MODELAGEM DE COMPORTAMENTO DE SOFTWARE: UM COMPILADOR NA FERRAMENTA LOTUS

Yan Augusto Gurgel da Silva(1); Márcia Machado Marinho(2); Gabrielle Silva Marinho(3); Paulo Henrique Mendes Maia(4)

1-Universidade Estadual do Ceará, yan.gurgel@gmail.com*; 2-Universidade Federal do Ceará, marinho.marcia@gmail.com;3-Universidade Estadual do Ceará,* gabrielle.marinho@uece.br*; 4-Universidade Estadual do Ceará, pauloh.maia@uece.br*

Resumo: LoTuS é uma ferramenta extensível e de código aberto para a criação de modelos de comportamento representados como *Labelled Trasition Systems* (LTS) de forma gráfica através de uma interface amigável que disponibiliza um mecanismo *drag and drop*. A ferramenta LoTuS foi desenvolvida por alunos do curso de graduação em Ciência da Computação da Universidade Estadual do Ceará em 2013 e, desde então, vem evoluindo com a incorporações de novas funcionalidades. Sua característica de extensão permitiu a criação de outras ferramentas, como TCG, para geração e seleção de casos de teste, e o LoTuS@Runtime, para monitoramento e análise de sistemas autoadaptativos. Apesar de facilitar a construção de modelos simples, a criação gráfica apresenta limitações quando se almeja especificar modelos maiores, uma vez que essa atividade pode tornar-se exaustiva. A possibilidade de usar uma linguagem de especificação para construir os comportamentos era algo que também já vinha sendo solicitado por usuários da ferramenta, bem como pela comunidade acadêmica. Para atender a essa necessidade, a ferramenta foi estendida com dois novos módulos: a tela de especificação textual e um compilador para tal linguagem. Na primeira, o usuário pode especificar um ou mais componentes do sistema utilizando um subconjunto da linguagem de especificação formal *Finite State Process* (FSP). A contribuição inicial deste estudo consiste na extensão da linguagem para permitir a modelagem de comportamentos probabilísticos, então permitindo a especificação de ambos os tipos de componentes. O segundo módulo consistiu no desenvolvimento de um compilador para a linguagem FSP estendida, permitindo que a especificação textual pudesse ser validada, gerando como saída a representação gráfica do LTS, seja ele probabilístico ou não. Após isso, o usuário pode utilizar os outros *plugins* do LoTuS para analisar seu modelo gerado, como detecção de *deadlock* e análise de propriedades probabilísticas. Assim, o compilador foi aplicado para construir vários modelos diferentes. Os testes realizados mostraram que ele é robusto o suficiente para permitir a geração de modelos de comportamento maiores, bem como verificar se a especificação está de acordo com a linguagem utilizada, dando ao usuário uma maior capacidade de uso da ferramenta. Ressalta-se que o LoTuS encontra-se disponível gratuitamente para *download* no *github*.

Palavras-chave: Compilador, Modelagem de comportamento, Especificação formal.

INTRODUÇÃO

A modelagem de comportamento consiste em uma técnica que permite desenvolvedores descreverem abstratamente e raciocinarem sobre o comportamento desejado de um sistema de *software*. Por sua simplicidade, o custo do desenvolvimento de modelos é considerado baixo quando comparado ao custo de construir um sistema por completo. Além disso, a modelagem traz como benefício a facilidade de entendimento sobre como o *software* deve se comportar em tempo de execução¸ e a possibilidade de antever possíveis falhas em tempo de projeto (MAGEE & KRAMER, 2006).

Para modelar e analisar o comportamento do sistema, geralmente é necessário especifica-lo formalmente e usar uma ferramenta automatizada que dê suporte à análise desejada. Posto que métodos formais oferecem uma precisão matemática na análise de programas que dão a possibilidade de certificar a corretude do programa, algo que o teste e a revisão informal não conseguem. O objetivo maior dos métodos formais é aumentar a qualidade do software ao tornar os requisitos, projeto e implementação do sistema mais explícitos e, portanto, deixando os defeitos mais facilmente detectados (GEER. 2011).

