro seguinte (*RabbitMQ Fila Consumo*):

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Grave.

2. Não entrega o serviço para o filtro seguinte (*Mensagem Descartada*):

    - **Domínio:** Conteúdo.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

Defeitos previstos no **Map_filter**:

1. Não consegue salvar, no banco de dados MySQL, a localização descrita na mensagem:

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Grave.

Defeitos previstos no **Url_filter**:

1. Não entrega o serviço para o banco de dados MongoDB (*Timeout*):

    - **Domínio:** Tempo.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

2. Não entrega o serviço para o banco de dados MongoDB (*RabbitMQ Fila Consumo*):

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.

- **Consistência:** Inconsistente.
- **Consequência:** Pequena.

3. Não entrega o serviço para o banco de dados MongoDB (*Endereco Incompleto*).

    - **Domínio:** Conteúdo.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

4. Não entrega o serviço correto para o banco de dados MongoDB (*Geral impressa True, Geral impressa True, Geral gaierror e Dominio Permissao*):

    - **Domínio:** Instável.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

Defeitos previstos no **Update_filter**:

1. Não entrega o serviço para o banco de dados MongoDB:

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

2. Não entrega o serviço para o filtro seguinte.

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

Defeitos previstos no **Unload_filter**:

1. Não entrega o serviço para o banco de dados MongoDB:

    - **Domínio:** Parada.
    - **Detectabilidade:** Sinalizada.
    - **Consistência:** Inconsistente.
    - **Consequência:** Pequena.

Referências

[1] Zabbix documentation. http://www.zabbix.com/documentation/2.0/manual/introduction, 2012 (Acesso em 01 dez. 2012). 35

[2] Thiago Araújo, Carla Wanderley, and Arndt von Staa. Um mecanismo de introspecção para depurar o comportamento de sistemas distribuídos. 2010. 35

[3] A. Avižienis, J.C. Laprie, and B. Randell. Fundamental concepts of dependability. Technical Report UCLA CSD Report 0100, Computer Science Department, University of California, Los Angeles, USA, 2001. 3, 4, 5

[4] Algirdas Avižienis, Jean Claude Laprie, Brian Randell, and Carl Landwehrn. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004. vii, 6, 7

[5] F. Benevenuto, J. Almeida, and A. Silva. Coleta e anÃ¡lise de grandes bases de dados de redes sociais online. pages 11–57, 2011. 16

[6] Manuel Castells. *The Rise of the Network Society*, volume 1 of *The Information Age: Economy, Society and Culture*. Blackwell, Malden, Mass., 1999. 15

[7] Marcelo de Mota Lopes. Aplicação de mineração de dados na identificação de padrões de falha em sistemas gerenciadores de banco de dados relacionais. http://www.pcs.usp.br/~gas/v2/images/Publications/dissertacao_marcelolopes.pdf, 2005. 9

[8] Bruno de O. S. Silva and Bruno A. P. de S. Souza. Uma abordagem exploratória de dependabilidade do twitter no contexto do observatório da web, 2011. Monografia apresentada como requisito parcial para conclusão do Curso de Computação – Licenciatura. 1, 19

[9] Thânia Clair de Souza Vargas. A história de uml e seus diagramas. https://projetos.inf.ufsc.br/arquivos_projetos/projeto_721/artigo.tcc.pdf. 9

[10] Walter dos Santos Filho, Gisele L. Pappa, Wagner Meira Jr, Dorgival Guedes, Adriano Veloso, Virgílio A. F. Almeida, Adriano M. Pereira, Pedro H. C. Guerra, Arlei L. da Silva, Fernando H. J. Mourão, Tiago R. de Magalhães, Felipe M. Machado, Letícia L. Cherchiglia, Lívia S. Simões, Rafael A. Batista, Filipe L. Arcanjo, Gustavo M. Brunoro, Nathan R. B. Mariano, Gabriel Magno, Marco Túlio C. Ribeiro, Leonardo V. Teixeira Altigran S. da Silva, BrunoWanderley Reis, and Regina Helena Silva. Observatório da web: Uma plataforma de monitoração, síntese e visualização de eventos massivos em tempo real. pages 110–120, 2010. vii, 19, 20, 21

[11] SANTANA V. F., MELO-SOLARTE D. S., NERIS V. P. A., MIRANDA L. C., and BARANAUSKAS M. C. C. Redes sociais online: Desafios e possibilidades para o contexto brasileiro. *XXXVI Seminário Integrado de Software e Hardware (SEMISH) / XXIX Congresso da Sociedade Brasileira de Computação (CSBC)*. 16

[12] Felipe Nunes Flores. Avaliando o impacto da qualidade de um algoritmo de stemming na recuperação de informações. Trabalho de conclusão de graduação, 2009. 22

[13] George Eduardo Freund. Análise estrutural para aumentar a eficiência de pesquisas "online". *Ciência da Informação*, 11(1), 1982. 22

[14] Lars Grunske. Specification patterns for probabilistic quality properties. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 31–40, New York, NY, USA, 2008. ACM. viii, 12, 13

[15] Mohamed Kaâniche, Jean-Claude Laprie, and Jean-Paul Blanquart. A framework for dependability engineering of critical computing systems. *Safety Science*, 40(9):731–752, 2002. 3

[16] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011. 10, 14

[17] Leonardo de Sousa Melo and Felipe Carvalho Gules. Avaliação de dependabilidade no observatório da web para manutenção preventiva com ferramenta de inspeção, 2013. Monografia apresentada como requisito parcial para a conclusão do bacharelado em Ciência da Computação. vi, viii, 21, 35, 37, 40, 48

[18] Bill Slawski. Google stopwords patent, August 2008. 23

[19] Adriano Veloso, Wagner Meira Jr., and Mohammed J. Zaki. Lazy associative classification. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 645–654, Washington, DC, USA, 2006. IEEE Computer Society. 22

[20] Taisy Silva Weber. *Tolerância a falhas: conceitos e exemplos*, volume 52. Distrito 4 da ISA (The Instrumentation, System and Automation Society), 2003. vii, 5, 6, 9