ESPECIFICAÇÃO E VERIFICAÇÃO FORMAL DE UM SISTEMA DE TROCA DENOMINADO ESCAMBO

JOÃO VITOR CHAGAS POSSOBOM VAZ[1]; SIMONE ANDRÉ DA COSTA CAVALHEIRO[3]

[1]UFPel - *jvcpvaz@inf.ufpel.edu.br*
[3]UFPel - *simone.costa@inf.ufpel.edu.br*

1. INTRODUÇÃO

Junto com o crescimento do tamanho e da complexidade dos sistemas computacionais, surgiu a necessidade de construir especificações mais precisas para descrever o comportamento desses sistemas de uma forma mais sistemática, confiável e sem ambiguidade. A utilização de métodos formais se torna uma interessante alternativa para a descrição desses sistemas, permitindo também a análise de seu comportamento e a verificação de propriedades de interesse por meio de ferramentas precisas e eficazes de análise, fundamentadas em técnicas baseadas na matemática. A especificação formal, por meio de uma precisa definição de um sistema utilizando uma linguagem com semântica definida, é uma excelente maneira de descobrir erros de projeto, possibilitando também analisar se um programa tem certas propriedades desejáveis.

Como um exemplo de método de especificação formal tem-se a gramática de grafos, a qual fornece um mecanismo onde as transformações locais em grafos podem ser modeladas de uma maneira visual e precisa (LEILA, 2000). Esse método se diferencia por possuir uma representação visual gráfica de uma forma natural para explicar situações complexas de modo intuitivo facilitando a compreensão, e sem a exigência de um conhecimento avançado sobre o seu formalismo. Na gramática de grafos basicamente os estados dos sistemas são modelados por grafos e as possíveis mudanças de comportamento nos sistemas são feitos por regras de transformação de grafos.

Dentre diversas ferramentas que permitem a verificação formal, a plataforma Rodin é uma alternativa para especificação e prova de propriedades que usa como linguagem de entrada o Event-B, onde o refinamento é um recurso utilizado para acrescentar de forma incremental detalhes numa especificação. Por sua vez, Event-B é um formalismo sustentado pelo rigor matemático (ABRIAL, 2010) e é usado para especificar e raciocinar sobre sistemas discretos complexos, o sistema e suas propriedades são especificadas usando a lógica de conjuntos e de predicados, utiliza de provas e refinamentos para mostrar que as invariantes são mantidas enquanto o desenvolvimento prossegue. A plataforma Rodin contém elementos de modelagem como: variáveis, invariantes, constantes e ações. Os usuários podem gerar seu modelo inicial e, a partir disto, descrever versões mais completas através do refinamento, conforme necessário. Fazendo isso são capazes de raciocinar, modificar e decompor seus modelos antes de iniciar a implementação efetiva de todo o seu sistema.

Este trabalho consiste na especificação de um sistema de trocas denominado Escambo elaborado pelo discente do curso em Ciência da Computação da UFPel Rafael Nascimento. O sistema estava previamente especificado em UML usando Casos de Uso. Esta descrição foi usada como base para a sua compreensão e especificação em gramática de grafos utilizando a ferramenta AGG (BERLIN, 2017), a qual foi posteriormente traduzida para linguagem Event-B. Por fim, descreveram-se propriedades desejáveis a serem verificadas na ferramenta Rodin.

2. METODOLOGIA

Para fazer a especificação do Escambo em gramática de grafos foi utilizado a modelagem já feita em UML usando casos de uso. A UML (Unified Modeling Language) é uma linguagem de modelagem informal, cujo objetivo é auxiliar a definir as características do software, tais como seus requisitos, seu comportamento e até suas necessidades físicas em relação ao equipamento no qual o sistema deverá ser implementado (GUEDES, 2014). O objetivo do sistema é oferecer a possibilidade de os usuários cadastrarem seus produtos e os oferecerem aos demais usuários como forma de escambo, permitindo que o produto cadastrado receba ofertas de outros usuários,

Foi analisado o referencial teórico (CRAIGEN, 1993), para a compreensão da Gramática de Grafos e para a realização da especificação do sistema Escambo. Com a especificação em gramática de grafos pronta na ferramenta AGG, foi feita a respectiva tradução para o Event-B.

Para a tradução para o Event-B (CAVALHEIRO; FOSS; RIBEIRO, 2017) foram seguidas as seguintes etapas: primeiro foi traduzido o grafo tipo, após foi traduzido o grafo inicial (por meio da instanciação de um grafo estado) e então foi feita a tradução das regras. Para a compreensão da ferramenta foi construído um pequeno sistema sobre o controle de carros na plataforma Rodin utilizando sua linguagem de entrada Event-B. Com a tradução pronta na plataforma Rodin, foi descrito algumas propriedades desejáveis a serem verificadas.