O comportamento de sistemas é geralmente modelado utilizando um formalismo de máquina de estados, como o *Labelled Transition Systems* –LTS (KELLER, 1976), no qual cada modelo se refere a um componente do sistema que interage com os demais componentes através de ações compartilhadas, *Probabilistic* LTS (PLTS) e *Discrete-time Markov Chains* (DTMC), no caso de modelagem de comportamento probabilístico. Há várias ferramentas de modelagem de comportamento que usam tanto LTSs, como LTSA (MAGEE & KRAMER, 2006), SPIN (HOLZMANN, 1997), e LTS-Min (BLOM et al. 2010), ou DTMCs, como PRISM (KWIATKOWSKA, 2007). Apesar de existirem outros formalismos, como as Redes de Petri (PETERSON, 1981) e suas derivações estocásticas, este trabalho foca no uso de LTS e PLTS para a modelagem e análise de comportamento tradicional e probabilístico de *software*, respectivamente.

Nesse contexto, a ferramenta LoTuS foi desenvolvida por alunos de graduação do Grupo de Engenharia de Software e Sistemas Distribuídos da Universidade Estadual do Ceará (UECE) e caracteriza-se por ser uma ferramenta de código aberto e extensível para modelagem gráfica e análise de comportamento de *software* utilizando modelos LTS e PLTS. Os principais benefícios da ferramenta são:

(i) Oferecer um mecanismo *drag and drop* que torna a modelagem mais fácil e intuitiva, possibilitando também ao usuário incrementar as transições com guardas e probabilidades, o que permite facilmente evoluir os modelos para se tornarem probabilísticos;

(ii) Permitir a geração de modelos a partir de outras fontes, como a diagramas de sequência UML, rastros de execução, ou outras ferramentas de modelagem, como o LTSA;

(iii) Disponibilizar algumas técnicas de análises de modelo, como detecção de *deadlocks*, simulação e execução, além da verificação probabilísticas de propriedades de alcançabilidade (*reachability properties*);

(iv) Fornecer uma API que permite desenvolvedores adicionarem novas funcionalidades através de *plugins*;

Apesar dessas vantagens, a criação gráfica apresenta limitações quando se almeja especificar modelos maiores, uma vez que essa atividade pode tornar-se exaustiva. A possibilidade de usar uma linguagem de especificação para construir os comportamentos era algo que também já vinha sendo solicitado por usuários da ferramenta, bem como pela comunidade acadêmica.

Portanto, este trabalho estende a ferramenta LoTuS através da criação de um módulo de especificação textual de modelos de comportamento e um compilador que processe essa especificação e a transforme em uma modelagem gráfica.

METODOLOGIA

A metodologia utilizada para o desenvolvimento da nova funcionalidade de especificação textual seguiu três etapas: pesquisa sobre linguagens de especificação formal de comportamento, desenvolvimento de um módulo de especificação textual para o LoTuS, e implementação de um compilador para a linguagem utilizada.

