UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTRATÉGIA PARA INTRODUÇÃO DE REQUISITOS DA NORMA UL1998 EM MODELOS UML STATECHART DE SISTEMAS EMBARCADOS

BÁRBARA CASTANHEIRA

ORIENTADORA: PROF. DRA. SANDRA CAMARGO PINTO FERRAZ FABBRI

São Carlos - SP
Março/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTRATÉGIA PARA INTRODUÇÃO DE REQUISITOS DA NORMA UL1998 EM MODELOS UML STATECHART DE SISTEMAS EMBARCADOS

BÁRBARA CASTANHEIRA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientadora: Dra. Sandra Camargo Pinto Ferraz Fabbri.

São Carlos - SP
Março/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de dissertação de mestrado da candidata Bárbara Castanheira, realizada em 08/03/2016.

Profa. Dra. Sandra Camargo Pinto Ferraz Fabbri
(UFSCar)

Prof. Dr. Fabiano Cutigi Ferrari
(UFSCar)

Profa. Dra. Silvia Regina Vergílio
(UFPR)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Profa. Dra. Silvia Regina Vergílio e, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa da aluna Bárbara Castanheira.

Profa. Dra. Sandra Camargo Pinto Ferraz Fabbri
Presidente da Comissão Examinadora
(UFSCar)

Dedico este trabalho aos meus pais e a minha orientadora Sandra

AGRADECIMENTO

Em primeiro lugar, agradeço à Profa. Dra. Sandra Fabbri, por ter me escolhido para orientada e durante os últimos anos dividir um pouco de toda a sua experiência e conhecimento no desenvolvimento deste trabalho. Por toda a sua dedicação, seu profissionalismo e sua amizade.

Agradeço a todos os colegas do LaPES, em especial, Elis Hernandes, Erik Aceiro Antônio e Guilherme Freire que me ajudaram academicamente e pessoalmente.

Agradeço o Departamento de Computação (DC) da Universidade Federal de São Carlos, pela infraestrutura e apoio durante o mestrado.

Agradeço a Tecumseh do Brasil pelo apoio e por ter me concedido a oportunidade e o tempo para frequentar as aulas e me dedicar a esta pesquisa.

Agradeço aos meus pais, que estiveram me incentivando a estudar desde meus primeiros anos e deram todo o suporte possível para que eu realizasse meus sonhos. Agradeço ao Danilo, meu marido, amor e companheiro de vida, por todo apoio e paciência nesta difícil, mas gratificante etapa.

Obrigada.

"All that man ignores, does not exist for him. Therefore the universe of each, is summed up to the size of his knowledge"

Albert Einstein

RESUMO

Contexto: Operando em diferentes contextos e segmentos, os sistemas embarcados (SEs) estão cada vez mais presentes no cotidiano da sociedade. Os SEs geralmente operam de forma invisível ao usuário mas, nem por isso, possíveis falhas de software deixam de oferecer riscos, ocasionando sérios danos materiais e financeiros. Levando em consideração a segurança dos usuários de SEs, entidades governamentais e instituições pelo mundo têm estudado e criado normas de desenvolvimento e de teste de SEs que garantam sistemas mais robustos e livres de defeitos que possam gerar falhas que ponham em risco o usuário. Contudo, as normas são habitualmente aplicadas no nível de código, o que aumenta, substancialmente, os recursos a serem empregados caso haja defeitos no SE. Objetivo: Apresentar a estratégia UL/ME que trata os defeitos descritos na norma de certificação de SEs, a UL 1998, no nível de modelagem, mais especificamente, no modelo de estados Statechart. O propósito é antecipar a identificação dos defeitos, de forma a gerar código com mais chance de atender aos requisitos da norma, evitando retrabalho e, consequentemente, melhorando a qualidade. Método: O trabalho foi realizado em três etapas: na primeira identificaram-se os defeitos tratados pela norma UL 1998, tabulando esses defeitos de forma a associá-los aos componentes de SEs; na segunda etapa foram desenhados modelos genéricos para cada componente e também os modelos de componentes de SEs que representam os testes requeridos pela norma e na terceira etapa, criaram-se duas proposta para utilização da Estratégia UL/ME, uma para SEs já implementados e outra para SEs em desenvolvimento. Essas propostas serviram para avaliação da estratégia utilizando dois SEs reais, um já implementado e outro em desenvolvimento. Resultados: De acordo com os requisitos da norma UL 1998, o uso da Estratégia UL/ME auxiliou a correção dos defeitos do primeiro SE avaliado o desenvolvimento do segundo SE. Conclusão: o uso da Estratégia UL/ME auxiliou na correção de um SE já implementado e na modelagem de um SE em desenvolvimento.

Palavras-chave: Sistemas Embarcados, Normas de certificação, modelos UML Statecharte e UL 1998, Estratégia UL/ME.

ABSTRACT

Context: Operating in different contexts and segments, embedded systems (ESs) are increasingly present in everyday society. The ESs usually operate invisibly to the user but, even so, possible software failures cease to pose risks, causing serious material and financial damage. Taking into consideration the safety of users of ESs, government entities and institutions around the world have studied and created standards development and SEs test to ensure systems that are more robust and free from defects that may cause faults that may endanger the user. However, standards are usually applied at the level of code, which increases substantially the resources to be used if there are defects in the SE. Objective: To present the UL/ME Strategy that addresses the shortcomings described in SEs certification standard, UL 1998 level modeling, more specifically, in the model of Statechart states. The purpose is to anticipate the identification of defects, to generate code that has more chance to meet the standard requirements, preventing rework and therefore improving quality. Methodology: The study was conducted in three stages: first stage identified the defects treated by the standard UL 1998, tabulating these defects, and then, link them to the SEs components. In the second stage were designed generic models for each component and also the ESs models that represents the tests required by the standard for each components, and in the third step, it was created two proposed for use UL/ME Strategy, one for ESs already implemented and other for ESs in development. These proposals were used for evaluation of the strategy using two real ESs, one already implemented and other in development. Results: According to the requirements of UL 1998 standard, the use of Strategy UL/ME helped to correct the defects of the first ES and evaluated the development of the second ES. Conclusion: the use of Strategy UL/ME assisted in the correction of an SE already implemented and modeling of an ES in development.

Keywords: Embedded Systems, certification standards, UML Statechart models and UL 1998, UL Strategy / ME.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC	Alternating current
BLDC	Brushless Direct Current
CMVV	Controladores de motor de velocidade variável
DC	Direct current
DVD	Digital Versatile Disc
ES	Engenharia de software
FAA	Federal Aviation Administration
LAPES	Laboratório de Pesquisa em Engenharia de Software
LED	Light Emitting Diode
LCD	Liquid Crystal Display
MEF	Máquina de Estados Finitos
OTP	One Time Programmable
PPG-CC	Programa de Pós-graduação em Ciência da Computação
PWM	Pulse-Width Modulation
SE	Sistemas Embarcados
StArt	State of the Art through Systematic Review
SysML	Systems Modeling Language
UFSCar	Universidade Federal de São Carlos
UL	Underwriters Laboratories
UML	Unified Modelling Language
USD	Universal Serial Bus

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta o contexto no qual este trabalho está inserido, expondo a importância dos sistemas embarcados nos dias de hoje, ressaltando os aspectos de certificação exigidos para esses sistemas. Além disso, com base nesse contexto, caracterizam-se os objetivos desta pesquisa.

1.1 Contexto

Os Sistemas Embarcados (SEs) estão cada vez mais presentes na vida humana, sendo possível perceber o seu uso em diferentes equipamentos e eletroeletrônicos, como smartphones, TVs inteligentes, aparelhos médicos, navegadores de carros, veículos aéreos tripulados, entre outros. Segundo Ebert e Jones (2009), em 2008 havia 30 microprocessadores por pessoa nos países desenvolvidos.

Esse número na inserção das tecnologias embarcadas demanda que o desenvolvimento de novas funcionalidades e a produção de novos SEs sejam feitos em prazos curtos para atender o mercado (JIA, SHELHAMER, *et al.*, 2014; LINDGREN e MÜNCH, 2015). Além disso, exige-se uma maior qualidade do SE, aliada com segurança e confiabilidade. Esse fato tem levado a comunidade acadêmica a explorar e estabelecer novos processos e técnicas de desenvolvimento de SEs. Isso pode ser evidenciado pela crescente investigação sobre a redução de possíveis falhas de um SE (KANDT, 2009; LIANG , BAI , *et al.*, 2015; DAVIS e GROLLEAU, 2015).

Uma das áreas que norteia as investigações sobre melhorias de software é a Engenharia de Software (ES), que envolve a interdisciplinaridade de áreas como a de Software e a de Sistemas Embarcados (EBERT e JONES, 2009; UMM-E-HABIBA e JAVED , 2014). Os fundamentos científicos da ES têm sido usados como base para muitos projetos em âmbito nacional e internacional. Tais projetos promoveram esforços na proposição de novas soluções para apoiar o processo de desenvolvimento de SEs de qualidade. Dentre eles, pode-se citar o do Instituto Nacional de Ciência e Tecnologia - Sistemas Embarcados Críticos (INCT-SEC, 2011) que buscou, junto a universidades, estabelecer avanços na área de sistemas embarcados críticos (SEC) com a proposição de novas estratégias e técnicas de desenvolvimento que reduzissem defeitos de software, agregando maior confiabilidade a estes sistemas. Como exemplo, podem ser citados os trabalhos de Freire (2011) e Antônio (2014), alunos do grupo de pesquisa do LAPES (Laboratório de Pesquisa em Engenharia de Software) da UFSCar. O Trabalho de Freire (2011) influenciou na escolha pela modelagem de SE usando UML Statechart, enquanto Antônio (2014) utilizou no seu trabalho parte do que foi aqui desenvolvido.

Outra vertente de estudos na área de confiabilidade dos SEs, especialmente os destinados ao uso doméstico, está nos órgãos Estatais que, desde o final do século passado, introduziram regulamentos com os requisitos de segurança no projeto de eletrodomésticos. Na Europa, exemplos desses requisitos estão definidos pela norma IEC60730 e nos EUA pela UL 1998. Essas normas exigem a inclusão de funcionalidades para evitar defeitos ou, pelo menos, garantir que qualquer defeito presente no software embarcado não gere uma falha no aparelho que coloque em risco a segurança do usuário (NAIR, DE LA VARA, *et al.*, 2015).

Para garantir que as normas sejam corretamente aplicadas ao produto, criou-se a certificação de produto, que envolve um processo de avaliação de requisitos pré-estabelecidos executado por órgãos não governamentais específicos para o país onde o produto será comercializado. O processo de certificação determina de forma sistemática, baseada nos princípios da ciência, da engenharia e da teoria de medição, se um sistema satisfaz padrões bem definidos e mensuráveis pela norma utilizada (WASSYNG, MAIBAUM , *et al.*, 2010).

Vale ressaltar que, embora a certificação de SEs seja uma oportunidade para se detectarem erros que ocasionem mau funcionamento do SE, em geral, o uso de requisitos de certificação ocorre apenas em fases tardias do processo de

desenvolvimento de software. Quando se trata das normas de certificação, pode-se perceber tanto em relatos da literatura quanto da prática, que tais requisitos são pouco explorados e aplicados de fato ao longo do processo de desenvolvimento de software (GALL, 2008).

1.2 Motivação e Objetivos

Motivado pelo contexto relatado anteriormente, o objetivo deste trabalho foi mapear requisitos de teste presentes nas normas de certificação de SEs, como UL1998, IEC61508, ISO 14971 e DO-178-B, para modelos ULM Statechart. O intuito desse mapeamento foi poder explorar esses defeitos ainda na fase de modelagem, uma vez que as normas serão somente aplicadas nas fases posteriores de desenvolvimento.

Ressalta-se que os projetistas raramente modelam o SE previamente, iniciando seu desenvolvimento, em geral, a partir de modelos de baixo nível, como o Simulink (FREIRE, 2011), ou a partir do próprio código.

No trabalho de mestrado de Freire (2011), foi explorada a contribuição de se usarem modelos de mais alto nível para melhorar a compreensão e o planejamento de um SE. Nesse estudo, o modelo investigado foi o Statechart e foi mostrado que a elaboração desse modelo, anteriormente ao modelo Simulink, torna este último mais compreensível e mais organizado. Assim, na pesquisa de Freire (2011), forneceram-se indícios que a adoção de um processo de desenvolvimento, em que a aplicação vai evoluindo gradativamente por meio dos artefatos que são elaborados, é uma abordagem que pode melhorar a qualidade da aplicação como um todo, uma vez que o fato de elaborar o Statechart melhora o modelo Simulink e, consequentemente, melhora o código, ou seja, o SE.

Corrigir erros em etapas anteriores de desenvolvimento também auxilia na economia de recursos (PRESSMAN, 2009). Esse cenário sugere que os erros sejam identificados e corrigidos já nas fases iniciais de desenvolvimento, em modelos de alto nível, para assim, evitar a inserção de defeitos em fases avançadas do desenvolvimento, como a codificação do software do SE, evitando retrabalho e minimizando o custo de desenvolvimento.

