UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RENAN FRANCISCO MACARRONI DA SILVA

VERIFICAÇÃO FORMAL USANDO *MODEL CHECKING* PARA SISTEMAS AUTOMOTIVOS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2019

RENAN FRANCISCO MACARRONI DA SILVA

VERIFICAÇÃO FORMAL USANDO *MODEL CHECKING* PARA SISTEMAS AUTOMOTIVOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof.ª Dr.ª Simone Nasser Matos

Coorientador: Prof. Dr. Max Mauro Santos

PONTA GROSSA

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

VERIFICAÇÃO FORMAL USANDO *MODEL CHECKING* PARA SISTEMAS AUTOMOTIVOS

por

RENAN FRANCISCO MACARRONI DA SILVA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 12 de novembro de 2019 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof.ª Dr.ª Simone Nasser Matos
Orientadora

_____
Prof. Dr. Max Mauro Dias Santos
Coorientador

_____
Prof. Dr. Gleifer Vaz Alves
Membro

_____
Prof. MSc. Victor Schnepper Lacerda
Membro

_____
Prof. MSc. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof.ª Dr.ª Mauren Louise Sguario
Coordenadora do Curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Certamente não lembrarei de todas as pessoas que fizeram parte do meu trajeto até aqui, desculpo-me por isso.

Primeiramente, à Deus, ou ao que dizem O ser.

Em especial, à minha orientadora Prof.ª Dr.ª Simone Nasser Matos, pelo carinho, atenção, persuasão e ensinamentos.

Ao meu coorientador Prof. Dr. Max Mauro Dias Santos pela oportunidade proporcionada nesta trajetória.

Ao Prof. Dr. Gleifer Vaz Alves e ao Prof. MSc. Clayton Kossoski, pela sabedoria e paciência.

Aos meus familiares e aos meus poucos e solitários amigos.

E, por fim, mas não menos importante, ao Spotify.

Obrigado.

Uma pessoa inteligente resolve um problema, um sábio o previne.

(EINSTEIN, Albert, 1955)

RESUMO

SILVA, R. F. M. **Verificação formal usando *model checking* para sistemas automotivos**. 2019. 74 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Sistemas computacionais tornam-se complexos, seja pelo seu número de tarefas, disputa de recurso e/ou pela sua precisão, e auxiliam na rotina do ser humano, porém a maioria desses não são devidamente testados e estão suscetíveis a falha, o que muitas vezes acarreta em retrabalho, mal desempenho, despesas e ameaças a vida humana. Métodos formais é uma área da computação que trabalha para minimizar o impacto que possíveis problemas que sistemas de tempo real oferecem aos envolvidos. Dentre as diferentes abordagens da área, este trabalho faz uso da verificação formal de modelos, embasado em axiomas e preposições da lógica temporal. Por meio do estudo de modelo de software automotivo presente no mercado, foi realizada uma verificação formal com a ferramenta UPPAAL em um sistema elétrico de janelas automotivo. A abordagem compreende sua modelagem, simulação, avaliação das especificações e verificação dos requisitos. O resultado mostra que a aplicação de verificação formal em sistemas de tempo real identifica falhas computacionais e oferece alta manutenibilidade, desde que este esteja corretamente modelado.

Palavras-chave: Sistema Embarcado. Sistema automotivo. Verificação formal. Autômatos temporais. UPPAAL.

ABSTRACT

SILVA, R. F. M. **Formal verification using model checking for automotive systems**. 2019. 74 p. Work of Conclusion Course (Graduation in Science Computing) - Federal Technology University - Paraná. Ponta Grossa, 2019.

Computer systems become complex, either because of their number of tasks, resource contention and / or accuracy, and aid in the human routine, but most of these are not properly tested and are susceptible to failure, which often entails rework, poor performance, expenses and threats to human life. Formal methods is an area of computing that works to minimize the impact that potential problems real-time systems have on those involved. Among the different approaches of this area, this work makes use of the formal verification of models, based on axioms and prepositions of temporal logic. Through the study of automotive software model present in the market, a formal verification was performed with the UPPAAL tool in an automotive window electric system. The approach comprises its modeling, simulation, specification evaluation and requirements verification. The result shows that the application of formal verification in real time systems identifies computational failures and offers high maintainability, provided it is correctly modeled.

Keywords: Embedded system. Automotive system. Model checking. Timed Automaton. UPPAAL.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS E ACRÔNIMOS

AUTOSAR	*AUTomotive Open System ARchitecture*
BDD	*Behavior Driven Development*
CTL	*Computation Tree Logic* (Lógica de Árvore de Computação)
CPAL	*Cyber-Physical Action Language*
CPS	*Cyber-Physical Systems*
FUSC	*Free Under Specific Condition* (Livre sobre condições específicas)
LTL	*Linear Temporal Logic* (Lógica Temporal Linear)
MBD	*Model-Based Design*
RTOS	*Real Time Operating Systems* (Sistemas Operacionais de Tempo Real)
RTS	*Real Time Systems* (Sistemas de Tempo Real)

SUMÁRIO

1 INTRODUÇÃO

Sistemas de tempo real (*Real Time System* [RTS]) é uma classe de sistemas computacionais embarcados em que os requisitos temporais devem ser monitorados na execução de tarefas. Tarefas (*tasks*) são um conjunto de instruções destinadas a disponibilizar alguma função ao usuário. Sistemas genéricos são compostos por tarefas, eles têm a característica de executar uma determinada tarefa no menor tempo possível, diferente dos RTS que nunca podem ultrapassar o intervalo de tempo pré-determinado para executar uma determinada função, o não cumprimento de uma meta temporal pode ser dado como falha crítica (KOPETZ, 2011).

Outra característica dos RTS é a prioridade entre as tarefas, o que infere em concorrência por recurso, regiões críticas no modelo e escalonamento por meio de preempção. Ou seja, se uma tarefa com alta prioridade for requisita, esta, por sua vez, pode interromper outro procedimento de baixa prioridade para ser processada (SAEEDLOEI, 2011).

A Microchip (2008) aborda um exemplo de prioridade entre as tarefas em uma máquina de lavar roupa, o sistema deve gerenciar diversas funções, entre elas o controle da água e as informações exibidas no *display*, como tempo de lavagem e secagem. Num eventual uso, o equipamento conclui que deve bloquear todo o abastecimento de água, caso contrário, o líquido irá transbordar. A tarefa de bloqueio possuí alta prioridade no sistema e provavelmente um *deadline* rigoroso, pois o derramamento do liquido pode gerar prejuízos. Neste caso, o escalonamento vai agir por meio da preempção do sistema, interrompendo a execução de tarefas com prioridades menores do que a sua e atualizar as informações impressas no *display*.

Por se tratar de sistemas críticos, Clarke e Wing (1996) afirmam que uma verificação rígida para minimizar suas falhas é importante. Esta etapa pode ser realizada com o uso de métodos formais. A verificação formal é ideal para ser aplicada em modelos onde as falhas de um sistema têm ligação com condições que envolvem diretamente a vida humana e/ou causam grande prejuízo monetário, como é o caso de equipamentos hospitalares, sistemas de aeronaves ou automotivos, por exemplo (FRAPPIER *et al*., 2010). O uso de especificação formal nesses cenários é importante, porém aplicar a técnica exige tempo hábil e profissionais qualificados para abstração do problema, o que, consequentemente, resulta em um maior

consumo de dinheiro. Contudo, como resultado, pode-se garantir a corretude computacional em sistemas de tempo real (CLARKE; WING, 1996).

Há trabalhos em repositórios científicos que mostram como fazer verificação formal em sistemas de forma genérica (LARSEN, 2014; PETTERSSON, 1999), pouco se encontra sobre o processo de abstração do problema para sua modelagem.

Este trabalho realizou a verificação formal de uma atividade de um software automotivo, um sistema elétrico de janelas (SANTOS *et al*., 2015), desde a compreensão da tarefa, sua modelagem, simulação e validação.

O sistema em questão trata-se do gerenciamento sobre duas janelas automotivas, uma do passageiro e outra do motorista, ambos podem movimentar sua respectiva janela, salvo o motorista que pode movimentar a do passageiro e tem prioridade sobre ela. Existe um sistema de segurança, que eleva todos os vidros até sua altura máxima. Além disso, há um sensor de obstruções, o qual quando acionado, indica que um objeto (ou membro humano e/ou animal) encontra-se no trajeto do vidro durante as fases de elevação (SANTOS *et al*., 2015).

De tal forma, os recursos disputados pelo modelo são as janelas, principalmente o vidro do passageiro, onde, além da concorrência pelo recurso, persiste a prioridade das ações do motorista sobre as ações do passageiro. Como o sistema traz o fator de risco sobre a vida humana, o sistema deve atender casos de obstrução com prioridade máxima, em seguida, a segunda maior prioridade é a do sistema de segurança, depois, as ações do motorista e, por fim, as ações do passageiro (SANTOS *et al*., 2015).

A aplicação de métodos formais, por tratar de axiomas e rigorosas condições matemáticas, elimina ambiguidade, inconsistência e *deadlocks* (bloqueios) no modelo, como afirma Dennis *et al*. (2012). Pressman e Maxim (2016) diz que, dentre os métodos que auxiliam nas etapas de desenvolvimento de software, por meio de provas matemáticas, a verificação formal pode assegurar requisitos lógicos em simulação representadas em máquinas de estado e, por sua vez, detectar e eliminar falhas previamente.

Uma máquina de estados é um autômato temporizado, do inglês *timed automaton*, e faz parte da classe de autômatos finitos, com a adição de um conjunto de relógios que funcionam de forma sincronizada. É usada para modelagem de sistemas em que o tempo é um fator importante para ser observado (ALUR, DILL,

1994). *Model checking* é uma técnica de verificação e validação automática de modelos de sistemas concorrentes com conjunto de estados finitos (PELED *et al*., 1999).

Zhongsheng *et al*. (2018) realiza uma verificação formal sobre uma modelagem de um sistema de elevador com três ferramentas, dentre elas, UPPAAL se destaca por, principalmente, gerenciar *clock* de sistemas e sincronizar seus modelos por meio de canais de comunicação, além de uma interface gráfica intuitiva.

A ferramenta UPPAAL, utilizada nesse trabalho, é um *software* de verificação formal de modelos. Seu ambiente de desenvolvimento auxilia na eliminação de falhas computacionais com três fases de desenvolvimento: modelagem – compreende-se os autômatos envolvidos no sistema, simulação – ocorre a explosão de estados e é possível analisar o sistema em processo de execução, e validação – por meio da lógica temporal o software verifica a corretude de requisitos do sistema (DAVID *et al*., 2015; PETTERSSON, 1999).

1.1 JUSTIFICATIVA

Sistemas computacionais podem monitorar, controlar e gerenciar uma planta física real – por meio de sensores e/ou atuadores –, Sistemas de tempo real englobam computadores que atendem tarefas em que a resposta correta deve ser dada no momento certo, e sua exatidão depende do tempo físico das próprias operações lógicas (KOPETZ, 2011).

Quando sistemas dessa categoria estão em operação, falhas computacionais retornam prejuízos monetários à desenvolvedora, além de colocar em risco à vida humana, como no caso de equipamentos de medição, instrumentos hospitalares, sistemas elétricos que controlam aviões, elevadores e automóveis, entre outros (ZHONGSHENG *et al*., 2018).

Testar software com essas características é uma atividade complexa, pois o operador deve considerar a variável de tempo, uma tarefa qualquer pode ser iniciada num dado momento, tal atividade, se iniciada, estimula outras tarefas que são sensíveis ao tempo que foram iniciadas e podem apresentar diferentes comportamentos. Torna-se complexo definir se o evento cronometrado observado foi

satisfatório. Este cenário traz inúmeros casos de teste, ou seja, o próprio teste do software se torna uma espécie de sistema de tempo real (PETTERSSON, 1999).

Ferramentas de *model checking* (verificação de modelos) têm a vantagem de verificar a segurança e corretude de sistemas de tempo real genéricos devido a sua capacidade de detectar automaticamente todos os estados possíveis no modelo, com o uso de máquina de estados e propriedades matemáticas. Um dos verificadores de modelos mais populares é o UPPAAL, utilizado para modelar, validar e verificar sistemas de tempo real (LARSEN *et al*., 2014).

A aplicação da técnica de especificação formal requer tempo hábil e profissionais qualificados para abstração do modelo do sistema, por efeito da demanda do mercado, muitos produtos em circulação pelo público não são submetidos à testes em ferramentas de *model checking* ou são testados em um conjunto de casos manualmente. Portanto, o sistema em prática pode atingir um estado não previsto e resultar em catástrofes (ZHONGSHENG *et al*., 2018).

1.2 OBJETIVOS

Especificar um sistema automotivo na ferramenta de *model checking* UPPAAL para descobrir falhas computacionais. Os objetivos específicos deste trabalho são:

- Modelar esquema computacional na ferramenta de *model checking* UPPAAL;
- Realizar testes sobre o modelo por meio da lógica temporal;
- Analisar os resultados obtidos.

1.3 ORGANIZAÇÃO DO TRABALHO

Esse trabalho está estruturado em sete capítulos. O Capítulo 2 apresenta as linguagens e ferramentas de verificação formal e sua contribuição na fase de desenvolvimento.

O Capítulo 3 descreve os conceitos sobre verificação formal e apresentação de ferramentas que contribuem na fase de desenvolvimento.

O Capítulo 4 narra as características de sistemas automotivos na indústria e qualidade dos testes de software.

O Capítulo 5 apresenta a aplicação da verificação formal sobre as especificações do sistema elétrico de janelas automotivas.

O Capítulo 6 narra os resultados obtidos com a aplicação da verificação formal. Por fim, o capítulo 7 descreve as conclusões, contribuições e trabalhos futuros que dão continuidade a este trabalho de pesquisa.

2 SISTEMAS DE TEMPO REAL

Kopetz (2011) afirma que sistema de tempo real – RTS (*Real Time System*), também conhecido como Sistema operacional de tempo real (*Real Time Operation System*), são *software* com um tempo limite de execução de tarefas, ou seja, os processos possuem um prazo para ser atendido e executado. Se o prazo estiver esgotado, caracteriza-se uma falha computacional. Os RTS são avaliados em três categorias (LARSEN *et al*., 2014):

- Requisitos Funcionais – diz respeito a realização de uma tarefa lógica atendida pelo *software*, como a comunicação de um sensor ou a alteração de uma variável, a tarefa muitas vezes pode ser um conjunto de outros procedimentos para atingir um objetivo.
- Requisitos Temporais – são funcionalidades do software que são sensíveis ao tempo, tais tarefas tem a característica de possuir um tempo mínimo e máximo para ser atendida e executada.
- Requisitos de Dependabilidade – é a qualidade do serviço oferecida pelo sistema, bem como uma prova de sua confiabilidade, principalmente em relação a disponibilidade de recursos compartilhados e segurança (*safety* – segurança como integridade dos dados).

Ressalta-se que um RTS não significa ser veloz, mas sim evoluir e atender necessidades de acordo com o ritmo do mundo real afim de interagir com o ambiente por meio de sensores e/ou atuadores (KOPETZ, 2011).

Este capítulo descreve informações sobre os RTS que foram importantes para o desenvolvimento deste trabalho. A Seção 2.1 apresenta os conceitos e definições sobre RTS e relata algumas aplicações dos RTS. A Seção 2.2 descreve os métodos para validação de um RTS. Por fim, a Seção 2.3 descreve as considerações finais deste capítulo.

2.1 CONCEITOS E APLICAÇÕES DE RTS

Um programa *P*, com entrada *I* (*input*), saída *O* (*output*), *T* é o tempo destinado para o sistema devolver uma resposta e, por fim, *D* diz respeito ao

deadline da operação, deste modo é dado a configuração de um RTS (PELED *et al*., 1999):

$$O = P\ (I);\ T < D \qquad (1)$$

Considerando requisitos temporais, pode-se subdividi-los em dois aspectos: sistemas de tempo real rígido (*hard*) e flexível (*soft*). Os sistemas em que nenhuma tarefa pode deixar de ser atendida no prazo, ou seja, todos os *deadlines* devem ser cumpridos rigorosamente são os de tempo real rígido. Respectivamente, no segundo caso alguns *deadlines* podem ser controladamente descumpridos, como pode acontecer em alguns captadores ou reprodutores de mídias, um pacote de dados pode ser corrompido e não causar problemas, porém a perda de vários pacotes pode gerar prejuízos (ALUR; DILL, 1994).

Outra característica desses sistemas é a prioridade entre as tarefas, o que infere em concorrência por recurso, regiões críticas no modelo, escalonamento de tarefas por meio de preempção, ou seja, se dete-rminada tarefa com alta prioridade for requisita, por sua vez, ela pode interromper uma outra de baixa prioridade para ser processada (SAEEDLOEI, 2011).

Existem algumas linguagens de programação de sistemas de tempo real, tais como: C, C++, Ada, JavaRT e *Assembly*. Porém, a maioria das aplicações são desenvolvidas principalmente em C e Java RT, devido a sua portabilidade e grande comunidade (BUTTAZZO; LIPARI, 2013). A *Semantic Scholar*, por exemplo, disponibiliza bibliotecas em C para programar sistemas de tempo real em diversas plataformas[1].

De forma geral, os critérios de seleção da linguagem podem variar bastante, a escolha deve ser baseada de acordo com a portabilidade, flexibilidade, custo, abstração, manipulação de recursos, entre outros. O uso de *Assembly*, por exemplo, traz maior controle sobre elementos de hardware e maior previsibilidade. Já a linguagem C, fornece um alto nível de abstração, mas ainda permite acesso aos detalhes do hardware (BUTTAZZO; LIPARI, 2013).

É comum o consumo de aplicações computacionais em que o tempo gasto para devolver uma saída não é algo significativo. Porém, existem casos em que o tempo pode ser crucial entre as operações por isto é necessário ter um controle e precisão sobre sistemas desta categoria.

[1]O endereço eletrônico para sistemas Linux pode ser encontrado nas referências deste projeto.

RTS são aplicadas comumente em circuitos embarcados, de forma que o usuário não possa ter acesso ao seu núcleo diretamente, como ocorre em equipamentos como micro-ondas e televisores. Outros exemplos desse cenário estão listados a seguir (BUTTAZZO; LIPARI, 2013):

- Meios de locomoção: automóveis, aeronaves, elevadores, ferrovias, entre outros;
- Clínicos: equipamentos de medição como radiografia, ressonância magnética, equipamentos cirúrgicos, entre diversos elementos;
- Industriais: usinas nucleares, hidroelétricas, industrias que fazem uso de componentes químicos, esteiras e maquinário de linha de produção, etc.

2.2 TESTE DE SOFTWARE

É comum na área de desenvolvimento de software setores com a finalidade de teste dos programas fabricados. Para atender a agilidade do mercado, busca-se testes de rápida aplicação com bons resultados e com uso de pouco recurso (ALUR; DILL, 1994).

Dois métodos de teste são utilizados para cenários diferentes: o teste de caixa-branca, para quando se tem acesso ao código-fonte do software, e o teste de caixa-preta, para quando não o tem (PELED *et al*., 1999).