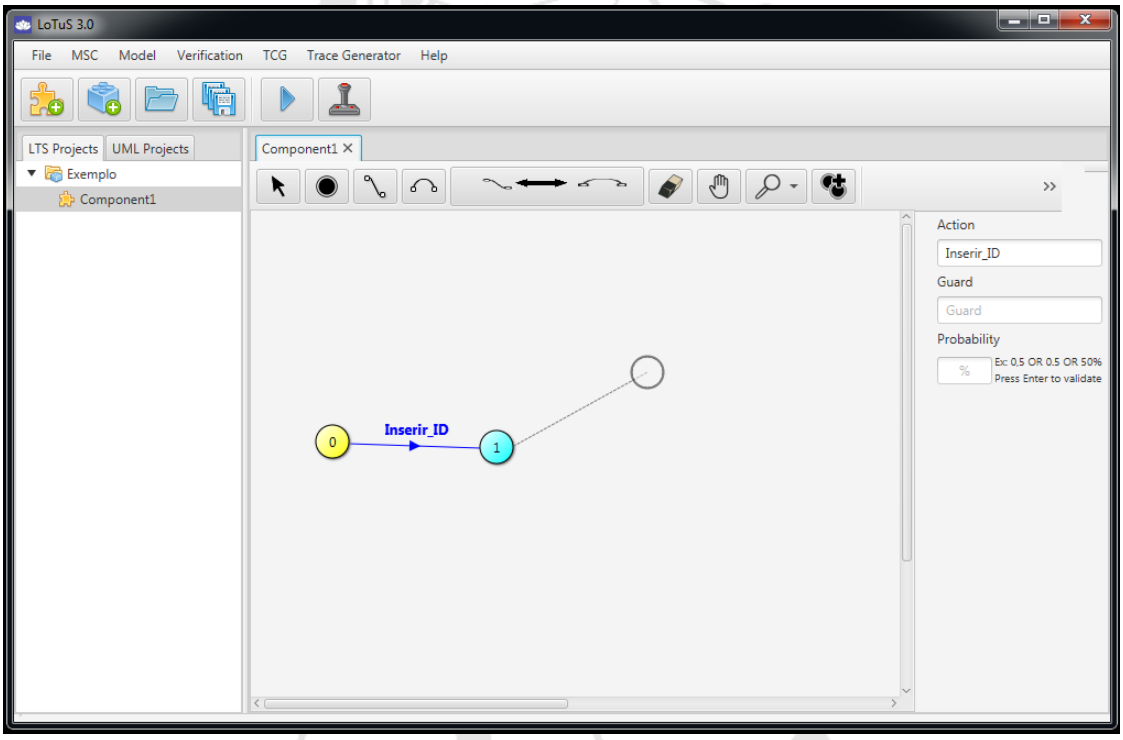
Na primeira etapa, foi realizado um estudo de diferentes linguagens de especificação formal de comportamento de software. A escolhida foi a *Finite State Process* (FSP) (MAGGE e KRAMER, 2006), pois apresenta uma sintaxe simples para o usuário final e por ser bastante utilizada no meio acadêmico. Contudo, apenas um subconjunto da linguagem foi implementado, pois era o suficiente para representar as principais formas de comportamento, como ações sequenciais, escolha determinística, e ações de parada). Além disso, para representar modelos probabilísticos, este trabalho estendeu a linguagem original, permitindo que as ações fossem precedidas por uma probabilidade de ocorrência.

Uma vez determinada a linguagem, a segunda etapa consistiu no desenvolvimento de um módulo no LoTuS para permitir que o usuário especifique o comportamento utilizando a linguagem FSP. Por fim, a última etapa envolveu o desenvolvimento de um compilador que valide se a especificação feita pelo usuário está sintaticamente correta e gere um modelo em LTS ou PLTS dependendo se o comportamento especificado foi não probabilístico ou probabilístico, respectivamente.

RESULTADOS E DISCUSSÃO

A ferramenta LoTuS utiliza a linguagem Java e JavaFX como fonte principal de código além de manipulação de arquivos em .xml e outras tecnologias para desenvolvimento. A figura 1 ilustra a interface da referida ferramenta, na qual pode se ver a tela de informações do projeto (à esquerda), tela de desenho do LTS (ao centro) e tela de informações do elemento selecionado (à direita), que neste caso é a transição "Inserir_ID".