3. RESULTADOS E DISCUSSÃO

Na plataforma Rodin foi possível perceber que uma especificação em Event-B possui dois componentes, denominados máquina e contexto. O contexto descreve sobre a parte estática de um sistema contendo basicamente constantes e axiomas. Axiomas são afirmações que são consideradas como verdadeiras para o resto do sistema, podendo ser marcadas como teoremas. Cada axioma consiste de um rótulo e um predicado. A máquina descreve o comportamento dinâmico do sistema por meio de variáveis que são alteradas de acordo com eventos. Uma guarda é a condição que precisa ser satisfeita para a ocorrência de um evento.

Como exemplo, na Figura 1 temos em gramática de grafos a regra logar do Escambo e na Figura 2 temos no contexto a tradução do lado esquerdo da regra logar (L) no Event-B. Esta regra possui uma NAC, condição negativa de aplicação, que estabelece que somente usuários que não estejam logados podem se logar.

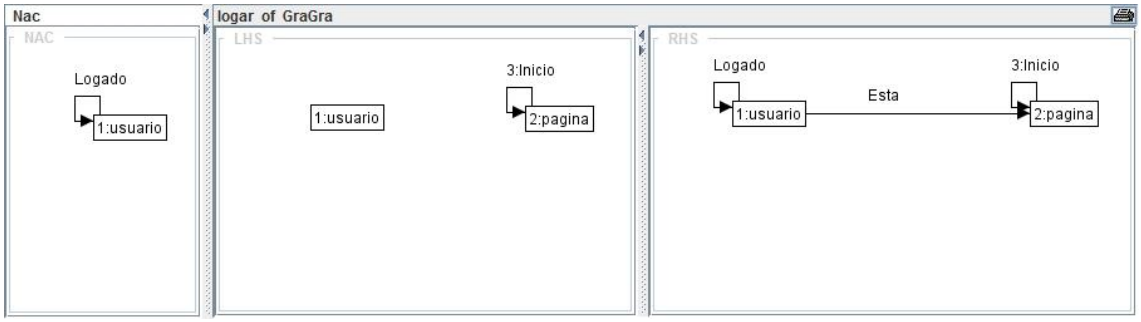

Figura 1 - Regra para logar no sistema Escambo

```
@axm_Vertlogar partition(Vertlogar, {usrlogar}, {paglogar})
@axm_Edgelogar partition(Edgelogar, {PagIniciologar})
@axm_srclogarType Sourcelogar ∈ Edgelogar → Vertlogar
@axm_srclogarDef partition(Sourcelogar,{PagIniciologar↦paglogar})
@axm_tgtlogarType Targetlogar ∈ Edgelogar → Vertlogar
@axm_tgtlogarDef partition(Targetlogar,{PagIniciologar↦paglogar})
@axm_tlogarV tlogarV ∈ Vertlogar → VertT
@axm_tlogarE tlogarE ∈ Edgelogar → EdgeT
@axm_tlogarV_def partition(tlogarV, {usrlogar ↦ usuario}, {paglogar ↦ pagina})
@axm_tlogarE_def partition(tlogarE, {PagIniciologar ↦ Inicio})
```