O teste de caixa-preta verifica as saídas do software usando entradas de vários tipos, quanto mais entradas e diversificadas mais rico é o teste. Como testar todos os casos é impossível na maioria dos cenários, uma abordagem realista para o teste de caixa-preta é escolher representantes para categorias de entrada, de forma que testar um elemento prove a qualidade do subconjunto, como, por exemplo, ao invés de testar todo o índice de 0 à N de uma lista, selecionar apenas inteiros que extrapolam os índices (como o índice -1 ou N+1), ou em casos onde a entrada esperada é do tipo texto, selecionar valores numéricos, ou vice-versa (PELED *et al*., 1999).

O teste de caixa-branca é uma técnica que faz uso do código fonte do software para modelar seus casos de teste, o que exige mais conhecimento técnico do profissional. Contudo, uma vantagem deste método é que, encontrar valores de

entrada é mais eficiente visto que a implementação do programa é dada como guia (PELED *et al*., 1999).

Ambos os procedimentos citados nesta seção podem ser usados em conjunto, contextualizando o teste de caixa-cinza. Nessa organização, os casos de teste são modelados conhecendo-se a estrutura interna do sistema, como no teste de caixa-branca, mas a execução é feita como no teste de caixa-preta, ignorando o aspecto da ciência do núcleo do software (ALUR; DILL, 1994).

O trabalho de Roquette (2018), traz uma abordagem dinâmica se comparada aos encontrados na literatura, porém pode-se ressaltar que, por mais que se cubra todos os casos de teste e alcance a validação – que é o cumprimento e corretude de todos os requisitos especificados – é difícil assegurar que uma falha não prevista aconteça, a verificação formal tratada neste trabalho testa todos os cenários de forma automática e busca a validação e segurança do aplicativo (DENNIS, 2012).

2.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Os testes de caixa preta, branca e cinza são eficientes, porém, aplicá-los em RTS é uma tarefa árdua e pode não apontar de fato todas as falhas do sistema, principalmente pelo cenário caótico da explosão de estados, ressaltando que, além de que sistemas desse gênero devem devolver resultados precisos, sistemas de tempo real possuem um prazo de tempo sobre o processo que deve ser cumprido.

Métodos formais são ideais em cenários desse tipo, visto que é difícil prever todas os cenários que o software será desafiado de forma manual. As ferramentas de verificação formal auxiliam na verificação automática dos requisitos implementados e valida o sistema como um todo.

3 VERIFICAÇÃO FORMAL

Métodos formais fazem uso de recursos para provar propriedades matemáticas em algoritmos a partir de axiomas, especificações e verificações formais, que é basicamente a descrição matemática de um determinado sistema somado com seu objetivo (DENNIS *et al*., 2012).

Pressman e Maxim (2016) comentam que dentre os métodos de criação de software existentes na literatura não garantem a ausência de erros. Os métodos formais trazem a promessa de construção de um sistema livre de falhas, visto que sua modelagem se baseia em máquina de estados, na qual se deve provar propriedades matemáticas sobre os eventos de um sistema. A aplicação dessa técnica trata de axiomas e rigorosas condições matemáticas. Ela pode eliminar ambiguidade, inconsistências e *deadlocks* (bloqueios de recursos) no modelo.

Métodos formais são usados por desenvolvedores que trabalham em projetos em que as falhas de um sistema computacional têm ligação com condições que envolvem diretamente a vida humana e/ou causam grande prejuízo monetário, como é o caso de sistemas de aeronaves ou exames clínicos. O uso da verificação formal é mais presente nesses casos pois a metodologia consome mais tempo na etapa de planejamento e desenvolvimento porque necessita de um delicado estudo sobre o problema. Em contrapartida, é inevitável o investimento em profissionais capacitados, pois há carência nesse nicho de mercado, o que, consequentemente, resulta em um maior consumo de dinheiro do lado empreendedor (PRESSMAN, MAXIM, 2016; DENNIS *et al*., 2012).

Na seção 3.1 são apresentados os conceitos de verificação formal. A seção 3.2 descreve sobre as técnicas de verificação formal. A seção 3.3 relata sobre as ferramentas usadas para aplicar a abordagem. Por fim, a última seção apresenta as considerações finais deste capítulo.

3.1 CONCEITOS

Conhecido como *Model Checking*, a verificação formal é uma técnica automática de verificação de modelos, simulação e validação de sistemas concorrentes com conjunto de estados finitos (PELED *et al*., 1999).

Na área da Ciência da Computação conhecida como Métodos Formais, a verificação de modelos trata de testar automaticamente um modelo dado uma especificação. Portanto, o modelo pode trazer abstrações de hardware, software e atuadores, como pessoas, por exemplo (DAVID *et al*., 2015).

A verificação formal faz uso de especificação formal para tratar seus modelos, que são descrições matemáticas que conceituam uma implementação, deste modo, são desenvolvidos autômatos focados nos objetivos da aplicação, guardas (que são condições lógicas testadas na transição de estados), invariantes, canais de comunicação entre os autômatos e eventuais variáveis (DENNIS *et al*., 2012).

Garis (2010) afirma que para realizar o conjunto de testes as ferramentas utilizam de lógica temporal, mais especificamente o conjunto CTL* - junção da lógica de árvore computacional (*Computation Tree Logic* [CTL]) com lógica temporal linear (*Linear Temporal Logic* [LTL]). A CTL prevê todos os possíveis caminhos que um sistema pode conter por meio de máquina de estados, enquanto a LTL define objetivos e condições para o futuro, com isso, o software de verificação de modelo podem testar casos como “Tarefa A só pode ocorrer quando Tarefa B for realizada” ou “Em algum momento agora ou no futuro Tarefa A vai acontecer”.

Neste contexto, um dos requisitos mais comuns testados em modelos é a ausência de *deadlock* (GARIS, 2010). *Deadlock* é o termo usado quando ocorre de duas ou mais tarefas aguardarem a liberação de recurso uma da outra, de forma que nenhumas das funcionalidades procedam e o sistema literalmente para (PELED *et al*., 1999).

3.2 TÉCNICAS DE VERIFICAÇÃO FORMAL

Um estudo de Dennis *et al*. (2012) mostra que a maioria das ferramentas de *model checking* utilizam máquinas de estados para realizar suas verificações, o que na prática significa combinar todos os estados de todos os modelos uns com os outros. Tal atividade é comumente conhecida como problema de explosão de estado (*Combinatorial explosion*) e resolver essa situação significa solucionar um problema do mundo real. Existem várias técnicas para isso, as mais relevantes são (LARSEN *et al*., 2014):

- Abstração: simplifica as propriedades de um sistema e tenta satisfazer as regras do sistema original. Geralmente isso não ocorre, de modo que um processo de refinamento pode ser necessário. Um exemplo desse cenário é testar a propriedade de exclusão mútua em um sistema considerando apenas as variáveis booleanas e o fluxo da programação.
- Refinamento de abstração orientada por contraexemplos: inicia a verificação com uma abstração qualquer. Quando uma falha é encontrada, a técnica analisa se houve uma violação ou se a saída está incompleta. Se sim, para o segundo caso, a abstração é refinada e o processo repetido.
- Algoritmos simbólicos: sempre evitam construir diretamente o gráfico para as máquinas de estados finitas; em vez disso, eles representam o gráfico implicitamente atribuindo a cada elemento um significado associado (ou associável a outro).
- Algoritmos de verificação de modelo delimitado: modificam o autômato para um número fixo de etapas, em seguida, verifica se uma violação de propriedade pode ocorrer em alguma delas. O processo pode ser repetido com valores cada vez maiores de etapas até que todas as possíveis falhas sejam descartadas.
- Redução parcial de pedidos: pode ser usada em máquinas de estados diretamente representadas, tem o objetivo de reduzir o número de intercalações de processos concorrentes a serem considerados. A ideia geral é que se "A" acontece antes de "B", pode-se então descartar toda possibilidade do cenário "BA".
- *On the fly*: termo utilizado para descrever algum processo que altera seu valor durante a execução do sistema. Também é definido como a ação de criar variáveis ou axiomas quando necessário.

Em especial, a última técnica citada é utilizada na área automotiva, pois ela infere sobre personalização e adaptação de um *software* perante o usuário. Considerando um veículo com tração nas quatro rodas, pode-se testar funcionalidades de trocar a tração de quatro para duas rodas enquanto o carro está em movimento (SCHAUFFELE, ZURAKAWA, 2005). Máquinas de estados podem ser representadas por meio de autômatos temporais, descritos na próxima seção.

3.2.1 Autômatos Temporais

Do inglês *timed automaton*, faz parte da classe de autômatos finitos, com a adição de um conjunto de relógios que funcionam de forma sincronizada. É usado para modelagem de sistemas quando o tempo é um fator importante para ser observado. Sua definição formal é dada por (PELED *et al*., 1999):

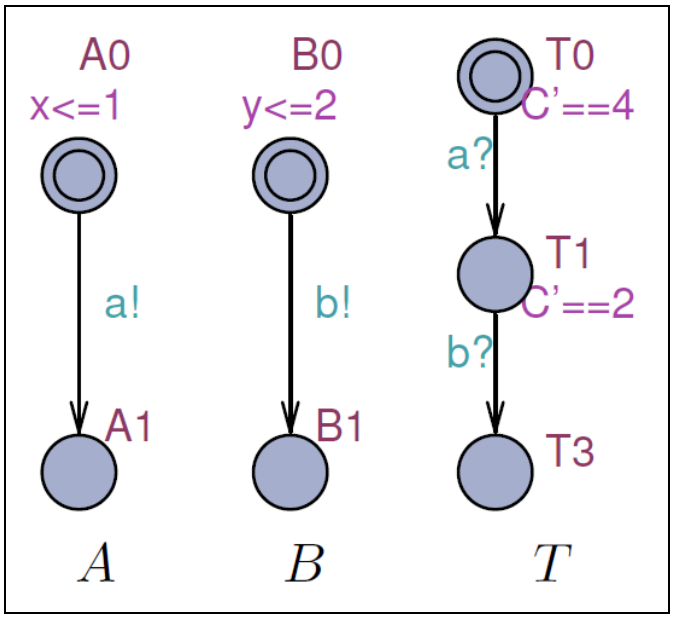
$$A = (\Sigma, S, S_0, C, E) \quad (2)$$

Onde:

- Σ é o conjunto do alfabeto;
- *S* o conjunto de estados;
- $S_0 \subseteq S$ o conjunto de estados iniciais;
- *C* é o conjunto de relógios (*clocks*);
- $E \subseteq S \times S_0 \times \Sigma \times 2^C \times \phi\ (C)$ são as regras de produção do autômato, representa as transições de um estado para outro (ALUR, DILL, 1994).

Na figura 1 se pode analisar um autômato temporal em que existem dois sistemas “A” e “B” disputando o recurso “T”, o vértice (estado) com um círculo interno representa o estado inicial do autômato. O *clock* é dado pelas variáveis “x” e “y”. Existe uma variável armazenando valores com o nome de “C’”. Também pode-se notar o uso de operadores lógicos em alguns estados, eles representam a condição que deve ser satisfeita para que a transição seja realizada (LARSEN *et al*., 2014).

Figura 1 - Autômato temporal modelado em UPPAAL



Fonte: Larsen *et al*. (2014)

A característica importante que une os três sistemas são os canais de comunicação, que podem ser definidos livremente pelo desenvolvedor. Quando há uma exclamação no final do termo, significa que uma mensagem está sendo transmitida por esse canal. A mensagem enviada é processada no próximo ciclo de execução do *clock* do computador, a transição que possui uma interrogação consome essas mensagens, o que indica que a transição nessa aresta só acontece quando o canal é utilizado pelos sistemas “A” e “B”. Por traz desse modelo a ferramenta permite a injeção de código C nos modelos para manipulação avançada dos dados (LARSEN *et al*., 2014).

3.3 FERRAMENTAS DE VERIFICAÇÂO FORMAL

Segundo Zhongsheng *et al*. (2018), existem várias ferramentas de *model checking* no mercado, dentre elas, uma comparação de três ferramentas foi realizada: SPIN (SPIN, 2016), UPPAAL (UPPAAL, 2015) e NuSMV (NuSMV, 1999). Como aponta o quadro 1, embora não seja diretamente simples tratar concorrência, UPPAAL possui um melhor conjunto de características, além de intuitivo com alta manutenibilidade (ZHONGSHENG *et al*., 2018).

Quadro 1 – Comparação de propriedades de verificação

Características	UPPAAL	SPIN	NuSMV
Safety (corretude)	Sim	Sim	Sim
Detecção de *not deadlock*	Sim	Sim	Sim
Atividade	Sim	Sim	Sim
Alcançabilidade	Sim	Sim	Sim
RTS	Sim	Não	Não
Interface gráfica	Sim	Sim	Não
Geração de contraexemplos	Sim	Sim	Não
Especificação gráfica	Sim	Não	Não
Concorrência	Não	Sim	Sim

Fonte: Adaptado de Zhongsheng *et al*. (2018)

O desempenho da verificação de tempo real por SPIN e NuSMV não é tão eficaz quanto pelo UPPAAL, pois ele possui relógios internos em seu sistema. NuSMV e SPIN podem por sua vez transmitir mais de uma mensagem por canal, o que o torna os mais adequados para a verificação de sistemas distribuídos, enquanto UPPAAL é melhor para à verificação de sistemas de tempo real. Contudo, vale ressaltar que NuSMV possui licença de uso livre enquanto SPIN e UPPAAL licença FUSC[2] (ZHONGSHENG *et al*., 2018).

SPIN é uma ferramenta de verificação formal distribuída de forma *open source* para a verificação de sistemas, é aplicada em diversas áreas de estudos teóricos e práticos, principalmente na automotiva. A modelagem no ambiente é dada por meio da linguagem PROMELA (*PROcess MEta LAnguage*), a qual permite a simulação interativa e verificações especificadas em LTL. Ainda na etapa de verificação, a ferramenta gera um algoritmo em linguagem C capaz de testar o modelo (FRAPPIER *et al*., 2010).

Desenvolvido em conjunto entre a Universidade Carnegie Mellon, Universidade de Gênova, Universidade de Trento e a ITC-IRST (*Istituto Trentino di Cultura* em Trento, Itália), o NuSMV, é uma extensão do verificador de modelo simbólico SMV, a primeira ferramenta de verificação de modelo baseada em diagramas de decisão binária (FRAPPIER *et al*., 2010). Segundo Frappier (*et al*., 2010), o usuário interage com a ferramenta de forma textual, suportando especificações expressas em CTL e LTL, destina-se à verificação de projetos de

[2] Livre sobre condições específicas (*Free Under Specific Condition* - e.g., livre para uso acadêmico).

tamanho industrial, para uso em conjunto com ferramentas de verificação e como ferramenta de pesquisa para técnicas formais de verificação.

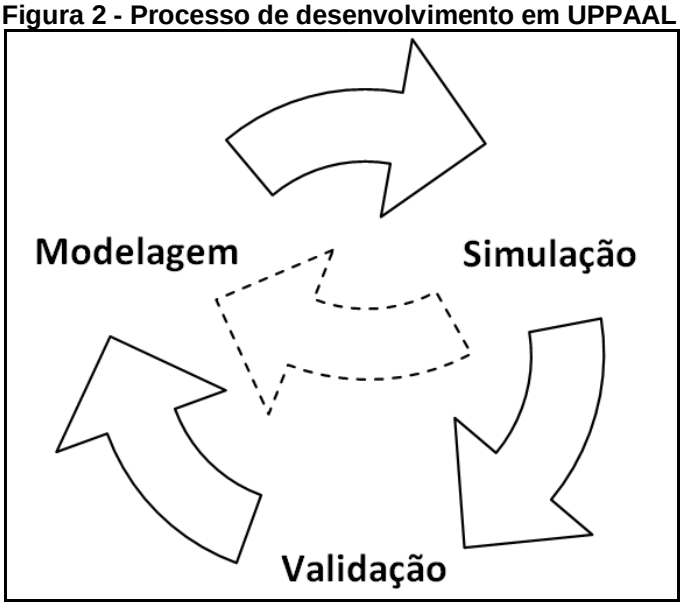
Zhongsheng *et al*. (2018) aponta que a ferramenta de *model checking* UPPAAL é intuitiva por conta de sua modelagem e programação via interface, apresenta facilidades ao reusar códigos – pois trabalha com arquivo de modelo de extensão *.xml* e arquivos de teste de extensão *.q* –, e em detectar e corrigir erros. Empresas como Philips e Lego Mindstorms possuem projetos em formato UPPAAL no próprio *site* da ferramenta, dado a facilidade em compreender sistemas completos de forma simples e com alta manutenibilidade em seus modelos. Além disso, o software oferece uma licença gratuita para uso com fins acadêmico e de pesquisa (DAVID *et al.*, 2015).

3.3.1 UPPAAL

Desenvolvido em C++ e Java entre uma parceria da Universidade de Uppsala (Suécia) com a Universidade de Aalborg (Dinamarca), UPPAAL é um software de verificação formal de modelos baseado em autômatos temporais, o que torna natural a simulação, verificação e validação de sistemas de tempo real, garantindo a eliminação de falhas computacionais (DAVID *et al*., 2015).

Como a modelagem exibida na figura 2, algumas ferramentas de *model checking*, são capazes de testar caminhos e propriedades em sistemas computacionais, por meio de lógica temporal. O ambiente de desenvolvimento do UPPAAL conta com três abas em seu processo de desenvolvimento: edição (modelagem), simulação e validação.

Figura 2 - Processo de desenvolvimento em UPPAAL



Fonte: Autoria própria

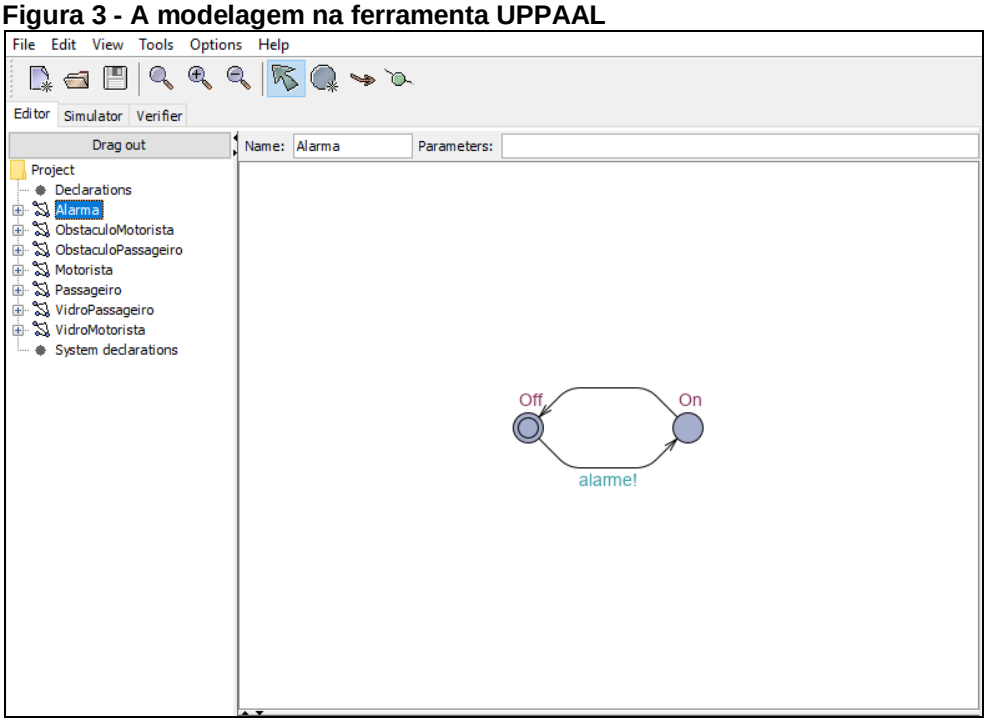
A primeira etapa é a Modelagem, como apresenta a figura 2, é o ambiente do programa em que será modelado o sistema desejado, a dinâmica da ferramenta pode ser visualizada nas figuras 3 e 4. Por exigir planejamento e corretude na abstração do modelo para o *software* essa fase consume a maior parte de tempo do desenvolvimento. O ambiente serve como uma linguagem de modelagem ou *design* para descrever o comportamento do sistema com redes de autômatos somados às variáveis de dados e relógios (CLARKE; WING, 1996).

A figura 3 exibe a modelagem gráfica de cada sistema e recurso enquanto a figura 4 apresenta a programação destes mesmos autômatos. Pode-se observar do lado esquerdo os autômatos modelados, ao topo, em "*Declarations*", encontra-se as declarações globais do sistema, exposta na figura 4, é onde são declaradas as variáveis globais e os canais de comunicação do sistema, que é por onde ocorre a sincronização entre os autômatos e a troca de informação (DENNIS *et al*., 2012).

Um canal de comunicação envia mensagens com o uso de uma exclamação e os canais expressos com uma interrogação "escuta" as mensagens. Os canais de comunicação são divididos em três grupos (LARSEN *et al*., 2014):

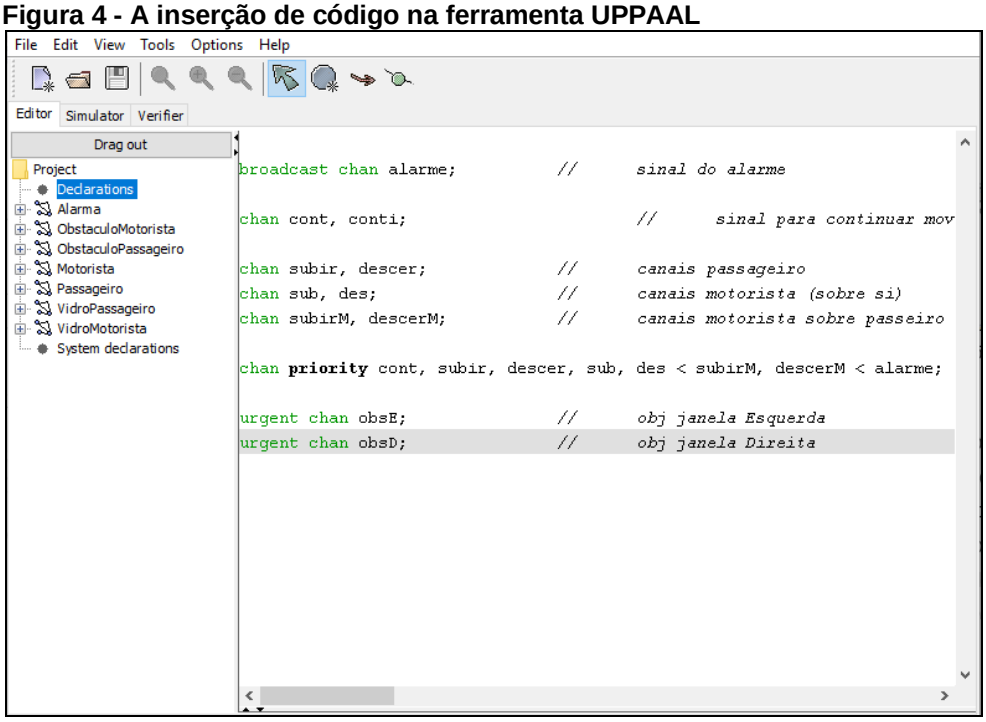
- Padrão – Uma mensagem é enviada e processada por vez;
- *Broadcast* – Quando ocorre a mensagem, ela é processada por todos os ouvintes;
- *Urgent* – Assim que ativada, para o clock de todos os modelos e é processado de imediato pelo sistema.

Figura 3 - A modelagem na ferramenta UPPAAL



Fonte: Adaptado de UPPAAL (2015)

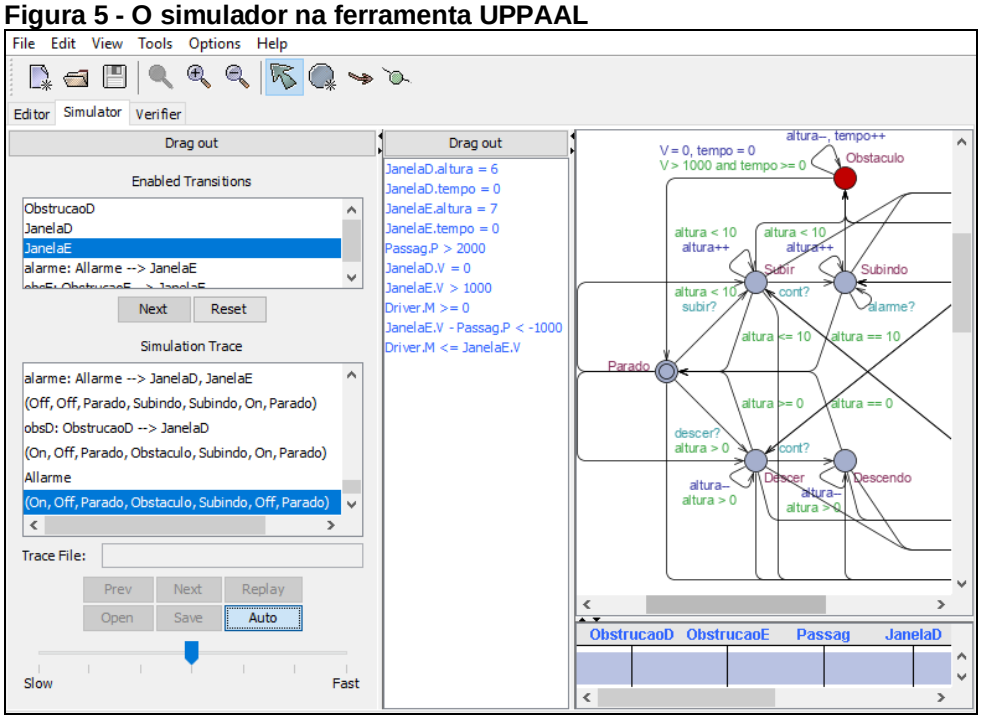
Figura 4 - A inserção de código na ferramenta UPPAAL



Fonte: Adaptado de UPPAAL (2015)

O simulador é a etapa de validação que permite a análise execuções dinâmicas de um sistema durante os estágios de projeto (ou modelagem). Os dois principais critérios para isso ser possível é o uso das técnicas *On the fly* em paralelo com algoritmos simbólicos. A figura 5 apresenta a simulação do modelo por meio de máquina de estados, a ferramenta pode gerar automaticamente um rastreamento de diagnóstico que explica por que uma propriedade é ou não satisfeita por uma descrição do sistema (LARSEN *et al*., 2014).

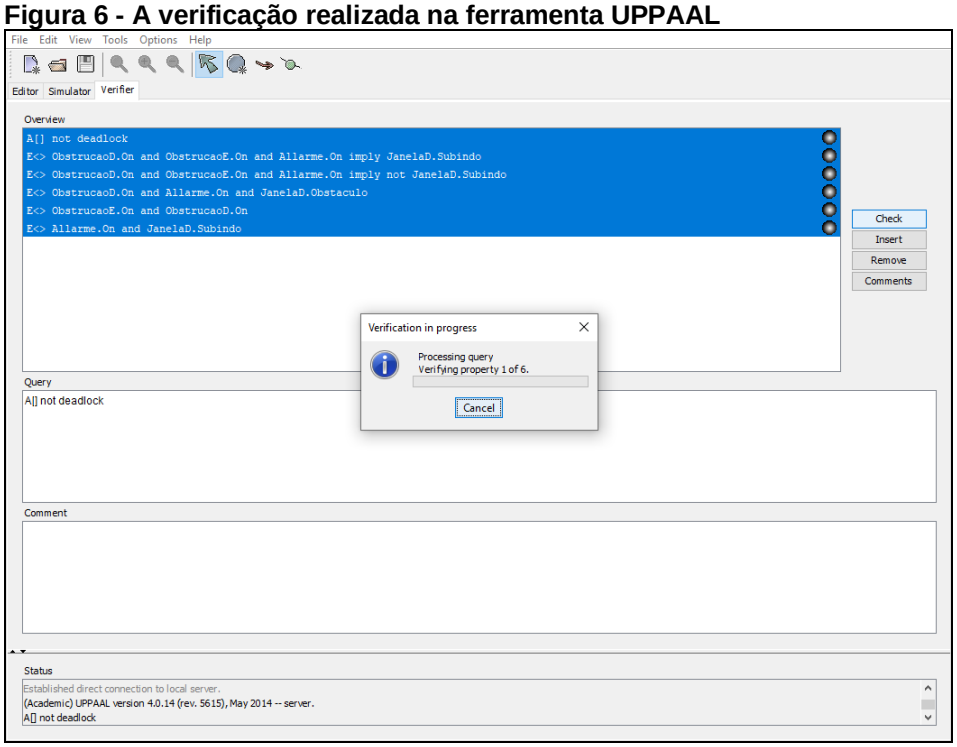
Figura 5 - O simulador na ferramenta UPPAAL



Fonte: Adaptado de UPPAAL (2015)

Pettersson (1999) afirma que o verificador de modelos pode explorar as propriedades invariantes e de alcançabilidade do projeto vasculhando os estados gerados na fase anterior. De tal maneira, o sistema pode ser testado por meio de propriedades que contém informações não tão óbvias do modelo, como por exemplo, capturar a probabilidade de um determinado estado ser atingido com o tempo de relógio em um momento específico ou, como exposto na figura 6, a ausência de *deadlock* no sistema.

Figura 6 - A verificação realizada na ferramenta UPPAAL



Fonte: Adaptado de UPPAAL (2015)

Na verificação, o sistema utiliza de *queries* para testar a lógica, valores de variáveis, alcance de estado e controle e passagem de tempo. O UPPAAL aceita os seguintes testes (LARSEN et al., 2014):

- Segurança – A[ ]X ou E[ ]Y: a primeira expressão verifica se para todos os caminhos possíveis (dado pela variável “A”) se X é satisfeito. A segunda expressão testa se existe um caminho (dado pela variável “E”) em que todos os estados satisfazem Y.
- Alcance (*reachability*) – E< >X: significa que, em algum momento, agora ou no futuro, há um momento onde X é satisfeito.
- Tempo de vida (*liveness*) – A< >X: para todos os caminhos possíveis, em algum momento, um estado satisfará X.
- Leva a (*leads to*) – X-->Y: sempre que X é processada, em algum momento posterior Y será satisfeito.

3.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Com a verificação formal, pode-se testar possíveis cenários e parâmetros de forma matemática e automatizada em sistemas computacionais, desde softwares

triviais à programas com regiões críticas de escala industrial, partindo do pressuposto da exatidão da representação do esquema no modelo.

Entre as ferramentas analisadas, o UPPAAL apresenta a característica de conseguir modelar os componentes de um sistema, desde hardware até o controle temporal do software, simular os modelos da etapa anterior como um sistema em execução de fato e, por fim, verificar propriedades lógicas, com a facilidade em manutenibilidade e testes compreensíveis.

4 SISTEMAS AUTOMOTIVOS

Sistemas automotivos buscam o desempenho em conforto, segurança, corretude e economia na produção de seus produtos. Recentemente o desenvolvimento de dispositivos automotivos transitam da disciplina de engenharia mecânica/elétrica para uma combinação de software em conjunto com essas mesmas engenharias. Essa mudança trouxe uma nova tendência nas indústrias automotivas em aderir ao desenvolvimento baseado em modelo em sua linha de processo (SCHAUFFELE, ZURAKAWA, 2005).

A seção 4.1 comenta sobre alguns softwares que auxiliam na compreensão de sistemas embarcados. A seção 4.2 descreve sobre os testes automotivos. Por fim, a seção 4.3 apresenta as considerações finais deste capítulo.

4.1 CONCEITOS

Quase todas as funções a bordo de um veículo moderno são controladas e/ou monitoradas eletronicamente. No entanto, o desenvolvimento de automóveis pede pré-requisitos contrastantes, como exigências de segurança e confiabilidade além de buscar custos baixos, ciclos de vida de produto mais longos juntamente com tempo de desenvolvimento mais curto, além de diversas outras invariantes possíveis no modelo (SCHAUFFELE, ZURAKAWA, 2005).

O *Controller Area Network* (CAN) é um protocolo de comunicação serial para sincronismo entre modelos, criado na Alemanha pela empresa BOSCH. Atualmente é utilizado na modelagem de veículos automotivos, navios, entre outros, pela indústria. Este protocolo trabalha com um conceito de múltiplos mestres, ou seja, módulos podem ser controladores ou operadores dependendo do cenário, além de suas mensagens serem enviadas como *broadcast*, que é o envio de um sinal para todos os canais disponíveis para transição. Seu custo é alto, tornando-se viável apenas para aplicações prioritárias, porém um ponto forte desta modelagem é possuir uma detecção de colisão de múltiplos acesso, ou seja, os módulos verificam o estado do barramento quando enviado uma mensagem, gerenciando as prioridades entre os canais (BABICH; DEOTTO, 2002).

LIN-Bus (*Local Interconnect Network*) é um sistema de barramento utilizado como uma alternativa ao CAN, por seu custo ser mais viável, visto que o modelo é menor e economiza mais energia. Desenvolvida em linguagem C, a diferença principal é que LIN-Bus possui um controlador principal que gerencia até 16 outros modelos, porém o sistema não apresenta nenhuma característica de detecção de colisão (LIN-BUS, 2019).

MOST (*Media Oriented Systems Transport*) é um padrão de barramento automotivo desenvolvido pela Microchip que tem a missão de conectar dispositivos multimídia de veículos entre si. O modelo abrange até 64 dispositivos para receber a configuração padrão para não precisar configurar o sistema ao conectar um novo dispositivo, herdado da tecnologia *plug and play*, mas caso seja necessário mais dispositivos podem ser conectados e configurados (MOST, 2019).

Desenvolvida sobre os procedimentos CAN e MOST, em uma tentativa de unir preço e performance, o barramento de comunicações *FlexRay* apresenta requisitos de redundância e tolerância de falhas. O protocolo fornece *triggers* (ativadores) a partir do *clock* e/ou eventos, e pode trabalhar de forma híbrida, com dados estáticos e eventos dinâmicos do tipo CAN (BABICH; DEOTTO, 2002).

4.2 TESTES EM SISTEMAS AUTOMOTIVOS

Segundo os autores Schauffele e Zurakawa (2005), sob o ponto de vista de gestão, recomenda-se que o responsável pelo trabalho de teste escolha as ferramentas, elabore um plano e projete uma solução que resulte no melhor equilíbrio entre requisitos, ambiente de desenvolvimento, prazo e custo. O processo de qualidade de software tem dois objetivos: verificar (garantir que a aplicação está agindo de acordo com o solicitado) e validar o software (certificar-se que o sistema entregará os resultados da forma correta).

Algumas definições dos termos utilizados no processo de teste em relação ao software são (SCHAUFFELE; ZURAKAWA, 2005):

- Exatidão: consistência em relação ao que foi solicitado;
- Confiabilidade: estatisticamente, apresenta falhas dentro de um valor máximo de erro aceitável;
- Segurança: não causa danos físicos às pessoas;

- Robustez: possui um comportamento aceitável dentro de um ambiente competitivo.

Tratando-se da parte elétrica do *software* de sistema automotivo, estes são divididos de acordo com os componentes a serem testados no veículo. Como por exemplo (SCHAUFFELE; ZURAKAWA, 2005):

- Sistemas elétricos, como vidros ou freios ABS;
- Testadores de bateria e sistema de carregamento;
- Ferramentas de análise de temperatura e controle de ar;
- Carregadores de bateria;
- Osciloscópios / visualizadores de forma de onda;
- Ferramentas de escaneamento;
- Testadores de circuito;
- Testadores especializados.

Dado a diversidade e a abrangência de requisitos que as tarefas oferecem, existem ferramentas específicas para realizar os testes e são descritos na próxima subseção.

4.2.1 Ferramentas

Os sistemas ciber-físico de linguagem de ação (CPAL) servem para descrever o comportamento funcional das tarefas, ou seja, seu código, e a arquitetura funcional do sistema – o conjunto de funções, como elas são acionadas e o fluxo de dados entre os procedimentos (SANTOS *et al*., 2015).

Lee (2008) afirma que sistemas ciber-físico (CPS) são integrações de computação e processos físicos. Esses sistemas implementam, modelam, verificam e simulam computadores enquanto são monitorados e controlados os processos físicos. Tal tecnologia tem um potencial econômico e social grande, a nova geração dos sistemas embarcados são utilizados na área automotiva

O trabalho de Michailidis (2010) exemplifica uma bateria de testes que podem ser executados no início do desenvolvimento de sistemas baseadas em AUTOSAR (*AUTomotive Open System ARchitecture*). Seus testes fundamentam-se no conceito de monitoramento do tempo que uma tarefa solicitada pelo controlador principal levará a até ser de fato processada. Devido aos conflitos de prioridade,

acesso aos recursos podem ocorrer de forma simultânea. Assim, uma pré-avaliação do sistema é efetivada com está ferramenta.

Roquette (2018) apresenta em seu trabalho uma abordagem de casos de teste utilizando do *Behavior Driven Development* (BDD) – uma técnica de teste proposta na computação pela área de engenharia de software. Segundo o autor, empresas automotivas adotam uma metodologia ágil que possibilita a implementação do software simultaneamente com a realização de teste, que são feitos de forma manual, empiricamente, o que pode resultar na não identificação de todos os cenários que uma aplicação pode enfrentar.

4.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

A pesquisa bibliográfica apresenta que, para testes de software em geral, os estudos e ferramentas são aplicados em sistemas automotivos, porém estabelecer um método completo e eficiente de verificação e validação torna-se um trabalho árduo visto que a manutenibilidade de uma aplicação com características RTS são complexos (SCHAUFFELE; ZURAKAWA, 2005).

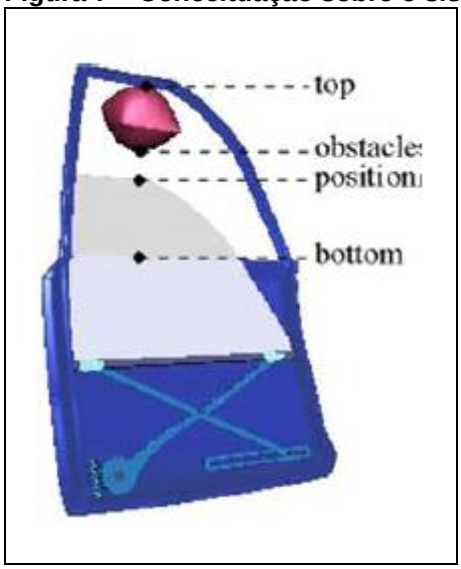
É com este objetivo que surgiu a motivação para o tema deste trabalho. A contribuição de *model checking* em sistemas automotivos traz a eliminação das falhas computacionais desejadas, além de testar o modelo de forma automática, sendo a modelagem o principal foco do desenvolvedor, uma opção de baixo custo e alta eficiência para o mercado.