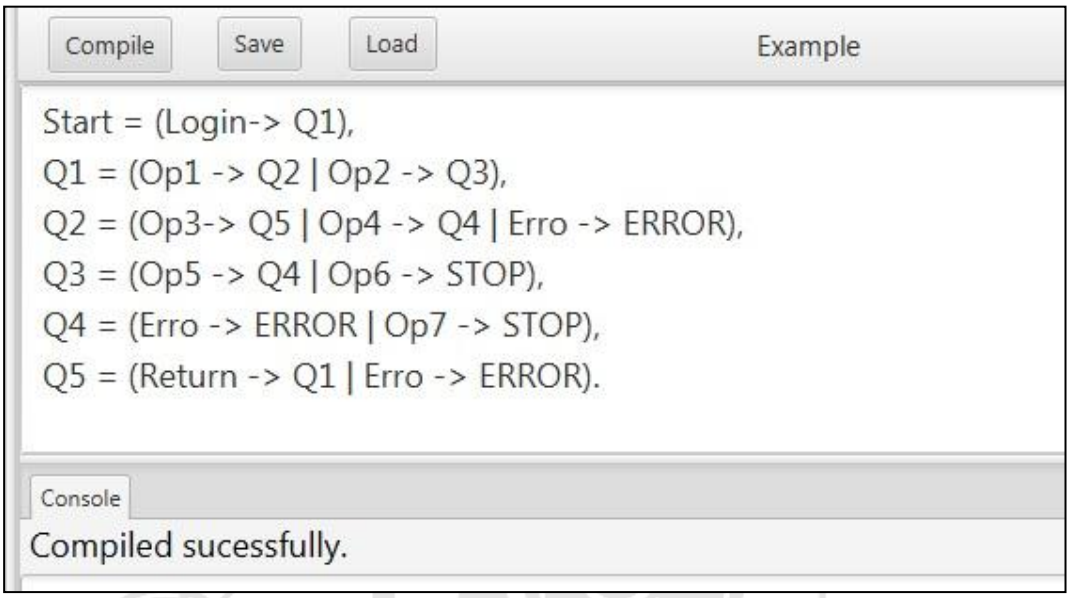
Figura 1 – Interface LoTuS



Fonte: LoTuS,2017.

O comportamento de cada componente do sistema modelado com a linguagem utilizada é composto por uma sequência *ação -> ESTADO*, que significa que o sistema, após executar *ação*, passa para o estado representado por *ESTADO*. As palavras reservadas *STOP* e *ERROR* representam os estados final e de erro, respectivamente. O usuário também pode usar a disjunção lógica "ou", representada pelo símbolo "|", para indicar que a possibilidade de mais de uma ação. A figura 2 ilustra um exemplo de um processo modelado com a linguagem proposta.