Figura 2 - Contexto da regra logar

No contexto, o Vertlogar representa todos os vértices da regra e Edgelogar representa todos os arcos de L. O Sourcelogar especifica a origem de cada arco da regra e Targetlogar o respectivo destino. O tlogarV define os tipos de cada vértice e tlogarE define os tipos de cada arco de L. VertT e o EdgeT, contradomínios de cada uma das relações anteriores, representam todos os tipos de vértices e todos os tipos de arcos, respectivamente. O mesmo processo foi usado para a construção do contexto das outras regras da gramática. Na Figura 3 é apresentada a descrição da regra logar na máquina.

```
logar:   not extended ordinary
ANY
  mV
  mE
  DelV
  PreservV
  DelE
  Dangling
  newE_Esta
  newE_Logado
WHERE
  grd_mV: mV ∈ Vertlogar → VertG not theorem
  grd_mE: mE ∈ Edgelogar → EdgeG not theorem
  grd_DelV:   DelV = mV[∅] not theorem
  grd_PreV:   PreservV = VertG ∖ DelV not theorem
  grd_DelE:   DelE = mE[∅] not theorem
  grd_Dang:   Dangling = dom((SourceG ▷ DelV) ∪ (TargetG ▷ DelV))∖DelE not theorem
  grd_newE_I126:  newE_Esta ∉ N ∖ EdgeG not theorem
  grd_newE_I128:  newE_Logado ∉ N ∖ EdgeG not theorem
  grd_diffI126I128:   newE_Esta ≠ newE_Logado not theorem
  grd_diffI128I126:   newE_Logado ≠ newE_Esta not theorem
  grd_tV: ∀v·v ∈ Vertlogar ⇒ tlogarV(v) = tGV(mV(v)) not theorem
  grd_tE: ∀e·e ∈ Edgelogar ⇒ tlogarE(e) = tGE(mE(e)) not theorem
  grd_srctgt: ∀e·e ∈ Edgelogar ⇒ mV(Sourcelogar(e)) = SourceG(mE(e)) ∧ mV(Targetlogar(e)) = TargetG(mE(e)) not theorem
  grd_logarNAC1:  ¬(∃ I131 · {I131} ⊆ EdgeG ∖ mE [Edgelogar] ∧ (tGE(I131) = Logado) ∧ (SourceG(I131) = mV(usrlogar)) ∧ (TargetG(I131) = mV(usrlogar))) not theorem
  grd_Identi1V:   DelV ∩ mV[{paglogar, usrlogar}] = ∅ not theorem
  grd_Identi1E:   Dangling = ∅ not theorem
THEN
  act_V:  VertG ≔ (VertG∖DelV)∪ ∅
  act_E:  EdgeG ≔ (EdgeG∖DelE)∪ {newE_Esta, newE_Logado}
  act_src:    SourceG ≔ (DelE ⩤ SourceG) ∪ {newE_Esta ↦ mV(usrlogar), newE_Logado ↦ mV(usrlogar)}
  act_tgt:    TargetG ≔ (DelE ⩤ TargetG) ∪ {newE_Esta ↦ mV(paglogar), newE_Logado ↦ mV(usrlogar)}
  act_tV: tGV ≔ (DelV ⩤ tGV) ∪ ∅
  act_tE: tGE ≔ (DelE ⩤ tGE) ∪ {newE_Esta ↦ Esta, newE_Logado ↦ Logado}
END
```

Figura 3 - Máquina da regra logar

Para cada regra é preciso criar um evento, especificando mudanças que devem ocorrer quando a regra é aplicada, de acordo com a especificação de suas pós-condições. As variáveis que definem o estado do sistema são atualizadas de acordo com os vértices e arestas criados e/ou deletados. Neste caso duas arestas são adicionadas. As guardas estabelecem também a

compatibilidade das arestas e vértices com os seus tipos. Especificam também as NACs, que descrevem situações que impedem a aplicação das regras.

Com a tradução pronta foram geradas as obrigações de prova garantindo a consistência da especificação. Aquelas que não foram provadas automaticamente, foram provadas interativamente usando a Perspectiva de Prova e os provadores que a ferramenta Rodin proporciona.

Com a prova da consistência da especificação realizada, algumas propriedades desejadas para o sistema podem ser provadas, tais como: um usuário não pode trocar duas vezes o mesmo produto e uma oferta não pode ser aceita e rejeitada simultaneamente.

4. CONCLUSÕES

A gramática de grafos é uma linguagem visual que permite explicar de uma forma natural e intuitiva situações mais complicadas, removendo a ambiguidade que poderia haver numa possível descrição em linguagem natural e facilitando o seu entendimento. Para a especificação da gramática que define o sistema Escambo, utilizou-se inicialmente a ferramenta AGG e posteriormente foi feita a tradução para o Event-B. O uso do AGG permitiu identificar falhas na especificação inicial da gramática. Por exemplo, foi possível perceber que algumas regras que deveriam poder ser aplicadas, não estavam podendo ser aplicadas, por estar faltando algum componente na regra. A identificação de erros e as respectivas correções no AGG impediu que estes erros se propagassem na especificação em Event-B. Correções na especificação em Event-B seriam mais custosas e trabalhosas.

Com o modelo no Event-B pronto e as obrigações de prova verificadas, garantindo a consistência do sistema, o próximo passo é fazer a verificação de algumas das propriedades definidas.

5. REFERÊNCIAS BIBLIOGRÁFICAS

ABRIAL, J.R. **Modeling in Event-B: System and Software**. Cambridge University Press, New York, 2010.

LEILA, R. **Métodos Formais de Especificação: Gramáticas de Grafos**. Instituto de Informática, Porto Alegre, 2000. Acessado em 13 set. 2019. Online. Disponível em: ftp://ftp.inf.ufrgs.br/pub/leila/eri.ps.gz

CAVALHEIRO, S.; Foss, L.; Ribeiro, L. **Theorem proving graph grammars with attributes and negative application conditions**. Theoretical Computer Science, Elsevier, 2017.

GUEDES, G.T. **UML 2 – Guia Prático**. São Paulo: Novatec Editora, 2014. 192 p.

BERLIN, T. **A brief Description of AGG**, Berlin 2 mar. 2017. Acessado em 13 set. 2019. Online. Disponível em: http://www.user.tu-berlin.de/o.runge/agg/index.html