5 APLICAÇÃO DA VERIFICAÇÃO FORMAL EM SISTEMAS AUTOMOTIVOS

A verificação formal proposta foi aplicada sobre um modelo elétrico de janelas de um automóvel. O protótipo foi elaborado pela Universidade Tecnológica Federal do Paraná (Ponta Grossa), Faculdade de Tecnologia de Curitiba (Santo André), Instituto Nacional de Telecomunicações (Santa Rita do Sapucaí) e Universidade de São Paulo (São Paulo) (SANTOS *et al*., 2015).

O protótipo diz respeito sobre um passageiro e um motorista que podem acionar mecanismos que movimentam para cima ou para baixo suas respectivas janelas e o motorista pode operar e tem prioridade sobre o vidro do passageiro. O projeto inclui um alarme, que deve fechar completamente o veículo quando acionado e dois obstáculos que podem ocasionalmente ser detectados em algumas das janelas por meio de sensores na operação de subida de algum dos vidros. A obstrução em questão deve ser atendida com prioridade máxima pelo sistema visto que a obstrução pode se tratar de um membro humano e ser um fator de risco à condição humana. A figura 7 ilustra o sistema de janelas.

Figura 7 – Conceituação sobre o sistema de janelas



Fonte: Santos *et al* (2015)

Dado o cenário do modelo, pode-se dizer que o sistema de gerenciamento de janelas se enquadra como um sistema de tempo real.

A seção 5.1 expõe os cenários de teste do protótipo desenvolvido por Santos *et al*. (2015). A seção 5.2 apresenta a modelagem do sistema elétrico de

janelas automotivas, desenvolvidas a partir do protótipo de Santos *et al*. (2015). A seção 5.3 descreve a simulação do modelo da seção 5.2. A última seção apresenta as considerações finais do capítulo.

5.1 CENÁRIO DE TESTE

O sistema de vidros elétricos é um subsistema de veículo automotor que infere em um conjunto de componentes mecânicos para gerenciar as tarefas das janelas. Existe uma prioridade sobre as interrupções do motorista sobre o passageiro, do acionamento do alarme sobre os autômatos e, por fim, do obstáculo sobre o todo.

A primeira etapa do processo é definir as tarefas a serem cumpridas pelo sistema. Os requisitos que devem ser satisfeitos pelo modelo são (SANTOS *et al*., 2015):

- REQ-01: o vidro deve ser totalmente aberto ou fechado em 4s.
- REQ-02: o vidro começa a se mover 200ms depois que o comando é selecionado.
- REQ-03: Após 4 segundos de movimento ininterrupto em uma direção, o motor deve desligar.
- REQ-04: A operação de tensão do sistema de controle está entre 12,5V e 14,5V.
- REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado entre 200ms e 1s, o vidro abre ou fecha completamente.
- REQ-06: Se um objeto obstrutivo for detectado, o vidro deverá descer aproximadamente 10cm.
- REQ-07: A detecção de obstáculos tem prioridade sobre os comandos do passageiro e motorista.
- REQ-08: Os comandos do interruptor do motorista têm prioridade sobre os do passageiro.
- REQ-09: Quando acionado, o sistema de segurança deve fechar completamente, a menos que haja um obstáculo.
- REQ-10: O sistema de controle opera apenas com a posição de ignição em "ligado".

Compreendendo os objetivos a serem atingidos, a próxima etapa se entende como a modelagem desses fatores e dos autômatos envolvidos. Este trabalho não abrangerá os requisitos que envolvem energização de placas (REQ-04 e REQ-10) e os que envolvem tempo (REQ-02), visto que para uma aplicação real há necessidade de inserir valores realistas para as variáveis do tipo *clock*.

O REQ-01 e o REQ-03, por apresentarem semelhança, foram tratados com a monitoração e controle de duas variáveis locais em cada janela denominada “altura”, visto que o funcionamento do vidro do motorista é independente da do passageiro, podendo ocorrer a movimentação simultânea de ambas as janelas.

5.2 MODELAGEM DO MODELO ELÉTRICO DA JANELA

Na ferramenta UPPAAL, a aba de edição do sistema é usada para criar e projetar o sistema a ser analisado. O modelo tem o foco de atender aos requisitos especificados na subseção anterior. Para isso, são feitas declarações de variáveis globais, modelos são projetados (com variáveis globais), canais de sincronização são criados para que haja comunicação entre os incorporados no sistema.

Além dos tipos conhecidos de variáveis, no software existem variáveis do tipo *chan*, são os canais de comunicação, para o sistema foram necessários três tipos: canal padrão – envia e recebe uma única mensagem por vez, canal *broadcast* – envia a mensagem para todos os participantes do grupo – e canal urgente – faz o relógio interno do sistema parar e envia seu sinal, usado em estados críticos e em situações de risco. Na figura 8 estão descritos as variáveis globais e os canais da aplicação.

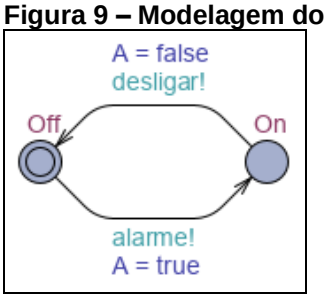
O REQ-09 traz dois aspectos para o sistema, a prioridade do alarme sobre todas as outras operações e o fator externo dos obstáculos, que pode ocorrer em ambos os vidros, o que há uma prioridade maior, pois o obstáculo em questão pode ser um ser humano. Na figura 9 pode-se analisar a modelagem no alarme de segurança enquanto nas figuras 10 e 11 os obstáculos são exibidos, que podem estar ativos ou inativos.

Figura 8 – Declarações de variáveis globais e canais de sincronização do sistema de janelas

```
bool U = false;                    //      movimentos motorista On e Off
bool E = false;                    //      movimentos motorista sobre passageiro On e Off
bool I = false;                    //      movimentos passageiro On e Off
bool A = false;                    //      alarme On ou Off
bool O = false;                    //      obstrucao On ou Off

chan cont, conti;                  //      sinal para continuar movimento
chan subir, descer;                //      canais passageiro
chan sub, des;                     //      canais motorista (sobre si)
chan subirM, descerM;              //      canais motorista sobre passeiro

broadcast chan alarme, desligar;//      sinal do alarme

urgent chan obsE;                  //      obj janela Esquerda
urgent chan obsD;                  //      obj janela Direita

//chan priority cont, subir, descer, sub, des < conti, subirM, descerM < alarme, desligar < obsE, obsD;
```

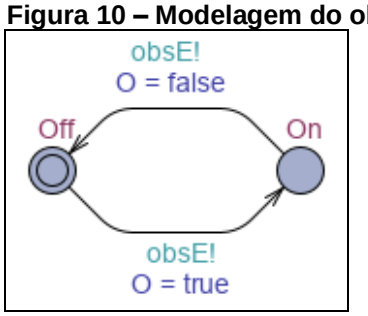
Fonte: Autoria própria

Figura 9 – Modelagem do sistema de segurança



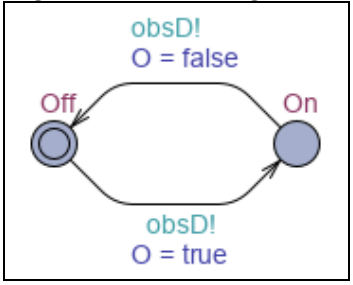
Fonte: Autoria própria

Figura 10 – Modelagem do obstáculo da janela esquerda



Fonte: Autoria própria

Figura 11 – Modelagem do obstáculo da janela direita



Fonte: Autoria própria

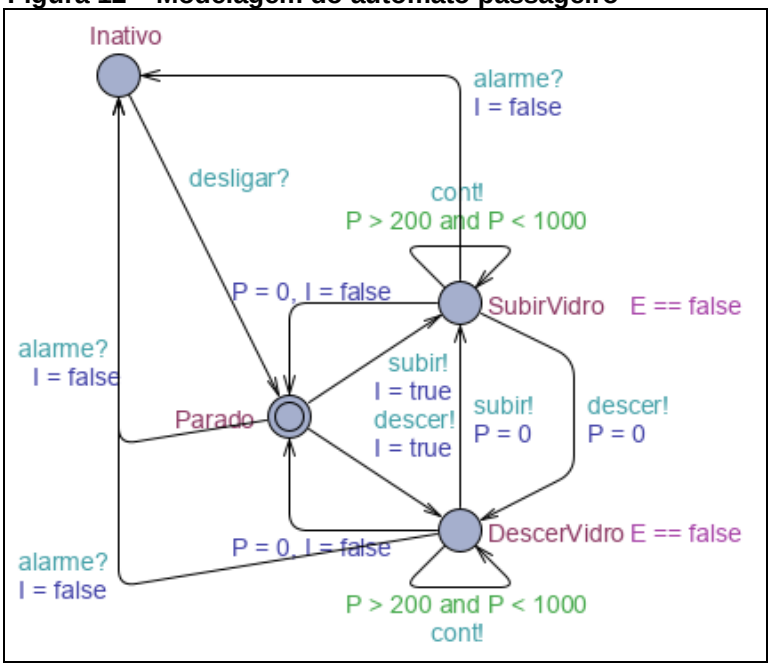
A figura 8 mostra as declarações globais do modelo, as primeiras cinco linhas tratam de variáveis booleanas. Estas variáveis dizem ao modelo quando determinado ator está realizando uma ação. A variável “I” representa as ações do passageiro sobre sua respectiva janela, quando com valor verdade. A *boolean* “U” assume valor verdade quando o motorista está realizando as ações em sua janela (lado esquerdo) enquanto que a *boolean* “E” assume valor verdade quando motorista exerce ações sobre a janela do passageiro (lado direito). Por sua vez a variável “A” representa o sistema de segurança ativo do veículo e “O” diz respeito de alguma das obstruções, de ambos os lados, serem ativada.

O sistema exige a modelagem dos autômatos passageiro e motorista que são os atores que devem manipular as operações sobre as janelas elétricas. Na figura 12, pode-se observar a modelagem do passageiro, o autômato inicia no estado “Parado” o passageiro pode realizar as ações de elevar ou abaixar o vidro, se ele permanecer no estado “SubirVidro” ou “DescerVidro” por P > 200 e P < 1000, onde P é uma variável local de *clock*, então o modelo enviará um sinal por meio do canal *cont* para o sistema de janela do passageiro, que fará com que o vidro continue determinando o movimento até fechar ou abrir completamente. Todos os três estados anteriores possuem um canal que leva o protótipo para “Inativo” quando o alarme for acionado.

O autômato motorista, exposto na figura 13, pode operar sua janela da mesma forma como descrita com o autômato passageiro inclusive com as mesmas condições. O motorista pode exercer operações sobre o vidro do passageiro e tem prioridade sobre o mesmo. Foram criados dois estados no autômato do motorista se comparado ao do passageiro: “SubirVidroPassageiro” e “DescerVidroPassageiro”, possibilitando ao motorista controle total sobre as janelas.

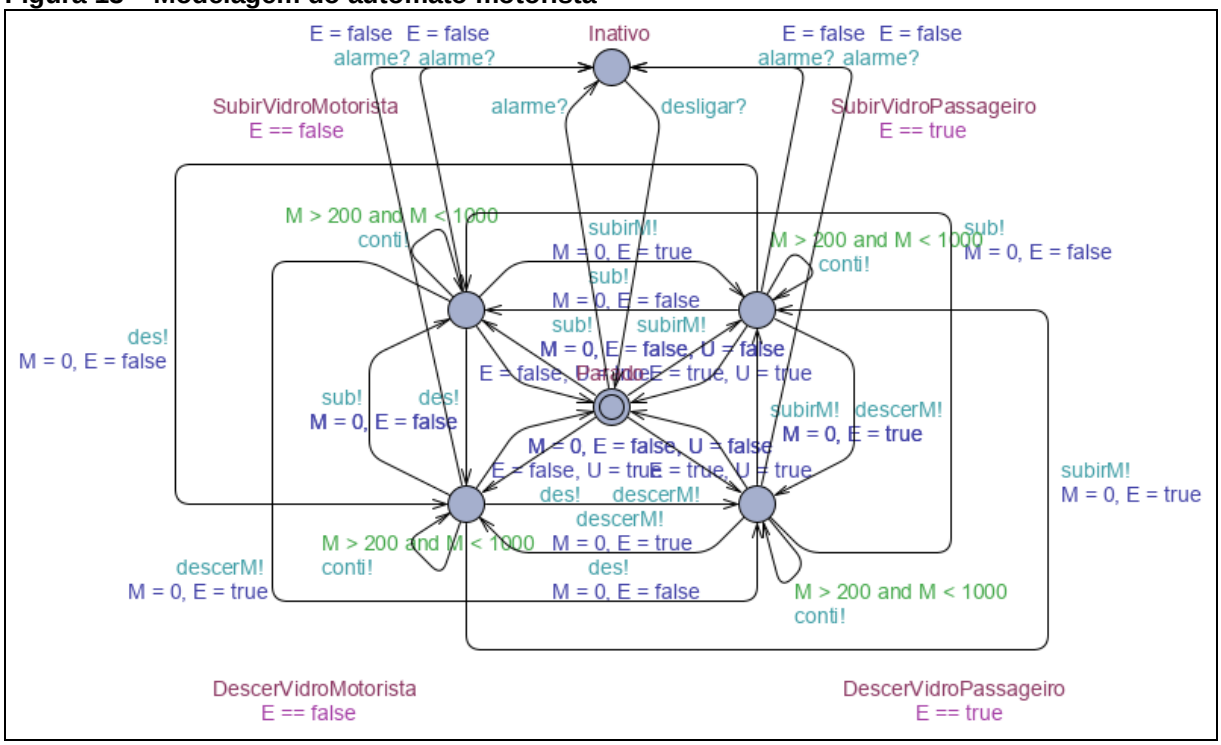
Quando o motorista opera ações em sua janela, a do lado esquerdo, a variável “U” recebe valor “true” e a “E” recebe “false”, enquanto que, quando o motorista executa procedimentos na janela do passageiro, lado direito, a variável “E” recebe valor “true” e a “U” recebe “false”. Deste modo, juntamente com o uso de invariantes nos estados de “SubirVidro” e “DescerVidro” do autômato passageiro, exposta na figura 12, o sistema gerencia a prioridade do motorista sobre o passageiro e impede o passageiro de permanecer nos estados citados quando o motorista está operando o vidro do lado direito, por conter a invariante “E==false”, obrigando o autômato transitar destes estados quando “E” assumir valor “true”.

Figura 12 – Modelagem do autômato passageiro



Fonte: Autoria própria

Figura 13 – Modelagem do autômato motorista



Fonte: Autoria própria

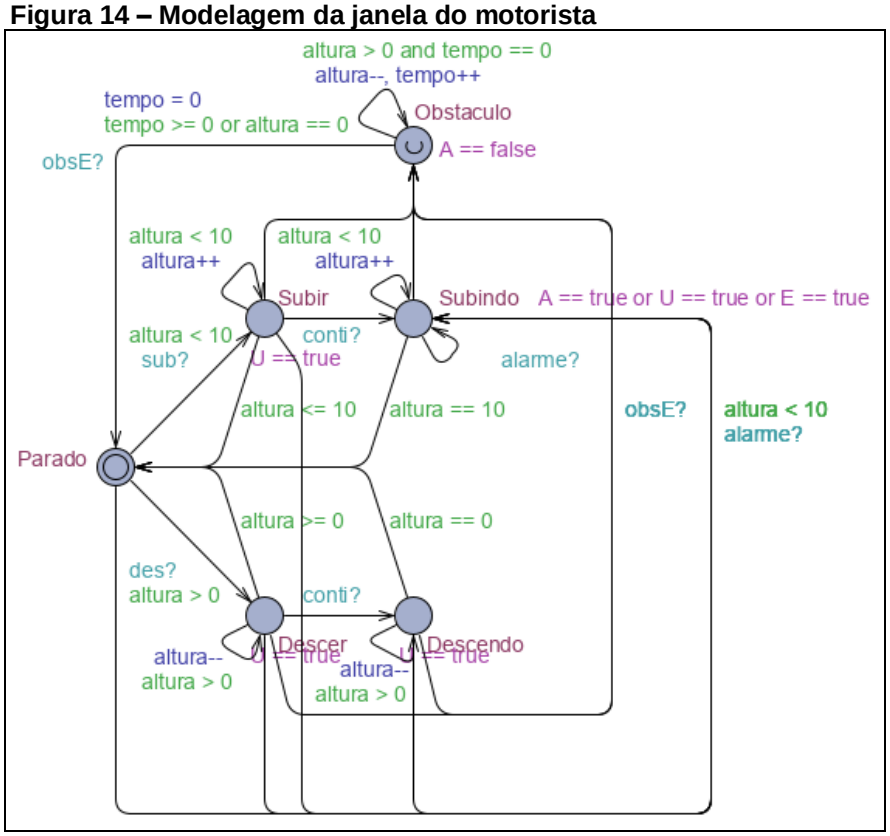
Para representar a realidade, foram criados dois modelos para as janelas elétricas, um para o vidro esquerdo do motorista, figura 14, e outro para o vidro direito do passageiro, figura 15. Desta forma, pode-se analisar os estados de cada

janela de forma independente uma da outra. Há guardas, que são condições lógicas a serem testadas antes de ocorrer a transição, em quase todas as arestas para limitar interações entre estados de acordo com a altura dos vidros. A figura 16 mostra as declarações das variáveis locais, o *clock* e a altura, iguais para ambos os vidros.

A modelagem da janela do motorista apresenta o estado inicial “Parado”, há um canal de sincronização entre o autômato motorista e a janela direita da figura 14, o canal “sub?” aguarda receber estímulo para levar o modelo do estado inicial para o estado “Subir”, existe um guarda que impede essa transição caso o vidro já esteja na sua altura máxima.

Na apresentação dos autômatos, nota-se que a representação gráfica de seus estados é semelhante, exceto pelos casos do estado inicial, sendo este o estado com um círculo interno adicional na sua representação gráfica, e a do estado urgente, exibindo um símbolo em forma de “u” na interface. Tanto na imagem 14 quanto na imagem 15 pode-se notar os estados “Obstaculo” atribuído como um estado urgente em ambos os modelos.

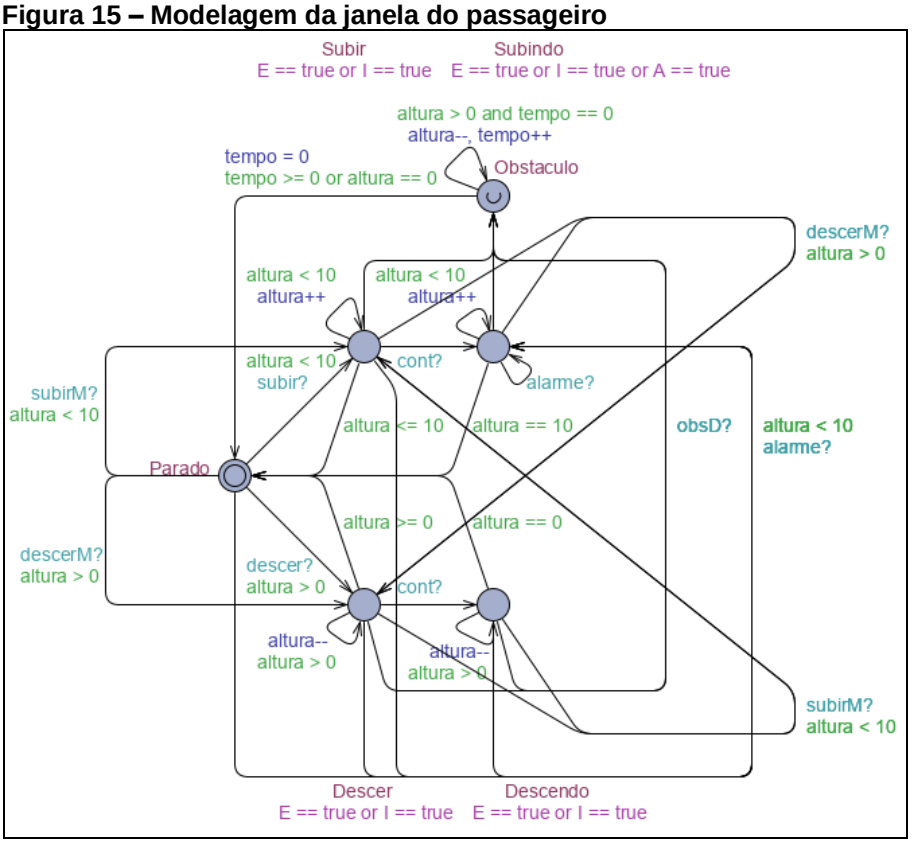
Figura 14 – Modelagem da janela do motorista



Fonte: Autoria própria

No estado “Subir” há um *loop* que incrementa a variável local “altura” que, por sua vez, diz respeito ao valor da elevação do vidro, se o autômato motorista da figura 13 permanecer no estado “SubirVidroMotorista” por M > 200 e M < 1000, onde *M* é o *clock* local, então um sinal será enviado por meio do canal “conti!”, que será interpretado pelo sistema de janela do motorista e leva-o ao estado “Subindo”, onde ira permanecer em *loop* até a variável “altura” atingir seu valor máximo e retornar ao estado “Parado”. O mesmo acontece com o movimento de descer o vidro, porém por meio dos canais “des” e “conti”.

Figura 15 – Modelagem da janela do passageiro



Fonte: Autoria própria

Figura 16 – Declarações das variáveis locais utilizadas nas janelas

```
clock V;

int [0,10] altura = 10; //altura do vidro
int [0, 1] tempo = 0;   //para controlar altura de elevação na obstrução
```

Fonte: Autoria própria

A figura 15 é o modelo da janela do passageiro, semelhante ao modelo da figura 14, sua diferença são as transições do estado “Parado” para os estados “Subir” e “Descer” por meio do respectivo canal “subirM” e “descerM”. O modelo da

figura 15 também possui arestas dos estados “Subir” e “Subindo” para “Descer” e dos estados “Descer” e “Descendo” para “Subir”. Elas foram criadas para que haja a interação do auômato motorista sobre a janela do passageiro.

A prioridade do motorista foi tratado com váriáveis globais, a booleana “U” para o motorista, “I” para o passageiro e “E” para a prioridade do motorista sobre o passageiro. Elas assumem o valor “true” quando seus respectivos autômatos estão realizando alguns dos movimentos de suas janelas e assumem valor “false” enquanto os autômatos permanecem no estado “Parado”.

Na modelagem do autômato passageiro (figura 12), observa-se a invariante “E== false” nos estados “Subir” e “Descer”. Invariantes são condições rotuladas em estados, os quais violam invariantes e saltam para o próximo estado possível (DAVID *et al*., 2015). O autômato motorista (figura 13) altera o valor da *boolean* “E” sempre que permanecer nos estados que controlam a janela do passageiro, tornando dessa forma o passageiro inoperante sobre o sistema. Já a *boolean* “U” assume valor “true” quando o motorista está realizando alguma ação, o mesmo vale para a *boolean* “I” e o autômato passageiro.

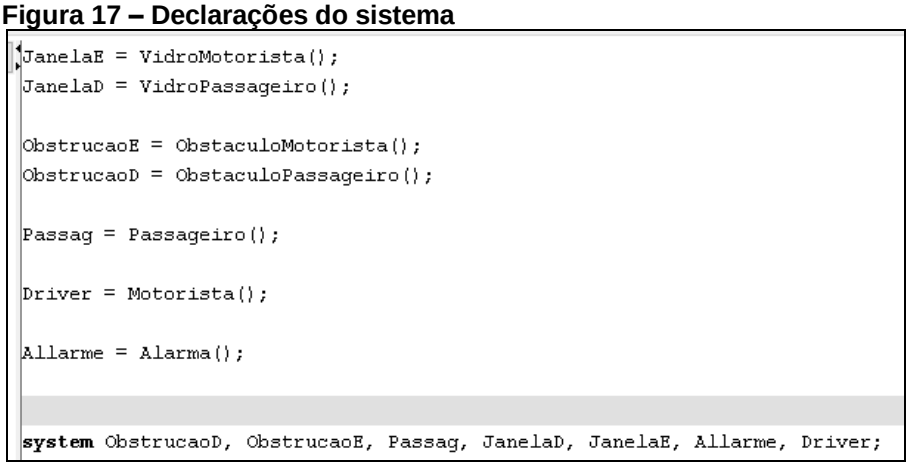
Em ambas as modelagem das janelas elétricas (figura 14 e 15), em todos os estados pode-se observar arestas que levam o modelo ao estado “Subindo”, essa aresta é percorrida com a ativação do sistema de segurança do sistema (figura 9). A ativação do alarme altera o valor da variável boolean “A” para “true” e envia uma mensagem para o canal *broadcast*, sendo atendida por todos os sistemas ouvintes. Há um laço no estado “Subindo" que verifica o estado da variável “A”.

Para garantir a prioridade absoluta sobre o risco que obstáculos oferecem aos usuários do veículo automotor, nas modelagens dos vidros elétricos, foi desenvolvido o estado “Obstaculo” definido como um estado urgente. Os sistemas seguem para esse estado toda vez que os canais urgentes “obsE” e “obsD” enviam um sinal. Estados urgentes param o tempo. Canais urgentes têm a característica de não ser possível atrasar no estado de origem se houver uma aresta de sincronização com um canal urgente. Vale ressaltar que guardas sobre relógios não são permitidos nas transições urgentes (DAVID *et al*., 2015).

Por fim, para a etapa de simulação, descrita na seção 5.3, foi necessário na modelagem injetar códigos no sistema para a ferramenta instanciar os modelos e compreender o protótipo como um todo, exposto na figura 17. É possível, por exemplo, gerar várias instâncias de obstáculos que podem ocorrer em alguns dos

lados da janela, desde que a modelagem e as declações dos sistemas estejam corretamente parametizadas.

Figura 17 – Declarações do sistema



Fonte: Autoria própria

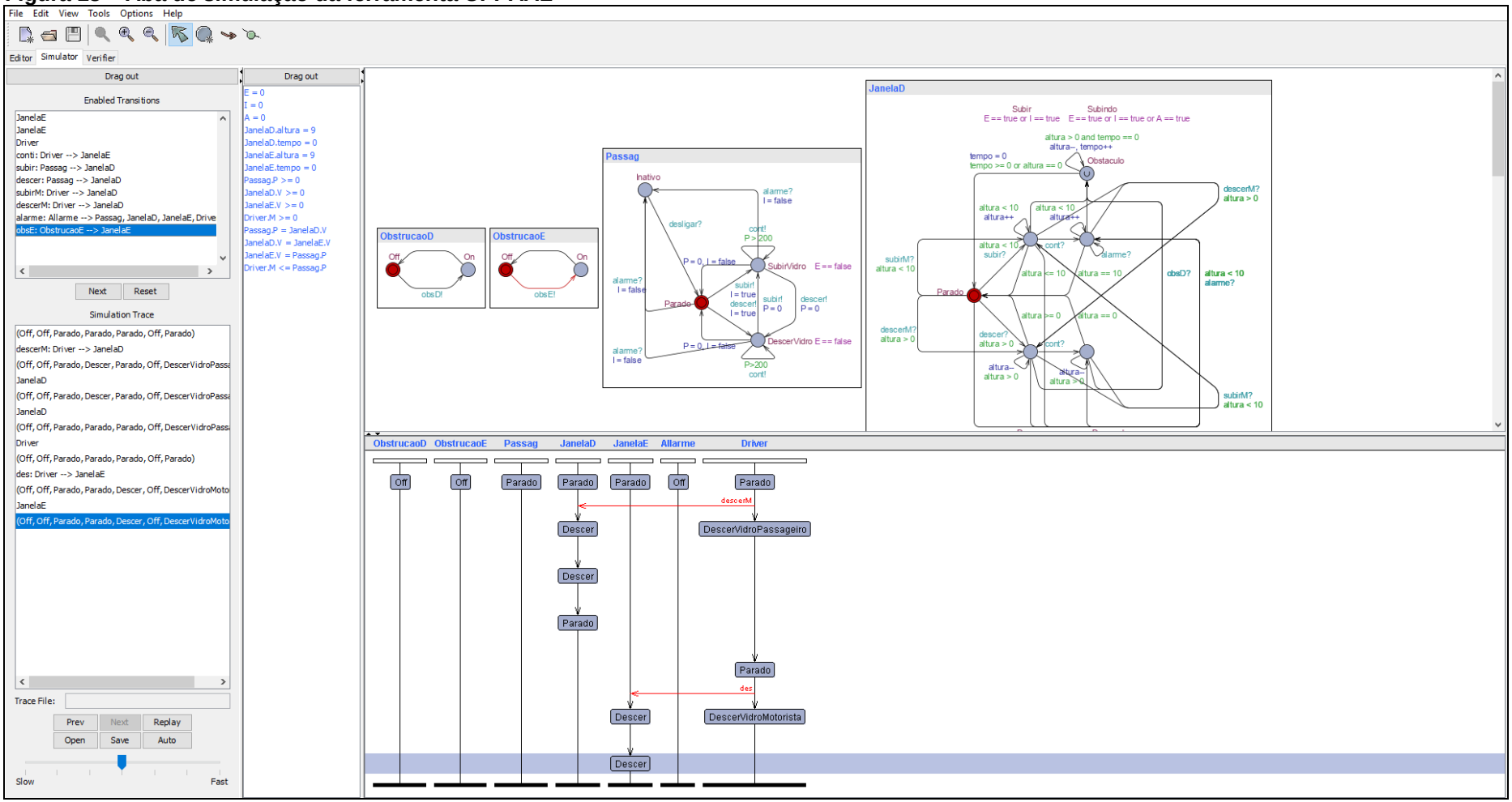
O arquivo com a modelagem descrita encontra-se no apêndice A deste documento e disponível para download no repositório do *GitHub*: <https://github.com/renan279/Eletric-Window-Automotive-System>.

5.3 SIMULAÇÃO DO MODELO PROPOSTO

A simulação é a fase que permite examinar as possíveis execuções dinâmicas de um sistema durante o estágio da modelagem. Nesse aspecto, o software fornece um meio barato de detecção de falhas antes da validação pela ferramenta de verificação formal.

A simulação consiste em quatro painéis, como pode-se observar na figura 18 em que na esquerda está o controle de simulação. As variáveis declaradas ao centro, no canto direito superior encontram-se os processos em simulação e o gráfico de sequência de mensagens no canto inferior direito.

Figura 18 – Aba de simulação da ferramenta UPPAAL



Fonte: Autoria própria

Para assegurar que o desenvolvimento do sistema de gerenciamento de janelas foi assinado de acordo com as especificações do cenário de teste, durante a própria etapa de simulação, um conjunto de testes foi elaborado no intuito de auxiliar no processo como um todo, alternando sempre as fases de simulação e modelagem. Os testes assinados foram:

- Passageiro: representando o gatilho de movimentação para cima ou para baixo da janela do lado direito.
- Motorista: representando os disparos em movimentar para cima e para baixo a janela do lado esquerdo como a do lado direito, além da prioridade do motorista sobre o passageiro.
- Limites superiores e inferiores da janela: limite máximo da variável altura definida com valor inteiro 10 e limite mínimo com valor 0.
- Obstáculo: detecção da obstrução por ambas as janelas.
- Alarme: *status* representa a segurança do sistema, movimentando as janelas para cima, porém, o movimento deve ser impedido no caso de uma obstrução.

Inicialmente, não foram atendidos todos os requisitos, justificando a declaração das variáveis *booleanas* declaradas na figura 8. Uma vez que as assinaturas urgentes foram satisfeitas, pode-se saltar para a terceira etapa, com as verificações por meio da lógica temporal.

5.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Com a criação do cenário do sistema de gerenciamento de janelas na ferramenta de verificação formal, pode-se observar que a fase de edição e simulação ocorrem lado a lado, como mostra a figura 2 da seção 3.3.3. Neste caso, é necessário alternar entre as duas etapas diversas vezes para atender aos requisitos da aplicação, ocupando o maior tempo de todo o processo. Assim, que a compreensão do propósito for definida e modelada de acordo, mesmo numa situação prematura, a ferramenta consegue apontar cenários de risco e falhas não obvias, o que foi determinante para criação de variáveis e estados.

Com uma compreensão da problemática exposta pela prioridade entre os autômatos, estados previamente não planejados foram desenvolvidos para atender quesitos como ao que o obstáculo oferece aos usuários do veículo na elevação dos vidros, por exemplo, observando-se que tal obstrução pode ser um objeto que ocasionaria em dano material ou um membro humano que ocasionaria em um dano físico, assim, um atraso ou não cumprimento da tarefa compromete o sistema como um todo.

Nesse aspecto, observa-se as especificações sobre situações críticas que a ferramenta oferece ao desenvolvedor, como no caso das obstruções do parágrafo anterior, a ferramenta traz soluções simplista e eficiente para atender cenários críticos com diferentes características.

6 RESULTADOS

Conforme a modelagem e simulação das etapas anteriores, pode-se aplicar a verificação automática no modelo definidas em forma de lógica temporal.

É importante ressaltar que para satisfazer um requisito foram necessárias algumas linhas de lógica temporal em alguns casos, os quais são detalhados nas seções deste capítulo.

A seção 6.1 conceitua a “verificação” da interface da ferramenta de *model checking* UPPAAL. A seção 6.2 apresenta os casos de teste para o sistema modelado com os requisitos exigidos. A seção 6.3 realiza uma comparação entre trabalhos relacionados com a modelagem desenvolvida neste trabalho. Por fim, a última seção relata as considerações finais deste capítulo.

6.1 VERIFICAÇÃO DE CONSULTAS

As consultas do sistema são verificadas na aba “verificação” na interface da ferramenta de verificação formal UPPAAL. Propriedades são editadas e verificadas quando o botão “checar” é pressionado. Na interface do sistema um círculo verde é dado aos resultados com valor verdade verdadeiro e um vermelho aos com valor verdade falso.

A linguagem de especificação de requisitos do UPPAAL suporta cinco tipos de propriedades, nelas existem algumas equivalências, o quadro 2 exibe essas propriedades. Os operadores lógicos utilizados pela ferramenta na etapa de verificação estão descritos da no capítulo 3 deste trabalho, na subseção 3.3.

Quadro 2 – Equivalência de propriedades na aba de verificação do software UPPAAL

Nome	Propriedade	Equivalência
Possivelmente	E < > P	
Alcançabilidade	A [ ] P	não E < > não P
Potencialmente sempre	E [ ] P	
Eventualmente	A < > P	não E [ ] não P
Leva a	P --> Q	A [ ] (P implica A < > Q)

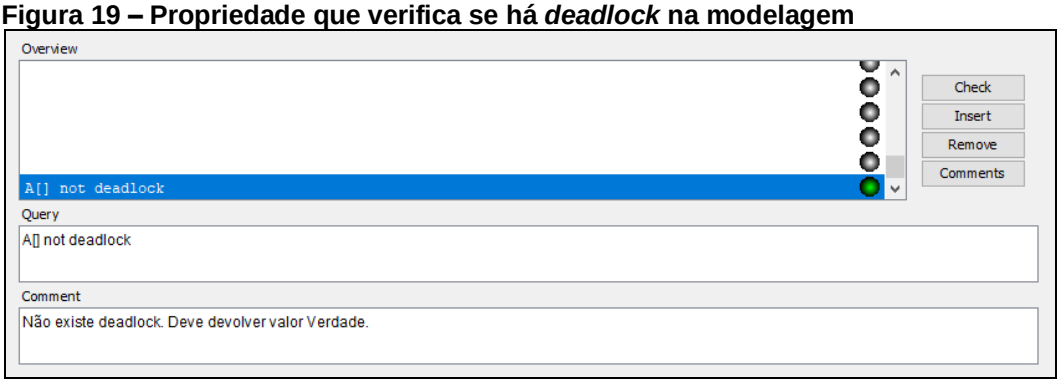
Fonte: Adaptado de UPPAAL (2015)

As propriedades de segurança são exploradas por meio de restrições e em termos de estados simbólicos representados pelos autômatos que compreende o sistema.

6.2 ANÁLISE DOS TESTES

O primeiro teste realizado verifica uma propriedade trivial, porém necessária em sistemas de tempo real, a ausência de *deadlock*. A figura 19 exibe esse teste na ferramenta UPPAAL.

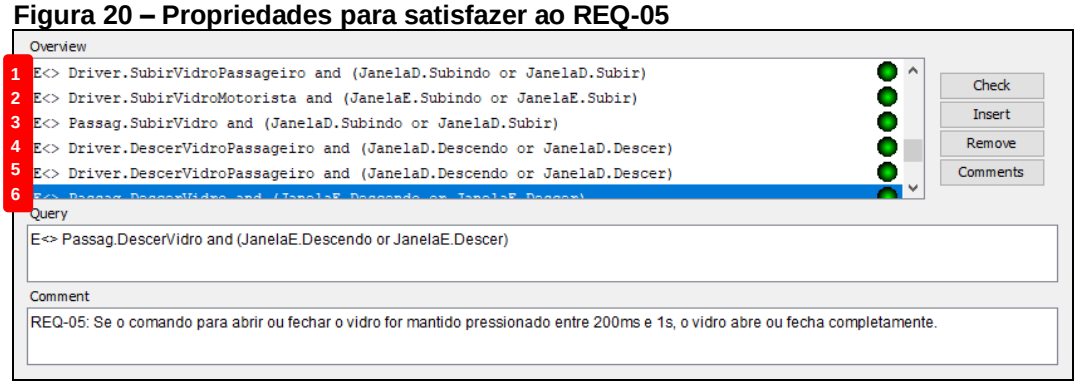
Figura 19 – Propriedade que verifica se há *deadlock* na modelagem



Fonte: Autoria própria

A figura 20 exibe seis preposições, suficientes para atingir o quinto requisito especificado, observa-se nas três primeiras linhas as propriedades de que existe um momento, atual ou futuro, em que o autômato motorista está pressionando a alavanca para elevar o vidro do lado do passageiro e a janela estará de fato nessa condição. Na segunda linha testa-se a mesma ação do autômato motorista sobre a janela do lado esquerdo. A terceira linha repete o teste envolvendo o passageiro e o vidro direito. Ao três últimas linhas apresentam os mesmos cenários, porém, com os autômatos pressionando a alavanca para abaixar as janelas.

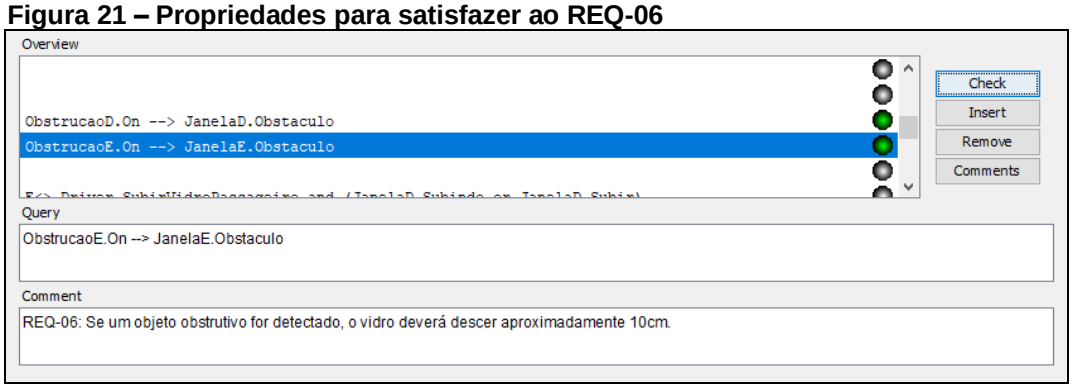
Figura 20 – Propriedades para satisfazer ao REQ-05



Fonte: Autoria própria

O sexto requisito relata a ênfase do sistema sobre o risco que a elevação das janelas oferece aos usuários do automotor, em cenários onde há um obstáculo ativo em algum dos vidros do veículo. Para assegurar tal situação, duas proposições de implicação, expostas na figura 21, inferem que: se há uma obstrução do lado esquerdo o vidro do motorista irá transitar para o estado urgente do modelo, impedindo a elevação da janela; o segundo teste verifica se há um obstáculo do lado direito, se houver, o vidro do passageiro salta para um estado urgente com as mesma características do teste anterior.

Figura 21 – Propriedades para satisfazer ao REQ-06



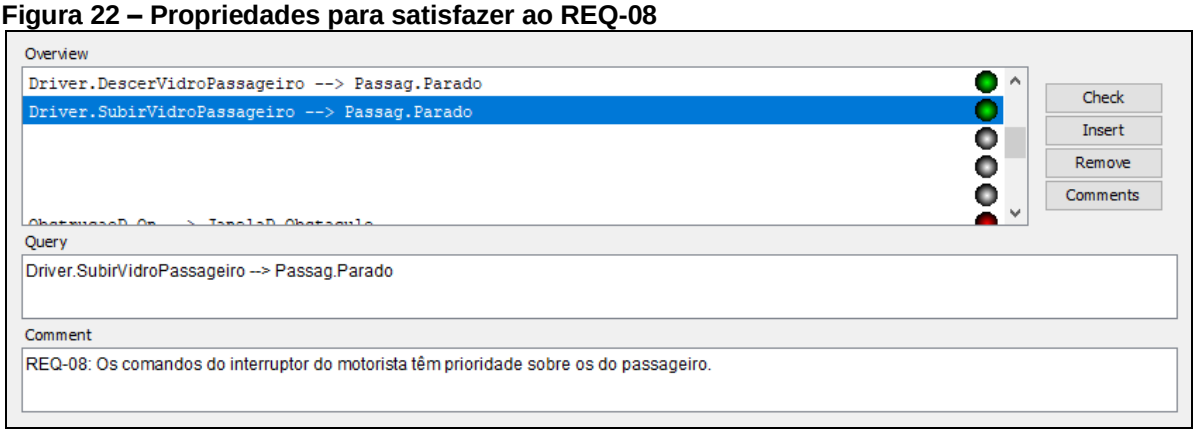
Fonte: Autoria própria

O REQ-07 diz respeito à obstrução ter prioridade sobre as ações do autômato motorista e do autômato passageiro, todavia, a prioridade da obstrução foi tratada com máxima urgência pois o cenário coloca em risco o ser humano, parando o relógio interno no sistema e executando os procedimentos necessários. Logo, está implícito a satisfação deste requisito.

A figura 22 verifica com implicações a prioridade do autômato motorista nas operações da janela direita sobre o autômato passageiro. O primeiro teste reflete

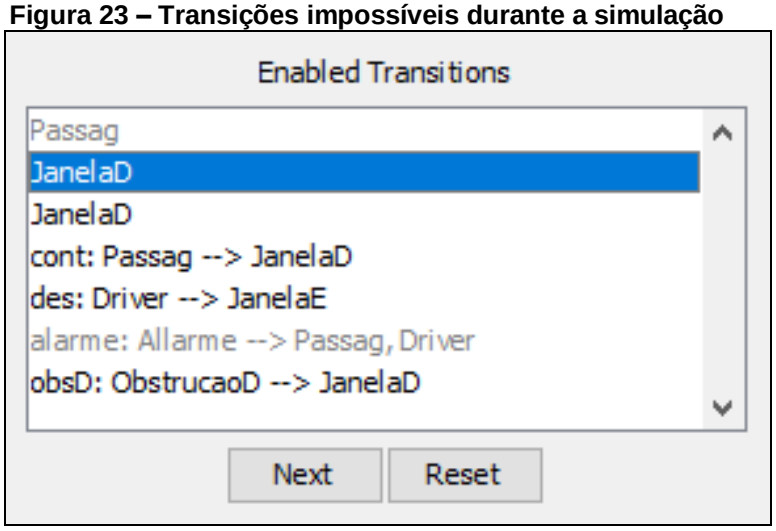
que quando o motorista movimentar o vidro para cima, o autômato passageiro salta para o estado “Parado”. O segundo teste insinua a mesma transição quando o motorista pressiona o movimento para abaixar o vidro passageiro. Essa situação ecoava durante as etapas de modelagem e simulação do sistema, como mostra a figura 23, quando o autômato principal estava realizando operação, tornava-se impossível o segundo autômato iniciar ações, representado em tom cinza claro.

Figura 22 – Propriedades para satisfazer ao REQ-08



Fonte: Autoria própria

Figura 23 – Transições impossíveis durante a simulação

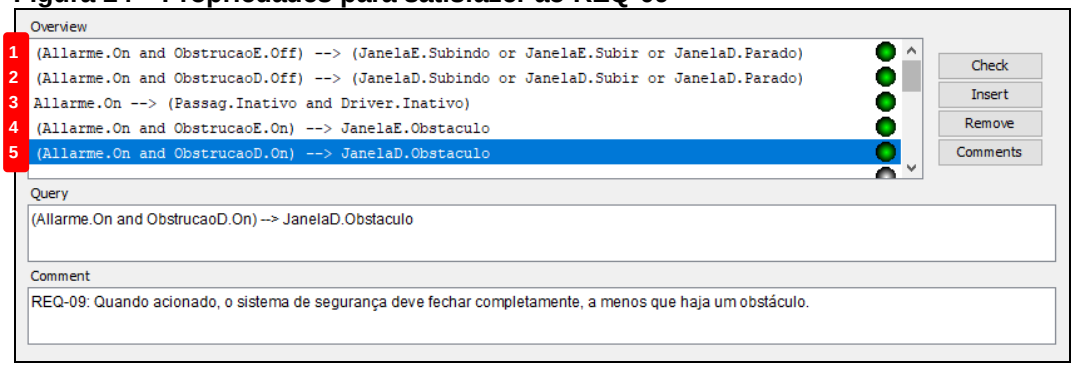


Fonte: Autoria própria

As últimas verificações apresentadas na figura 24 consideram o caso de que haja um obstáculo enquanto o sistema de segurança estiver ativo, que envolve a elevação de ambos os vidros, as janelas não executarão o comando. As duas primeiras linhas mostram que, se não houver obstáculos no trajeto das janelas, elas irão incrementar sua altura até o valor máximo. A terceira linha implica que, caso o sistema de segurança seja ligado, os usuários do veículo tornam-se inoperantes

sobre o automotor, muito semelhante ao REQ-10. As próximas linhas manifestam implicações de que, se o alarme estiver ativo e há obstáculos em algum dos lados, então, o sistema de janela saltará para o estado que atende ao risco que a obstrução oferece.

Figura 24 – Propriedades para satisfazer ao REQ-09



Fonte: Autoria própria

Vale ressaltar que, para validação, houve uma espera de alguns poucos minutos para conclusão de todos os testes, visto que a ferramenta realiza uma explosão de estados para verificar todos os caminhos possíveis que o modelo pode gerar. O quadro 3 mostra os requisitos do computador no qual foram executados os testes.

Quadro 3 – Requisitos do computador operado para testes

Processador Intel I7 de 3ª geração 3.3 GHz
Placa de Vídeo de 2 GB DDR 3 1600 MHz
Memória RAM de 4 GB DDR3 de 1600 MHz
Memória física de 500 GB SATA (taxa de transferência de até 140MB/s)
Placa de Rede Wireless PCI de 150 MB/s

Fonte: Autoria própria

O quadro 4 apresenta os resultados dos testes aplicados sobre a modelagem desenvolvida na ferramenta de verificação formal. O ambiente gráfico do software exibe um sinal verde quando a propriedade expressa valor lógico verdade, vermelho quando expressa valor lógico falso. Contudo, exibir um valor lógico verdade nem sempre é o desejado, isso depende de como as propriedades foram desenvolvidas para interpretação. A última linha da tabela exibe a propriedade

verificada na figura 19: *A[ ] not deadlock* – para todos os caminhos possível, não haja deadlock; o desejado neste caso é o sinal verde como o exibido pela ferramenta, porém nota-se que, se a propriedade não estivesse em negação, o valor lógico desejado seria o falso.

Conforme apresenta o quadro 4, o REQ-01 e o REQ-03 foram parcialmente atendidos, pois estes foram assegurados com a declaração de duas varáveis locais independentes denominadas “altura”, que monitoram e controlam o valor de elevação das janelas. De tal forma, o controle e a implementação na ferramenta tornam-se mais robusto, tal aspecto pode sugerir alterações nos requisitos previamente dispostos, para aprimorar o modelo como um todo.

Para satisfazer todos os requisitos, mais de uma propriedade foi necessária em todos os casos, salvo o caso da ausência de *deadlock*.

Quadro 4 – Resultados satisfeitos

Requisito	Objetivo	Status
REQ-01: O vidro deve ser totalmente aberto ou fechado em 4s	Atendido Parcialmente	Satisfeito
REQ-02: O vidro começa a se mover 200ms depois que o comando é selecionado.	Não atendido	-
REQ-03: Após 4 segundos de movimento ininterrupto em uma direção, o motor deve desligar.	Atendido Parcialmente	Satisfeito
REQ-04: A operação de tensão do sistema de controle está entre 12,5V e 14,5V.	Não atendido	-
REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado entre 200ms e 1s, o vidro abre ou fecha completamente.	Atendido	Satisfeito Valor lógico verdade
REQ-06: Se um objeto obstrutivo for detectado, o vidro deverá descer aproximadamente 10cm.	Atendido	Satisfeito Valor lógico verdade
REQ-07: A detecção de obstáculos tem prioridade sobre os comandos do passageiro e motorista.	Atendido	Satisfeito Valor lógico verdade
REQ-08: Os comandos do interruptor do motorista têm prioridade sobre os do passageiro.	Atendido	Satisfeito Valor lógico verdade
REQ-09: Quando acionado, o sistema de segurança deve fechar completamente, a menos que haja um obstáculo.	Atendido	Satisfeito Valor lógico verdade
REQ-10: O sistema de controle opera apenas com a posição de ignição em "ligado".	Não atendido	-
Não há *dealdock*	Atendido	Satisfeito Valor lógico verdade

Fonte: Autoria própria

O arquivo com o cenário de teste encontra-se no apêndice B deste documento e disponível para download no repositório do *GitHub*: <https://github.com/renan279/Eletric-Window-Automotive-System>.

6.3 ANÁLISE DO TRABALHO PROPOSTO EM RELAÇÃO AOS DA LITERATURA

O quadro 5 apresenta uma análise da diferença e a característica dos trabalhos de Santos *et al*. (2015) intitulado de “Rapid Control Prototyping for Automotive Software in Power Windows Systems”, de Roquette (2018) nomeado de “Uma Abordagem Utilizando do Behavior Driven Development para geração de casos de teste: Um estudo de caso na área automotiva” juntamente com as características deste trabalho.

Quadro 5 – Comparação entre trabalhos relacionados

Autor	Característica
Santos *et al*. (2015)	Prototipagem e teste de requisitos por meio de *Model-Based Design* (MDB) das funções de um sistema elétrico de janelas automotivas.
Roquette (2018)	Geração de casos de teste por meio da abordagem *Behavior Driven Development* (BDD) e outras técnicas de engenharia de software para a prototipagem do sistema elétrico de janelas automotivas proposta por Santos *et al*. (2015).
Verificação Formal Proposta	Verificação formal sobre uma modelagem feita a partir da prototipagem do sistema elétrico de janelas automotivas proposta por Santos *et al*. (2015).

Fonte: Autoria própria

No trabalho de Santos *et al*. (2015) há um protótipo de janelas elétrica automotiva desenvolvido a partir de requisitos pré-estabelecidos, explicados na seção 5.1 deste trabalho. Santos *et al*. (2015) utiliza do processo de desenvolvimento definido como *Model-Based Design* (MDB), que é o desenvolvimento de software baseado em modelos para sistemas embarcados. Portanto, semelhante ao proposto neste trabalho, por meio de sistemas modelados com determinadas tarefas, é possível especificar, projetar, simular, verificar e testar sistemas embarcados durante todo o projeto.

Roquette (2018) propõe uma geração de casos de teste sobre o protótipo de Santos *et al*. (2015) por meio de técnicas de engenharia de software, utilizando em suas etapas principais uma abordagem denominada como *Behavior Driven*

Development (BDD). O BDD é uma metodologia de desenvolvimento ágil com o intuito de assimilar linguagem de programação com regras de negócio, priorizando o comportamento do programa. Segundo Roquette (2018), a abordagem BDD, por tal característica, apontou erros no protótipo de Santos *et al*. (2015) não previstos anteriormente.

Diferente do trabalho de Roquette (2018), onde o autor traz um conjunto de cenários para teste sobre o mesmo protótipo elétrico de janelas para sistemas automotivos apresentado por Santos *et al*. (2015), e embora que na verificação formal também haja um conjunto de testes, por conta da lógica temporal, com a abordagem de *model checking* proposta neste trabalho é possível testar todos os cenários de uma aplicação, validando e assegurando as modelagens desenvolvidas na ferramenta (DENNIS *et al*., 2012).

6.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Com a apresentação deste capítulo, pode-se analisar a modelagem desenvolvida sobre o protótipo proposto por Santos *et al*. (2015), bem como suas etapas de simulação e verificação, ressaltando que, embora abordado com fases, não se trata de um processo linear, visto que conforme falhas são detectadas há retrocessos e avanços entre as camadas do processo.

Conforme exposto nos capítulos 5 e 6 deste trabalho, repara-se no difícil entendimento imediato da situação da modelagem, mesmo considerando a baixa quantidade de especificações, é inevitável a grande quantidade de arestas e a explosão de estados, que resulta num grande consumo de memória volátil e processamento do computador. Porém, é notável a eficiência da verificação formal em realizar testes complexos e garantir requisitos e propriedades lógicas em sistemas de tempo real.

O processo de verificação formal trouxe sugestões de alterações nos requisitos a serem cumpridos pelo sistema. O REQ-01 e o REQ-03 tratam sobre o controle da altura da janela, pois seu objetivo é semelhante. Para a modelagem foram desenvolvidas variáveis locais que gerenciam a altura das janelas de forma independente, tornando o sistema simplista para manutenção e, por sua vez, aprimorando os requisitos iniciais previamente propostos.

Inicialmente não se foi alcançado todos os objetivos planejados para a aplicação, vários guardas e invariantes foram avaliados nesse ponto para assegurar requisitos. Arestas para o estado urgente – que atende ao sensor que detecta obstáculos – foram necessárias para agilizar e tornar o uso do equipamento fluído. Outra vez, desenvolvendo soluções simplistas, como as invariantes, pode-se notar a escalabilidade e manutenibilidade que a ferramenta de *model checking* propõe.

7 CONCLUSÕES

Disputa de recursos, prejuízo monetário e fator de risco de vida humana são alguns dos pontos que pequenos descuidos na implementação de softwares de sistemas de tempo real podem trazer aos desenvolvedores. Com o objetivo de eliminar essa perda, ferramentas de verificação formal fornecem garantias na criação de sistemas capazes de assegurar propriedades especificadas.

Este trabalho utilizou a verificação formal aplicada em sistema automotivos e contribui com a comunidade científica com a modelagem do protótipo de um sistema elétrico de janelas automotiva.

Ao longo da construção deste trabalho, observa-se a importância da verificação formal em cenários em que ocorre momentos críticos em tempo de execução e/ou disputa de recurso, como é a urgência gerada pelo caso do obstáculo nesse trabalho – que pode vir a ser um objeto qualquer bem como um membro humano e/ou animal. A modelagem de sistemas desse gênero em ferramentas de *model checking* é relevante e providencia segurança aos envolvidos.

O trabalho também contribui com a modelagem do sistema elétrico de janela automotiva a partir do protótipo desenvolvido por Santos *et al* (2015), bem com a elaboração formal de um conjunto de teste para o modelo desenvolvido, disponíveis para *download* no repositório do *GitHub* <https://github.com/renan279/Eletric-Window-Automotive-System>.

Vale ressaltar que, com a modelagem e aplicação da técnica, sugere-se um aprimoramento adequado aos REQ-01 (que trata o tempo total em que a janela deve ser aberta ou fechada) e REQ-03 (que trata o desligamento do motor após a total abertura ou fechamento do vidro), ambos são semelhantes porém com propósitos diferentes. Como o projeto não aborda diretamente mecanismos motores, os requisitos citados foram atendidos com a monitoração e controle de uma variável “altura” que diz respeito sobre o valor de elevação de cada janela, de forma independente uma da outra.

A verificação formal proposta gerou várias ambuiguidades durante as etapas de modelagem e simulação. Facilidades da ferramenta UPPAAL impedem verificações como a alcançabilidade da propriedade “A[ ]”, tornando-se inviável o uso, como acontece com o uso da prioridade de canais nas declarações na etapa de

modelagem, dado pela palavra reservada “*chan priority*”. Além disso, a explosão de estados foi um obstáculo considerável, dito que implica numa demora em cada verificação juntamente com o uso de memória volátil da máquina.

A verificação formal proposta apresenta resultados satisfatórios em sistemas onde há disputa de recurso em tempo de execução, em que os envolvidos podem sofrer danos físicos e/ou materiais, visto que a técnica é capaz de testar todos os cenários previstos pela simulação e assegurar a corretude da modelagem por meio da verificação dos requisitos.

7.1 TRABALHOS FUTUROS

Trabalhos futuros que podem dar continuidade a esta pesquisa estão listados a seguir:

- Compreensão e modelagem para verificação formal de outras tarefas e recursos do veículo automotor.
- Adicionar mais autômatos ao modelo, de forma que o sistema agora atenda à quatro passageiro no veículo juntamente com suas respectivas janelas, sendo um deste passageiros o motorista.
- Geração de códigos-fontes dos modelos descritos e aplicação em placas para testes físicos.
- Modelagem, simulação e verificação na ferramenta UPPAAL PORT, baseada em componentes de sistema embarcado e de tempo real, pode ser usado como *plugin* em ambientes de desenvolvimento como o Eclipse. Disponível em: <it.uu.se/research/group/darts/uppaal/port/>.
- Aprimorar o modelo desenvolvido nesse trabalho por meio da ferramenta Urpal, que usa um modelo UPPAAL como entrada, executa testes no autômato, conforme descrito previamente pelo desenvolvedor, e grava um segundo modelo como resultado do teste em um arquivo. A ferramenta também oferece facilidades como: duplicar autômatos, remover transições com base em seu rótulo de ação, renomear canais e todas suas ocorrências e dimensionar o layout um modelo. Disponível em: <github.com/tbrk/urpal>.

REFERÊNCIAS

ALUR, R; DILL, D. L. Fundamental Study. A theory of timed automata. **Theoretical Computer Science,** v. 126, n.1, p. 183-235, 1994.

BABICH, F.; DEOTTO, L. Formal methods for specification and analysis of communication protocols. **Communications Surveys & Tutorials, IEEE**, v. 4, n. 1, p. 2–20, 2002.

BUTTAZZO, G. C; LIPARI, G. Ptask: An educational C library for programming real-time systems on Linux. **18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)**, p. 1-8, 2013.

CLARKE, E. M; WING, J. M. **Formal Methods:** State of the Art and Future Directions. Carnegie Mellon University: ACM, 1996.

DAVID, A. *et al*. UPPAAL SMC tutorial. **International Journal on Software Tools for Technology Transfer**, v. 17, n. 4, p. 397-415, 2015.

DENNIS, L. *et al*. Model checking agent programming languages**. Automated Software Engineering**, v. 19, n. 1, p. 5–63, 2012.

FRAPPIER, M. et al. Comparison os Model Checking Tools for Information Systems. In: 12TH INTERNATIONAL CONFERENCE ON FORMAL ENGINEERING METHODS (ICFEM). v. 6447, 2010. **Anais...** Springer, Berlin, Heidelberg: Universidade de Sherbrooke, 2010.

GARIS, A. Lógica Temporal en Verificación de Modelos de Software: Origen y Evolución hasta tiempos actuales. **Red de Revistas Científicas de América Latina y el Caribe, España y Portugal**, v. 11, n. 21, p. 151-161, 2010.

KOPETZ, H. **Real-Time Systems**: Design Principles for Distributed Embedded Applications. 2. ed. Austria: Springer Science & Business Media, 2011. 388 p.

LARSEN, K. G. *et al*. Testing real-time systems using UPPAAL. **Research Gate**, p. 77-117. 10.1007/978-3-540-78917-8_3, 2014. Disponível em: <https://www.researchgate.net/publication/221352048_Testing_real-time_systems_using_UPPAAL>. Acesso em: 17 set. 2018.

LEE. E. A. Cyber Physical Systems: Design Challenges. **11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)**, p. 363-369, 2008.

LIN-BUS. **Local interconnect network bus**. 2019. Disponível em: <https://www.ni.com/pt-br/innovations/white-papers/09/introduction-to-the-local-interconnect-network--lin--bus.html/>. Acesso em: 16 out. 2019.

MICHAILIDIS, A. et al. Test Front Loading in Early Stages of Automotive Software Development Based on AUTOSAR. **IEEE Europe Conference & Exhibition in Design, Automation & Test, Dresden**, p. 435-440, 2010.

Microchip Technology. **Concepts of Real Time Systems**. 2008. Disponível em: <http://www.microchip.com/stellent/groups/sitecomm_sg/documents/devicedoc/en543053.pdf>. Acesso em: 28 mai. 2018.

MOST. **Media oriented system transport**. 2019. Disponível em: <https://www.vector.com/int/en/know-how/technologies/networks/most/>. Acesso em: 16 out. 2019.

NuSMV. **Symbolic model checker tool**. 1999. Disponível em: < http://nusmv.fbk.eu/>. Acesso em: 16 out. 2019.

PELED, D. A. et al. **Model Checking**. 1. Ed. Massachusetts: The MIT Press. 1999.

PETTERSSON, Paul. **Modelling and Verification of Real-Time Systems Using Timed Automata**: Theory and Practice. 1999. 206 f. Tese (Doutorado) - Doctor of Philosophy, Uppsala University, Sweden, 1999.

PRESSMAN, R. S; MAXIM, B. R. **Engenharia de Software**: Uma Abordagem Profissional. 8. Ed. São Paulo: AMGH. 2016.

ROQUETTE, J. H. **Uma Abordagem Utilizando do Behavior Driven Development para geração de casos de teste**: Um estudo de caso na área automotiva. 2018. 71f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação - Universidade Tecnológica Federal do Paraná). Ponta Grossa, 2018.

SAEEDLOEI, N. **Modeling and verification of real-time and cyber-physical systems**. 2011. 165 f. Tese (Doutorado) – Doctor of Philosophy in Computer Science, The University of Texas at Dallas. Texas, 2011.

SANTOS, M. M. D. et al. Rapid Control Prototyping For Automotive Software In Power Windows Systems. **International Journal of Innovative Computing, Information and Control**, v. 11**,** n. 4, p. 1341-1356, 2015.

SCHAUFFELE, J.; ZURAKAWA, T. **Automotive software engineering**: Principles, processes, methods and tools. 1 ed. Warrendale: SAE International, 2005. 385 p.

SPIN. **Open-source software verification tool**. 2016. Disponível em: < http://spinroot.com/spin/whatispin.html>. Acesso em: 16 out. 2019.

TAURION, C. **Software Embarcado**: A nova onda da Informática: chips e softwares em todos objetos. 1. ed. Rio de Janeiro: Brasport, 2005.

UPPAAL. **Model checker tool**. 2015. Disponível em: <http://www.uppaal.org/>. Acesso em: 30 set. 2018.

YI, W.; PETTERSSON, P.; DANIELS, M. Automatic verification of real-time communicating systems by constraint-solving. **Citeseer**, Springer, Boston (MA), v. 6, p. 243–258, 1994.

ZHONGSHENG, Q. *et al*. Modeling Distributed Real-time Elevator System by Three Model Checker. **International Journal of Online Engineering,** Nanchang. China, Jiangxi University Of Finance And Economics, v. 14, n. 4, p. 1861-2121. 2018.

APÊNDICE A - Modelagem do sistema de janelas automotivo

```
<?xml version="1.0" encoding="utf-8"?><!DOCTYPE nta PUBLIC '-//Uppaal
Team//DTD Flat System 1.1//EN'
'http://www.it.uu.se/research/group/darts/uppaal/flat-
1_1.dtd'><nta><declaration>bool U = false;                  //    movimentos
motorista On e Off

bool E = false;               //    movimentos motorista sobre passageiro On
e Off

bool I = false;               //    movimentos passageiro On e Off

bool A = false;               //    alarme On ou Off

bool O = false;               //    obstrucao On ou Off

chan cont, conti;             //    sinal para continuar movimento

chan subir, descer;           //    canais passageiro

chan sub, des;                //    canais motorista (sobre si)

chan subirM, descerM;         //    canais motorista sobre passeiro

broadcast chan alarme, desligar;//  sinal do alarme

urgent chan obsE;             //    obj janela Esquerda

urgent chan obsD;             //    obj janela Direita

//chan priority cont, subir, descer, sub, des &amp;lt; conti, subirM, descerM
&amp;lt; alarme, desligar &amp;lt; obsE,
obsD;</declaration><template><name>Alarma</name><location id="id0" x="128"
y="0"><name x="120" y="-32">On</name></location><location id="id1" x="0"
y="0"><name x="-8" y="-32">Off</name></location><init
ref="id1"/><transition><source ref="id0"/><target ref="id1"/><label
kind="synchronisation" x="40" y="-56">desligar!</label><label
kind="assignment" x="40" y="-72">A = false</label><nail x="96" y="-
32"/><nail x="32" y="-32"/></transition><transition><source
ref="id1"/><target ref="id0"/><label kind="synchronisation" x="40"
y="32">alarme!</label><label kind="assignment" x="40" y="48">A =
true</label><nail x="32" y="32"/><nail x="96"
y="32"/></transition></template><template><name>ObstaculoMotorista</name><l
ocation id="id2" x="128" y="0"><name x="120" y="-
32">On</name></location><location id="id3" x="0" y="0"><name x="-8" y="-
32">Off</name></location><init ref="id3"/><transition><source
ref="id2"/><target ref="id3"/><label kind="synchronisation" x="40" y="-
72">obsE!</label><label kind="assignment" x="32" y="-56">O =
false</label><nail x="96" y="-32"/><nail x="32" y="-
32"/></transition><transition><source ref="id3"/><target ref="id2"/><label
kind="synchronisation" x="48" y="32">obsE!</label><label kind="assignment"
x="40" y="48">O = true</label><nail x="32" y="32"/><nail x="96"
y="32"/></transition></template><template><name>ObstaculoPassageiro</name><
location id="id4" x="64" y="0"><name x="56" y="-
32">On</name></location><location id="id5" x="-64" y="0"><name x="-72" y="-
32">Off</name></location><init ref="id5"/><transition><source
ref="id4"/><target ref="id5"/><label kind="synchronisation" x="-24" y="-
```

72">obsD!</label><label kind="assignment" x="-24" y="-56">O =
false</label><nail x="32" y="-32"/><nail x="-32" y="-
32"/></transition><transition><source ref="id5"/><target ref="id4"/><label
kind="synchronisation" x="-16" y="32">obsD!</label><label kind="assignment"
x="-24" y="48">O = true</label><nail x="-32" y="32"/><nail x="32"
y="32"/></transition></template><template><name>Motorista</name><declaratio
n>clock M;</declaration><location id="id6" x="-32" y="-224"><name x="-48"
y="-256">Inativo</name></location><location id="id7" x="-128" y="64"><name
x="-296" y="176">DescerVidroMotorista</name><label kind="invariant" x="-
264" y="192">E == false</label></location><location id="id8" x="-128" y="-
64"><name x="-296" y="-208">SubirVidroMotorista</name><label
kind="invariant" x="-272" y="-192">E == false</label></location><location
id="id9" x="64" y="64"><name x="120"
y="176">DescerVidroPassageiro</name><label kind="invariant" x="168"
y="192">E == true</label></location><location id="id10" x="64" y="-
64"><name x="112" y="-208">SubirVidroPassageiro</name><label
kind="invariant" x="144" y="-192">E == true</label></location><location
id="id11" x="-32" y="0"><name x="-56" y="-32">Parado</name></location><init
ref="id11"/><transition><source ref="id6"/><target ref="id11"/><label
kind="synchronisation" x="0" y="-208">desligar?</label><nail x="0" y="-
192"/></transition><transition><source ref="id11"/><target
ref="id6"/><label kind="synchronisation" x="-112" y="-
208">alarme?</label><nail x="-64" y="-
192"/></transition><transition><source ref="id10"/><target
ref="id6"/><label kind="synchronisation" x="64" y="-
240">alarme?</label><label kind="assignment" x="80" y="-256">E =
false</label><nail x="96" y="-224"/></transition><transition><source
ref="id9"/><target ref="id6"/><label kind="synchronisation" x="120" y="-
240">alarme?</label><label kind="assignment" x="144" y="-256">E =
false</label><nail x="128" y="-224"/></transition><transition><source
ref="id7"/><target ref="id6"/><label kind="synchronisation" x="-160" y="-
240">alarme?</label><label kind="assignment" x="-232" y="-256">E =
false</label><nail x="-192" y="-224"/></transition><transition><source
ref="id8"/><target ref="id6"/><label kind="synchronisation" x="-216" y="-
240">alarme?</label><label kind="assignment" x="-168" y="-256">E =
false</label><nail x="-160" y="-224"/></transition><transition><source
ref="id8"/><target ref="id10"/><label kind="synchronisation" x="-48" y="-
128">subirM!</label><label kind="assignment" x="-72" y="-112">M = 0, E =
true</label><nail x="-96" y="-96"/><nail x="32" y="-
96"/></transition><transition><source ref="id7"/><target ref="id8"/><label
kind="synchronisation" x="-224" y="-16">sub!</label><label
kind="assignment" x="-232" y="0">M = 0, E = false</label><nail x="-192"
y="32"/><nail x="-192" y="-32"/></transition><transition><source
ref="id9"/><target ref="id7"/><label kind="synchronisation" x="-48"
y="96">des!</label><label kind="assignment" x="-72" y="112">M = 0, E =
false</label><nail x="32" y="96"/><nail x="-96"
y="96"/></transition><transition><source ref="id10"/><target
ref="id9"/><label kind="synchronisation" x="128" y="-
8">descerM!</label><label kind="assignment" x="80" y="8">M = 0, E =
true</label><nail x="128" y="-32"/><nail x="128"
y="32"/></transition><transition><source ref="id7"/><target
ref="id7"/><label kind="guard" x="-224" y="80">M > 200 and M <
1000</label><label kind="synchronisation" x="-224"