Figura 2 – Exemplo



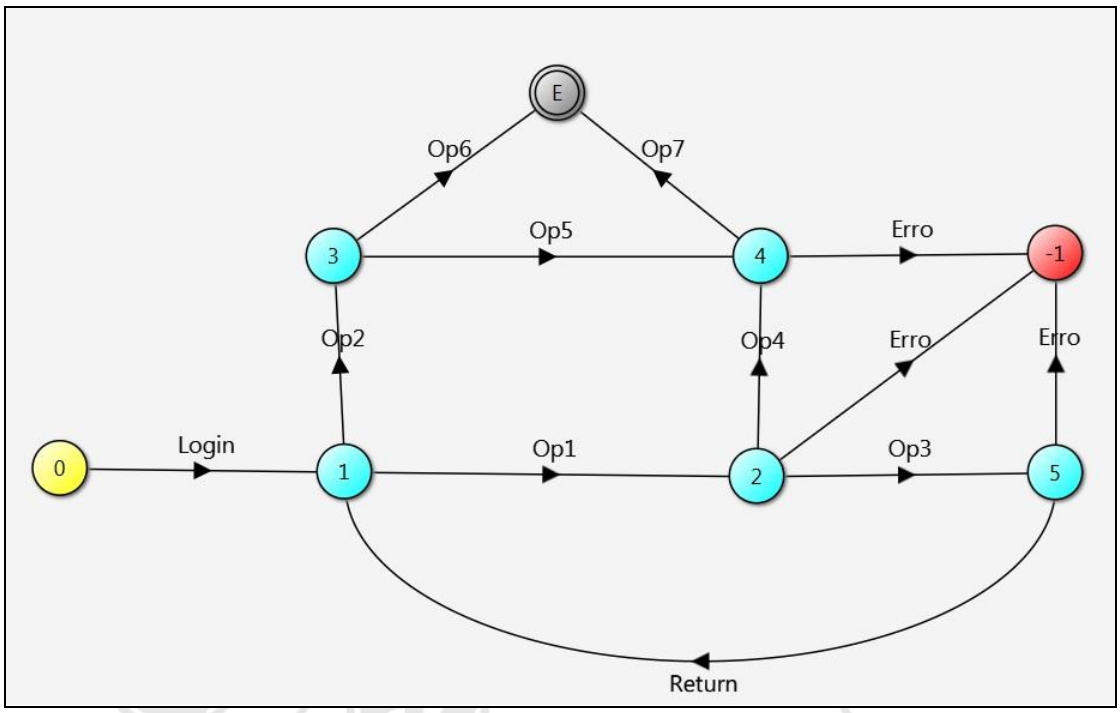
Fonte: LoTuS,2017.

A tela do módulo de especificação possui três botões: um para compilar a especificação, um para salvar a especificação de modo que ela fique organizada dentro de um arquivo padrão .txt, e outro para carregar a especificação reconhecendo apenas arquivos no formato .txt, como pode ser visto na Figura 2. Após a digitação da especificação para criar o LTS/PLTS o usuário deve clicar no botão "Compile" onde será disparado uma série de verificações para analisar o que foi escrito. Se a especificação estiver dentro da lógica da linguagem definida o próximo passo do compilador é iniciar a construção daquela especificação.

O objetivo do compilador é traduzir as sequências de caracteres que representam o programa fonte em código executável. No caso do LoTuS, reconhecer a linguagem e traduzi-la para um LTS/PLTS é uma tarefa complexa o suficiente de forma que o compilador pode ser dividido em processos menores interconectados.

Cada um dos processos tem acesso a tabelas de informações globais sobre o programa fonte. Na compilação existe também um módulo responsável pelo tratamento ou recuperação de erros cuja função é diagnosticar, através de mensagens adequadas, os erros léxicos, sintáticos e semânticos encontrados.

Figura 3 – LTS do exemplo gerado pelo compilador



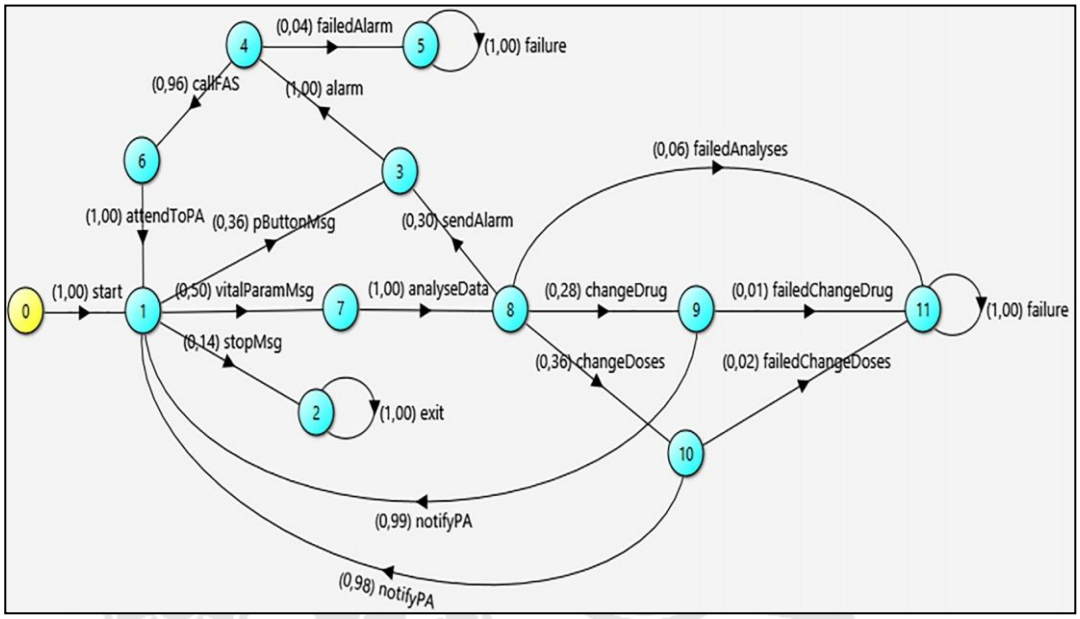
Fonte: LoTuS,2017.

