UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATEUS RICARDO PALUDO BALBINOT

UMA ABORDAGEM DECLARATIVA PARA O DESENVOLVIMENTO DE SISTEMAS CRÍTICOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2016

MATEUS RICARDO PALUDO BALBINOT

UMA ABORDAGEM DECLARATIVA PARA O DESENVOVLIMENTO DE SISTEMAS CRÍTICOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Marcelo Teixeira

PATO BRANCO

2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

UMA ABORDAGEM DECLARATIVA PARA O DESENVOLVIMENTO DE SISTEMAS CRÍTICOS

por

MATEUS RICARDO PALUDO BALBINOT

Este trabalho de conclusão de curso foi apresentado no dia 24 de novembro de 2016, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Banca examinadora:

Prof. Dr. Marcelo Teixeira
Orientador

Prof. Dr. Richardson Ribeiro

Profª. Me. Eliane Maria de Bortoli Favero

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª. Me. Soelaine Rodrigues Ascari
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar aonde a maioria não chega, faça o que a maioria não faz.

Bill Gates

RESUMO

BALBINOT, Mateus R. P. Uma abordagem declarativa para o desenvolvimento de sistemas críticos. 2016. 28 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

O desenvolvimento de sistemas computacionais de grande porte, complexos e com múltiplos requisitos, em geral, demanda uma habilidade maior de um programador. Limitações do raciocínio humano sobre problemas de grande porte e de alta complexidade podem levar à incorporação de erros na codificação do software, muitas vezes por falhas na identificação dos requisitos, por envolver um número grande de variáveis relacionadas, validações, etc. Em sistemas críticos, como um sistema de controle de tráfego aéreo, por exemplo, erros podem ser fatais e, portanto, devem ser evitados ao máximo. Ao mesmo tempo, esse tipo de sistema envolve um extenso e complexo encadeamento de fatores, tais que dificultam a programação livre de erros. Uma alternativa que pode ser usada para facilitar tal programação envolve o uso modelos capazes de fornecer uma visão mais macro do sistema, de tal forma que a programação possa ser otimizada. Autômatos finitos são modelos capazes de descrever a dinâmica de um sistema baseando-se no conceito de eventos. Usando esses modelos pode-se expressar o comportamento de vários elementos de hardware e de software. Neste trabalho, autômatos serão usados para representar um sistema de tráfego aéreo. Será mostrado que a abordagem permite obter automaticamente as sequências operacionais do controle aéreo de forma a descartar qualquer possibilidade de ocorrer uma sequência de eventos não prevista pelos requisitos impostos.

Palavras-chave: Desenvolvimento dirigido a modelos. Sistemas críticos. Engenharia de Software automatizada.

ABSTRACT

BALBINOT, Mateus R. P. Desenvolvimento De Sistemas Críticos Com Geração Automática De Código E Garantias De Qualidade. 2016. 117 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

The development of large, complex, and with multiple requirements computer systems, generally requires a greater ability from a programmer. Considering the limitations of human reasoning about large and high complexity problems, software coding tends to incorporate errors. On critical systems, such as an air traffic control system, for example, errors can be fatal and, therefore, are unacceptable. At the same time, this kind of system involves a long and complex chain of factors such it difficult an error-free programming. An alternative that can be used to facilitate this task involves the use of models that provide a more macro vision of the system, such that programming can be optimized. Finite Automata are models that describe the dynamics of a system based on the concept of events. Using these models one can express the behavior of many hardware and software elements. In this work, automata are used to represent an air traffic system. It is shown that the approach can automatically obtain the operating sequences of the air traffic control in order to rule out any possible sequence of events not covered by the imposed requirements.

Keywords: Model-Driven Develepment. Critical systems. Automated software engineering.

LISTA DE FIGURAS

LISTA DE SIGLAS

CASE	*Computer-Aided Software Engineering*
UML	*Unified Modeling Language*
IEEE	*Instituto de Engenheiros Eletricistas e Eletrônicos*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo será apresentada a introdução com considerações iniciais, objetivos, justificativa e estruturação do trabalho.

1.1 CONSIDERAÇÕES INICIAIS

Na abordagem convencional de desenvolvimento de software, as etapas de análise são seguidas pelas etapas de implementação, quando um programador, ou uma equipe de programadores, codifica o sistema final conforme sua natureza e requisitos. Embora essa seja uma abordagem amplamente usada na prática, é inevitável observar que o processo que leva ao produto final (software) consiste em uma atividade cuja evolução é altamente dependente dos conhecimentos da equipe de desenvolvimento.

Ainda que, tais equipes sejam especializadas e capazes de conduzir o processo de desenvolvimento de maneira aceitável, é inegável que seres humanos são propensos a erros e, não raramente, eles surgem, sendo observados, sobretudo, quando o sistema está em operação (BEZERRA *et al*, 2015).

Em sistemas complexos, cujos requisitos envolvem aspectos críticos de segurança, confiabilidade, desempenho, concorrência, etc., a abordagem convencional de desenvolvimento de software pode não representar a melhor escolha de projeto, já que a constatação tardia de eventuais problemas no software pode ocasionar em maiores desastres, envolvendo vidas por exemplo. Tais sistemas são, caros, complexos de serem desenvolvidos, críticos de serem testados e, para eles, erros operacionais são inadmissíveis.

Em geral, um projeto envolvendo o desenvolvimento de um sistema dessa classe, complexa, é associado a uma etapa sólida de testes e verificações. Ou seja, primeiramente o sistema é desenvolvido, para que suas propriedades sejam então testadas a *posteriori*. Obviamente, essa verificação pós-implementação implica custo de projeto.

Nesse trabalho, será apresentada uma alternativa à abordagem convencional de desenvolvimento de softwares. Fazendo uso de conceitos e ferramentas de modelagem que dão ênfase aos eventos, será mostrado que um sistema crítico (código) final pode ser obtido de maneira automatizada, a partir de versões de modelos implementados com alto nível de abstração. Será mostrado, ainda, que tal produto final detém certas garantias de qualidade, por exemplo: paralelismo, concorrência, restrições comportamentais e assim por diante, que não precisam ser verificadas *a posteriori*, ou seja, elas são garantidas por construção, a partir do uso de métodos e operações formais. Isso tende a reduzir o custo de projeto substancialmente. Outra contribuição estimada para o trabalho é que, na abordagem proposta, a forma de desenvolvimento facilita o raciocínio sobre requisitos complexos, o que pode ser decisivo para o caso de sistemas de grande porte e de natureza crítica.

O exemplo de um sistema (parcial) de tráfego aéreo é usado nesse trabalho para ilustrar o método de desenvolvimento utilizado. Nesse exemplo parte-se de uma visão abstrata do processo e através das regras de composição apresentadas chega-se ao código do sistema. Como requisitos, assume-se que é desejável a manipulação do maior número possível de aeronaves, mas, em contrapartida, as garantias de segurança para o sistema devem ser preservadas. O uso desse exemplo, em particular, é motivado pelo fato de que essa tarefa pode não ser trivial perante a abordagem convencional de desenvolvimento, já que em na abordagem convencional a eficiência do sistema seria altamente dependente do desenvolvedor.

1.2 OBJETIVOS

A seguir serão apresentados os objetivos gerais e específicos do o presente trabalho.

1.2.1 Objetivo Geral

Apresentar uma alternativa à abordagem convencional de desenvolvimento de software, fazendo uso de ferramentas de modelagem e geração de código para a construção de sistemas cujo comportamento reflita o exato cumprimento do conjunto de especificações. A abordagem será testada no contexto de um sistema de controle de tráfego aéreo.

1.2.2 Objetivos Específicos

Como objetivos específicos pretende-se:

- Analisar o processo de controle de tráfego aéreo;
- Modelar o processo e seus requisitos;
- Gerar o código a partir do modelo obtido.

1.3 JUSTIFICATIVA

Programação de sistemas complexos é uma tarefa crítica cuja complexidade depende de múltiplos fatores, tais como o número de componentes, o escopo do ambiente explorado, concorrência, paralelismo, restrições comportamentais, entre outros. Geralmente é desejável que múltiplos componentes interajam uns com os outros e com o ambiente de uma maneira concomitante, compartilhando recursos de um modo permissivo ao máximo e sem bloqueios.

Programar uma estrutura de software que fornece tais propriedades em conjunto pode ser uma tarefa difícil de ser resolvida. Uma alternativa é adotar

estratégias baseadas em modelos, a fim de expressar as propriedades do sistema e seus requisitos.

Nesse caso, as operações automatizadas podem ser processadas através de um modelo, a fim de calcular sub comportamentos de interesse.

Com o presente trabalho será apresentada uma abordagem alternativa, por meio da qual será gerado o código a partir da modelagem feita através de ferramentas apropriadas, que programam algoritmos capazes de garantir um código matematicamente correto, conciso e sem adversidades que comprometeriam o conjunto de especificações exigidas.

1.4 ESTRUTURAS DO TRABALHO

No capítulo 2, Referencial Teórico, será dada uma introdução mais específica para cada assunto abordado no trabalho, contando com os subtópicos: sistema de controle aéreo, sistemas dirigidos a eventos, modelagem, teoria dos autômatos, modelagem do sistema, modelagem dos requisitos, obtenção do sistema final.

No capítulo 3, Materiais e Métodos, serão apresentados os softwares utilizados e suas descrições, e os métodos usados para a realização do trabalho, contando respectivamente com os tópicos: materiais e métodos.

Para o capitulo 4, Resultados, serão mostrados os resultados da modelagem proposta: explicação e detalhes dos modelos e o que cada modelo faz. Será mostrado em um subitem as simulações do autômato.

Por fim, no capitulo 5, Conclusão, será apresentada a Conclusão, onde serão retomadas as vantagens, considerações finais, juntamente com a contribuição do trabalho para a comunidade acadêmica e profissional e para a sociedade como um todo.

2 REFERENCIAL TEÓRICO

No presente capítulo serão apresentados os assuntos necessários para o entendimento do trabalho.

2.1 SISTEMAS DIRIGIDOS A EVENTOS

A maioria dos sistemas, senão todos, compartilham da característica de possuírem uma sequência lógica de execução. Quando essa sequência corresponde a sinais que ocorrem de maneira abrupta ou involuntária, alterando o estado do sistema, diz-se que o sistema é orientado a eventos (MENDES, 1995).

A principal característica dessa classe de sistemas é que a sequência de ações a ser seguida pelo software depende da ocorrência assíncrona de eventos. Tem-se um estado, acontece um evento, o software segue para o próximo estado e assim sucessivamente. Inclusive, pode-se dizer que neste paradigma o software segue um *loop*, de entradas de eventos e mudança de estados. Utilizando o exemplo abordado nesse trabalho, o controle de tráfego aéreo, pode-se constatar este fato, em que um avião pede decolagem, decola, solicita pouso e finalmente pousa, e assim sucessivamente.

Através da orientação a eventos, o entendimento dos processos, suas funções, e até mesmo o desenvolvimento em si são facilitados devido ao maior nível de abstração com que o sistema é visto, dado que é necessário somente identificar os elementos que deverão ser modelados, desprezando as partes desses elementos que podem não ser relevantes em determinado contexto. O arranjo e a integração entre esses elementos, depois de modelados, podem ser obtidos automaticamente.

2.1.1 Modelagem de sistemas computacionais

De forma sucinta, é através da modelagem que um desenvolvedor vê as funções, as sequências, as informações e demais itens necessários acerca do que o software deverá conter e como ele deverá ser programado.

É bastante comum utilizar ferramentas para realizar uma modelagem, e estas ferramentas recebem o nome de ferramentas *CASE* (do inglês, ***C****omputer-****A****ided* ***S****oftware* ***E****ngineering*). Tais ferramentas podem ser subdivididas em várias categorias e classificações, cada qual com sua função, como por exemplo gerência de projeto, controle de versão, edição e ferramentas de prototipagem, dentre outras.

A modelagem, apesar de muitas vezes não receber a devida atenção, é uma das etapas mais importantes do desenvolvimento em razão de esquematizar todo o software, descomplicar a escrita do código e resultar na economia de tempo e outros custos (de desenvolvimento e manutenção, principalmente).

Para realizar a modelagem de um sistema existem diversas técnicas, abordagens e objetivos que devem ser considerados antes do inicio do trabalho. Por exemplo, para a modelagem de um controle aéreo em alto nível, como o aqui proposto, poderia ser usada uma abordagem declarativa baseada em diagrama (Linguagem de Modelagem Unificada - UML, por exemplo) (BEZERRA *et al*, 2015), um modelo de simulação (Redes de Petri, por exemplo) (MORAES, CASTRUCCI, 2001), etc. Um caso particular de diagrama de transição são os autômatos, que basicamente rotulam cada transição do diagrama com "etiquetas" representando os eventos reais do sistema, responsáveis pelas transições entre estados. Autômatos serão apresentados em mais detalhes a seguir.

2.1.2 Teoria Dos Autômatos

Inicialmente, a proposta do uso de autômatos era a de modelar a função do cérebro humano, mas eles acabaram se mostrando extremamente úteis para vários outros propósitos computacionais, como por exemplo (HOPCROFT, 2003, p.2):

- Projetar e verificar o comportamento de circuitos digitais;

- O analisador léxico de um compilador;
- Softwares para examinar grandes corpos de texto, como coleções de páginas da Web, a fim de encontrar ocorrências de palavras, frases ou outros padrões.

O exemplo mais simples que pode ser citado sobre um autômato, é o de um interruptor (figura 1), onde pode ocorrer o evento "pressionar" e o interruptor muda do estado ligado para o estado desligado e vice-versa:

Figura 1 - Exemplo autômato (HOPCROFT, 2003, p.3)

Assim sendo, sabe-se que autômatos finitos são máquinas de estados finitas que aceitam ou rejeitam cadeias de símbolos, gerando um único ramo de computação para cada cadeia. Autômatos, inclusive, suportam a modelagem de especificações que ajustam o comportamento de um sistema conforme seus requisitos. Autômatos suportam, ainda, o processamento de operações para o cálculo de sistemas (HOPCROFT, 2003, p.2).

O método é inspirado em um mecanismo formal de modelagem, fundamentado sobre a teoria dos conjuntos. Nesse formalismo, cada parte do sistema, e seus requisitos, são construídas individualmente e, então, as partes são compostas para resultar no sistema em si. Essa composição é definida de tal forma que, caso um evento esteja sendo desabilitado por algum modelo de requisito, ele é garantidamente desabilitado na composição final.

2.1.3 Modelagem De Sistemas Usando Autômatos

Uma modelagem pode ser feita de diversas maneiras e com vários métodos, dependendo da natureza do sistema a ser desenvolvido. Atualmente é adotada normalmente a Linguagem de Modelagem Unificada (UML), cuja modelagem se baseia na elaboração de vários tipos de diagramas, cada um com sua finalidade no processo de desenvolvimento de software (ex. identificação de requisitos, definição de classes e seus atributos e métodos). Citando caso análogo, o diagrama de classes que é um diagrama indispensável para representar a arquitetura do sistema. Mas para realizar a modelagem de um sistema, tendo em vista os objetivos deste trabalho, serão usados os autômatos por permitir representar de maneira clara sistemas que possuem a sua dinâmica dirigida a eventos, além de garantirem certas propriedades de segurança no sistema final, possibilitarem a geração automática de código e etc.

Formalmente, um autômato é uma 5-tupla $(\sum, Q, q°, Q^w, \rightarrow)$ em que:

- $\sum$: é o conjunto de eventos ou alfabeto;
- Q: é o conjunto de estados;
- $q°$: é o estado inicial do autômato;
- Q^w: é o subconjunto de estados marcados (tarefas completadas);
- $\rightarrow \subseteq Q \times \sum \times Q$ representa a relação de transição que leva de um estado, para outro estado, com uma transição etiquetada por um elemento de $\sum$.

Neste trabalho, primeiramente será usado um autômato para modelar somente o sistema em si, ou seja, todas as sequências possíveis em cada elemento do sistema. Posteriormente, definem-se, também por autômatos, os seus requisitos de segurança, justiça, confiabilidade, etc.

A intenção é ilustrar o que o sistema deverá fazer, mas não como deverá fazer porque os requisitos são responsáveis por implementar tais restrições sobre o sistema. Essa é a visão mais abstrata, e simples, possível do sistema (que contém sua estrutura e organização).

Tem-se um exemplo de modelagem por autômato abaixo, na Figura 3.

Figura 2 - Exemplo de modelagem

Pela figura 2 observa-se que os círculos representam os estados e que os estados iniciais estão marcados. A partir do momento que ocorrer um dos eventos (a ou b, c ou d) os autômatos irão assumir outro estado. Por exemplo, o autômato AB pode representar o acelerador de um carro, cujos estados seriam solto (estado inicial) e pressionado, e os eventos seriam pressionar e soltar, enquanto o autômato CD poderia representar a embreagem com os estados: solto (estado inicial) e pressionado, e os eventos seriam pressionar e soltar também.

No caso proposto, irá acontecer um evento a no estado inicial e somente depois um b. Em paralelo, acontecerá um c e somente depois um d. Nota-se que os comportamentos representados pelos dois modelos são paralelos e assíncronos, ou seja, eles são independentes e podem ocorrer a qualquer tempo, em qualquer ordem. Respeitando apenas o fato de que um b ocorre somente após um a, e um c ocorre somente após um b. Mas se houver necessidade que outra a ordem seja adicionalmente garantida (exemplo: sequência a, b, c, d) ou que seja seguida uma ordem diferente (exemplo: sequência a, c, b, d), então um ou mais requisitos terão de ser implementados e combinados com tais modelos para restringi-los e adequá-los a tais comportamentos.

2.1.4 Modelagem De Requisitos Usando Autômatos

Após a coleta de requisitos e antes da modelagem destes requisitos, é necessário definir os requisitos em si, através de análise. A análise de requisitos consiste em: reconhecer o problema, avaliar o problema e propor sua solução, modelar, especificar os requisitos e revisão. Os requisitos ainda podem ser divididos em: requisitos do projeto, produto, funcionais e não funcionais (PRESSMAN et al, 2016). Ainda, a obtenção de tais requisitos pode ser feita de algumas maneiras,

como por exemplo: entrevista, *brainstorming*, questionário e pesquisa ou observação.

Segundo a IEEE (1990), a análise de requisitos é um processo que envolve o estudo das necessidades do usuário para se encontrar uma definição correta ou completa do sistema ou requisito de software.

Para a abordagem declarativa (KULESZA, 2010), é com a modelagem dos requisitos que o sistema começa a funcionar como deve, e todas as propriedades detectadas e modeladas começam a ser aplicadas. Logo, é nessa etapa que o sistema propriamente dito começa a tomar forma, e, de todas as sequências operacionais possíveis no modelo do sistema, apenas uma parte delas sobrevive. Claramente, é necessário que os requisitos estejam devidamente modelados para não haver discrepâncias, em relação ao que se espera do sistema.

Utilizando o esquema criado no item anterior (Figura 2), supõe-se que é necessário que aconteça o evento c somente depois de acontecer o evento b, ou seja, o autômato CD opera apenas após a conclusão do autômato AB. Então o estado inicial do autômato será obrigatoriamente com o evento "a" e, no estado inicial, c deve ser proibido. Somente este requisito pode mudar todo o comportamento do sistema quando em operação. Um modelo para esse requisito é apresentado na sequência.

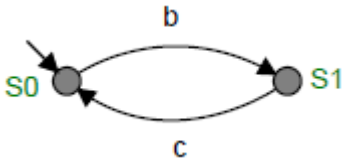

Figura 3 - Modelagem do requisito

Através do modelo na figura 3, já se pode obter o inicio obrigatório com o evento a, bloqueando a execução de c antes de b. Esse modelo também impede 2 ocorrências consecutivas de b sem que haja um c.

2.1.5 Obtenção Do Sistema Final

Após a conclusão da modelagem do sistema e da modelagem de seus requisitos, é necessário realizar a combinação dos modelos. Em autômatos, uma

forma de combinar dois ou mais modelos é por composição síncrona, denotada por || e definida a seguir. Sejam dois autômatos:

$$A = < \Sigma_A \quad , Q_A, x_A^{\circ}, Q_a^{\omega}, \rightarrow_A >$$

e

$$B = < \Sigma_B \quad , Q_B, x_B^{\circ}, Q_B^{\omega}, \rightarrow_B >.$$

A composição A||B gera um único elemento (autômato) construído conforme a seguinte estrutura:

$$A||B = < \Sigma_A \cup \Sigma_B,\ Q_A \times Q_B, (q_A^{\circ},\ q_B^{\circ}), Q_A^{\omega} \times Q_B^{\omega},\ \rightarrow >,$$

Nessa composição, os conjuntos de eventos dos modelos compostos são unidos. Já os estados são combinados por produto cartesiano, para que a composição reflita exatamente todas as possibilidades de evolução paralela dos modelos que estão sendo combinados. Os estados iniciais dos modelos compostos são juntados em um único estado. Por fim, a estrutura de transição é definida de modo que (TEIXEIRA *et al*, 2015):

$$-(q_A, q_B) \xrightarrow{\sigma} (q'_A, q'_B) \text{ se } \sigma \in \Sigma_A \cap \Sigma_B;$$

$$-(q_A, q_B) \xrightarrow{\sigma} (q'_A, q_B) \text{ se } \sigma \in \Sigma_A \setminus \Sigma_B;$$

$$-(q_A, q_B) \xrightarrow{\sigma} (q_A, q'_B) \text{ se } \sigma \in \Sigma_B \setminus \Sigma_A.$$

Ou seja, se a partir de um par de estados, um em A e um em B, um mesmo evento é possível (pertence à intersecção dos alfabetos), então a transição é fundida em uma mesma transição. Do contrário, caso o evento seja habilitado por apenas um dos modelos A ou B, então ele é proibido de ocorrer na composição, que assim evolui apenas com as sequências possíveis nos dois modelos. Essa é a forma como uma composição pode refletir um requisito que proíbe um evento de ocorrer.

Retomando o esquema dos itens anteriores, a composição do sistema sem requisitos será como mostrado na Figura 4:

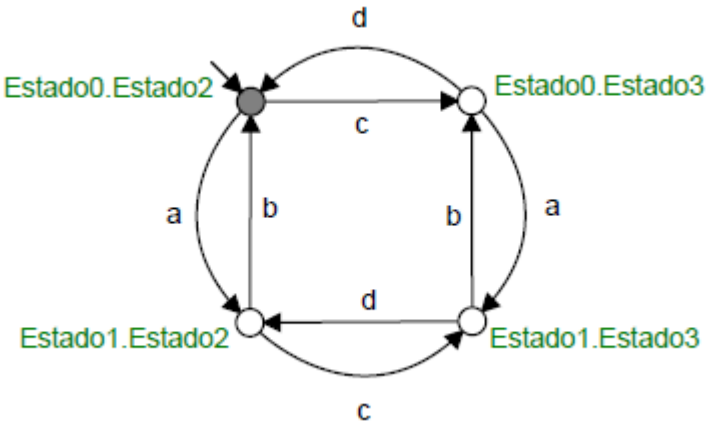

Figura 4 - Composição do esquema, sem o requisito

Nota-se que a composição habilita no estado inicial a ou c, o que é esperado do sistema em paralelo. Mas após o requisito ser incluso na composição (que sequêncializa os componentes), será obtido o seguinte resultado, conforme mostrado na Figura 5, obrigatoriamente iniciando com o evento a:

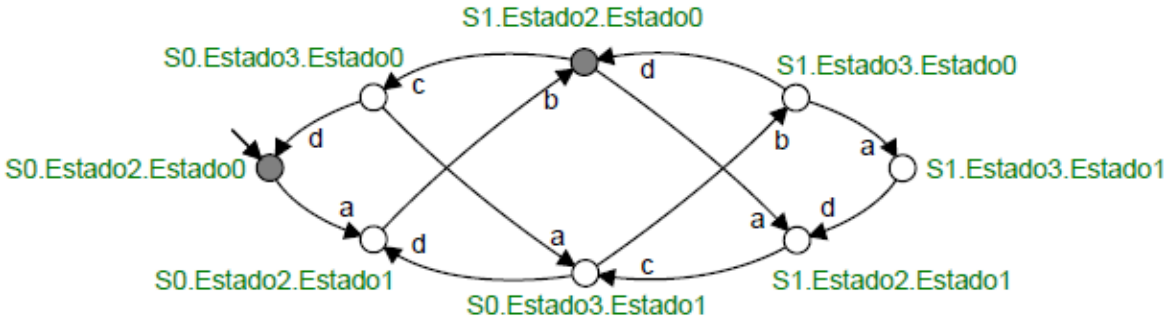

Figura 5 - Composição final com o requisito

Note que no estado inicial, agora apenas o evento a é possível. Esse modelo reflete, portanto, um comportamento com todas as sequências possíveis de operação do sistema, exceto aquelas que levariam a violar o requisito proposto. Então, essa estrutura pode ser usada para gerar o código implementável do sistema.

Atualmente, há softwares capazes de gerar o código com base nesses autômatos, e para o trabalho proposto, o sistema e seus requisitos serão modelados, simulados, combinados e, por fim, o código será obtido de forma automatizada, tendo a possibilidade de se realizar ajustes conforme necessidades ou preferências do programador. Para a modelagem, simulação e composição será usado o Supremica (AKESSON, FABIAN, et al., 2014), e para a obtenção de código será

usado o Deslab (TORRICO, 2016), que suporta a exportação para código C compatível com implementações em microcontroladores. Mas, para alguns tipos de códigos, como é o caso do XML, o próprio Supremica pode ser usado para a conversão.

O código gerado garante a construção de um sistema que opera de maneira maximamente paralela (por uma questão de aproveitamento de recursos e desempenho) e que respeita formalmente o conjunto de requisitos.

3 MATERIAIS E MÉTODO

Neste capitulo serão apresentadas as ferramentas utilizadas para a realização da modelagem, composição e geração do código do exemplo avaliado.

3.1 MATERIAIS

Ferramenta	**Finalidade**	**Descrição**
Supremica	Ferramenta para montar os modelos.	Supremica é uma aplicação desenvolvida com base na plataforma Java e é distribuída gratuitamente. Dentre outras, esta ferramenta é utilizada para modelar processos, circuitos, especificações e sistemas por meio da utilização de autômatos (MENEZES, 1998).
DESLAB	Ferramenta para gerar o código.	Em seu manual (TORRICO C., 2016), é descrito como "uma ferramenta funcional que permite realizar operações básicas com autômatos, a síntese de supervisores dentro da teoria clássica, e a geração de código para microcontroladores"

3.2 MÉTODOS

Preliminarmente a modelagem, foram coletados os requisitos através do conhecimento comum já adquirido, incluindo sites e livros para consultas de regras que sejam obrigatórios (como acontece com os requisitos de R1 a R4, citados no na primeira sessão do capitulo 4). Para esta etapa, os requisitos poderiam ser obtidos através de técnicas convencionais, *brainstorming* por exemplo.

Após o processo de análise geral do sistema e de requisitos, foi iniciada a modelagem propriamente dita. Inicialmente foram modelados os dois aviões: colocados os nós, simbolizando os estados, as arestas indicando os eventos e

vinculando os eventos às arestas. Após a modelagem dos dois aviões, são unidos os autômatos, compondo-os de forma a obter um único autômato contendo todas as transições que representam o sistema.

Ainda, um terceiro componente foi necessário: a torre de comando. O modelo da torre foi preciso para representar as ações possivelmente tomadas pela torre. Da mesma forma, a torre foi modelada fazendo uso dos nós e arestas, porém um único nó foi usado e empregando o conceito de *auto-loop* nas arestas.

Com a modelagem dos aviões concluída, a próxima etapa é a modelagem dos requisitos, para que o sistema funcione de maneira que haja as propriedades de segurança, exclusão mútua da pista e comando da torre (a torre decide quem autorizar para pousos ou decolagens). Os requisitos foram modelados semelhantemente aos aviões, colocando os nós e arestas e vinculando os eventos às arestas.

Finalmente com os aviões e requisitos devidamente modelados, foi feita a composição de todos os autômatos, resultando no sistema final. Com a composição final pronta, a mesma foi exportada para um arquivo no formato XML que contém todas as transições. As transições foram inseridas manualmente no Deslab e o mesmo pode gerar automaticamente o código.

4 RESULTADOS

Nesta seção são apresentados os resultados obtidos com este trabalho.

4.1 ANÁLISE DO PROCESSO DE CONTROLE DE TRÁFEGO AÉREO

Com este trabalho foi desenvolvido um sistema de controle de trafego aéreo, cujo enfoque foi na coordenação de decolagens e pousos. O processo pode ser resumido no contexto de 3 passos básicos:

1. Um avião faz uma solicitação (decolagem ou pouso) para a torre,
2. A torre autoriza
3. O avião realiza a ação

A pista é assumida ser única, enquanto a quantidade de aviões é maior do que 1. Ou seja, o trabalho explora o fato de que o recurso (a pista) é compartilhado com inúmeros usuários (aviões), o que tende a aumentar a complexidade na coordenação dos elementos do sistema.

Então, foi feito um levantamento dos requisitos reais que um aeroporto pode conter. Foram selecionados seis deles, que foram considerados os mais relevantes sob o ponto de vista de segurança e levando-se em conta o fato de que os dois componentes (aviões) competem pelo recurso (pista). Entende-se que esses requisitos servem para o alcance dos objetivos do trabalho, muito embora outros requisitos possam ser incorporados em trabalhos futuros. Os requisitos assumidos são:

R1. A1 (Avião 1) só decola depois de autorizado pela torre
R2. A1 só pousa depois de autorizado pela torre
R3. A2 (Avião 2) só decola depois de autorizado pela torre
R4. A2 só pousa depois de autorizado pela torre
R5. Exclusão mutua da pista para pousos e decolagens
R6. Desabilitar comando da torre enquanto não houver demanda

Abaixo tem-se a figura (figura 6) ilustrativa do processo real:

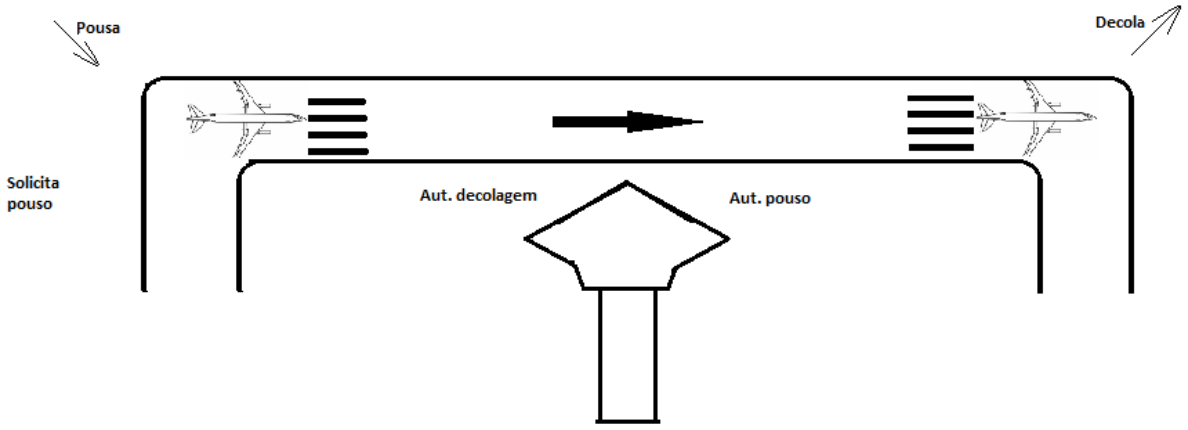

Figura 6 - Ilustração da situação usada como exemplo

Note que a pista suporta pousos e decolagens de múltiplas aeronaves. Na ilustração, o avião da esquerda está em processo de pouso, enquanto o da direita acaba de deixar a área de exclusão mútua, em um procedimento de decolagem. Esses eventos são coordenados pela torre ilustrada no centro da figura.

É interessante enfatizar que este sistema de tráfego aéreo será somente o meio para demonstrar a modelagem e aplicação de autômatos no desenvolvimento de sistemas complexos, porém pode-se utilizá-los para vários outros fins, inclusive para a coordenação de ambientes aeroportuários mais extensos, como em geral é o caso.

O escopo do sistema foi abstraído o máximo possível, a fim de preservar a viabilidade do trabalho e sua melhor compreensão. Para fins didáticos e sem perda de generalidade, foi assumido que a quantidade de aviões é igual a 2, muito embora a abordagem seja naturalmente extensível a uma quantidade maior de aviões e de requisitos. O tempo entre uma solicitação e autorização também não é considerado, uma vez que o trabalho objetiva coordenar as sequências operacionais, e não o momento em que elas ocorrem. Assim, somente os eventos em si (decolagem e pouso) são observados, já que esse é o fator que importa para a tomada de decisão na ação.

4.2 ALTERNATIVA PARA A ABORDAGEM CONVENCIONAL DE DESENVOLVIMENTO

O uso de autômatos para a modelagem de sistemas complexos se mostra promissora, reduzindo custos, tempo de modelagem e codificação, complexidade de compreensão e desenvolvimento, dentre outros aspectos.

Após a modelagem dos estados e suas ligações, especificações, é possível utilizar softwares para gerar uma base de código contendo toda a estrutura básica de variáveis e condições *if-else*: considerando que autômatos admitem somente um estado por vez, dependendo do evento acontecido.

Esta forma de se realizar modelagem se dá focando os estados e os eventos que levam a mudança para outro estado. "*Estacionado*", por exemplo, é um dos estados do avião, e ao ocorrer o evento "*askDec*" (pedido de decolagem), passa ao estado "*PrepararDecolagem*" (aguardando autorização da torre de comando para decolar de fato).

4.3 MODELAGEM DO PROCESSO E SEUS REQUISITOS

Para a coordenação dos dois aviões, foi feito inicialmente o modelo dos dois aviões, contendo todas as sequências operacionais possíveis.

Os aviões foram modelados de forma que o processo comece quando ele pede autorização para a torre de comando para decolar. O processo sempre segue a mesma ordem dos eventos: solicita decolagem, decola, solicita pouso, pousa, conforme a Figura 7:

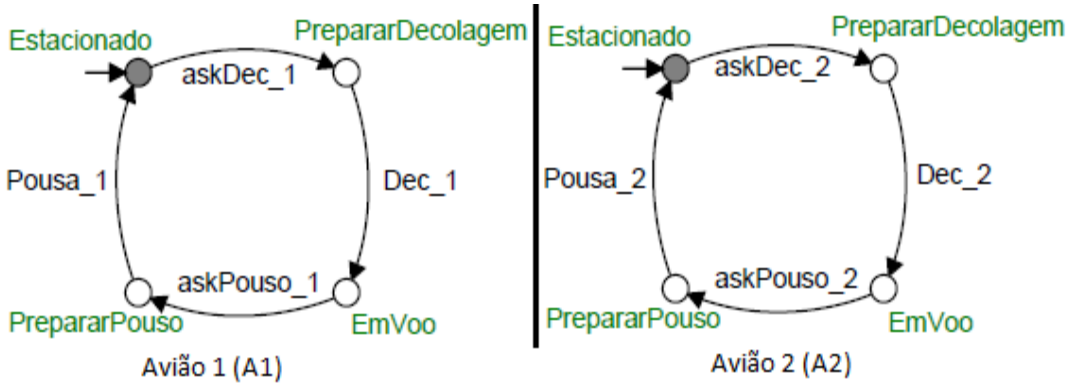

Figura 7 - Modelagem dos aviões

Observa-se que “*Estacionado*” é o estado inicial, “*PrepararDecolagem*” é o momento em que o avião está esperando autorização para decolar, “*EmVoo*” indica que o avião está no ar e, finalmente, “*PrepararPouso*” representa o momento que o avião aguarda pela autorização de pouso.

Observa-se também os eventos indicando a ação tomada. A seguir, descrevem-se os eventos presentes nos modelos

Evento	Função
askDec_1 e askDec_2	Pedido de decolagem
autDec_1 e autDec_2	Autorização da decolagem
Dec_1 e Dec_2	Decolagem
askPouso_1 e askPouso_2	Pedido de pouso
autPouso_1 e autPouso_2	Autorização do pouso
Pousa_1 e Pousa_2	Pousar

Tabela 1 - Eventos e suas funções

Ademais, foi realizada a modelagem da torre (T) de comando, resultando no seguinte autômato (figura 8):

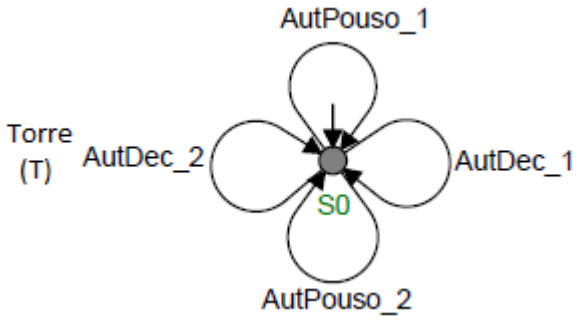

Figura 8 - Modelo da torre de comando (T)

A torre é composta pelos 4 eventos de autorização. Eles foram modelados fazendo auto-loop para que, além de evitar o travamento do autômato, seja feita a tomada de decisão para a autorização tanto da decolagem quanto do pouso de algum dos aviões, desta forma emulando o comportamento real de uma torre de comando.

Devido ao fato dos eventos serem compartilhados entre os modelos, quando um evento está em execução em um determinado autômato, o mesmo evento está em execução em outro autômato. Com a torre acontece o mesmo, porém neste

modelo não foi necessário usar mais de um estado dado que a torre não tem mudança de estados, somente desempenhando a função de autorizar pousos e decolagens.

O modelo completo do sistema, que reflete seu comportamento perante a ausência de controle, pode ser obtido pela composição S = A1 || A2 || T. Agora restam modelar os requisitos capazes de coordenar o modelo S da maneira desejada e segura.

Com os requisitos R1, R2, R3 e R4 (figura 8), têm-se que tanto A1 quanto A2 podem somente decolar ou pousar após autorização pela torre.

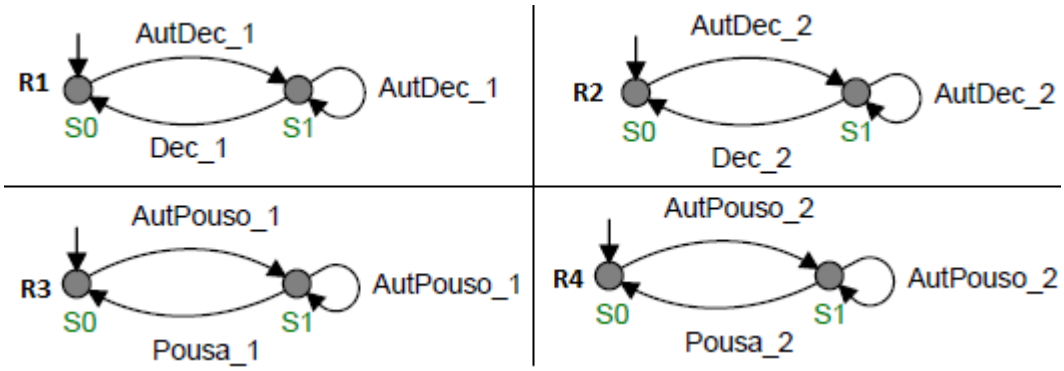

Figura 9 – Modelagem dos requisitos R1, R2, R3 e R4

Os requisitos R1, R2, R3 e R4 definem uma ordem: primeiro a autorização da ação e depois a realização da ação, cumprindo assim as funções definidas para os requisitos. Tais ações são coordenadas através da torre de comando e a mesma fica encarregada de decidir quando autorizar. O auto-loop nos estados “S1” são necessários para prevenir bloqueios de eventos na composição final, pois expressam uma ação minimamente restritiva. Tais eventos são compartilhados com a torre.

Pelo quinto requisito (R5) (figura 10), similarmente é implementada a propriedade de exclusão mútua da pista, ou seja, impede que mais de um avião decole ou pouse ao mesmo tempo.

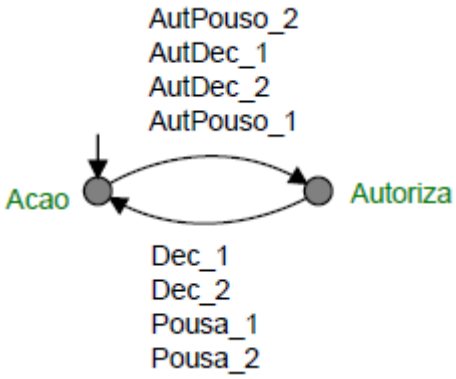

Figura 10 – Modelagem do requisito R5

O autômato do requisito R5 deixa a pista bloqueada caso a mesma já estiver sendo usada. Como os eventos são compartilhados entre os autômatos, os pedidos de pousos ou decolagens irão ficar ativos neste autômato, mas somente será permitida a liberação de um avião por vez devido a mudança de estados. Por exemplo, os dois aviões solicitam pouso e o avião 1 foi autorizado primeiro, o estado deste modelo irá mudar para "*Autoriza*", assim sendo, o outro avião ficará automaticamente em espera.

Através do sexto requisito (R6), o qual é subdividido em 4, conforme cada tipo de ação possível pretende-se desabilitar o comando da torre enquanto não houver demanda. Funciona de forma a prevenir que a torre fique sobrecarregada, fazendo com que ela sempre esteja disponível para desempenhar sua função, que é escolher qual avião autorizar.

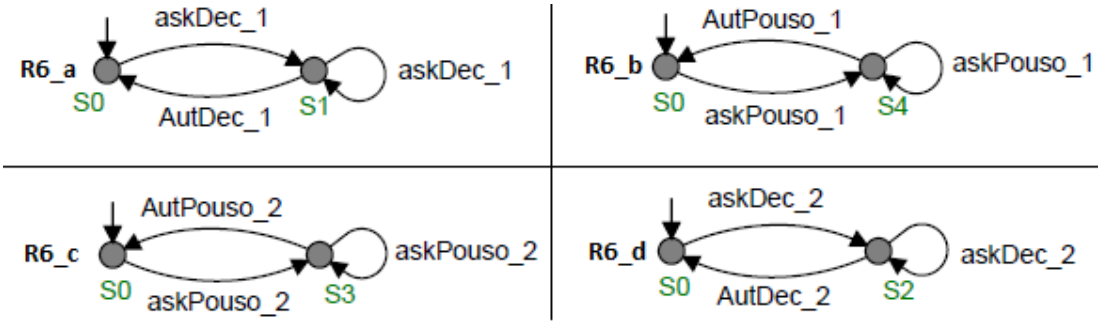

Figura 11 - Modelagem requisito 6

Observa-se que, com o requisito R6 = R6_a || R6_b || R6_c || R6_d, não há funcionamento quando não houver uma solicitação (proibição de autorizações no estado inicial), afinal, é necessário que primeiro venha à solicitação. Caso não houvesse esse requisito, a torre não teria a liberdade para escolher qual avião

autorizar antes, sendo que se sabe que a torre de comando possui total autoridade sobre quem proporcionar pouso ou decolagem.

Por fim, pode-se obter uma versão completa do modelo dos requisitos por meio da composição R = R1 || R2 || R3 || R4 || R5 || R6. Quando associado ao modelo do sistema, esse requisito R o ajusta ao comportamento esperado. Esse resultado pode ser obtido por meio da composição K = S || R, em que S é o modelo do sistema e R é o modelo dos requisitos. O autômato K pode então ser usado para a geração do código final do sistema.

4.3.1 Simulação dos resultados sem requisitos

Pelo Supremica foi possível realizar os testes dos modelos, e verificar se os estados e eventos estavam sendo executados conforme desejado.

Primeiramente foram realizados testes somente com os autômatos dos aviões, conforme figura 12.

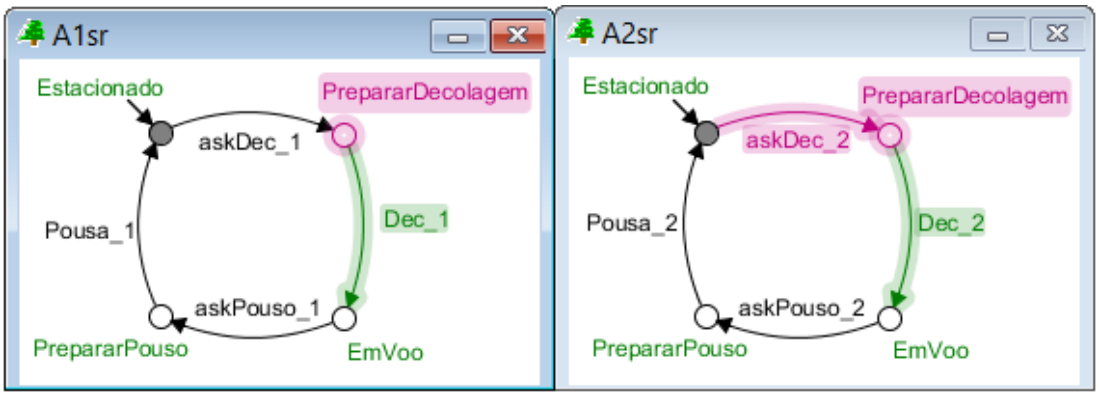

Figura 12 - Simulação sem uso dos requisitos

Pode-se constatar que sem o uso dos requisitos o sistema liberaria qualquer estado e evento a qualquer momento, como na imagem acima (Figura 12), na qual os dois aviões estariam decolando ao mesmo tempo, resultando em um acidente.

4.3.2 Simulação dos resultados com requisitos

A partir do momento que os requisitos são incluídos, já temos o impedimento de determinados estados (figura 13).

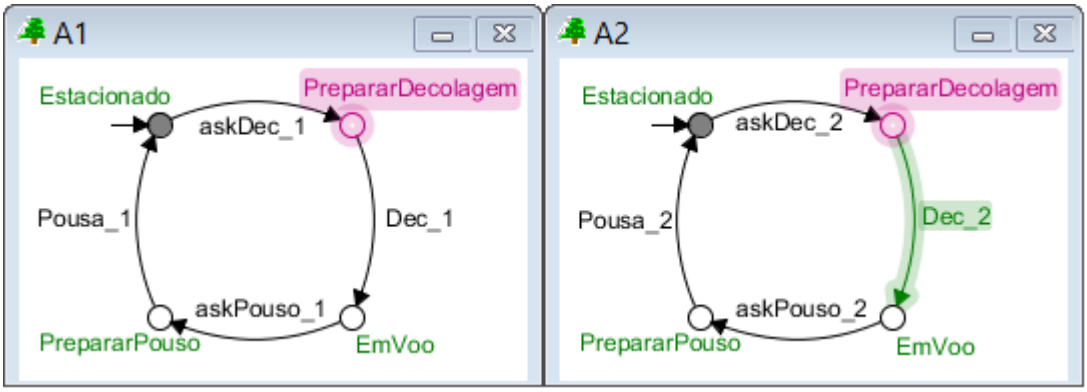

Figura 13 - Simulação com aplicação dos requisitos

É possível certificar-se que enquanto o avião dois (A2) não decolar, o avião um (A1) fica em espera até que A2 esteja de fato em voo. As próximas figuras (figuras 14, 15 e 16) mostram como os autômatos dos requisitos se comportam diante deste fato.

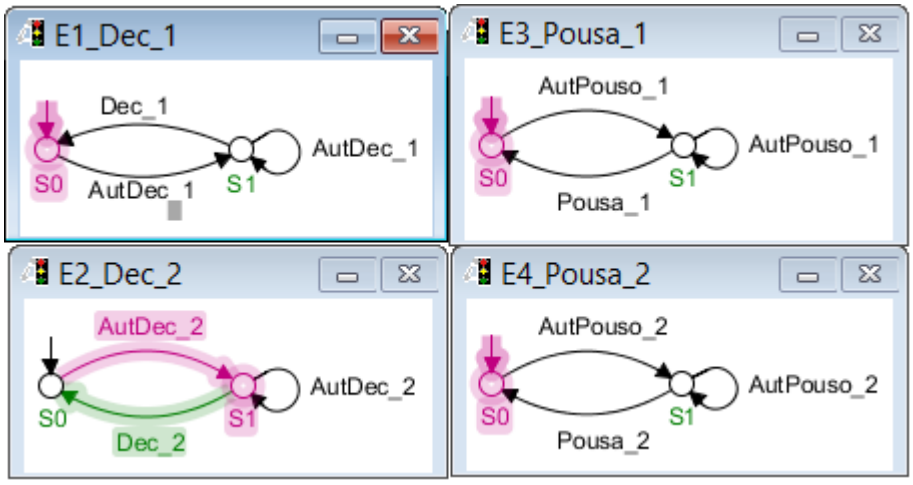

Figura 14 - Especificações 1, 2, 3 e 4 no mesmo momento de simulação dos aviões

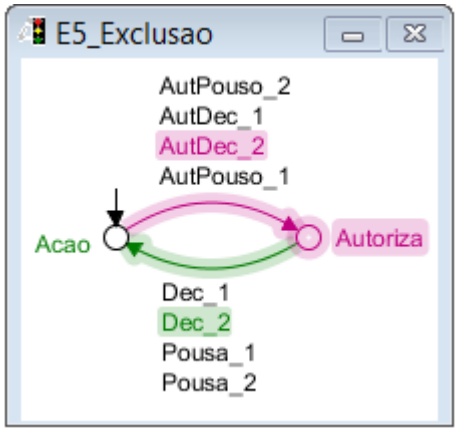

Figura 15 - Especificação 5 no mesmo momento de simulação dos aviões

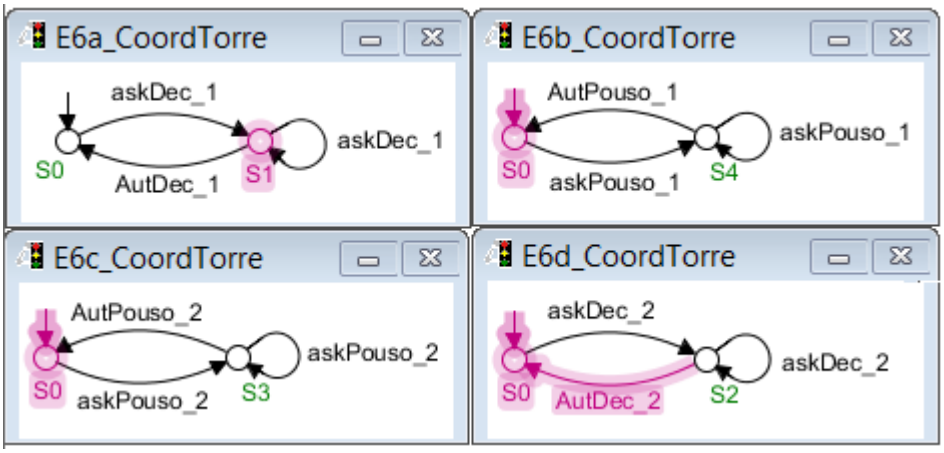

Figura 16 - Especificação 6 no mesmo momento de simulação dos aviões

A simulação mostra que os resultados desejados foram obtidos. A imagem da composição total não será mostrada devido ao número de estados gerados, mas a visualização individual de cada autômato não altera a verificação do sistema.

4.4 GERAÇÃO DO CÓDIGO A PARTIR DO MODELO OBTIDO

Conforme comentado anteriormente, é possível obter um código a partir dos modelos feitos. Para chegar a este código, primeiro foi necessário exportar os modelos para um arquivo XML para obter todas as entradas, os eventos subsequentes e as saídas por escrito.

A estrutura do arquivo XML é dividida em eventos, estados e transições. Para a seção dos eventos tem-se os eventos somente, descritos em *tags* como, por exemplo:

“*<Event label="AutDec_1" id="0"/>*”

A seção dos estados comporta a combinação dos estados de todos os autômatos incluídos na composição, e suas *tags* são escritas desta maneira:

<State name="EmVoo.EmVoo.S0.S0.S0.S0.Acao.S0.S0.S0.S0" id="0"/>.

Na seção das transições estão todos os eventos de transição, contendo um evento, uma saída e uma entrada, por exemplo:

<Transition event="11" dest="4" source="0"/>

Foi utilizada somente a seção das transições para inserção dos dados no programa Deslab.

Com o software Deslab, foi criado um novo projeto, inseridas as entradas, eventos e saídas (figura 17) obtidas do arquivo XML proveniente do Supremica.

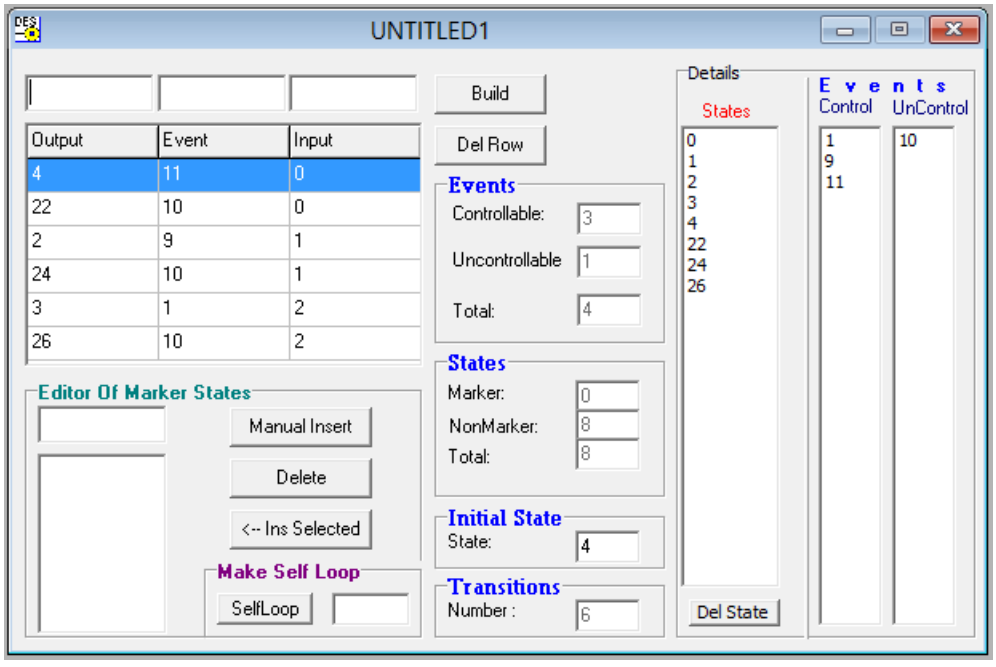

Figura 17 - Inserção das entradas, eventos e saídas de estados no Deslab

E então foi permitido obter o obter o código na linguagem C (figura 18). O Deslab reconhece os estados e eventos, sendo assim possível obter um algoritmo base confiável.

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;       //Desabilita o WDT
    config_clk();               //configura o clk
    config_timer();         //configura o timer
    config_io();                //configura entradas e saídas

    unsigned int k;
    char occur_event;           //Evento ocorrido
    unsigned int current_state = 22;    //Estado atual inicializado com estado inicial
    int g=0;            //Flag para gerador aleatório de eventos
    _enable_interrupt();        //Habilita interrupção geral

    while(1)
    {

        if(n_buffer == 0)//se não existir evento no buffer então gerar um evento interno(evento controlável)
        {
            if(TACTL&TAIFG) //Se o timer estourar, habilita a geração de eventos
            {
                gerar_evento=1;
            }
            if(gerar_evento==1)
            {
                switch(g)   //Aqui é implementado um gerador automático de eventos controláveis
                {
                    case(0):
                        occur_event=1;
                        g++;
                        break;
                    case(1):
                        occur_event=3;
                        g++;
                        break;
                    case(2):
                        occur_event=5;
```

Figura 18 - Parte do código gerado através do Deslab, na linguagem C

Nota-se que o código gerado simplesmente reflete a estrutura de uma máquina de estados, fazendo o uso de *IF-else* e *switch-case*, porém não há modificações, já que o objetivo é somente mostrar o código. Inclusive o próprio Deslab insere comentários pelo algoritmo, explicando o que cada parte do código deve fazer e assim facilitando possíveis alterações e ajustes necessários. Ressalta-se ainda que com a quantidade de estados gerados, a economia de tempo obtida e a confiabilidade deste código podem ser fatores decisivos para a escolha desta abordagem.

5 CONCLUSÃO

Ainda que seres humanos estejam propensos a fatores que contribuem para a ocorrência de erros na codificação de um software, a abordagem declarativa com o uso de autômatos proposta neste trabalho diminuiu a probabilidade de tais ocorrência ainda na etapa de modelagem, devido a própria metodologia que induz a maior abstração do modelo e consequentemente facilita o processo

Enquanto a modelagem de um sistema é deixada de lado por alguns desenvolvedores, é dada uma importância maior por outros. Para os sistemas complexos, a modelagem é imprescindível para a compreensão do sistema como um todo além de facilitar o processo de codificação do software. Através de metodologias formais também há garantia de que as propriedades do sistemas serão implementadas e estarão funcionando totalmente, já que metodologias formais de desenvolvimento “são técnicas baseadas em formalismos matemáticos para a especificação, desenvolvimento e verificação dos sistemas de softwares e hardwares” (SOUSA, 2013).

Devido a grande maioria dos sistemas serem comerciais e industriais ser menos utilizada na localidade onde o mesmo foi realizado, a proposta de abordagem deste trabalho praticamente não é usada, que já fazem uso de outras linguagens e metodologias. A intenção é justamente contribuir com sua difusão na comunidade acadêmica, e também contribuir para as empresas com uma visão de desenvolvimento diferenciada, na qual o principal objeto de trabalho são os eventos. Aqui a utilização da teoria dos autômatos auxilia de tal forma que é possível obter um melhor nível de abstração, aperfeiçoar o tempo de desenvolvimento, aperfeiçoar custos e principalmente aperfeiçoar a qualidade do sistema a ser desenvolvido.

Finalmente, conclui-se que, não só em sistemas complexos, mas principalmente neles, a modelagem é o processo mais importante, afinal, por esta etapa é possível obter um software operativo de forma que suas propriedades e suas funções sejam preservadas, operando da forma mais próxima possível do esperado.

REFERÊNCIAS

AKESSON K., FABIAN M., FLORDAL H., MALIK R., VAHIDI A., SKOLDSTAM M., CENGIC G., **Supremica**, 2014. Disponível em: <http://www.supremica.org/> Acesso em 09 de mai de 2016.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 2. ed. rev. atual. Rio de Janeiro, RJ: Elsevier, 2015. 416 p.

HOPCROFT, John E.; et al. **Introdução à teoria de autômatos, linguagens e computação.** Rio de Janeiro, RJ: Campus, c2003. 560 p.

IEEE – Institute of Eletricaland Eletronics Engineers. **Standards Glossary of Software Engineering Terminology:** Std 610.12, N.Y.,1990. 84p.

KULESZA, Raoni et al. Ginga-J: Implementação de Referência do Ambiente Imperativo do Middleware Ginga. **WebMedia** 2010, 2010.

MENDES, Rafael Santos. Modelagem e Controle de Sistemas a Eventos Discretos. **Markus, M. e Pires, P.(organizadores): Manufatura integrada por computador. Belo Horizonte, Fundação CEFETMINAS**, 1995.

MENEZES, Paulo Blauth. **Linguagens formais e autômatos**. Sagra-Dcluzzato, 1998. Dísponível em < http://othonbatista.com.br/arquivos/unifacs/formais/aulas/blauth/aula06.pdf>. Acesso em 30 de Nov de 2016.

MORAES, Cícero Couto de; CASTRUCCI, Plínio. **Engenharia de automação industrial**. Rio de Janeiro: LTC, c2001. 295 p.

SUPREMICA. Disponível em: <http://www.supremica.org>. Acesso em 09 de mai de 2016.

PRESSMAN, Roger; MAXIM, Bruce. Engenharia de Software-8ª Edição. McGraw Hill Brasil, 2016. 940 p.

SOUSA, Dyego, **Transcrição de Métodos Formais para Especificação do Software**, 2013. Disponível em <https://prezi.com/cfgcixdsds2l/metodos-formais-para-especificacao-do-software/>. Acesso em 01 de nov 2016.

TORRICO C., **Deslab** 3.6, 2016. Disponível em: <https://sites.google.com/site/controlediscreto9/instaladores>. Acesso em 09 de mai de 2016.

TEIXEIRA, MARCELO; RIBEIRO, RICHARDSON; BARBOSA, MARCO; MARIN, LUCIENE. A Formal Method Applied to the Automated Software Engineering with Quality Guarantees. In: 2014 **IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)**, 2014, Naples. 2014 IEEE International Symposium on Software Reliability Engineering Workshops. p. 108.