y="96">conti!</label><nail x="-144" y="112"/><nail x="-168"
y="88"/></transition><transition><source ref="id9"/><target
ref="id9"/><label kind="guard" x="88" y="112">M > 200 and M <
1000</label><label kind="synchronisation" x="88"
y="128">conti!</label><nail x="96" y="104"/><nail x="80"
y="120"/></transition><transition><source ref="id10"/><target
ref="id10"/><label kind="guard" x="72" y="-120">M > 200 and M <
1000</label><label kind="synchronisation" x="104" y="-
104">conti!</label><nail x="88" y="-104"/><nail x="104" y="-
88"/></transition><transition><source ref="id8"/><target ref="id8"/><label
kind="guard" x="-248" y="-136">M > 200 and M < 1000</label><label
kind="synchronisation" x="-200" y="-120">conti!</label><nail x="-168" y="-
104"/><nail x="-144" y="-120"/></transition><transition><source
ref="id9"/><target ref="id11"/><label kind="assignment" x="-88" y="16">M =
0, E = false, U = false</label><nail x="32"
y="16"/></transition><transition><source ref="id10"/><target
ref="id11"/><label kind="assignment" x="-80" y="-48">M = 0, E = false, U =
false</label><nail x="32" y="-16"/></transition><transition><source
ref="id8"/><target ref="id11"/><label kind="assignment" x="-80" y="-48">M =
0, E = false, U = false</label><nail x="-96" y="-
16"/></transition><transition><source ref="id7"/><target ref="id11"/><label
kind="assignment" x="-88" y="16">M = 0, E = false, U = false</label><nail
x="-96" y="16"/></transition><transition><source ref="id10"/><target
ref="id7"/><label kind="synchronisation" x="-352" y="-
56">des!</label><label kind="assignment" x="-424" y="-40">M = 0, E =
false</label><nail x="64" y="-160"/><nail x="-320" y="-160"/><nail x="-320"
y="64"/></transition><transition><source ref="id7"/><target
ref="id10"/><label kind="synchronisation" x="264"
y="32">subirM!</label><label kind="assignment" x="264" y="48">M = 0, E =
true</label><nail x="-128" y="160"/><nail x="224" y="160"/><nail x="256"
y="160"/><nail x="256" y="-64"/></transition><transition><source
ref="id8"/><target ref="id9"/><label kind="synchronisation" x="-320"
y="96">descerM!</label><label kind="assignment" x="-352" y="112">M = 0, E =
true</label><nail x="-192" y="-96"/><nail x="-256" y="-96"/><nail x="-256"
y="128"/><nail x="64" y="128"/></transition><transition><source
ref="id9"/><target ref="id8"/><label kind="synchronisation" x="200" y="-
128">sub!</label><label kind="assignment" x="200" y="-112">M = 0, E =
false</label><nail x="96" y="96"/><nail x="192" y="96"/><nail x="192" y="-
128"/><nail x="-128" y="-128"/></transition><transition><source
ref="id7"/><target ref="id9"/><label kind="synchronisation" x="-56"
y="64">descerM!</label><label kind="assignment" x="-72" y="80">M = 0, E =
true</label></transition><transition><source ref="id8"/><target
ref="id7"/><label kind="synchronisation" x="-160" y="-
16">des!</label><label kind="assignment" x="-232" y="0">M = 0, E =
false</label></transition><transition><source ref="id10"/><target
ref="id8"/><label kind="synchronisation" x="-48" y="-96">sub!</label><label
kind="assignment" x="-72" y="-80">M = 0, E =
false</label></transition><transition><source ref="id9"/><target
ref="id10"/><label kind="synchronisation" x="72" y="-
8">subirM!</label><label kind="assignment" x="80" y="8">M = 0, E =
true</label></transition><transition><source ref="id11"/><target
ref="id10"/><label kind="synchronisation" x="-16" y="-
64">subirM!</label><label kind="assignment" x="-8" y="-32">E = true, U =

```
true</label></transition><transition><source ref="id11"/><target
ref="id9"/><label kind="synchronisation" x="-32"
y="48">descerM!</label><label kind="assignment" x="-8" y="32">E = true, U =
true</label></transition><transition><source ref="id11"/><target
ref="id7"/><label kind="synchronisation" x="-80" y="48">des!</label><label
kind="assignment" x="-112" y="32">E = false, U =
true</label></transition><transition><source ref="id11"/><target
ref="id8"/><label kind="synchronisation" x="-72" y="-64">sub!</label><label
kind="assignment" x="-120" y="-32">E = false, U =
true</label></transition></template><template><name>Passageiro</name><decla
ration>clock P;</declaration><location id="id12" x="-96" y="-192"><name
x="-112" y="-224">Inativo</name></location><location id="id13" x="96"
y="64"><name x="112" y="56">DescerVidro</name><label kind="invariant"
x="192" y="56">E == false</label></location><location id="id14" x="96" y="-
64"><name x="112" y="-72">SubirVidro</name><label kind="invariant" x="192"
y="-72">E == false</label></location><location id="id15" x="0" y="0"><name
x="-64" y="-8">Parado</name></location><init
ref="id15"/><transition><source ref="id15"/><target ref="id12"/><label
kind="synchronisation" x="-152" y="-48">alarme?</label><label
kind="assignment" x="-144" y="-32">I = false</label><nail x="-96"
y="16"/></transition><transition><source ref="id13"/><target
ref="id12"/><label kind="synchronisation" x="-152"
y="72">alarme?</label><label kind="assignment" x="-152" y="88">I =
false</label><nail x="-96" y="96"/></transition><transition><source
ref="id14"/><target ref="id12"/><label kind="synchronisation" x="104" y="-
200">alarme?</label><label kind="assignment" x="104" y="-184">I =
false</label><nail x="96" y="-192"/></transition><transition><source
ref="id12"/><target ref="id15"/><label kind="synchronisation" x="-56" y="-
144">desligar?</label></transition><transition><source ref="id13"/><target
ref="id14"/><label kind="synchronisation" x="104" y="-
16">subir!</label><label kind="assignment" x="104" y="0">P =
0</label></transition><transition><source ref="id14"/><target
ref="id13"/><label kind="synchronisation" x="168" y="-
16">descer!</label><label kind="assignment" x="168" y="0">P =
0</label><nail x="160" y="-32"/><nail x="160"
y="32"/></transition><transition><source ref="id13"/><target
ref="id13"/><label kind="guard" x="32" y="96">P &gt; 200 and P &lt;
1000</label><label kind="synchronisation" x="80" y="112">cont!</label><nail
x="64" y="96"/><nail x="128" y="96"/></transition><transition><source
ref="id14"/><target ref="id14"/><label kind="guard" x="32" y="-120">P &gt;
200 and P &lt; 1000</label><label kind="synchronisation" x="80" y="-
136">cont!</label><nail x="64" y="-96"/><nail x="128" y="-
96"/></transition><transition><source ref="id13"/><target
ref="id15"/><label kind="assignment" x="-32" y="64">P = 0, I =
false</label><nail x="0" y="64"/></transition><transition><source
ref="id14"/><target ref="id15"/><label kind="assignment" x="-32" y="-80">P
= 0, I = false</label><nail x="0" y="-64"/></transition><transition><source
ref="id15"/><target ref="id13"/><label kind="synchronisation" x="48" y="-
8">descer!</label><label kind="assignment" x="48" y="8">I =
true</label></transition><transition><source ref="id15"/><target
ref="id14"/><label kind="synchronisation" x="56" y="-
40">subir!</label><label kind="assignment" x="48" y="-24">I =
```

```
true</label></transition></template><template><name>VidroPassageiro</name><
declaration>clock V;

int [0,10] altura = 10; //altura do vidro

int [0, 1] tempo = 0;   //para controlar altura de elevação na obstrução

</declaration><location id="id16" x="160" y="-176"><name x="160" y="-
208">Obstaculo</name><label kind="invariant" x="176" y="-184">A ==
false</label><urgent/></location><location id="id17" x="160" y="128"><name
x="168" y="256">Descendo</name><label kind="invariant" x="144" y="272">E ==
true or I == true</label></location><location id="id18" x="160" y="-
64"><name x="160" y="-296">Subindo</name><label kind="invariant" x="128"
y="-280">E == true or I == true or A == true</label></location><location
id="id19" x="64" y="128"><name x="40" y="256">Descer</name><label
kind="invariant" x="0" y="272">E == true or I ==
true</label></location><location id="id20" x="64" y="-64"><name x="32" y="-
296">Subir</name><label kind="invariant" x="-16" y="-280">E == true or I ==
true</label></location><location id="id21" x="-32" y="32"><name x="-96"
y="16">Parado</name></location><init ref="id21"/><transition><source
ref="id17"/><target ref="id20"/><label kind="guard" x="392" y="208">altura
&lt; 10</label><label kind="synchronisation" x="392"
y="192">subirM?</label><nail x="224" y="224"/><nail x="384" y="224"/><nail
x="384" y="192"/></transition><transition><source ref="id19"/><target
ref="id20"/><label kind="guard" x="392" y="208">altura &lt;
10</label><label kind="synchronisation" x="392"
y="192">subirM?</label><nail x="224" y="224"/><nail x="384" y="224"/><nail
x="384" y="192"/></transition><transition><source ref="id18"/><target
ref="id19"/><label kind="guard" x="392" y="-144">altura &gt;
0</label><label kind="synchronisation" x="392" y="-
160">descerM?</label><nail x="224" y="-160"/><nail x="384" y="-160"/><nail
x="384" y="-128"/></transition><transition><source ref="id20"/><target
ref="id19"/><label kind="guard" x="392" y="-144">altura &gt;
0</label><label kind="synchronisation" x="392" y="-
160">descerM?</label><nail x="224" y="-160"/><nail x="384" y="-160"/><nail
x="384" y="-128"/></transition><transition><source ref="id21"/><target
ref="id19"/><label kind="guard" x="-192" y="96">altura &gt; 0</label><label
kind="synchronisation" x="-192" y="80">descerM?</label><nail x="-128"
y="32"/><nail x="-128" y="128"/></transition><transition><source
ref="id21"/><target ref="id20"/><label kind="guard" x="-200" y="-24">altura
&lt; 10</label><label kind="synchronisation" x="-184" y="-
40">subirM?</label><nail x="-128" y="32"/><nail x="-128" y="-
64"/></transition><transition><source ref="id18"/><target
ref="id18"/><label kind="synchronisation" x="184" y="-
48">alarme?</label><nail x="176" y="-32"/><nail x="192" y="-
48"/></transition><transition><source ref="id20"/><target
ref="id18"/><label kind="guard" x="360" y="-16">altura &lt;
10</label><label kind="synchronisation" x="360" y="0">alarme?</label><nail
x="88" y="-32"/><nail x="88" y="256"/><nail x="352" y="256"/><nail x="352"
y="-64"/></transition><transition><source ref="id21"/><target
ref="id18"/><label kind="guard" x="360" y="-16">altura &lt;
10</label><label kind="synchronisation" x="360" y="0">alarme?</label><nail
x="-32" y="256"/><nail x="352" y="256"/><nail x="352" y="-
64"/></transition><transition><source ref="id17"/><target
```

```
ref="id18"/><label kind="guard" x="360" y="-16">altura &lt;
10</label><label kind="synchronisation" x="360" y="0">alarme?</label><nail
x="160" y="256"/><nail x="352" y="256"/><nail x="352" y="-
64"/></transition><transition><source ref="id19"/><target
ref="id18"/><label kind="guard" x="360" y="-16">altura &lt;
10</label><label kind="synchronisation" x="360" y="0">alarme?</label><nail
x="64" y="256"/><nail x="352" y="256"/><nail x="352" y="-
64"/></transition><transition><source ref="id19"/><target
ref="id19"/><label kind="guard" x="-16" y="160">altura &gt; 0</label><label
kind="assignment" x="-8" y="144">altura--</label><nail x="32"
y="144"/><nail x="56" y="160"/></transition><transition><source
ref="id20"/><target ref="id20"/><label kind="guard" x="-24" y="-128">altura
&lt; 10</label><label kind="assignment" x="-16" y="-
112">altura++</label><nail x="32" y="-80"/><nail x="56" y="-
104"/></transition><transition><source ref="id16"/><target
ref="id16"/><label kind="guard" x="88" y="-248">altura &gt; 0 and tempo ==
0</label><label kind="assignment" x="96" y="-232">altura--,
tempo++</label><nail x="120" y="-192"/><nail x="144" y="-
216"/></transition><transition><source ref="id17"/><target
ref="id17"/><label kind="guard" x="96" y="168">altura &gt; 0</label><label
kind="assignment" x="112" y="152">altura--</label><nail x="128"
y="144"/><nail x="160" y="160"/></transition><transition><source
ref="id18"/><target ref="id18"/><label kind="guard" x="72" y="-128">altura
&lt; 10</label><label kind="assignment" x="96" y="-
112">altura++</label><nail x="128" y="-80"/><nail x="152" y="-
96"/></transition><transition><source ref="id17"/><target
ref="id16"/><label kind="synchronisation" x="296" y="-
16">obsD?</label><nail x="176" y="192"/><nail x="288" y="192"/><nail
x="288" y="-128"/><nail x="160" y="-128"/></transition><transition><source
ref="id19"/><target ref="id16"/><label kind="synchronisation" x="296" y="-
16">obsD?</label><nail x="80" y="192"/><nail x="288" y="192"/><nail x="288"
y="-128"/><nail x="160" y="-128"/></transition><transition><source
ref="id17"/><target ref="id21"/><label kind="guard" x="144" y="56">altura
== 0</label><nail x="128" y="32"/></transition><transition><source
ref="id18"/><target ref="id21"/><label kind="guard" x="144" y="-16">altura
== 10</label><nail x="128" y="32"/></transition><transition><source
ref="id20"/><target ref="id21"/><label kind="guard" x="48" y="-16">altura
&lt;= 10</label><nail x="32" y="32"/></transition><transition><source
ref="id19"/><target ref="id21"/><label kind="guard" x="48" y="56">altura
&gt;= 0</label><nail x="32" y="32"/></transition><transition><source
ref="id16"/><target ref="id21"/><label kind="guard" x="-40" y="-200">tempo
&gt;= 0 or altura == 0</label><label kind="synchronisation" x="-80" y="-
176">obsD?</label><label kind="assignment" x="-40" y="-216">tempo =
0</label><nail x="-32" y="-176"/></transition><transition><source
ref="id19"/><target ref="id17"/><label kind="synchronisation" x="88"
y="104">cont?</label></transition><transition><source ref="id21"/><target
ref="id19"/><label kind="guard" x="-24" y="104">altura &gt; 0</label><label
kind="synchronisation" x="-24"
y="88">descer?</label></transition><transition><source ref="id20"/><target
ref="id18"/><label kind="synchronisation" x="88" y="-
64">cont?</label></transition><transition><source ref="id21"/><target
ref="id20"/><label kind="guard" x="-24" y="-64">altura &lt;
10</label><label kind="synchronisation" x="-16" y="-
```

```
48">subir?</label></transition><transition><source ref="id18"/><target
ref="id16"/><label kind="synchronisation" x="296" y="-
16">obsD?</label></transition><transition><source ref="id20"/><target
ref="id16"/><label kind="synchronisation" x="296" y="-
16">obsD?</label><nail x="64" y="-128"/><nail x="160" y="-
128"/></transition></template><template><name>VidroMotorista</name><declara
tion>clock V;

int [0,10] altura = 10; //altura do vidro

int [0, 1] tempo = 0;

</declaration><location id="id22" x="160" y="-176"><name x="160" y="-
208">Obstaculo</name><label kind="invariant" x="176" y="-184">A ==
false</label><urgent/></location><location id="id23" x="160" y="128"><name
x="168" y="136">Descendo</name><label kind="invariant" x="150" y="143">U ==
true</label></location><location id="id24" x="160" y="-64"><name x="176"
y="-88">Subindo</name><label kind="invariant" x="240" y="-88">A == true or
U == true or E == true</label></location><location id="id25" x="64"
y="128"><name x="72" y="136">Descer</name><label kind="invariant" x="54"
y="143">U == true</label></location><location id="id26" x="64" y="-
64"><name x="72" y="-88">Subir</name><label kind="invariant" x="54" y="-
49">U == true</label></location><location id="id27" x="-32" y="32"><name
x="-96" y="16">Parado</name></location><init
ref="id27"/><transition><source ref="id24"/><target ref="id24"/><label
kind="synchronisation" x="216" y="-48">alarme?</label><nail x="176" y="-
32"/><nail x="192" y="-48"/></transition><transition><source
ref="id26"/><target ref="id24"/><label kind="guard" x="360" y="-16">altura
&lt; 10</label><label kind="synchronisation" x="360"
y="0">alarme?</label><nail x="88" y="-32"/><nail x="88" y="256"/><nail
x="352" y="256"/><nail x="352" y="-64"/></transition><transition><source
ref="id27"/><target ref="id24"/><label kind="guard" x="360" y="-16">altura
&lt; 10</label><label kind="synchronisation" x="360"
y="0">alarme?</label><nail x="-32" y="256"/><nail x="352" y="256"/><nail
x="352" y="-64"/></transition><transition><source ref="id23"/><target
ref="id24"/><label kind="guard" x="360" y="-16">altura &lt;
10</label><label kind="synchronisation" x="360" y="0">alarme?</label><nail
x="160" y="256"/><nail x="352" y="256"/><nail x="352" y="-
64"/></transition><transition><source ref="id25"/><target
ref="id24"/><label kind="guard" x="360" y="-16">altura &lt;
10</label><label kind="synchronisation" x="360" y="0">alarme?</label><nail
x="64" y="256"/><nail x="352" y="256"/><nail x="352" y="-
64"/></transition><transition><source ref="id25"/><target
ref="id25"/><label kind="guard" x="-16" y="160">altura &gt; 0</label><label
kind="assignment" x="-8" y="144">altura--</label><nail x="32"
y="144"/><nail x="56" y="160"/></transition><transition><source
ref="id26"/><target ref="id26"/><label kind="guard" x="-24" y="-128">altura
&lt; 10</label><label kind="assignment" x="-16" y="-
112">altura++</label><nail x="32" y="-80"/><nail x="56" y="-
104"/></transition><transition><source ref="id22"/><target
ref="id22"/><label kind="guard" x="88" y="-248">altura &gt; 0 and tempo ==
0</label><label kind="assignment" x="96" y="-232">altura--,
tempo++</label><nail x="120" y="-192"/><nail x="144" y="-
216"/></transition><transition><source ref="id23"/><target
```

```
ref="id23"/><label kind="guard" x="96" y="168">altura &gt; 0</label><label
kind="assignment" x="112" y="152">altura--</label><nail x="128"
y="144"/><nail x="160" y="160"/></transition><transition><source
ref="id24"/><target ref="id24"/><label kind="guard" x="72" y="-128">altura
&lt; 10</label><label kind="assignment" x="96" y="-
112">altura++</label><nail x="128" y="-80"/><nail x="152" y="-
96"/></transition><transition><source ref="id23"/><target
ref="id22"/><label kind="synchronisation" x="296" y="-
16">obsE?</label><nail x="176" y="192"/><nail x="288" y="192"/><nail
x="288" y="-128"/><nail x="160" y="-128"/></transition><transition><source
ref="id25"/><target ref="id22"/><label kind="synchronisation" x="296" y="-
16">obsE?</label><nail x="80" y="192"/><nail x="288" y="192"/><nail x="288"
y="-128"/><nail x="160" y="-128"/></transition><transition><source
ref="id23"/><target ref="id27"/><label kind="guard" x="144" y="56">altura
== 0</label><nail x="128" y="32"/></transition><transition><source
ref="id24"/><target ref="id27"/><label kind="guard" x="144" y="-16">altura
== 10</label><nail x="128" y="32"/></transition><transition><source
ref="id26"/><target ref="id27"/><label kind="guard" x="48" y="-16">altura
&lt;= 10</label><nail x="32" y="32"/></transition><transition><source
ref="id25"/><target ref="id27"/><label kind="guard" x="48" y="56">altura
&gt;= 0</label><nail x="32" y="32"/></transition><transition><source
ref="id22"/><target ref="id27"/><label kind="guard" x="-40" y="-200">tempo
&gt;= 0 or altura == 0</label><label kind="synchronisation" x="-80" y="-
176">obsE?</label><label kind="assignment" x="-40" y="-216">tempo =
0</label><nail x="-32" y="-176"/></transition><transition><source
ref="id25"/><target ref="id23"/><label kind="synchronisation" x="88"
y="104">conti?</label></transition><transition><source ref="id27"/><target
ref="id25"/><label kind="guard" x="-24" y="104">altura &gt; 0</label><label
kind="synchronisation" x="-24"
y="88">des?</label></transition><transition><source ref="id26"/><target
ref="id24"/><label kind="synchronisation" x="88" y="-
64">conti?</label></transition><transition><source ref="id27"/><target
ref="id26"/><label kind="guard" x="-24" y="-64">altura &lt;
10</label><label kind="synchronisation" x="-16" y="-
48">sub?</label></transition><transition><source ref="id24"/><target
ref="id22"/><label kind="synchronisation" x="296" y="-
16">obsE?</label></transition><transition><source ref="id26"/><target
ref="id22"/><label kind="synchronisation" x="296" y="-
16">obsE?</label><nail x="64" y="-128"/><nail x="160" y="-
128"/></transition></template><system>JanelaE = VidroMotorista();

JanelaD = VidroPassageiro();

ObstrucaoE = ObstaculoMotorista();

ObstrucaoD = ObstaculoPassageiro();

Passag = Passageiro();

Driver = Motorista();

Allarme = Alarma();

system ObstrucaoD, ObstrucaoE, Passag, JanelaD, JanelaE, Allarme,
Driver;</system></nta>
```

APÊNDICE B - Testes realizados no sistema de janelas automotivo

```
//This file was generated from (Commercial) UPPAAL 4.0.14 (rev. 5615), May
2014


/*REQ-09: Quando acionado, o sistema de seguran\u00e7a deve fechar
completamente, a menos que haja um obst\u00e1culo.*/


(Allarme.On and ObstrucaoE.Off) --> (JanelaE.Subindo or JanelaE.Subir or
JanelaD.Parado)


/*REQ-09: Quando acionado, o sistema de seguran\u00e7a deve fechar
completamente, a menos que haja um obst\u00e1culo.*/


(Allarme.On and ObstrucaoD.Off) --> (JanelaD.Subindo or JanelaD.Subir or
JanelaD.Parado)


/*REQ-09: Quando acionado, o sistema de seguran\u00e7a deve fechar
completamente, a menos que haja um obst\u00e1culo.*/


Allarme.On --> (Passag.Inativo and Driver.Inativo)


/*REQ-09: Quando acionado, o sistema de seguran\u00e7a deve fechar
completamente, a menos que haja um obst\u00e1culo.*/


(Allarme.On and ObstrucaoE.On) --> JanelaE.Obstaculo


/*REQ-09: Quando acionado, o sistema de seguran\u00e7a deve fechar
completamente, a menos que haja um obst\u00e1culo.*/


(Allarme.On and ObstrucaoD.On) --> JanelaD.Obstaculo


/*REQ-08: Os comandos do interruptor do motorista t\u00eam prioridade sobre
os do passageiro.*/


Driver.DescerVidroPassageiro --> Passag.Parado


/*REQ-08: Os comandos do interruptor do motorista t\u00eam prioridade sobre
os do passageiro.*/


Driver.SubirVidroPassageiro --> Passag.Parado


/*REQ-06: Se um objeto obstrutivo for detectado, o vidro dever\u00e1 descer
aproximadamente 10cm.*/
```

```
ObstrucaoD.On --> JanelaD.Obstaculo


/*REQ-06: Se um objeto obstrutivo for detectado, o vidro dever\u00e1 descer
aproximadamente 10cm.*/


ObstrucaoE.On --> JanelaE.Obstaculo


/*REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado
entre 200ms e 1s, o vidro abre ou fecha completamente.*/


E<> Driver.SubirVidroPassageiro and (JanelaD.Subindo or JanelaD.Subir)


/*REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado
entre 200ms e 1s, o vidro abre ou fecha completamente.*/


E<> Driver.SubirVidroMotorista and (JanelaE.Subindo or JanelaE.Subir)


/*REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado
entre 200ms e 1s, o vidro abre ou fecha completamente.*/


E<> Passag.SubirVidro and (JanelaD.Subindo or JanelaD.Subir)




/*REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado
entre 200ms e 1s, o vidro abre ou fecha completamente.*/


E<> Driver.DescerVidroPassageiro and (JanelaD.Descendo or JanelaD.Descer)


/*REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado
entre 200ms e 1s, o vidro abre ou fecha completamente.*/


E<> Driver.DescerVidroMotorista and (JanelaE.Descendo or JanelaE.Descer)


/*REQ-05: Se o comando para abrir ou fechar o vidro for mantido pressionado
entre 200ms e 1s, o vidro abre ou fecha completamente.*/


E<> Passag.DescerVidro and (JanelaD.Descendo or JanelaD.Descer)


/*N\u00e3o existe deadlock. Deve devolver valor Verdade.*/


A[] not deadlock
```