Todas as verificações que o compilador deve fazer foram programadas em Java e o usuário pode ver o resultado da verificação no console pertencente ao compilador do LoTuS. O console pode mostrar mensagens de *Error*, *Warning* ou *Compiled sucessfully* cada um com uma cor distinta, dando ao usuário o retorno do que possivelmente está errado e retornando à primeira linha onde o possível erro foi encontrado.

Passando por todas as verificações referentes a linguagem, o próximo passo é montar o LTS/PLTS na tela da ferramenta. Para isso foi definido que dentro da especificação se teria pontos chaves para poder iniciar a construção. O primeiro ponto consiste em identificar todos os estados principais, pois a construção dos modelos é feita linha por linha a partir de um ponto inicial. Para realizar essa tarefa, foi feita uma varredura na especificação onde todos os pontos iniciais, ou seja, as declarações presentes antes dos sinais de igualdade seriam armazenadas bem como as palavras reservadas.

Identificando todos os estados principais, a estrutura de armazenamento utilizada foi o *HashMap*, com o qual se pode ter múltiplas informações referentes a um mesmo estado, facilitando assim a construção via código.

Figura 4 – LTS probabilístico



Fonte: LoTuS,2017.

CONCLUSÕES

Este trabalho apresentou uma extensão da ferramenta de modelagem LoTuS para permitir a especificação textual de modelo de comportamento de software não probabilístico e probabilístico. O compilador foi testado e aplicado para construir vários modelos que puderam, em seguida, utilizar as funcionalidades do LoTuS, como simulação, verificação e execução. A nova ferramenta está disponível para download gratuitamente no GitHub para ser utilizada pela comunidade.

REFERÊNCIAS BIBLIOGRÁFICAS

Magee, J. and Kramer, J. (2006). Concurrency: State Models and Java Programs. Wiley.
Milner, R. (1989). Communication and concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

GEER, P. A. . Formal Methods in Practice: Analysis and Application of Formal Modeling to Information Systems. PhD thesis, State University of New York Institute of Technology at Utica/Rome, 2011.

Keller, R. M. (1976). Formal verification of parallel programs. Commun. ACM, 19:371–384.

HOLZMANN, G. J. The model checker spin. IEEE Transaction on Software Engi-neering, v.23, n.5, p.279–295, 1997.

BLOM, S.; VAN DE POL, J.; WEBER, M. Ltsmin: Distributed and symbolic re-achability. In Proceedings of the 22Nd International Conference on Computer Aided Verification, CAV'10, p. 354–359. Springer-Verlag, 2010.

KWIATKOWSKA, M. Quantitative verification: models, techniques and tools. In ESEC/FSE'07: The 6th Joint Meeting on European Software Engineering Conference and the ACMSIGSOFT Symposiumon the Foundations of Software Engineering, p. 449–458, Dubrovnik, Croatia, 2007.

Peterson, J. L. (1981). Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA.