Joinville/SC – 26 a 29 de Setembro de 2017
UDESC/UNISOCIESC
"Inovação no Ensino/Aprendizagem em Engenharia"

COBENGE 2017
XLV CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA

C4uC: UMA FERRAMENTA DE GERAÇÃO DE CÓDIGO PARA MICROCONTROLADORES E O SEU USO COMO INSTRUMENTO DE APOIO AO ENSINO DE DISCIPLINAS DE AUTOMAÇÃO

Gabriel Hermann Negri[1] – negri.gabriel@gmail.com
Christian Joezer Meirinho[1,2] – christianmeirinho@gmail.com
Camila Rossi Scalabrin[1] – c.scalabrin@gmail.com
Gabriel Soares Dalpiaz[1] – dalpiaz.gb@gmail.com
André Bittencourt Leal[1] – leal.ab@gmail.com
Mariana Santos Matos Cavalca[1] – mariana.cavalca@udesc.br
[1]Universidade do Estado de Santa Catarina, Departamento de Engenharia Elétrica
Rua Paulo Malschitzki, 200
89.219-710 – Joinville – SC
[2]KaVo
Rua Chapecó, 86
89221-040 – Joinville – SC

Resumo: *Nas disciplinas de automação, os alunos aprendem a programar circuitos lógicos programáveis (CLPs) e microcontroladores e aplicam os métodos de controle em estudos de caso. Entretanto, o desenvolvimento da lógica de controle é usualmente feito de forma empírica, por tentativa e erro. Dessa forma, a formação do engenheiro pode ser prejudicada por não haver uma metodologia formal para resolução de problemas em automação no programa de disciplinas de automação. Para sanar esta deficiência na formação dos alunos, alguns cursos contemplaram o uso da Teoria de Controle Supervisório (TCS) de Sistemas a Eventos Discretos (SEDs) como metodologia formal para a solução de problemas de automação. Tais sistemas, diferentemente de sistemas com dinâmica contínua como circuitos elétricos de primeira ordem e sistemas massa-mola, são modelados por estados e transições, ativadas pela ocorrência de eventos. Entretanto, a geração manual de código para a implementação prática de sistemas supervisórios pode se tornar uma tarefa demasiadamente trabalhosa e desmotivar os alunos a investigarem os aspectos práticos da disciplina. Dessa forma, objetiva-se com o presente artigo, divulgar uma ferramenta computacional de geração de código para microcontrolador Arduino, sendo tal microcontrolador de fácil aquisição e aprendizado. O gerador, chamado C4uC (Código para Microcontrolador), foi utilizado por alunos do curso de Engenharia Elétrica e, no presente artigo, reporta-se a aplicação testada pelos alunos e os princípios de funcionamento da ferramenta. Destacam-se, ainda, as vantagens da utilização do gerador em uma disciplina baseada na teoria de controle supervisório de SEDs.*

Palavras-chave: *Automação, Sistemas a Eventos Discretos, Microcontrolador Arduino, Geração Automática de Código.*

1. INTRODUÇÃO

Usualmente, o termo "controle de sistemas" (ou sistemas de controle) é relacionado ao controle da resposta transiente de sistemas dinâmicos com variáveis contínuas no tempo, cuja modelagem é realizada por meio de equações diferenciais. Este é o caso do controle de corrente em circuitos elétricos utilizando controladores do tipo PID (proporcional, integral e derivativo), por exemplo. Entretanto, nas últimas décadas, um outro tipo de sistema pode ser considerado, cuja dinâmica é regida por operações lógicas, através da ocorrência de eventos discretos. Tais sistemas são chamados Sistemas a Eventos Discretos (SEDs) (CASSANDRAS; LAFORTUNE, 2010). Um exemplo de SED é uma máquina de pagamento via cartão, que deve não somente atuar de acordo com uma sequência lógica (inserir valor, verificar senha e efetuar o pagamento) mas tratar eventos que não estão na sequência de operação desejada, como, por exemplo, a digitação errada da senha ou solicitação de cancelamento durante o processo. Sistemas eletrônicos, computacionais e de automação destacam-se pela quantidade de SEDs em suas aplicações (CASSANDRAS; LAFORTUNE, 2010). A linguagem matemática de autômatos possibilita representar e modelar matematicamente SEDs (MAAS; PINOTTI; LEAL, 2012), tornando possível a aplicação de teorias de controle nesses sistemas automatizados e computacionais.

Automação é uma área da engenharia em constante desenvolvimento com destaque para indústria 4.0 e internet das coisas (WOLLSCHLAEGER; SAUTER; JASPERNEITE, 2017). Dessa forma, é necessário que educadores de engenharia estejam sensíveis à necessidade de adequar o ensino a essas novas tecnologias e nova era do ensino (RICHERT et al., 2016). Aulas teóricas e práticas, similares ao ambiente industrial, tornam-se necessárias para que o estudante esteja capacitado e confiante para atuar no emergente mercado da automação industrial (LEAL et al., 2013). Além das aulas em laboratórios, iniciativas que extrapolam o conhecimento da automação para fora das aplicações industriais permitem uma maior absorção dos conceitos fundamentais da disciplina (ALAYÓN; GONZÁLEZ; TOLEDO, 2013). Entretanto, o desafio do ensino na era da informação é alto. Torna-se muito importante que sejam desenvolvidos métodos para que o conteúdo apresentado seja tão atrativo ao acadêmico quanto outras atividades, como o acesso a mídias sociais (BAO et al., 2015).

Existem diversas ferramentas de simulação e modelagem em computador para autômatos em sistemas industriais (DELIGIANNIS; MANESIS, 2006) e mais recentemente ferramentas online para o ensino da linguagem (CASTRO-SCHEZ et al., 2017; NETO; TERRA, 2016). Há também o desenvolvimento de ferramentas que auxiliam o entendimento de Teoria dos Autômatos Finitos por meio de simulação (JUKEMURA; NASCIMENTO; UCHÔA, 2005), limitadas, entretanto, a cursos de computação. Entretanto, em engenharia não é suficiente que o *software* seja capaz de interpretar linguagens e autômatos. É necessário que os *softwares* permitem a simulação e implementação prática do sistema como um todo, considerando todos componentes, como os sensores, atuadores e a dinâmica da própria planta estudada.

Como visto, já foram desenvolvidas muitas ferramentas para simulação e ensino de

autômatos, porém, poucas para auxiliar a implementação de sistemas de controle supervisório em engenharia. Dessa forma, o presente artigo apresenta uma ferramenta desenvolvida com o propósito de auxiliar e reduzir o tempo de implementação prática de autômatos supervisores no microcontrolador Arduino, que é de baixo custo, fácil aquisição e programação.

Este artigo apresenta inicialmente a disciplina de automação na qual a TCS (Teoria de Controle Supervisório) de SEDs é lecionada. Posteriormente a ferramenta proposta é introduzida e, por fim, um estudo de caso com acadêmicos que utilizaram a ferramenta é apresentado, seguido pelas conclusões.

2. Teoria de Controle Supervisório de SEDs

Nesta seção, apresenta-se de forma resumida a Teoria de Controle Supervisório (TCS) de SEDs e também a disciplina de Laboratório de Automação da Manufatura, ministrada no curso de Engenharia Elétrica da Universidade do Estado de Santa Catarina (UDESC), na qual a TCS é utilizada como metodologia de projeto.

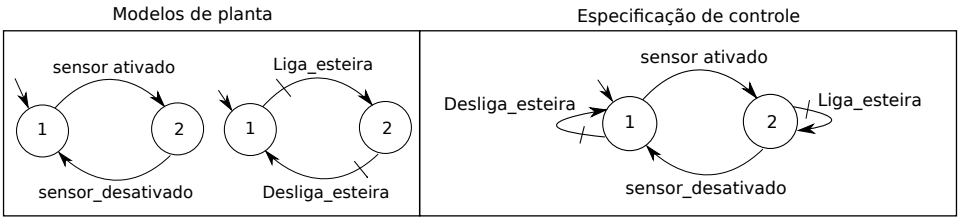
2.1. Apresentação da TCS

Um autômato é um conjunto de estados e regras de transição entre os estados que pode ser utilizado para modelar um sistema a eventos discretos. Um evento discreto, por sua vez, representa o ocorrência de algo que possa modificar a situação atual da planta. Eventos podem ser classificados como controláveis, os quais podem ser diretamente impedidos de ocorrer pela ação de controle, ou não controláveis, que ocorrem livremente na planta. As transições oriundas de eventos controláveis usualmente são marcadas com um traço perpendicular no desenho do autômato. O método utilizado para encontrar um autômato que represente o comportamento simultâneo de diversos autômatos é chamada de composição síncrona (CASSANDRAS; LAFORTUNE, 2010).

A Figura 1 mostra um exemplo projeto de controle para um sistema simples de manufatura. Trata-se de uma esteira, que deve ser ligada quando um sensor indicar a presença de uma peça sobre a esteira e desligada quando o objeto atingir a posição final, desativando o sensor. O problema é dividido, inicialmente, na modelagem da planta (sensor e esteira) e em especificações de controle (apenas uma especificação foi utilizada nesse exemplo). Os modelos de planta apresentam, primeiramente, um autômato para o sensor que, no estado 1 encontra-se desativado. Na ocorrência do evento de ativação do sensor, o autômato transita para o estado 2, de onde só é possível que ocorra o evento de desativação do sensor. Ambos os eventos são não controláveis, pois podem ocorrer a qualquer instante na planta, respeitando a alternância modelada no autômato. O segundo modelo de planta mostra que, inicialmente, no estado 1, a esteira encontra-se desligada e, portanto, o único evento que pode ocorrer nesse estado é a ativação da esteira, que faz com que o autômato transite para o estado 2. No estado 2, o evento `Liga_esteira` é proibido, de forma que o autômato transita de volta ao estado 1 com o desligamento da esteira. Nesse caso ambos os eventos são controláveis, pois dependem da ação de controle. A especificação de controle mostra que, inicialmente, enquanto o evento `sensor_ativado` não ocorrer, o evento `Liga_esteira` é proibido, podendo a esteira apenas ser desligada. No estado 2 ocorre o oposto: a esteira não pode ser desligada enquanto o sensor não for

desativado.

Figura 1 – Exemplo de autômato



A obtenção de um autômato único, chamado de supervisor, que determina a ação de controle a partir da ocorrência de eventos, se dá através dos seguintes procedimentos: i) composição síncrona dos elementos de planta, resultando no autômato G; ii) composição síncrona das especificações de controle, resultando no autômato E; iii) composição síncrona entre G e E, resultando em K; iv) finalmente, é encontrada a máxima sublinguagem de K que é controlável em relação a G, resultando no autômato supervisor S. Estes passos constituem uma abordagem formal para síntese de controladores em automação e podem ser realizados com apoio de uma ferramenta computacional como Nadzoru (PINHEIRO et al., 2015) e IDES (RUDIE, K., 2010), dentre outras. Para mais detalhes, a referência (CURY, 2001) é indicada ao leitor .

O exemplo mostra um autômato muito simples e pode ser facilmente implementado em linguagem de programação com estruturas condicionais (*if, else*). Entretanto, a planta a ser considerado pode ser muita mais complexa do que o exemplo apresentado, com diversos elementos interagindo entre si. Assim, torna-se indispensável o uso de uma metodologia formal.

2.2. Laboratório de Automação da Manufatura

No curso de Engenharia Elétrica da UDESC, a disciplina de Laboratório de Automação da Manufatura (LAM) trata da metodologia para resolução de problemas de automação, com ênfase em plantas de manufatura. Tais plantas caracterizam-se por um trabalho coordenado entre diversos elementos, incluindo sensores e atuadores. Dessa forma, utiliza-se a Teoria de Controle Supervisório de SEDs como ferramenta de metodologia de projeto de controle.

Para uma planta de manufatura, entretanto, o cálculo do supervisor, conforme explicado previamente, modelando os elementos da planta e as especificações de controle, pode resultar em um autômato supervisor com milhares de estados e transições. Dessa forma, o uso de uma ferramenta de geração automática de código se faz importante. No âmbito da disciplina, desenvolveu-se uma ferramenta para facilitar a implementação do controle supervisório, através da geração automática de código para microcontrolador Arduino, nomeada C4uC (Código para Microcontrolador), utilizando como entrada um arquivo em formato Grail. Tal formato é o padrão utilizado na ferramenta *Grail for supervisory control of DES* (REISER; CUNHA; CURY, 2006) e também pode ser gerado pelo *software* IDES (RUDIE, K., 2010).

3. Ferramenta C4uC

Nesta seção, apresenta-se tecnicamente a lógica de funcionamento do gerador C4uC. Na disciplina em questão, as principais ferramentas para modelagem e cálculo do autômato supervisor são IDES e Nadzoru (PINHEIRO et al., 2015). No presente trabalho, considerou-se o formato de arquivo Grail (.fm), gerado pelo *software* IDES, como entrada para o gerador de código para Arduino.

3.1. Arquivo de entrada supcon.fm

O gerador C4uC recebe como entrada um arquivo no formato Grail (extensão .fm), que é um dos formatos suportados pelo *software* IDES. O arquivo é composto por diversas linhas, uma para cada transição existente no autômato supervisor, sendo cada linha composta por: "estado atual (número), nome de um evento possível em tal estado (string), próximo estado (número)". Como não há distinção entre eventos controláveis e eventos não controláveis, for imposta uma restrição, com o uso do C4uC, para que os nomes do eventos controláveis iniciem com letra maiúscula, enquanto os nomes dos eventos não controláveis iniciem com letra minúscula. Existem, além de linhas contendo transições, entretanto, linhas indicando se um determinado estado é final e uma linha informando o estado de origem. Dessa forma, o gerador, primeiramente filtra o arquivo de entrada, removendo tais linhas e gera um arquivo `supcon.aux`, contendo somente as transições.

3.2. Geração dos grafos de eventos controláveis e não controláveis

Após a leitura do arquivo de entrada, o arquivo filtrado é lido, linha a linha. Atribui-se o número 1 para o primeiro evento não controlável que aparecer em tal leitura, assim como ao primeiro evento não controlável. Para cada evento, diferente dos já numerados, que aparecer, atribui-se a numeração subsequente. Caso o evento seja controlável, adiciona-se um par de inteiros à posição correspondente ao número do estado de origem, no grafo de eventos controláveis, contendo o número do evento e o número do estado de destino lido. Caso o evento seja não controlável, faz-se o mesmo procedimento, mas adicionando a transição ao grafo de eventos não controláveis. Tal separação em dois grafos é feito porque a prioridade de transição deve ser dada aos eventos não controláveis, ou seja, primeiro verifica-se se algum evento não controlável aconteceu, pois não se tem controle sobre sua ocorrência. Somente caso não haja eventos não controláveis acontecendo, testa-se a possibilidade de geração de um evento controlável.

3.3. Escrita do cabeçalho, de variáveis globais e template para funções de escrita e leitura de eventos

Com os grafos gerados, parte-se para a geração do código. Primeiramente, inclui-se a diretiva `#include <avr/pgmspace.h>` para se utilizar a escrita em memória de programa, necessária devido ao possivelmente elevado número de transições do supervisor. Após, são declaradas as variáveis globais `s` (número do estado atual), `inp` (número do evento não controlável lido, atribuído -1 caso não haja nenhum evento) e `out` (número do evento controlável que pode ser gerado, atribuído -1 caso não haja nenhum evento). São então criados os escopos das funções `read_I()`, `write_O()` e `setup()`, que devem ser editados pelo usuário após a geração do código, pois dependem da implementação em

hardware a ser realizada.

3.4. Escrita das transições

Como a placa Arduino Uno, atualmente definida como o dispositivo padrão de implementação do C4uC, possui uma limitação de tamanho do código em texto, utilizou-se uma abordagem diferente da intuitiva (usando estruturas condicionais de se/senão, para cada estado). A implementação empregada foi definir três vetores de inteiros de 16 bits, com tamanho igual ao número de transições do supervisor. Como cada transição é descrita por estado de origem, evento e estado de destino, seguiu-se a seguinte lógica: na primeira posição de cada vetor, escreve-se o estado de origem, o evento e estado de destino da primeira transição. Na segunda posição de cada vetor, escreve-se o trio correspondente à segunda transição, e assim por diante. Observa-se que, para cada estado de origem, são escrita, primeiramente, as transições de eventos não controláveis e, por fim, a possível transição por evento controlável, se existir. A seguir observa-se como exemplo um exemplo, um trecho do código gerado para o estudo de caso que será apresentado na Seção 4:

Figura 2 – Exemplo de trecho de codigo gerado

```
current_state[] =  {..., 6, 6,  6,  6,   6,  6,  7, 7,  7,  7,  7, ...}
event[]         =  {..., 1, 2,  3,  5,   6,  9,  0, 7,  3,  5,  6, ...}
next_state[]    =  {..., 3, 12, 16, 17, 18, 15, 12, 3, 19, 20, 21, ...}
```

No exemplo, podem ser observadas as transições que partem dos estados 6 e 7. Em tal exemplo, há um total de 9 possíveis eventos não controláveis, numerados de 0 a 8. Dessa forma, os eventos controláveis são numerados a partir de 9. No estado 6, há a possibilidade de ocorrência dos eventos 1, 2, 3, 5 e 6. Caso nenhum destes eventos ocorra, o evento número 9, controlável, será disparado, levando o supervisor ao estado 15. No estado 7 não há eventos controláveis, ou seja, o supervisor não sairá do estado 7 enquanto nenhum dos possíveis eventos não controláveis nesse estado não ocorra. As autotransições com eventos não controláveis (*uncontrollable self-loops*) não são implementadas para reduzir o consumo de memória, pois tais transições não fazem efeito prático no supervisor.

3.5. Escrita da rotina principal

Finalmente, escreve-se a rotina principal para o programa, que consiste nos seguintes passos:

1. Ler a entrada, ou seja, verificar se ocorreu o disparo de algum evento não controlável, com a função `read_I()`, atribuindo um valor simbólico -1 ao evento ocorrido caso nenhum evento não controlável tenha sido disparado;

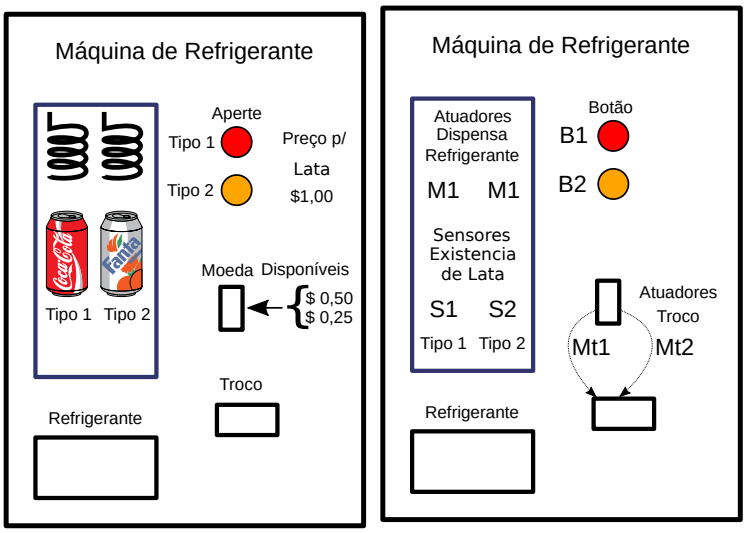
2. Varrer o primeiro vetor, `current_state`, buscando o estado atual do supervisor. Quando encontrado, verificar se o evento controlável ocorrido é igual a um dos eventos não controláveis. Caso positivo, assinalar a mudança de estado e encerrar o ciclo. Caso contrário verificar se há evento controlável a ser gerado. Caso sim,

marcar o evento a ser gerado e encerrar o ciclo. Caso nenhum transição seja gerada, encerrar o ciclo e aguardar o próximo ciclo (aguarda 100 ms por padrão).

4. Estudo de caso

O gerador de código foi utilizado por alunos da disciplina de LAM no segundo semestre de 2016. Apresenta-se, nessa seção, o trabalho das equipes de alunos para controle supervisório de um modelo de simulação de uma máquina servidora de refrigerantes. Tal máquina é composta por dois tipos de refrigerantes. Cada um é acionado pelo seu botão que, por sua vez, acionava um comando elétrico para um solenoide que empurrava a lata de refrigerante correspondente. Ainda, há um sensor que detecta a presença de latas disponíveis, para cada tipo. Apesar da máquina ser simulada, embarcou-se o programa do supervisor em uma placa Arduino Uno, utilizando botões para gerar os eventos não controláveis e LEDs para indicar a ativação das saídas (eventos controláveis). O preço das bebidas é $ 1.00. A máquina aceita moedas de $ 0.50 e $ 0.25. No momento em que é inserida uma moeda na máquina um sensor detecta esta ação. Caso o valor colocado exceda o preço do produto, têm-se na planta dois atuadores, um para cada valor aceito pela máquina, que devolvem o troco conforme o valor inserido. Na Figura 3, mostra-se a fisionomia da máquina de refrigerante em questão e também a disposição de sensores e atuadores.

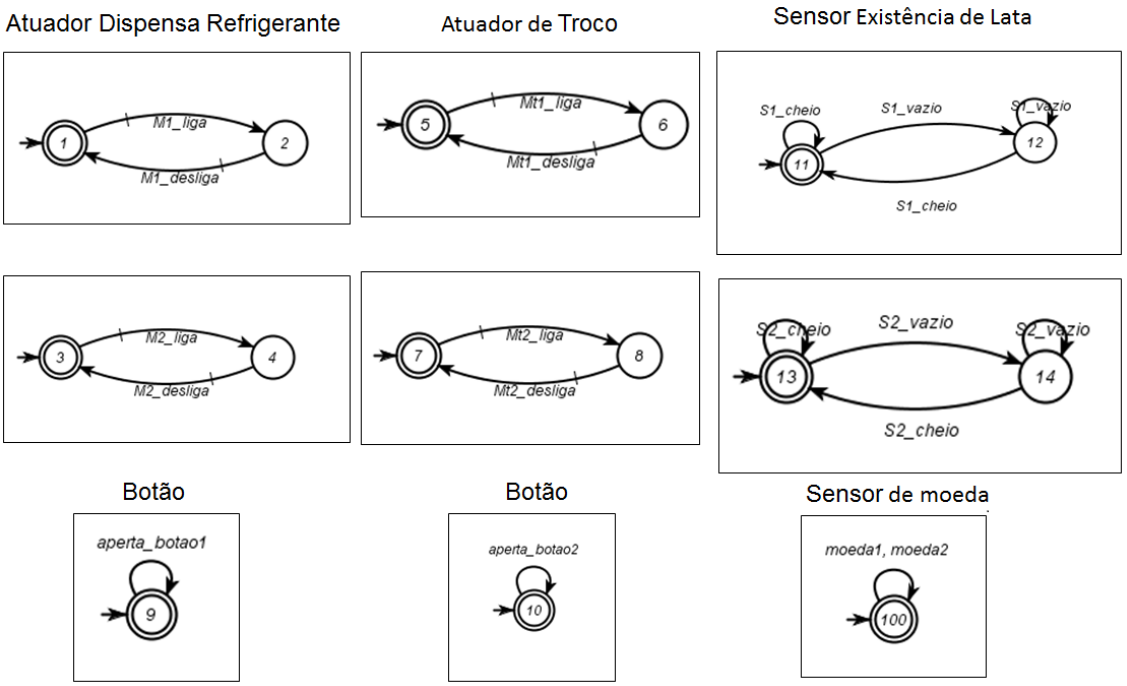
Figura 3 – Planta considerada



A partir dessa planta obteve-se um modelo em autômatos. Para fazê-lo, separou-se cada processo da máquina e foram observados quais eram os estados possíveis e quais eventos desencadeavam um novo estado. Com isso, modelou-se a máquina de refrigerante conforme consta na Figura 4.

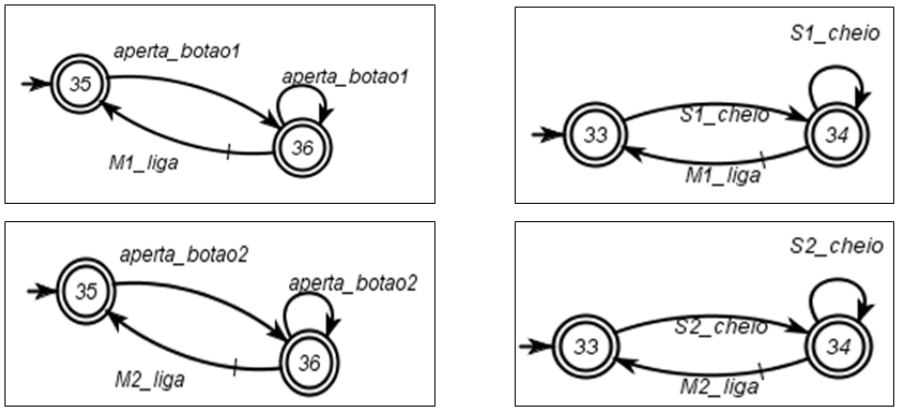
Para realizar uma modelagem utilizando a linguagem de autômatos além de modelar a planta é necessário modelar especificações. Essas especificações são muito importantes, pois delimitam a planta de forma física e lógica. A especificação imposta ao pagamento se tornou bastante complexa. Existe, primeiramente, uma parcela dessa especificação que

Figura 4 – Planta modelado com autômatos



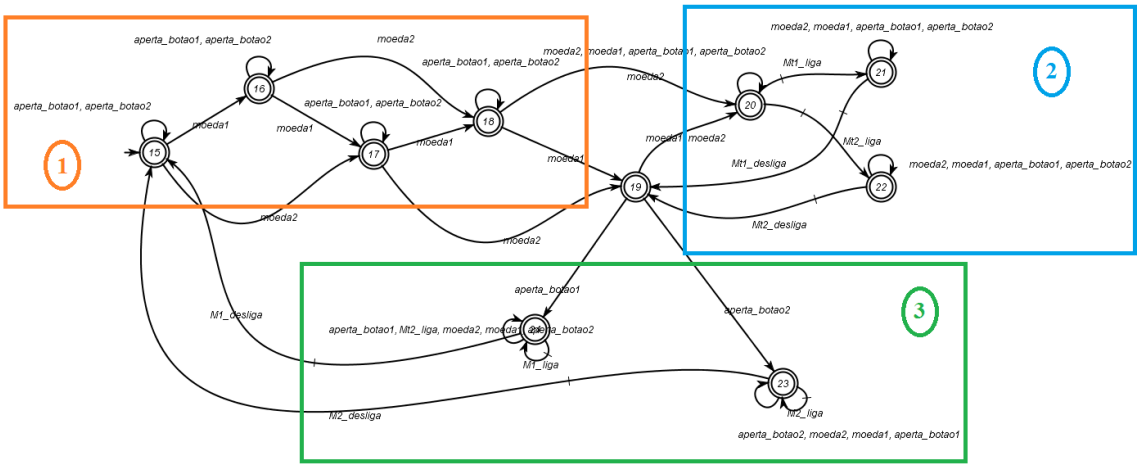
é relacionada à contagem do valor necessário para adquirir o produto. A segunda parcela é relacionada ao troco e a terceira ao atuador do refrigerante. As Figuras 5 e 6 apresentam as especificações que foram aplicadas a planta.

Figura 5 – Especificações de controle, parte 1



Para obter o modelo completo da máquina de refrigerantes (planta mais especificações), faz-se uma composição síncrona da planta com as especificações. Após ter realizado essa etapa do projeto, obtém-se um único autômato que engloba todos os estados e eventos (transições) possíveis. Ao final desse processo, esse projeto obteve 6048 transições e 704 estados, com redução para 4932 transições após a remoção das auto-transições não controláveis.

Figura 6 – Especificações de controle, parte 2: especificação de troco



O próximo passo do projeto é a escrita de um código que seja compilável e, em seguida, embarcá-lo em um microcontrolador (ou algum dispositivo semelhante). Nesse ponto, vê-se a necessidade de escrever um código que interprete o arquivo que contém os estados e as transições e o traduza para a linguagem C++ (ou semelhante), dado que são muitos estados e transições para que o código seja escrito manualmente. Para a máquina de refrigerante, a implementação foi feita na plataforma Arduino. Com o código gerado pelo C4uC, a complementação do mesmo se dá de forma simples. Além das usuais configurações (pinos de entrada/saída e variáveis auxiliares), apenas é necessária a edição das funções `read_I()` e `write_O()` que são responsáveis pela leitura e realização de eventos, respectivamente. Em `read_I()` foram escritas as condicionais responsáveis pela leitura dos botões e sensores e têm como resultado a atribuição de um valor para a variável `inp` de acordo com a numeração dos eventos não controláveis. `write_O()` foram escritas as condicionais que realizam o acionamento/desligamento dos motores de acordo com o valor recebido da variável `out` (evento controlável gerado).

5. Conclusão

Como resultado, foi verificado que o programa supervisor funcionou de acordo com o esperado, respeitando a lógica de controle imposta. Além disso, destaca-se que a ferramenta proporcionou facilidade de implementação em um microcontrolador altamente acessível. Entretanto, fez-se necessário o estudo e entendimento do sistema de geração de código e foi necessário que os alunos implementassem o sistema de escrita e leitura. Dessa forma, a ferramenta C4uC apresenta-se como um facilitador para uma das partes mais críticas da disciplina, que é a implementação prática de autômatos de grande porte, permitindo que o aluno possa focalizar sua atenção à teoria para então verificar e entender seu funcionamento prático. O *software* está disponível para *download* gratuito na plataforma SourceForge (NEGRI, G. H., LEAL, A. B., 2017).

Agradecimentos

Os autores agradecem à UDESC pela concessão, por meio do PROMOP, da bolsa de pós-graduação a Gabriel. H. Negri; ao Programa de Educação Tutorial (PET); ao professor Daniel G. N. Maas, por ter disponibilizado o uso do C4uC aos alunos na disciplina de LAM; e ao Bruno Bertoldi, pelo *feedback* em relação ao uso do C4uC.

REFERÊNCIAS BIBLIOGRÁFICAS

ALAYÓN, S.; GONZÁLEZ, C.; TOLEDO, P. A laboratory experiment for teaching automation inspired by the smart home. *Computer Applications in Engineering Education*, v. 21, n. S1, p. E121–E131, 2013.

BAO, L. et al. Study and exploration on the featured teaching of automation control theory during the information age. *Procedia - Social and Behavioral Sciences*, v. 177, p. 325–330, 2015.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2. ed. [S.l.]: Springer, 2010.

CASTRO-SCHEZ, J. et al. Knowledge-based systems to enhance learning: A case study on formal languages and automata theory. *Knowledge-Based Systems*, v. 122, p. 180–198, 2017.

CURY, J. E. R. *Teoria de Controle Supervisório de Sistemas a Eventos Discretos*. 2001. Apostila publicada no V Simpósio Brasileiro de Automação Inteligente.

DELIGIANNIS, V.; MANESIS, S. Building automata-type simulation models for undergraduate teaching industrial systems analysis. *IFAC Proceedings Volumes*, v. 39, n. 6, p. 297–302, 2006.

JUKEMURA, A. S.; NASCIMENTO, H. A. D. do; UCHÔA, J. Q. Gam - um simulador para auxiliar o ensino de linguagens formais e de autômatos. In: *XXV Congresso da Sociedade Brasileira de Computação*. São Leopoldo: [s.n.], 2005.

LEAL, A. B. et al. Implantaçao de laboratório de automação da manufatura como meio para melhorar o ensino de automação em curso de engenharia elétrica. In: *XLI Congresso brasileiro de educação em engenharia*. Gramado: [s.n.], 2013.

MAAS, D. N.; PINOTTI, A. J.; LEAL, A. B. Síntese e implementação de controle supervisório monolítico para um ice maker. In: *Anais do XIX Congresso Brasileiro de Automática, CBA 2012*. Campina Grande: [s.n.], 2012.

NEGRI, G. H., LEAL, A. B. *C4uC*. 2017. Disponível em: <https://sourceforge.net/projects/c4uc/>.

NETO, J.; TERRA, R. Lfapp: Um aplicativo móvel para o ensino de linguagens formais e autômatos. In: *24º Workshop sobre Educação em Computação*. Porto Alegre: [s.n.], 2016.

PINHEIRO, L. P. et al. Nadzoru: A software tool for supervisory control of discrete event systems. *IFAC-PapersOnLine*, v. 48, p. 182–187, 2015.

REISER, C.; CUNHA, A. E. D.; CURY, J. E. The environment grail for supervisory control of discrete event systems. In: *2006 8th International Workshop on Discrete Event Systems*. Ann Arbor, MI, USA: [s.n.], 2006.

RICHERT, A. et al. Educating engineers for industry 4.0: Virtual worlds and human-robot-teams: Empirical studies towards a new educational age. In: *2016 IEEE Global Engineering Education Conference (EDUCON)*. Abu Dhabi: [s.n.], 2016. p. 142–149.

RUDIE, K. *Integrated Discrete Event Systems (IDES)*. 2010. Disponível em: <https://qshare.queensu.ca/Users01/rudie/www/software.html>.

WOLLSCHLAEGER, M.; SAUTER, T.; JASPERNEITE, J. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, v. 11, n. 1, p. 17–27, 2017.