Assim, considerando-se o contexto dos SEs no qual as normas de certificação são aplicadas somente quando se tem o código pronto, e no qual o uso de modelos Statechart – antes do desenvolvimento do código – é uma alternativa viável para o projeto do SE, identificou-se como uma relevante oportunidade de pesquisa a caracterização e avaliação dos requisitos relativos às normas de certificação UL 1998 e DO-178B em tais modelos.

Dessa forma, o objetivo foi estudar algumas normas de certificação existentes, suas similaridades e então explorar como os requisitos dessas normas podem ser mapeados para o modelo Statechart. Além disso, ao ser construído o modelo Statechart, ele pode passar por uma inspeção que investigue, no próprio modelo, se defeitos explorados pelas normas de certificação estariam presentes no modelo e, com a evolução da aplicação, muito provavelmente, seriam transportados para o código.

1.3 Metodologia e Desenvolvimento do Trabalho

A metodologia utilizada para desenvolver este trabalho consistiu nas seguintes etapas: (1) Caracterização das principais lacunas de pesquisas existentes na área de certificação de software de SE, por meio de revisão bibliográfica utilizando a ferramenta *StArt* (State of the Art through Systematic Review), desenvolvida no LAPES (Laboratório de Pesquisa em Engenharia de Software). (2) Com base na revisão bibliográfica observou-se que as normas de certificação são pouco exploradas em níveis de desenvolvimento anteriores a própria escrita do código do SE. Assim foi decidido criar uma estratégia de identificação de defeitos, segundo os requisitos de normas de certificação de SE, em nível de modelo, de forma a auxiliar o desenvolvimento de SEs. (3) Foi realizado um estudo comparativo entre as normas de certificação de SEs e foi elaborada uma tabela que apresenta os testes requeridos em comum pelas normas, para cada componente que pode ser utilizado no SE. Vale notar que esses testes serão referenciados no presente trabalho como requisitos da norma de certificação e que foi decidido usar a norma UL 1998 como norma de certificação, pois muito dos requisitos presentes nesta norma são comuns a outras normas de certificação e pela afinidade da pesquisadora

com essa norma. Com base nos requisitos da norma UL 1998 identificados na Etapa (2) foi desenvolvida uma estratégia para auxiliar a identificação desses requisitos em nível de modelo UML Statechart e criadas propostas de uso dessa estratégia. (4) Finalmente, foram usados dois SEs reais para avaliar o uso da estratégia criada na Etapa (3) para na identificação dos requisitos da norma em nível de modelo.

1.4 Organização do Trabalho

O Capítulo 1 contextualiza o universo da pesquisa os problemas no contexto de SEs que servirão de inspiração para a busca de novas soluções.

O Capítulo 2, apresenta o embasamento teórico a partir da revisão bibliográfica realizado ao longo de todo o desenvolvimento desse trabalho, com o auxílio da ferramenta *StArt*, desenvolvida no LAPES.

O Capítulo 3 descreve as etapas da criação da Estratégia UL/ME e propostas para o uso da estratégia.

O Capítulo 4 apresenta duas avaliações de SEs reais utilizando a Estratégia UL/ME e as propostas para o seu uso do capítulo anterior.

O Capítulo 5 apresenta, por fim as conclusões do estudo realizado e suas limitações, que poderão ser exploradas em pesquisas futuras.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma revisão da literatura, destacando as principais contribuições que serviram de inspiração para a criação da Estratégia UL/ME.

2.1 Considerações iniciais

Este capítulo apresenta a fundamentação teórica utilizada para o um mapeamento de defeitos descritos em normas de certificação de SEs em nível de modelo, conforme será explanado no Capítulo 3 em seguida.

Vale ressaltar que a revisão bibliográfica do presente trabalho foi conduzida ao longo de todo o desenvolvimento desse estudo utilizando a ferramenta *StArt*, em desenvolvimento desde 2010 por membros do LAPES.

Este capítulo está organizado da forma descrita a seguir. Como primeira etapa, será apresentada na Seção 2.2 uma visão geral de Sistemas Embarcados (SEs). A Seção 2.3 abordará a técnica de modelagem de SEs UML Statechart que será utilizada na criação da Estratégia UL/ME. A seção 2.4 apresenta testes de SEs. Em seguida, na Seção 2.5 são tratadas as principais normas de certificação utilizadas como garantia de qualidade de processos e produtos da área de SEs, como por exemplo, UL 1998 e DO-178C. Na Seção 2.6 é apresentado um breve estudo sobre taxonomia de defeitos em nível de modelo. Na Seção 2.7 encerra-se o capítulo.

2.2 Sistemas Embarcados (SEs)

A literatura apresenta diferentes definições para SE, uma das preconizadas neste estudo é a do SE como uma combinação de software embarcado, também denominado firmware, e hardware, projetada para desempenhar uma tarefa específica. Ou seja, é um sistema micro processado que suporta uma aplicação determinada (BARR, 1999; CARRO e WAGNER, 2003) e não inúmeras aplicações como um computador. Sendo assim geralmente não é permitido carregar novas aplicações ou acrescentar novos periféricos, o que mais uma vez os diferencia de computadores de propósito geral (SIMON, 1999).

Sendo o SE dedicado a tarefas específicas, por meio de engenharia pode-se otimizar o projeto reduzindo tamanho, recursos computacionais e custo do produto (WOLF, 2012). Nesse sentido, os próprios fabricantes de microcontroladores têm investido cada vez mais em produtos dedicados a desenvolvimentos específicos, chamados domínios de aplicação, tais como: acionamento de motores, automobilística, telecomunicação, geração e economia de energia, área médica, área aviônica e área industrial, entre outros. Um microcontrolador dedicado, e a própria limitação de funções de um SE, pode sugerir que o desenvolvimento de um software embarcado seja mais simples e consuma menos recursos que os softwares de sistemas computacionais, no entanto, o contato direto do SE com o usuário e as implicações na segurança do mesmo, nem sempre viabilizam essa economia. Entre os anos de 1990 e 2000 quarenta por cento dos marca-passos de um fabricante americano foram recolhidos do mercado pois apresentavam defeitos de firmware (EBERT e JONES, 2009).

SEs como marca-passos, assim como todos os SEs cujas falhas podem ocasionar acidentes graves ou mesmo perda de vida humana, são classificados como Sistemas Embarcados Críticos (SECs). Outros exemplos de SECs são controladores de voo, sistemas automotivos, sistemas de gestão de reator nuclear, aparelhos médicos responsáveis por tratamentos com radiação.

Esse trabalho não está endereçado aos SECs, pois esses sistemas exigiriam pesquisas muito mais aprofundadas, dados os padrões rígidos que determinam seu desenvolvimento, mas sim aos SEs mais comuns, como lavadoras de roupas, celulares, *smartTVs*, *smartWhachs*, geladeiras e *home theaters*, os quais quando

possuem defeitos de hardware ou software que os levam a estados desconhecidos, não chegam a causar acidentes tão graves quanto os SECs, mas podem gerar fogo, perigo de choque ou ferimentos a quem os manuseia. Esses problemas, embora de menor periculosidade ao usuário, trariam grandes desconfortos ao mesmo além de consequências financeiras e legais às empresas responsáveis por tais SEs.

2.2.1 Características e composição do SE

Uma das principais características do SE explorada nesse estudo é a de ser um sistema de natureza tipicamente reativa. Um sistema reativo é um sistema que possui um conjunto de estados e que continuamente interage com seu ambiente externo. Cada interação é executada a partir de uma entrada, então o sistema realiza o processamento de alguma computação e produz uma saída determinada (MARWENDEL, 2011). Sabendo disso, neste trabalho será estudada a possibilidade de uma entrada produzir uma saída diferente da esperada segundo as especificações da norma. Isso será considerado como um defeito de software que, para que não cause uma falha, deverá ser endereçado a um estado seguro ao usuário, bem como produzir um alerta que avise o mesmo do funcionamento incorreto do SE.

Os SEs podem também ser caracterizados quanto a sua resposta ao tempo. Os sistemas que não tem restrições de tempo real, por não possuírem relógio interno para temporizar seus processamentos, ou simplesmente não tem restrições temporais, são sistemas cujos processamentos e ações ocorrem de forma concatenada, ou seja, em uma ordem previamente determinada. Sistemas de tempo real são sistemas que possuem um relógio interno e são capazes de responder a eventos provenientes de uma fonte externa com certas restrições operacionais de tempo (LI e YAO, 2003). Nesses sistemas o processamento, ou seja, o tempo em que deve haver uma resposta, ou saída, depois de um determinado estimulo externo, ou entrada, é limitado. No caso de um sistema de tempo real não cumprir com rigor as restrições de tempo inicialmente estabelecidas, perdas significativas podem ser evidenciadas na qualidade final obtida pelo processamento do sistema embarcado (MARWENDEL, 2011). Esse atraso pode ser considerado também um erro e pode ser proveniente de um defeito no desenvolvimento do software como será abordado mais explicitamente no Capítulo 3. Os sistemas de tempo real são os

escolhidos como objeto de interesse dessa pesquisa por representarem maioria dos dispositivos embarcados (BARRY , 2010).

Outra definição importante do SE diz respeito a sua composição física. De forma genérica um SE é composto por uma unidade de processamento, no presente trabalho: um microcontrolador, componentes de hardware e periféricos. A unidade de processamento de um SE é reduzida se comparada com computadores de uso geral. Sendo assim, os microprocessadores usados nas CPUs das máquinas conhecidas, são substituídos por microcontroladores, que além do poder de processamento reduzido ao estritamente necessário, colaborando com redução de custos de projeto, já possuem diversos componentes e periféricos integrados no mesmo chip e podem ser dedicados a desempenhar tarefas específicas de acordo com o domínio de aplicação, o que facilita o desenvolvimento.

Os componentes de hardware que compõe o SE ou, de forma facilitada, o próprio microcontrolador, que é em si um SE, são recursos de hardware que podem ser implementados de diversas formas, por exemplo, a entrada analógica é um componente que pode servir para ler sinais de tensão e corrente, contudo esse sinal pode representar diversas grandezas físicas como uma onda sonora, temperatura, luminosidade. Fica implícito aqui que os componentes não tratam ou diferenciam o tipo de iteração que fazem com o ambiente, portanto é função software embarcado fazer o tratamento da grandeza física segundo suas características. É importante ressaltar que o número de componentes de um SE é finito, porém a sua forma de implementação ou de possibilidade de periféricos é infinita ou, de forma simplista, dependem da criatividade do desenvolvedor, de novas tecnologias a serem exploradas e de novos produtos a serem criados. No contexto deste trabalho os componentes de um SE são:

- Unidade de memória: corresponde a parte responsável por guardar os dados no microprocessador, pode ser volátil, isto é apagável como o caso das memórias flash comumente presentes em unidades Universal Serial Bus (USB), ou não volátil, isto é, indeléveis, como as memórias One Time Programmable (OTP).
- Unidade Central de Processamento (CPU): responsável pelas operações matemáticas. No interior da CPU se encontram os registros que ajudam a executar com maior rapidez várias operações desejadas.

- Barramento interno: responsável pela comunicação interna. Existem dois tipos de barramento que são o de dados e o de endereços e que tem funções específicas. O número de linhas do barramento de dados depende da quantidade de memória que se deseja endereçar e o número de linhas do barramento de endereço depende da largura da palavra de dados.
- Unidade de entrada/saída: as entradas e saídas são denominadas I/O em que “I” significa entrada (input) e “O” significa saída (output). Elas podem ser digitais, analógicas ou configuráveis.
- Comunicação serial: principal meio de comunicação do microprocessador com outros dispositivos e pode usar vários protocolos como RS232, um protocolo usado para troca serial de dados, ou RS485, outro protocolo para troca serial de dados.
- Temporizador interno: base de tempo interna para SEs de tempo real.
- Conversores A/D (Analógico/Digital) e D/A (Digital/Analógico): conversores que convertem grandezas físicas analógicas como corrente, som, tensão em valores digitais e conversores que transformam uma grandeza digital, como um botão por exemplo, em um sinal analógico.

Finalmente, os periféricos são a dispositivos que fazem a interface dos componentes do SE com o ambiente. Exemplos de periféricos são: teclados, interfaces seriais, memórias externas, fones, microfones, luzes do tipo Light Emitting Diode (LED), displays de cristal líquido (LCD), telas Touchscreen, interface serial, USB. Como dito anteriormente, as iterações dos componentes de um SE são praticamente infinitas, já que diferentes periféricos vêm sendo criados ao longo dos anos.

A Figura 2.1 apresenta um modelo ilustrativo de microcontrolador, seus componentes e possíveis periféricos.

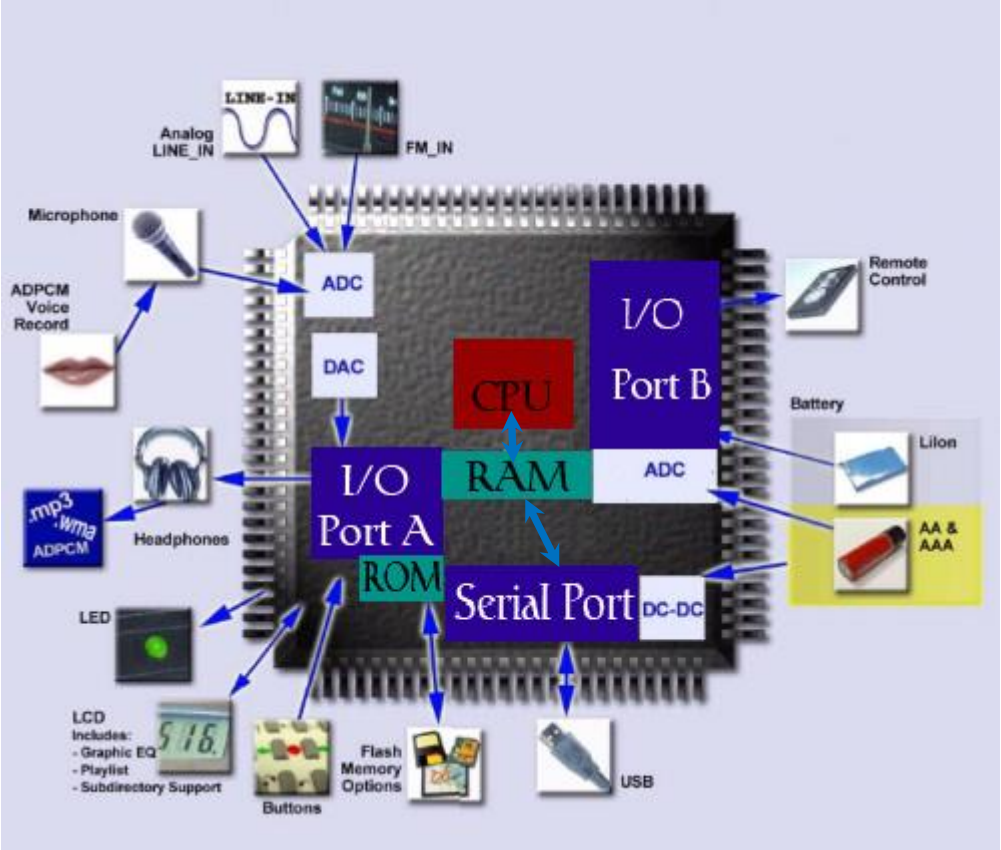


Figura2. 1 - Microcontrolador ST, componentes e periféricos (http://www.st.com).

É possível observar na Figura 2.1, o exemplo, um gravador de voz com um microfone. A onda sonora captada pelo microfone é lida como um sinal analógico por uma entrada do microprocessador, convertida num sinal digital pelo conversor A/D, tornando-se então um dado que pode ser manipulado pelo microcontrolador. Posteriormente esse dado pode ser novamente convertido em sinal analógico por um conversor D/A e ser captado por um fone de ouvido ligado a uma saída do microcontrolador.

2.3 Modelagem de SEs

A modelagem é uma etapa importante do desenvolvimento de Sistemas Embarcados e deve ser feita numa linguagem de alto nível e visual. No presente trabalho a modelagem será utilizada com objetivo de detectar previamente defeitos

de software mapeados pelas normas de certificação através da análise de cada componente da unidade de hardware programável, o microcontrolador.

No tópico 2.3.1 será apresentado com maior detalhamento o artefato de modelagem UML Statechart que será a ferramenta de modelagem escolhida para ser utilizada neste trabalho, devido aos critérios já apresentados no Capítulo 1.

2.3.1 Diagrama UML Statechart

A Unified Modeling Language ™ (UML®) é uma linguagem de modelagem visual padrão destina-se a ser utilizado para modelagem de processos empresariais e semelhantes, análise, projeto e implementação de sistemas baseados em software. Nesse estudo será explorado um dos cinco modelos de modelagem para sistemas dinâmicos que a UML possui, o UML Statechart.

O diagrama UML Statechart foi inicialmente desenvolvido por Harel em 1987, com o propósito de criar uma técnica gráfica formal que representasse sistemas reativos, cuja definição foi apresentada em 2.2.1. Nessa abordagem, os Statechart podem ser entendidos como uma extensão de Máquinas de Estados Finitas (MEF) com adição dos conceitos de hierarquia, ortogonalidade e comunicação em broadcast. A hierarquia permite que os estados de um sistema possam ser decompostos em subestados, de forma a representar o comportamento interno; a ortogonalidade permite representar funcionalidades que são executadas concorrentemente; e a comunicação em broadcast, permite que sejam representadas transições em estados concorrentes a partir de um mesmo evento. Além disso, os Statechart são úteis na introdução de características modulares e hierárquicas para as MEFs clássicas e estão ganhando popularidade para a modelagem de software embarcado em tempo real (NAUGHTON, MCGRATH e HEFFERNAN, 2006), também definido em 2.2.1.

A modelagem UML Statechart é uma variante do modelo de estados proposto por Harel (FORCE, 2010), embora ambos apresentam os mesmos conceitos semânticos e sintáticos no que se refere a hierarquia, ortogonalidade e comunicação em broadcast.

A representação de Statechart pode ser definida por:

- Hierarquia: Pode ser representada como decomposição XOR (ou

Statechart = Diagrama de Estados + Hierarquia + Ortogonalidade + Comunicação em Broadcast

exclusivo) ou estados aninhados. Essa característica permite que um estado seja um estado composto, contendo outros subestados.

- Ortogonalidade: Também conhecida como decomposição AND, na qual um estado pode ser composto por dois ou mais estados ortogonais que executam de forma independente e concorrente.
- Comunicação em Broadcast: permite que estados independentes possam realizar uma transição por meio de um evento habilitado por outro estado. Os estados não concorrem entre si, mas a ocorrência de um evento em estado implica na transição de outros estados que aguardam pelo mesmo evento.

2.3.2 Elementos do Statechart

Os elementos básicos do Statechart são os estados e as transições representadas por setas direcionadas. Uma transição conecta um ou mais estados (HAREL, 1987). Estados são representados por retângulos com cantos arredondados (Figura 2.2).

Figura2. 2 - Modelo de estado Statechart.

Uma transição ocorre quando um evento no estado S1, habilita uma transição de S1 para outro estado, ou para o próprio S1. Essa transição é representada por: uma seta direcionada, contendo um rótulo com o evento que habilita transição; uma condição para a transição é a ação que dever ser executada, conforme apresentado na Figura 2.3, a seguir.

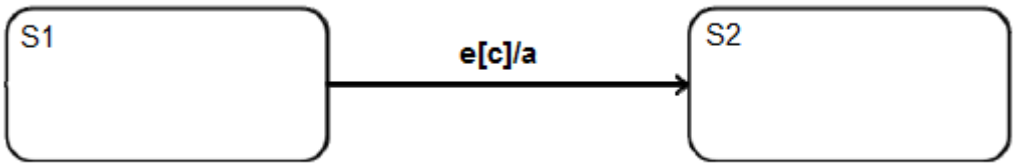


Figura2. 3 - Modelo de transição Statechart

A Figura 2.3 ilustra uma transição entre os estados S1 e S2. A partir do estado S1, uma transição é representada por uma seta direcionada até o estado S2. A transição tem um rótulo representado por ***e[c]/a***. A letra ***e*** representa o evento que precisa ocorrer para que a transição para S2 ocorra e a letra entre colchetes ***[c]***, representa a condição para que um evento ocorra. Por fim ***a*** é a ação que deve ser tomada ao entrar no estado S2.

Conforme explicitado na Seção 2.3.1, a hierarquia permite descrever a decomposição de um estado, sendo assim há um estado maior, denominado superestado ou estado composto, que contém subestados executados internamente. A Figura 2.4 ilustra um estado composto que contém dois subestados.

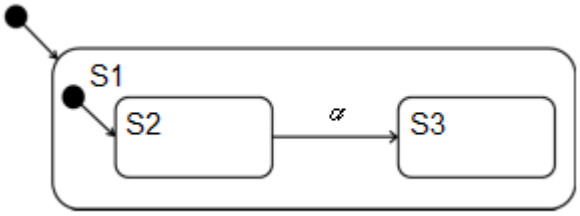


Figura2. 4 - Modelos de estado composto Statechart.

A Figura 2.4 mostra o estado composto S1, um estado composto, que contém outros dois subestados, S2 e S3. O pequeno círculo preenchido indica o estado inicial. Assim nessa figura, o Statechart tem início no estado S1 e, uma vez entrando no estado S1, inicia-se o estado S2, que por meio do evento α dispara uma transição para o estado S3.

A ortogonalidade, também conhecida como decomposição *AND*, representa que, estando em um estado, todos os sistemas contidos nele devem ocorrer de forma concorrente. A notação para descrever estados concorrentes está representada pela linha tracejada na Figura 2.5.

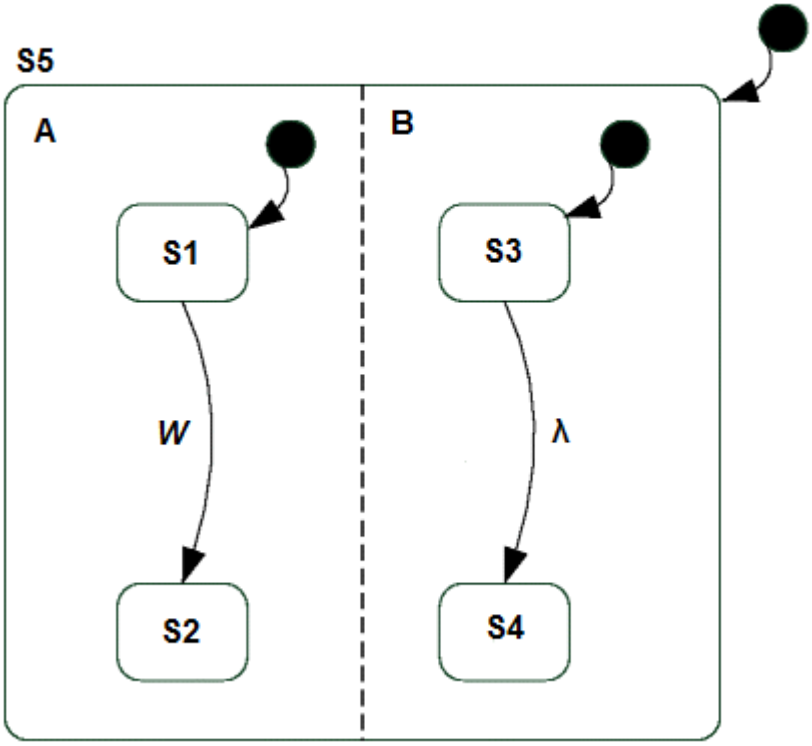


Figura2. 5 - Modelo de Ortogonalidade Statechart.

Na Figura 2.5 o estado composto S5 possui duas regiões separadas por uma linha tracejada, indicando que os estados S1 e S2 ocorrem concorrentemente com os estados S3 e S4. Os eventos W e λ disparam uma transição de S1 para S2 e de S3 para S4, respectivamente.

Por fim, a comunicação broadcast é uma característica incorporada ao Statechart de Harel (1987), em que a ocorrência de um evento permite que duas ou mais transições ocorram. Para isso estas transições precisam estar associadas à eventos de mesmo nome. A Figura 2.6 apresenta um modelo de comunicação broadcast.

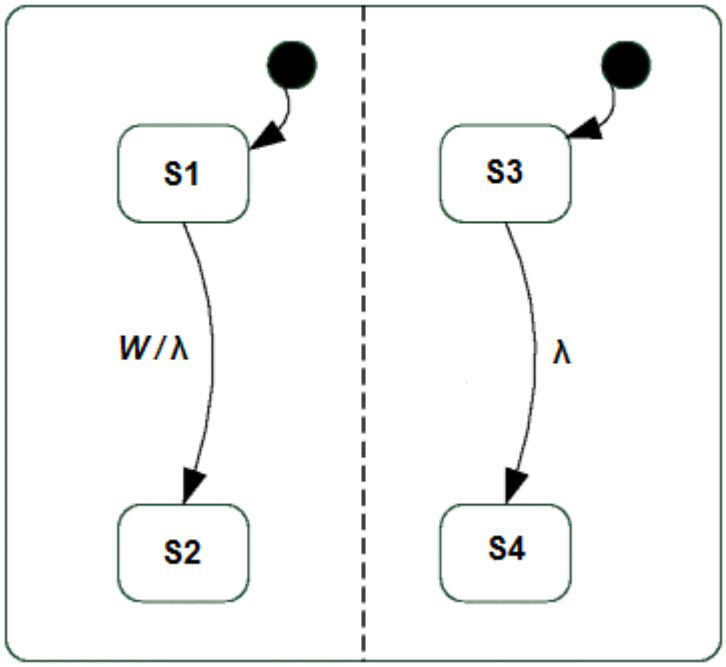


Figura2. 6 - Modelo de Comunicação em Broadcast Statechart.

Na Figura 2.6 ilustra um exemplo de comunicação em broadcast. Um evento W provoca a transição de S1 para S2 e gera uma saída (que também é um evento) λ. Então, λ dispara a transição de S3 a S4 no estado ortogonal.

2.4 Teste de Software em SEs

Teste de software é o processo no qual são aplicadas técnicas e estratégias de acionamento do SE, com a intenção de garantir que o mesmo possua as atribuições descritas em sua especificação. Infelizmente, não existe ainda uma forma completamente padronizada e definida para derivarem-se casos de teste somente segundo a especificação do produto (PFALLER, FLEISCHMANN, *et al.*, 2006).

De forma geral, o teste de software é dividido em três grandes categorias descritas a seguir.

- Teste funcional ou caixa preta: tem por objetivo encontrar discrepâncias entre o programa e as especificações externas por meio de entradas

previamente escolhidas e do conhecimento da saída que tal entrada deve produzir. Uma especificação externa é uma descrição do comportamento do programa do ponto de vista do usuário (requisitos do sistema) (MYERS, SANDLER, *et al.*, 2004). Exemplos de critérios de teste funcionais são: particionamento de equivalência, análise do valor limite, grafo de causa efeito e erro por adivinhação (error guessing).

- Teste Estrutural ou caixa-branca: avalia a estrutura interna de um programa, conduzindo para a avaliação de sua estrutura lógica interna. Essa técnica permite avaliar características internas de um programa tais como o fluxo de controle. Exemplos de critérios de teste estruturais são: todos os nós e todos os arcos (MYERS, SANDLER , *et al.*, 2004).
- Técnica Baseada em Defeitos: tem por objetivo os defeitos que são cometidos com maior frequência pela equipe de desenvolvimento. Exemplos de critérios são: análise de mutantes e semeadura de erros (MYERS, SANDLER, *et al.*, 2004).

Neste trabalho interessa o auto teste orientado para testes estruturais (GIZOPOULOS, PASCHALIS e ZORIAN, 2013), pois ele é o teste presente nas normas de certificação, como a UL 1998 (DESAI, 1998). A forma como esse tipo de teste será utilizado no nível de modelos UML Statechart evidencia-se nos modelos criados no Capítulo 3.

2.5 Normas de Certificação Para SEs

De acordo com exposto no Capítulo 1, os SEs estão presentes em dispositivos utilizados comumente por pessoas. Essa presença maior não só da sua inserção na sociedade, mas da própria variedade e oferta desses sistemas, pode estar disseminando sistemas passíveis de erros decorrentes de defeitos em seus desenvolvimentos. Em decorrência desses erros, falhas que causam acidentes aos usuários passaram a ter mais atenção de Estados que, por conseguinte, começaram a ser desenvolvidos em vários países que definem altos padrões de qualidade, certificações de produto que garantem que defeitos em softwares embarcados sejam

minimizados ou que não permitam que eles resultem em falhas que comprometam o consumidor final, tais como choques e queimaduras.

Sendo assim, a certificação é uma prática cujo objetivo é estabelecer padrões de qualidade ao produto. No geral, seguem os critérios da legislação estabelecida no país onde o produto será utilizado, do tipo de utilização, do usuário, do tipo de desenvolvimento do produto e da arquitetura deste. De forma genérica, a certificação de sistemas embarcados deve garantir:

1) Validação do produto: avaliar se foi desenvolvido o produto correto; e
2) Verificação do produto: avaliar se o produto foi desenvolvido da forma correta.

A forma como a validação e a verificação serão realizadas irá depender da classificação do produto, que é determinada pelos fatores citados anteriormente.

Para fornecer um panorama mais abrangente sobre os tipos de normas de certificação existentes para os diferentes tipos de aplicação e mercado, algumas normas serão citadas no presente trabalho: ISO 14971: Dispositivos médicos, DO 178-b: Indústria Aeronáutica e de Defesa, 61508: Indústria e UL 1998: Indústria e bens de consumo.

Por questão de afinidade com a norma e de todas as normas dedicadas a SEs de tempo real serem bem parecidas como será mostrado ao longo dessa seção, este estudo irá preconizar o uso da UL1998, que por essa razão será mais bem explicada também ao longo desse capítulo.

Vale ressaltar que as normas citadas pertencem a órgãos nacionais ou internacionais que são terceirizados como UL (*Underwrite Laboratories*) ou VDE (*Association for Electrical, Electronic & Information Technologies*). Nesse sentido, a certificação é um meio de transferir a responsabilidade dos riscos inerentes ao sistema do desenvolvedor para o certificador.

O processo de criação e aplicação de certificação é tipicamente aplicado por quem desenvolve e vende o SE, pelo cliente que requisita e compra do software ou por um terceiro, que cobra a certificação garantindo imparcialidade na análise.

No contexto desta pesquisa, a principal vantagem de se aplicar uma norma num SE é garantir que caso um erro ocorra nele, o sistema não irá para um estado indefinido, ou seja, agregar confiabilidade ao sistema. A exemplo da importância de não permitir que um defeito num software embarcado leve o sistema a um estado indefinido, tem-se o caso do foguete Mariner, desviado de seu percurso de voo logo

após o lançamento. O controle da missão destruiu o foguete 293 segundos após a decolagem. Essa falha decorreu de um defeito inserido pelo programador, ao passar para o computador uma fórmula que haviam lhe entregado escrita manualmente, na qual havia se esquecido de uma barra da operação de divisão.

2.5.1 ISO 14971: Dispositivos médicos

O padrão ISO 14971 é usado para dispositivos médicos que possuem componentes programáveis. Segundo esse padrão, a verificação não é simplesmente teste, pois este não é capaz de mostrar a ausência de erros.

Sendo assim, nas subseções DO-178B o termo "verificar" é usado no lugar de "teste" já que o processo de verificação de *software* utiliza uma combinação de revisões, verificações e testes.

De acordo com Caffery e Dorling (2010), para o processo de gestão de risco ISO 14971 definem-se as seguintes áreas de estudo e documentação de desenvolvimento de software.

- Nível de preocupação;
- Descrição Software;
- Risco de Dispositivos e Análise de Risco;
- Especificação de Requisitos de Software;
- Design da Arquitetura;
- Especificações de Design;
- Análise de rastreabilidade de requisitos;
- Verificação, Validação e Teste;
- História nível de revisão; e
- Anomalias não resolvidas;

Fazem parte do escopo dessa norma os testes funcionais e estruturais, os quais analisam a funcionalidade do *firmware* e o código em si, respectivamente.

2.5.2 DO 178-B: Indústria Aeronáutica e de Defesa

Com a evolução da indústria aeronáutica fez-se necessário o desenvolvimento da norma DO-178B *Software Considerations in Airborne Systems and Equipment Certification.* Há nesse padrão considerações especiais para os

sistemas que dizem respeito a problemática da aeronáutica militar. A indústria aeronáutica teve o desafio de se adaptar rapidamente à tecnologia em rápida evolução do *software* embarcado de tempo real. Juntamente a isso, muitas entraram no mercado comercial e se depararam com a necessidade de certificação pela *Federal Aviation Administration* (FAA) ou seu representante. Para os mercados nacionais e outros, os produtos aeronáuticos também devem ser certificados por agências reguladoras.

A certificação significa que os aspectos de *software* de um sistema são seguros e de navegabilidade. Isto é, eles foram desenvolvidos, tal como definido nas orientações de *software* de certificação para o nível de rigor e disciplina exigido pelo seu nível crítico, como determinado por uma avaliação do risco funcional.

2.5.3 IEC 61508: Indústria

A *IEC (International Electrotechnical Commission)* 61508 é um padrão de segurança aplicável a todos os ramos da indústria. A primeira premissa do padrão é a existência de um equipamento sob controle (ESC), um sistema que o controla, e que entre eles, há um risco. O sistema de controle pode ser integrado com o ESC como, por exemplo, por meio de um microprocessador, ou remotamente a partir dele. A ameaça é um "risco da energia mal direcionada".

A segunda premissa é a de que funções de segurança devem ser implementadas a fim de reduzir os riscos decorrentes da interação entre ESC e seu sistema de controle. Funções de segurança podem estar presentes em um ou mais sistemas de proteção, bem como dentro do próprio sistema de controle. Em princípio, a sua separação do sistema de controle é mais recomendada.

Qualquer sistema que for designado para implementar as funções de segurança necessárias de um ESC é definido como um sistema de segurança e é a esse que o padrão IEC 61508 se aplica.

Na Tabela 2.1 é possível observar como a IEC 61508 se relaciona com os componentes de microprocessador, descritos na Seção 2.2.1.

Tabela 2. 1- IEC60508 - Cobertura para modos de falha

Componente		**Falha**
1. CPU	Registradores	Falha DC
	Contador de Programa	-
2. Interrupção		Nenhuma interrupção ou Interrupções muito frequentes
3. Relógio*		Frequência errada
4. Memória	Não volátil	Falhas de único bit e de bits duplos
	Volátil	Falha DC e de ligações cruzadas dinâmicas
	Endereçamento	Falha DC
5. Caminhos de dados internos		Falha DC Endereço errado ou múltiplos endereços
6. Comunicação externa	Dados	Falhas de único bit e de bits duplos
	Endereço	Endereçamento errado
	Tempo	Erro de temporização
7. Entradas/Saídas	E/S digitais	Circuito aberto ou curto-circuito
	E/S analógica: Conversores A/D e D/A	Circuito aberto ou curto-circuito
	E/S Analógica Multiplexadas	Endereçamento errado

Na primeira coluna da Tabela 2.1 estão enunciados de forma genérica os componentes que podem estar presentes num microcontrolador. Alguns deles estão subdivididos, pois podem ser diferenciados, como a memória que pode ser volátil ou não volátil, ou memória de endereçamento ou barramento interno. Na coluna à direita estão listadas as falhas potenciais inerentes a cada componente do microprocessador. A definição dessas falhas, bem como a forma como essas falhas foram pensadas e posteriormente definidas como falhas potenciais a serem cobertas pela norma, serão explicadas na Seção 2.3.4, quando será mostrada inclusive a semelhança entre IEC61508 e UL1998.

2.5.4 UL1998

Seguindo a tendência de O desenvolvimento da norma UL 1998 teve início em 1988 com uma revisão das normas existentes. O objetivo dessa revisão foi identificar requisitos relevantes que poderiam ser escolhidos para a aplicação práticas que beneficiassem o consumidor de produtos industriais.

Durante o desenvolvimento inicial, verificou-se que muitos dos padrões disponíveis no momento baseavam-se principalmente nos requisitos do processo de engenharia. Esses critérios de processo forneceram uma fundamentação à UL 1998, no entanto, um objetivo fundamental era aumentar esses critérios deixando a codificação à prova de falhas e tolerante a falhas de requisitos de projeto de software que estivessem relacionados à segurança.

A Norma UL1998 é limitada a aplicações específicas, com um único componente programável embutido no produto, no qual uma falha pode resultar em um risco de incêndio, choque elétrico ou ferimentos as pessoas. A norma UL 1998 alcançou o *status* de Padrão ANSI (*American National Standards Institute*) em fevereiro de 1999.

É considerado um componente embutido programável qualquer *hardware* onde possa ser embarcado um *software* e cujo mesmo possa sofrer alterações. Uma configuração mais geral, pode também incluir, um sistema operacional ou *software* executável, um *software* de comunicação, um microcontrolador, uma entrada de rede e saída de hardware e um software de interface do usuário.

Os requisitos da norma UL 1998 se destinam principalmente a evitar os riscos que ocorrem no processo de desenvolvimento e manutenção do *software*. Ao aplicar esses requisitos à norma deve ser capaz de examinar as diferentes fases do ciclo de desenvolvimento de *software*, as etapas de projeto, a implementação e os testes. A norma exige que seja feita uma análise para determinar o conjunto de riscos inerentes à aplicação. A análise deve identificar os estados e as transições que são capazes de resultar em risco, na prática isso significa não só identificar riscos, mas também traçar uma relação entre eles e suas causas para assim agir no ponto certo, assim a norma também é provida das ações que deverão ser tomadas caso os riscos sejam identificados. Essas frequentemente estão relacionadas a testes estruturais conhecidos o que facilita sua aplicação e garante eficiência do método empregado.

Para uma maior clareza de como a análise de risco e o procedimento, ou teste, a ser seguido a fim de minimizar esse risco para cada componente do SE, tem-se uma versão revisada e simplificada da Tabela UL1998 na Tabela 2.2:

Tabela 2. 2 - UL1998 - Cobertura para modos de falha

Componente		**Falha**	**Exemplos de testes aceitáveis**
1. CPU	Registradores	Stuck-at	Teste funcional; ou Auto teste periódico usando: - Teste de memória estático - Proteção de escrita com redundância single-bit
	Contador de Programa	Stuck-at	Teste funcional; ou Auto teste periódico; ou Monitoramento de tempo; ou Monitoramento de sequência
2. Interrupção		Nenhuma interrupção ou Interrupções muito frequentes H2;S1	Auto teste periódico; ou Monitoramento de tempo
3. Relógio		Frequência errada H2;S2	Monitoramento de frequência; ou Monitoramento de tempo
4. Memória	Não volátil	Falhas de único bit	Teste periódico com checksum; ou Múltiplos checksums; ou Proteção usando single bit
	Volátil	Falha DC	Teste de memória estático periódico; ou Proteção usando single bit
	Endereçamento (Não volátil e volátil)	Stuck-at	Proteção com checksum incluindo o endereço
5. Caminhos de dados internos		Endereço errado ou múltiplos endereços	Proteção com checksum incluindo o endereço
6. Comunicação externa	Dados	Erros de único bit	Proteção com redundância multi-bit; ou CRC; ou Redundância; ou Protocolo
	Endereço	Endereçamento errado	Proteção com redundância multi-bit; ou CRC; ou Redundância; ou Protocolo
	Tempo	Erro de temporização	Monitoramento de tempo
7. Entradas/Saídas	E/S digitais	Circuito aberto ou curto-circuito	Verificação de plausibilidade
	E/S analógica: Conversores A/D e D/A	Circuito aberto ou curto-circuito	Verificação de plausibilidade
	E/S Analógica Multiplexadas	Endereçamento errado	Verificação de plausibilidade

Na primeira coluna da Tabela 2.2 têm-se os componentes que compões o sistema microcontrolado onde será embarcado o código, assim como na Tabela 2.1, a primeira coluna está subdividida: à esquerda temos uma definição mais genérica dos componentes de um SE e à direita uma mais específica. Essa abordagem foi

feita com o único intuito de facilitar a compreensão ao leitor com menos intimidade na área de embarcados. Na segunda coluna estão listadas as falhas potenciais de cada componente e por fim, na terceira coluna, foram enumerados os testes aceitáveis para garantir que a possível defeito presente no código referente a cada componente não gere falha que incorra no risco ao usuário.

2.6 Taxonomia de Defeitos em nível de modelo

A taxonomia de defeitos costuma ser uma referência para os defeitos que se desejam encontrar numa técnica de leitura voltada a correção em modelos SE. De maneira geral, quanto ao defeito do software, deve se basear na sua categoria, prioridade, situação e severidade (WALIA e CARVER, 2009).

Este trabalho usará, de forma um pouco otimizada, a própria taxonomia descrita pela UL1998, que contempla os defeitos de software cuja localização em nível de modelo de SE são um dos principais interesses desse estudo e a taxonomia criada por Antônio (2014), cujo trabalho de doutorado também utilizou a norma UL 1998.

2.7 Considerações Finais

No presente capítulo foram considerados os principais fundamentos teóricos que serviram de embasamento para elaboração da Estratégia UL/ME, que será apresentada no Capítulo 3. Foi apresentada uma visão geral de Sistemas Embarcados (SEs). Depois foram apresentadas as principais características da técnica de modelagem de SEs UML Statechart, que será utilizada no presente estudo. Foram também abordados os testes de SEs mais comumente utilizados em SEs, destacando-se o auto teste, que é o utilizado para cumprirem-se os requisitos das normas de certificação. Foi também apresentado um apanhado geral de exemplos de normas de certificação de domínios de aplicação médico, aviônico e industrial, em especial da norma UL 1998, que será a utilizada neste trabalho, por ter

similaridades com as outras normas e por ser a norma adequada para certificação dos SEs reais que serão utilizados para avaliação da Estratégia UL/ME no Capítulo 4. Por fim foi feita uma breve abordagem sobre a taxonomia de defeitos em SE, decidindo-se pelo uso da taxonomia utilizada pela própria norma de certificação, UL 1998, juntamente com a taxonomia criada por Antônio (2014).

Com base nos estudos apresentados neste capítulo, no capítulo seguinte, apresentam-se as etapas para a criação de uma estratégia de identificação de defeitos, de acordo com os requisitos da norma de certificação UL 1998, em modelos de SEs e propostas para o uso dessa estratégia.

Capítulo 3

ESTRATÉGIA UL 1998 DE MODELAGEM

Nesse capítulo são exploradas as relações entre a norma UL1998 e os recursos da modelagem UML Statechart para SEs com objetivo de criar uma estratégia com modelos UML Statechart que facilitem a certificação de SEs, que é a maior motivação do presente estudo.

3.1 Considerações Iniciais

Este capítulo apresenta as fases realizadas para contemplar o objetivo proposto no Capítulo 1, que é o de criar uma estratégia baseada em modelos UML Statechart que facilitem a correção e o desenvolvimento de SEs que atendam aos requisitos da norma UL 1998, que será denominada Estratégia UL/ME.

Partindo deste objetivo e da revisão literária realizada no Capítulo 2, desenvolveram-se modelos genéricos de componentes de SEs, descritos em UML Statechart, para identificar falhas potenciais e apresentar, além desse modelo de falha, um possível modelo genérico que corrija ou enderece uma possível falha a um estado seguro, de acordo com a norma de certificação UL1998.

A primeira fase deste estudo ocorreu em conjunto com Antônio (2014), realizando um mapeamento de defeitos tratados pela norma UL 1998 e criando uma tabela, baseada na Tabela 2.2, cuja utilização pelo desenvolvedor fosse facilitada por uma melhor descrição das falhas a serem observadas, dos testes aceitos pela

norma e de palavras chaves que auxiliassem na detecção de cada componente do SE citado pela UL1998.

Na segunda fase foram desenhados modelos genéricos para cada componente e também os modelos de componentes SE que representam os testes aceitos pela norma. Observando-se que esses testes seriam, na forma de aplicação geral da norma seguida pela maioria dos desenvolvedores, inseridos somente no código escrito, ou seja, já numa fase mais avançada do desenvolvimento em que a correção dos erros gera gastos de recursos maiores que os das fases de modelagem do projeto de SE.

Na terceira fase foram detalhadas as falhas de interesse da norma, os testes aplicados a cada componente para evitar que possíveis defeitos de software venham se manifestar nas falhas descritas pela mesma.

Por fim, apresentam-se, na quarta fase, duas propostas de uso da Estratégia UL/ME em SE, a primeira para correção de SE que apresentem defeitos segundo a norma e a segunda para desenvolvimento em engenharia avante.

Assim, este capítulo está estruturado da seguinte forma: na Seção 3.2 apresenta-se a criação de uma tabela que correlaciona os componentes de um SE e as possíveis falhas que os componentes podem manifestar, com palavras chaves, que tem por objetivo facilitar a identificação destes componentes em nível de modelo. A criação destas palavras chaves foi feita por adaptação e reuso da taxonomia de defeitos criada por Antônio (2014). Na Seção 3.3, com base na tabela elaborada na seção anterior, foram criados modelos UML Statechart genéricos que representassem os componentes do SE modelados em não conformidade com a norma UL1998. Em seguida, foram criados modelados UML Statechart com os testes requeridos pela norma UL 1998. Na Seção 3.4 apresentam-se os passos das propostas de uso da Estratégia UL/ME. Na Seção 3.5 encerra-se o capítulo.

3.2 Criação de palavras chave e taxonomia de defeitos

Com base na experiência adquirida no trabalho com Antônio (2014) utilizada em seu doutorado (ANTÔNIO, 2014), desenvolveu-se a Tabela 3.1, cujo objetivo é estabelecer a correlação entre os componentes de SEs e suas falhas, segundo a

norma UL 1998 e já retratados na Tabela 2.2, com palavras chaves que possam auxiliar na detecção desses componentes em nível de modelo e uma Taxonomia de defeitos que possa ser utilizada como marcação dessas falhas componente detectadas no modelo.

Tabela 3. 1 - Mapeamento da UL 1998: Palavras chave e taxonomia de defeitos.

Componente		Falha	Palavras Chave	Taxonomia de defeito
1. CPU	Registradores	Stuck-at	Registrado	<<UL98CPURegister>>
	Contador de Programa	Stuck-at	Contador	<<UL98CPUProgramCounter>>
2. Interrupção		Nenhuma interrupção ou Interrupções muito frequentes H2;S1	Comunicação serial, RS232, RS485, modem MNP2, modem V.2, botão, chamar, PWM, Frequência, botão, chave, ligado, desligado, clique, a decisão	<<UL98CPUInterrupt>>
3. Relógio		Frequência errada H2;S2	Relógio, tempo (abreviações da palavra tempo)	<< UL98Clock>>
4. Memória	Não volátil	Falhas de único bit	Memória externa, memória estática, usb	<<UL98NoVoltatileMemory>>
	Volátil	Falha DC	EEPROM, Memoria Flash	<< UL98VoltatileMemory >>
	Endereçamento (Não volátil e volátil)	Stuck-at	Endereço de memória	<< UL98MemoryAddress >>
5. Caminhos de dados internos		Endereço errado ou múltiplos endereços	Não se aplica	Não se aplica por tratar da topologia interna do microcontrolador.
6. Comunicação externa	Dados	Erros de único bit	HDLC, Ethernet, PPP e USB.	<< UL98CommunicationData >>
	Endereço	Endereçamento errado	IP, protocolo	<< UL98CommunicationAddress >>
	Tempo	Erro de temporização	Tempo de comunicação, relógio de comunicação, tempo limite de comunicação,	<< UL98CommunicationTime >>
7. Entradas/Saídas	E/S digitais	Circuito aberto ou curto-circuito	LED, lâmpada, chave seletora, microfone	<< UL98DigitalIO>>
	E/S analógica: Conversores A/D e D/A	Circuito aberto ou curto-circuito	Temperatura, acelerômetro resistência, som, tensão, corrente, fone de ouvido, portão, chute	<< UL98AnalogIO >>
	E/S Analógica Multiplexadas	Endereçamento errado		<< UL98AnalogMultiplex >>

O mapeamento da UL 1998 presente na Tabela 3.1 foi construído da seguinte forma: a partir de cada componente do sistema (coluna 1) e da possível falha associada a este (coluna 2), definiram-se as palavras-chave (coluna 3). As palavras-chave foram extraídas a partir do no trabalho com Antônio (2014) e complementadas com o estudo de exemplos de Modelos UML Statechart de SEs de vários domínios de aplicação distintos. Esses modelos foram obtidos a partir da pesquisa bibliográfica e estão apresentados, bem como as palavras chaves que vieram de cada modelo, no Apêndice A.

No presente estudo, a construção da Tabela 3.1 foi importante para melhor visualizar as possibilidades da norma, especialmente no que se refere a coluna “Palavras Chaves”. Nessa coluna foram exemplificados diferentes periféricos que podem se conectar ao mesmo componente do SE. Essa diferenciação depende das funcionalidades atribuídas ao SE e do seu domínio de aplicação. Por exemplo, numa aplicação para doméstica de uma torradeira (Figura 5.2 do Anexo A), o componente “saída digital” pode ser identificado pela palavra *lamp* (lâmpada). Numa aplicação do mesmo domínio, porém com outras funcionalidades, como da Figura 5.3 do Anexo A, portão de garagem, o mesmo componente de SE, uma saída digital, é identificado pela palavra *abre_portão* (open_gate). Na Figura 5.7 do Anexo A apresenta-se o exemplo de domínio de aplicação robótico, nela uma das saídas digitais é *chute_bola* (Kick_ball), que é um sinal para que o robô da aplicação chute a bola. Embora, aparentemente diferentes, no modelo UML Statechart de ambos os SE, *lâmpada, abre_portão* e *chute_bola*, são saídas digitais, podendo ser modelados simplesmente como um componente de saída digital e, segundo a norma UL 1998, os testes para atestar sua confiabilidade, seriam os mesmos.

Referindo-se ao modelamento dos SEs citados no parágrafo anterior, se o desenvolvedor os nomeasse simplesmente como: saída digital a *lâmpada*, o *abre_portão* ou o *chute_bola*, a relação com o componente de SE dar-se-ia facilmente. Se, por outro lado, o desenvolvedor especificar o componente segundo seus periféricos e domínio de aplicação, a coluna “Palavras Chaves” da Tabela 3.1, pode servir de auxílio para um desenvolvedor que não esteja contextualizado com a aplicação, permitindo que ele identifique o componente do SE.

Ainda se tratando da Tabela 3.1, observa-se que algumas palavras chaves se repetem para diferentes componentes, contudo, como dito anteriormente, o uso da tabela destina-se a desenvolvedores SE, que não possuam conhecimento

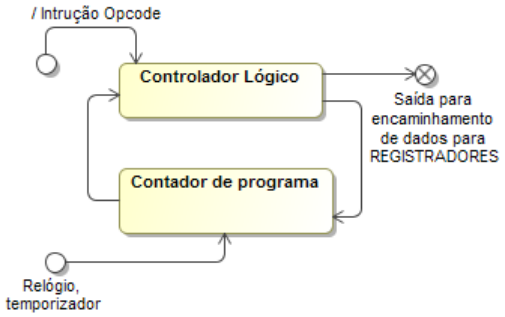
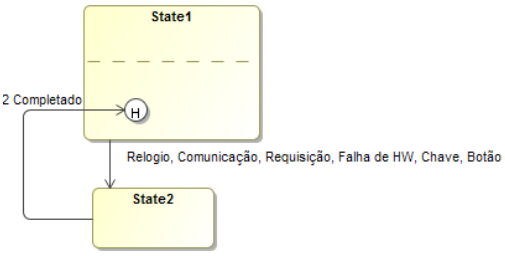
aprofundado de todos os domínios de aplicação, mas supõe-se minimamente capacitados, podendo discernir entre uma entrada de temperatura é ligada ao um componente de entrada analógica ou a uma entrada analógica multiplexada (linha 8 da Tabela 3.1, informação inerente ao conhecimento da arquitetura do microcontrolador.

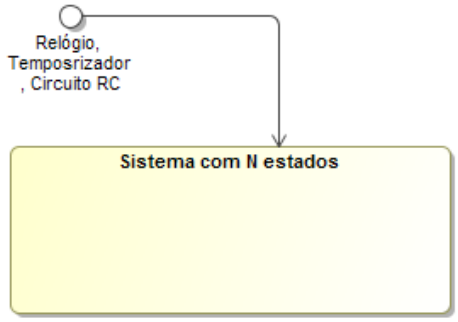
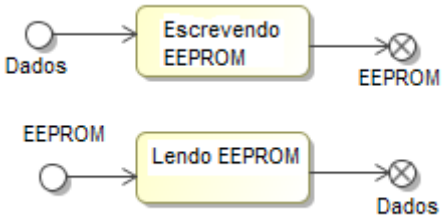
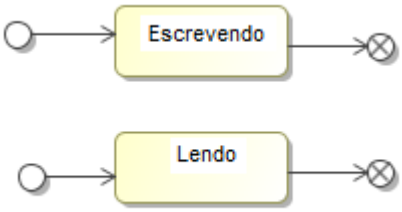
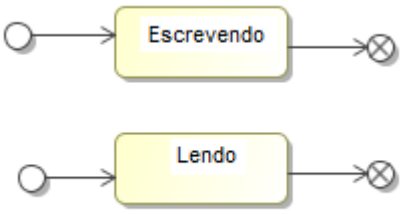
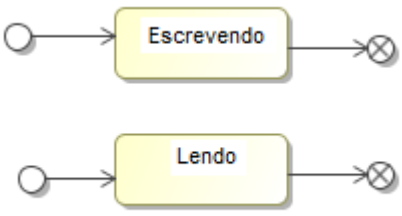
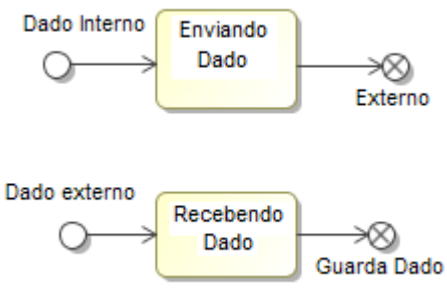
3.3 Criação dos modelos genéricos de componentes de SEs baseados na UL 1998

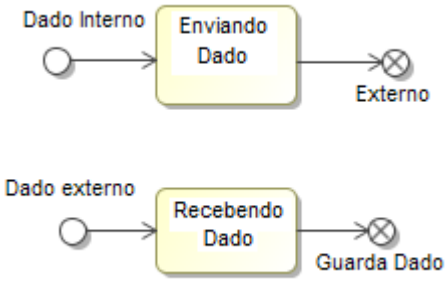
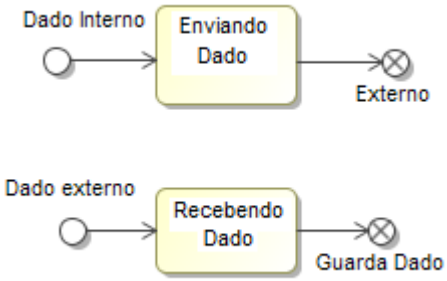
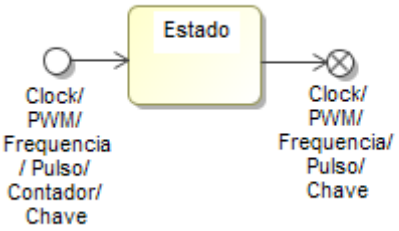
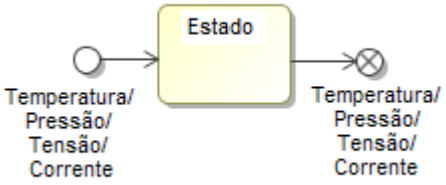
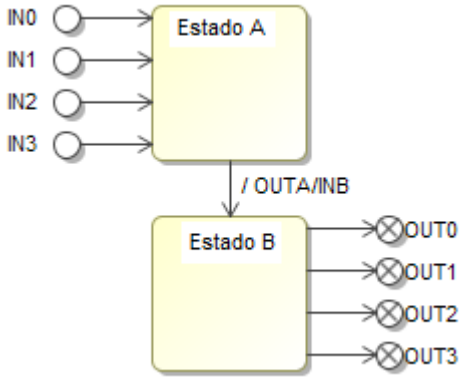
Depois de construída a tabela da Seção 3.2, foram modelados exemplos de componentes em UML Statechart de forma genérica para que, dispondo deste modelo e das informações da Tabela 3.1, qualquer profissional da área de SE tenha condições, mesmo que de maneira superficial, de correlacionar modelos de SEs com a norma UL 1998.

A Tabela 3.2, a seguir, apresenta o mapeamento da UL 1998.

Tabela 3. 2 - Mapeamento da UL 1998: Modelo de Componente.

Componente		Falha	Exemplos de testes aceitáveis	Detecção do modo potencial de falha no modelo
1. CPU	Registradores	Stuck-at H1	Teste funcional; ou Auto teste periódico usando: - Teste de memória estático - Proteção de escrita com redundância single-bit	
	Contador de Programa	Stuck-at H1	Teste funcional; ou Auto teste periódico; ou Monitoramento de tempo; ou Monitoramento de sequência	
2. Interrupção		Nenhuma interrupção ou Interrupções muito frequentes H2;S1	Auto teste periódico; ou Monitoramento de tempo	

3. Relógio		Frequência errada H2;S2	Monitoramento de frequência; ou Monitoramento de tempo	
4. Memória	Não volátil	Falhas de único bit H3	Teste periódico com checksum; ou Múltiplos checksums; ou Proteção usando single bit	
	Volátil	Falha DC H4	Teste de memória estático periódico; ou Proteção usando single bit	
	Endereçamento	Stuck-at H1	Proteção com checksum incluindo o endereço	
5. Caminhos de dados internos		Endereço errado ou multiplos endereços S3	Proteção com checksum incluindo o endereço	
6. Comunicação externa	Dados	Erros de único bit H3;S3	Proteção com redundância multi-bit; ou CRC; ou Redundância; ou Protocolo	

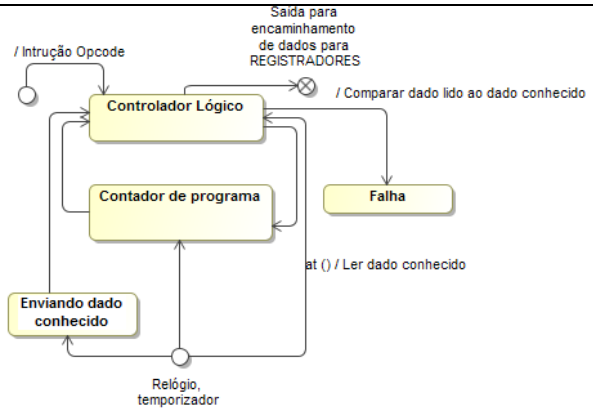
	Endereço	Endereçamento errado S3	Proteção com redundância multi-bit; ou CRC; ou Redundância; ou Protocolo	
	Tempo	Erro de temporização S3	Monitoramento de tempo	
7. Entradas/Saídas	E/S digitais	Circuito aberto ou curto-circuito H4	Verificação de plausibilidade	
	E/S analógica: Conversores A/D e D/A	Circuito aberto ou curto-circuito H4	Verificação de plausibilidade	
	E/S Analógica Multiplexadas	Endereçamento errado S3 H4	Verificação de plausibilidade	

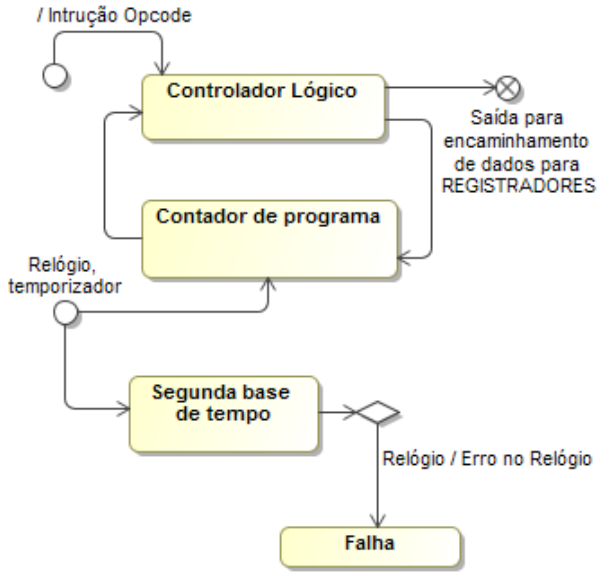
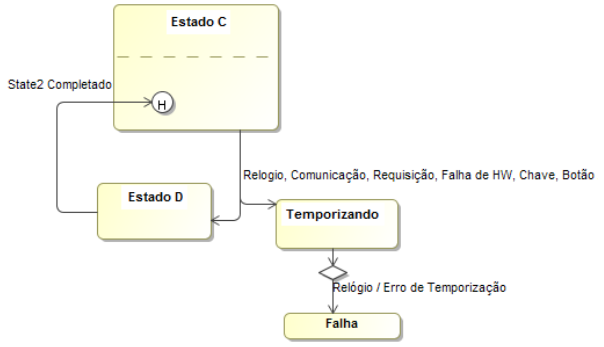
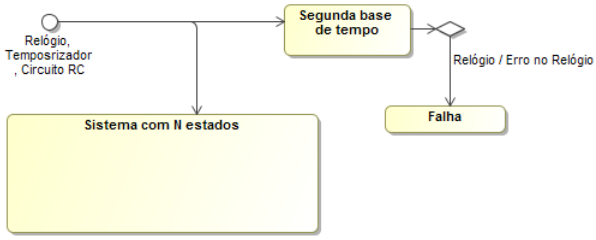
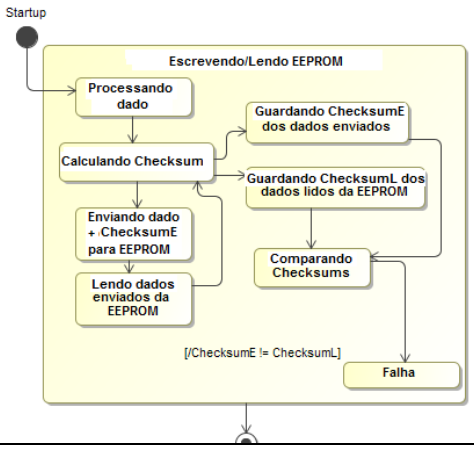
A Tabela 3.2 é uma versão da Tabela 2.2 acrescida, dos modelos UML Statechart dos componentes de SE que a UL 1998 propõe verificar. Como citado anteriormente, ter conhecimento de que cada componente pode diferenciar-se dependendo da aplicação a que se destinam auxiliou no desenvolvimento dos modelos, como ficou explícito na Tabela 3.1, contudo a Tabela 3.2, por ter um propósito diferente da Tabela 3.1, pode contemplar todo o protocolo UL 1998.

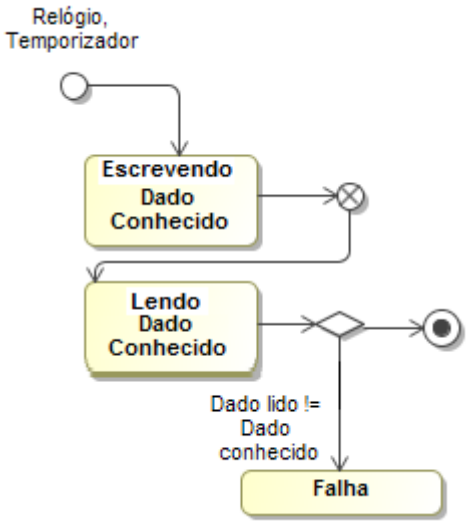
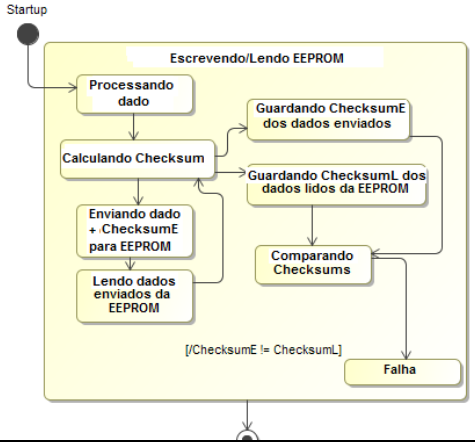
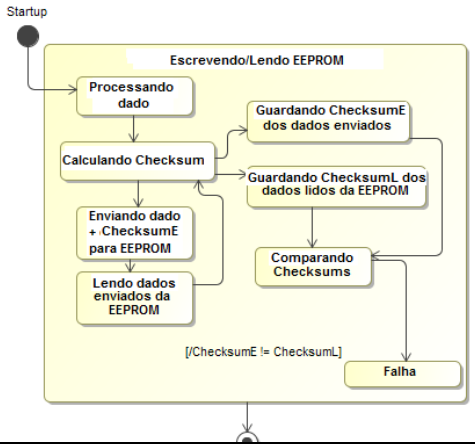
Dispondo de modelos genéricos a serem identificados em Statechart que representam componentes de SEs que não comtemplam os requisitos da norma UL 1998, o próximo passo desta pesquisa foi remodela-los, com objetivo de, nesse processo, inserir os testes propostos na coluna da Tabela 3.2 “Exemplos de testes aceitáveis”. Evidentemente, seria um trabalho muito extenso modelar todos os testes possíveis, assim, por facilidade de pesquisa, elencaram-se os testes que se repetiam mais vezes, independente do componente, escolhendo-se apenas um teste para cada componente.

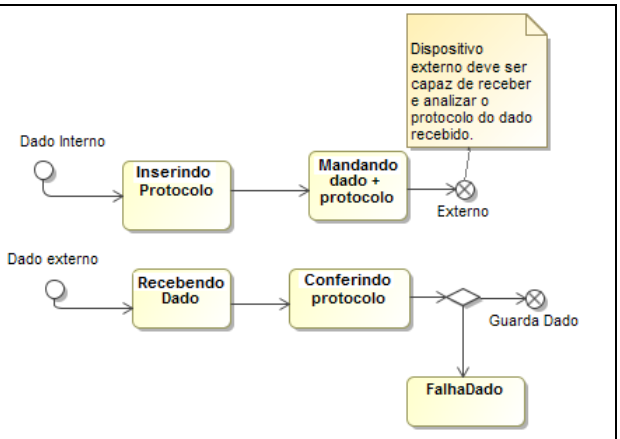
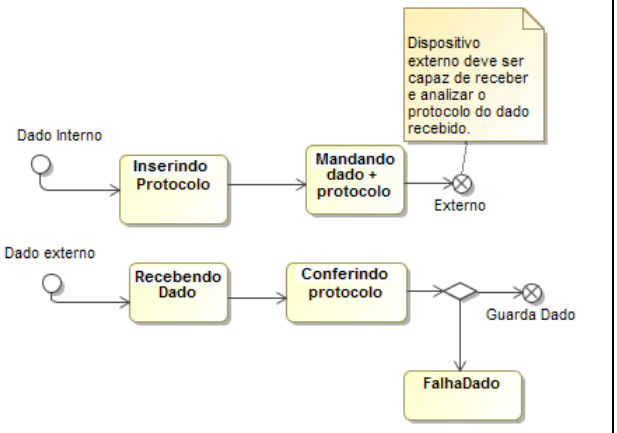
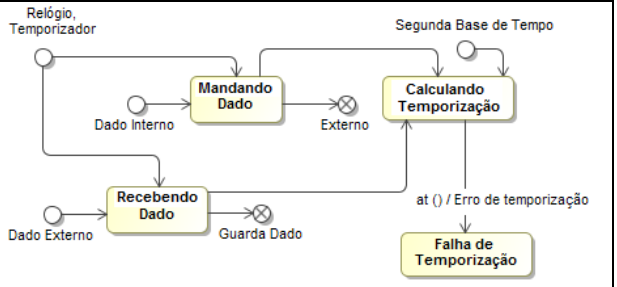
A Tabela 3.3, a seguir, apresenta o mapeamento da UL 1998, levando-se em consideração o exposto no parágrafo anterior.

Tabela 3. 3 - Mapeamento da UL 1998: Modelo de Teste de Componente.

Componente		Falha	Exemplos de testes aceitáveis	Teste de prevenção Proposto em modelo
1. CPU	Registradores	Stuck-at [H1]	Teste de memória estático[T1]	

	Contador de Programa	Stuck-at H1	Monitoramento de tempo T2	
2. Interrupção		Nenhuma interrupção ou Interrupções muito frequentes H2;S1	Monitoramento de tempo T2	
3. Relógio		Frequência errada H2;S2	Monitoramento de tempo T2	
4. Memória	Não volátil	Falhas de único bit H3	Teste periódico com checksum T3	

<table>
<tr>
<td rowspan="2"></td>
<td>Volátil</td>
<td>Falha DC [H4]</td>
<td>Teste de memória estático periódico [T4]</td>
<td> 
</td>
</tr>
<tr>
<td>Endereçamento</td>
<td>Stuck-at [H1]</td>
<td>Checksum incluindo o endereço (Não volátil e volátil) [T5]</td>
<td> 
</td>
</tr>
<tr>
<td colspan="2">5. Caminhos de dados internos</td>
<td>Endereço errado ou multiplos endereços [S3]</td>
<td>Checksum incluindo o endereço [T5]</td>
<td> 
</td>
</tr>
</table>

6. Comunicação externa	Dados	Erros de único bit H3;S3	Protocolo T6	
	Endereço	Endereçamento errado S3	Protocolo T6	
	Tempo	Erro de temporização S3	Monitoramento de tempo T2	
7. Entradas/Saídas	E/S digitais	Circuito aberto ou curto-circuito H4	Verificação de plausibilidade T7	Não se aplica
	E/S analógica: Conversores A/D e D/A	Circuito aberto ou curto-circuito H4	Verificação de plausibilidade T7	Não se aplica
	E/S Analógica Multiplexadas	Endereçamento errado S3 H4	Verificação de plausibilidade T7	Não se aplica

H*: Falhas de hardware; S*: Falhas de Software; T*: Testes aplicados

Para facilitar a leitura da Tabela 3.3, a coluna “Detecção do modo potencial de falha no modelo” foi suprimida, dando mais espaço a coluna “Teste de prevenção

Proposto em modelo". Para melhor caracterizar as falhas de hardware e de software e os testes a serem aplicados, representados pelos modelos da coluna "Teste de prevenção Proposto em modelo", foram introduzidas as legendas H*, S* e T*, respectivamente, em que "*" é o número da falha ou teste. Esses itens têm seu detalhamento apresentado no Apêndice A. O Apêndice A servirá também para criação de testes para o SE em na atividade de criação de testes de acordo com a norma UL 1998, proposta na Seção 3.4.

É importante notar que, para alguns componentes, nenhum dos testes sugeridos pela UL 1998 são aplicáveis aos modelos propostos, pois dependeriam da introdução de hardware adicional, o que não convém modelar, visto que o presente trabalho está voltado à aplicação das normas de certificação no contexto de software embarcado.

3.4 Propostas de uso da Estratégia UL/ME

Nessa seção serão apresentadas duas propostas de utilização da Estratégia UL/ME para auxiliar o desenvolvimento de SE que contemplem os requisitos da norma UL 1998. A Estratégia UL/ME é um processo que faz uso das tabelas criadas neste capítulo para auxiliar o desenvolvedor a criar softwares embarcados livres dos defeitos que foram abordados até o presente momento.

Para facilitar o entendimento da estratégia, as propostas de uso da mesma serão divididas em etapas que por sua vez, estarão subdivididas em atividades.

A primeira proposta de uso da Estratégia UL/ME utiliza engenharia reversa e reengenharia para corrigir códigos já implementados que apresentem defeitos, de acordo com os requisitos UL 1998. A segunda proposta de uso da Estratégia UL/ME se aplica no desenvolvimento avante de SEs.

3.4.1 Estratégia UL/ME utilizando Engenharia Reversa e Reengenharia

Nesta primeira proposta o objetivo do uso da Estratégia UL/ME é corrigir SEs que já tenham sido implementados e código, mas não estejam consistentes com os padrões da norma UL 1998. Na Tabela 3.4 apresenta as atividades realizadas em

cada etapa do processo de utilização da Estratégia UL/ME na correção de um SE já implementados.

Tabela 3. 4 - Etapas e atividades realizadas para correção de um SE

Etapas	Atividades
Etapa A: Fazer engenharia reversa no código para construção do modelo original	(I) Traduzir o código original em modelo UML Statechart. Se necessário, utilizar-se das palavras chaves da Tabela 3.1 para identificar no código os componentes do SE.
Etapa B: Identificar os componentes do SE no modelo original	(II) Encontrar as palavras chaves descritas na Tabela 3.1
	(III) Marcar com os estereótipos da Tabela 3.1 os componentes que foram identificados em (II)
Etapa C: Corrigir o modelo do código original	(IV) Comparar os componentes marcados em (III) com os modelos da Tabela 3.2 e marcar os que estão representados da mesma forma da tabela como modelos a serem corrigidos
	(V) Corrigir os modelos marcados na atividade (IV) de acordo os modelos descritos na Tabela 3.3
Etapa D: Traduzir o modelo correto para código	(VI) De forma objetiva, utilizando softwares que traduzam modelos UML Statechart em código, ou, de forma subjetiva, traduzir o modelo corrigido em código
Etapa E: Avaliar código final.	(VII) Criar testes de acordo com as especificações do SE que garantam que ele manteve suas funcionalidades depois de passar por todo processo descrito até (VI)
	(VIII) Executar os testes criados em (VII) e verificar se o código final comtempla todas as funcionalidades propostas em sua especificação. Em caso negativo retornar às atividades anteriores em busca de erros que incorram na perda de funcionalidades do SE
	(IX) Criar testes de acordo com a norma UL 1998 que verifiquem se o código novo atende aos requisitos da norma
	(X) Executar os testes criados em (IX) e verificar se o código final comtempla todas as funcionalidades propostas em sua especificação. Em caso negativo retornar às atividades anteriores em busca de erros no uso da Estratégia que resultem num código que não atende aos requisitos da UL 1998.

A seguir, descrevem-se as atividades mencionadas na Tabela 3.4.

Etapa A: atividade (I) – Traduzir o código original em modelo UML Statechart:

Traduzir o código embarcado original, que não está de acordo com os requisitos da norma, em um modelo UML Statechart que represente o código. Este processo pode ser feito de forma objetiva, através de ferramentas, ou se forma subjetiva, pelo próprio desenvolvedor. Fazendo-o de forma subjetiva, deverá contar com seu conhecimento em técnicas de modelagem ou com técnicas disponíveis na literatura para auxilia-lo (YU, WANG , *et al.*, 2005) evitando incorrer em modelos que não representem o código.

Etapa B: atividade (II) – Encontrar as palavras chaves descritas:

Procurar pelas palavras chaves da Coluna 3 da Tabela 3.1 no modelo Statechart criado na atividade anterior.

Etapa B: atividade (III) – Marcar com os estereótipos no modelo:

Utilizar novamente a Tabela 3.1 para correlacionar as palavras chaves encontradas no modelo na atividade anterior, com a Taxonomia de defeitos da quarta coluna da mesma tabela. Utilizar a taxonomia de forma que fique explícito no modelo onde está cada componente do SE de interesse deste trabalho. Além de escrever o estereótipo entre “<<>>”, aconselha-se demarcar toda a área que representa este estereótipo e, consequentemente, o componente do SE.

Etapa C: atividade (IV) – Comparar os componentes marcados:

Agora que os componentes do SE estão demarcados, comparar cada componente do modelo do código original com o respectivo componente representado na Tabela 3.2, Coluna 4. Caso o modelo do componente SE esteja igual ao modelo representado na quarta coluna da tabela, marque-o como incorreto. Se o componente não estiver modelado de forma similar ao da Tabela 3.2, compare-lo com o componente da Tabela 3.3. Caso esteja similar a este, marque o componente como correto. Caso o componente do modelo do código original não se assemelhe ao componente respectivo da Tabela 3.2 ou 3.3, marque-o como inconclusivo.

Etapa C: atividade (V) – Corrigir os modelos de componentes:

Os componentes marcados como incorretos no modelo do código original na atividade anterior devem ser corrigidos com auxílio dos modelos da Coluna 4 da Tabela 3.4. Depois de corrigir todos os componentes marcados como incorreto, ter-se-á um modelo corrigido.

Etapa D: atividade (VI) – Traduzir o modelo UML Statechart em um novo código:

Traduzir o modelo corrigido, gerado na atividade anterior em código embarcado na mesma linguagem do código original. Da mesma forma que na Atividade I, essa tradução pode ser feita de forma objetiva ou subjetiva.

Etapa E: atividade (VII) – Criar testes de acordo com a especificação do SE

A especificação de um SE diz respeito às funcionalidades inerentes a este sistema. Criar uma tabela de testes, que podem ser do tipo caixa preta, que verifiquem se as principais funções do SE ainda estão presentes no mesmo. Essa atividade visa garantir que, durante as atividades para correção do código original em relação aos requisitos da norma UL 1998, nenhum defeito que afete as funcionalidades do SE foi inserido. É importante criar um critério que defina se o SE passou ou falhou no teste, bem. Por exemplo, se o SE que está passando por esta proposta de uso da Estratégia UL/ME é um leitor de DVD (Digital Versatile Disc), criar um teste ler um DVD conhecido, se o SE realizar a leitura de forma esperada, ele passou no teste, se ele não for capaz de ler o DVD ou lê-lo de forma errada, o SE falhou. Este é um exemplo meramente ilustrativo.

Etapa E: atividade (VIII) – Executar testes de especificação

Executar os testes da etapa anterior. Caso o SE falhe em algum teste, o que significa ter perdido uma funcionalidade a qual deveria atender, as atividades anteriores devem ser revisadas a fim de encontrar o erro. Não é possível aqui especificar qual ou quais atividades geraram a perda de funcionalidade do SE. Uma possível solução é repetir todas as atividades desde o princípio.

Etapa E: atividade (IX) – Criar testes de acordo com a norma UL 1998

Utilizar a terceira coluna da Tabela 3.3, *Exemplos de testes aceitáveis,* e o Apêndice A para criar uma tabela que contenha cada componente do SE, o teste que deve ser realizado a fim de verificar a presença dos requisitos da norma UL 1998 no componente e o critério usado para verificar se o SE passou ou falhou no teste. Por exemplo, o software de um componente de memória não volátil, segundo a Tabela 3.3, deve realizar um teste periódico que use checksum (relação matemática entre os dados enviados e recebidos). Sabendo a periodicidade desse teste, forçar comunicação do SE com a memória não volátil e verificar se o SE está usando checksum. A informação da periodicidade do teste, ou de quanto em quanto tempo ele é realizado, está presente no próprio código. A verificação da presença do checksum pode ser feita por uso de ferramentas de debug ou meios de comunicação externa do SE.

Etapa E: atividade (X) – Executar testes de acordo com a norma UL 1998

Executar os testes da etapa anterior. Caso o SE falhe em algum teste, verificar se o modelo corrigido está realmente de acordo com a Tabela 3.3. Em caso afirmativo o, retomar as atividades a partir da Etapa D.

3.4.2 Estratégia UL/ME utilizando Engenharia Avante

Nesta segunda proposta o objetivo do uso da Estratégia UL/ME desenvolver modelos e, posteriormente, o código de um SE que atenda aos padrões da norma UL 1998. Na Tabela 3.6 apresenta as atividades realizadas em cada etapa do processo de utilização da Estratégia UL/ME para desenvolvimento de um SE.

Tabela 3. 5 - Etapas e atividades realizadas no desenvolvimento de um SE

Etapas	Atividades
Etapa A': Modelar o SE	(I) Desenvolver um modelo UML Statechart
Etapa B': Traduzir o modelo correto para código	(II) De forma objetiva, utilizando softwares que traduzam modelos UML Statechart em código, ou, de forma subjetiva, traduzir o modelo corrigido em código
Etapa C': Avaliar	(III)

código final.	Criar testes de acordo com as especificações do SE
	(IV) Executar os testes criados em (III) e verificar se o código final comtempla todas as funcionalidades propostas em sua especificação.
	(V) Criar testes de acordo com a norma UL 1998 que verifiquem se o código novo atende aos requisitos da norma
	(VI) Executar os testes criados em (V) e verificar se o código final comtempla todas as funcionalidades propostas em sua especificação.

A seguir serão apresentadas as etapas e atividades da Tabela 3.5.

Etapa A': atividade (I) – Modelar o SE:

Modelar o SE em UML Statechart de acordo com sua especificação e usar a Tabela 3.3 para auxiliar a modelagem dos componentes do SE que estão identificados na quarta coluna desta tabela.

Etapa B': atividade (II) – Traduzir o modelo UML Statechart em um novo código:

Traduzir o modelo, na linguagem do código que se deseja embarcar, de forma objetiva ou subjetiva.

Etapa C': atividade (III) – Criar testes de acordo com a especificação do SE

Criar uma tabela de testes, que podem ser do tipo caixa preta, que verifiquem se as principais funções do SE, segundo sua especificação, estão presentes no mesmo.

Etapa C': atividade (IV) – Executar testes de especificação

Executar os testes da etapa anterior. Caso o SE falhe em algum teste, o que verificar se o modelo gerado na atividade I contempla todas as funcionalidades do software. Em caso afirmativo conferir a atividade II.

Etapa C': atividade (V) – Criar testes de acordo com a norma UL 1998

Utilizar a terceira coluna da Tabela 3.3, *Exemplos de testes aceitáveis,* e o Apêndice A para criar uma tabela que contenha cada componente do SE, o teste que deve ser realizado a fim de verificar a presença dos requisitos da norma UL 1998 no componente e o critério usado para verificar se o SE passou ou falhou no teste.

Etapa C': atividade (X) – Executar testes de acordo com a norma UL 1998

Executar os testes da etapa anterior. Caso o SE falhe em algum teste, verificar se o modelo está realmente de acordo com a Tabela 3.3. Em caso afirmativo o, procurar por erros na atividade II.

3.5 Considerações Finais

Como definido anteriormente, a grande motivação do presente trabalho é criar meios para facilitar a correção de defeitos, segundo a norma de certificação UL 1998, em SEs já implementados e, facilitar também, o desenvolvimento de SEs cujos requisitos da norma, abordados no presente capítulo, sejam contemplados já em nível de modelo e não num momento mais avançado do desenvolvimento.

Assim, neste capítulo apresentou-se o estudo realizado em conjunto com Antônio (2012) que resultou na Tabela 3.1, a qual é serve à Estratégia UL/ME como ponto de partida para identificação de componentes do SE pelo uso de suas palavras chaves. Posteriormente, a partir dessa primeira investigação e do conhecimento obtido e relatado no Capítulo 2, foi apresentada a Tabela 3.2 a qual representa modelos de componentes genéricos que não contemplam os requisitos da norma, auxiliando a detecção de defeitos em nível de modelo. Identificados os modelos de um SE que não contemplam a norma UL 1998, apresentou-se como corrigir esses modelos, na Tabela 3.3, de forma que os mesmos atendam aos requisitos mapeados no presente trabalho.

Finalmente, apresentam-se duas propostas de uso da estratégia criada, sendo a primeira para correção e, a segunda, para desenvolvimento.

No Capítulo 4 serão apresentadas duas avaliações da Estratégia UL/ME que utilizaram as propostas descritas na Seção 3.4 num SE já implementado e num SE que será desenvolvido. Essa seria uma segunda etapa desta pesquisa, cujo objetivo é verificar a aplicabilidade do uso da Estratégia UL/ME.

Capítulo 4

AVALIAÇÃO DA ESTRATÉGIA UL/ME

Neste capítulo serão apresentadas duas avaliações da aplicação da Estratégia UL/ME apresentada no Capítulo 3, em SEs reais. Na primeira identificaram-se requisitos da norma UL 1998 em um modelo construído pela engenharia reversa de uma aplicação que já estava implementada. Na segunda, identificaram-se esses requisitos em uma aplicação para a qual foi construído primeiro o modelo e depois o código.

4.1 Considerações Iniciais

Neste capítulo apresentam-se duas avaliações utilizando Controladores de Motores de Velocidade Variável (CMVV) desenvolvidos na Tecumseh do Brasil. A primeira avaliação (CMVV1) iniciou-se com um código embarcado já desenvolvido e que deveria atender aos requisitos da norma de certificação UL 1998. No entanto, a validação do software não foi concluída com êxito.

Para corrigir esse código foi feita engenharia reversa e gerado o seu modelo UML Statechart. Em seguida, aplicando-se a Estratégia UL/ME, foram usadas palavras chaves para auxiliar a identificação dos componentes do SE no modelo, que foi então corrigido e refeito, quando necessário, com o objetivo de atender a UL 1998. Em seguida, desenvolveu-se um novo código baseado no modelo corrigido e depois, testou-se o código para validar a presença dos requisitos que compõem a norma UL 1998. O resultado, como será mostrado na Seção 4.3, foi um código cujo tratamento dos seus componentes está de acordo com os requisitos da norma UL 1998.

A segunda avaliação (CMVV2) da Estratégia UL/ME deu-se num projeto ainda em fase inicial e sem código. O primeiro passo foi a modelagem do SE utilizando como suporte a Estratégia UL/ME. Depois de desenvolvidos, os modelos foram traduzidos em código. O código foi validado da mesma forma que na avaliação, obtendo também sucesso no que diz respeito ao tratamento dos seus componentes estarem de acordo com os requisitos da norma UL 1998.

Para melhor entendimento das avaliações apresentadas neste capítulo, na Seção 4.2 faz-se uma breve introdução aos CMVV, descrevendo-se as principais características e funcionalidade presentes nesse domínio de aplicação de SE. Na Seção 4.3 apresenta-se a avaliação do SE que já estava implementado e na Seção 4.4 apresenta-se a avaliação relativa à engenharia avante. Por fim, na Seção 4.5 apresentam-se as considerações finais deste capítulo.

4.2 Controladores de Motor de Velocidade Variável (CMVV)

O controlador de motor de velocidade variável de interesse do presente trabalho é um SE que está sendo cada vez mais utilizado na indústria de refrigeração atual, devido às vantagens energéticas inerentes à sua tecnologia de variação das velocidades do motor (BELMAN-FLORES, 2015). Essa tecnologia permite que ele funcione em velocidades baixas, retardando seu desligamento e diminuindo o número de vezes que ele é desligado. O desligamento de motores é um ponto crítico de sua eficiência energética, pois quando o motor tem de ser religado há um pico de energia, resultando em alto gasto energético, em se tratando da rede elétrica doméstica.

Assim, o motor de velocidade variável difere dos motores convencionais utilizados em refrigeradores e ares-condicionados por ter uma velocidade de giro, geralmente designada em rotações por minuto (RPM), variável e regulada eletronicamente e por ser mais eficiente em termos de consumo de energia elétrica para seu funcionamento. Já os motores convencionais têm sua velocidade fixada num valor proporcional à frequência da própria rede elétrica doméstica, que no caso do Brasil é de 60Hz, não necessitando de eletrônica adicional para seu funcionamento, que é menos eficiente.

De forma genérica, um controlador de motor de velocidade variável transforma a entrada da rede elétrica doméstica de corrente alternada (AC) em corrente direta (DC) usando uma ponte retificadora. Posteriormente, a componente DC é chaveada em forma de pulsos PWM (Pulse-Width Module) impondo diferentes velocidade ao rotor do motor de forma proporcional a frequência. A Figura 4.1 mostra o diagrama de blocos simplificado de um controlador de motor de velocidade variável.

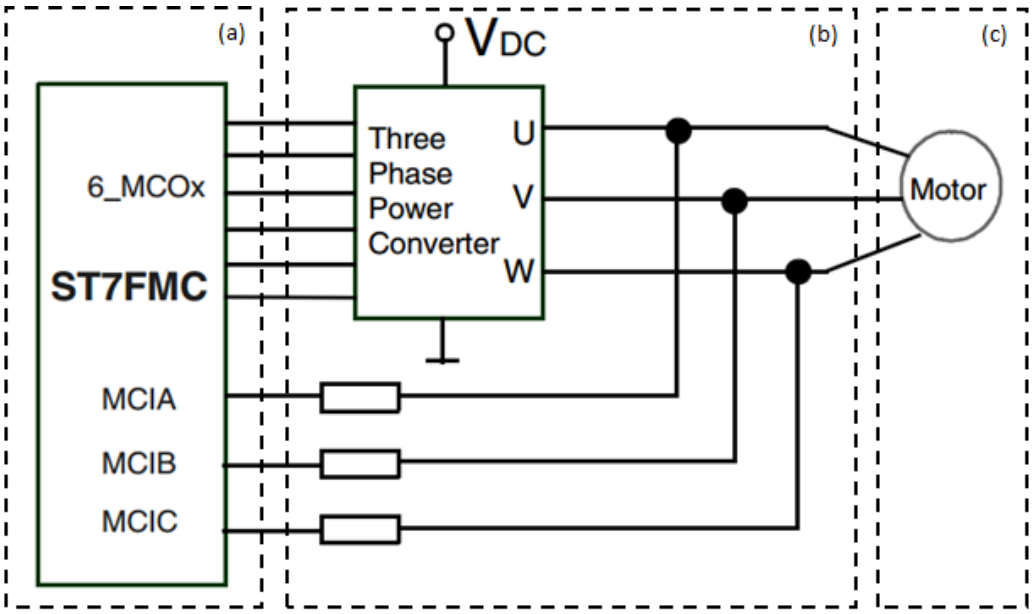


Figura 4. 1 - Exemplo de diagrama de blocos de um controlador de motor de velocidade variável.

A Figura 4.1 foi retirada de uma nota de aplicação da empresa desenvolvedora de eletrônica *ST Microeletrônics* e a série ST7FMC é uma classe de microcontroladores da marca dedicada ao acionamento de motores de velocidade variável. A Figura 4.1 é, mais especificamente, o exemplo de um dos motores de velocidade variável mais comum na indústria da refrigeração, o motor BLDC (Brushless Direct Current) sem sensor, o que significa que o próprio controlador eletrônico é responsável por inferir a velocidade do compressor através de medidas elétricas que serão processadas pelo SE. Na Figura 4.1 o SE é representado por (a) e (b), onde, mais especificamente, (a) é o microcontrolador que possui os componentes do SE que deverão ter seu funcionamento regulamentado pela norma UL 1998 e (b) é a ponte trifásica, ou de forma mais genérica, o hardware que será controlado por (a) a fim de acionar corretamente o motor.

Vale ressaltar que o SE do CMVV é de utilização doméstica, portanto não deve oferecer risco de choque elétrico ao usuário ou risco de fogo, necessitando então de normas de certificação, como a UL 1998, que garantam que seus riscos estão controlados para que possa ser comercializado.

Nas Seções 4.3 e 4.4 a seguir, serão apresentadas as avaliações de CMVV, enunciados brevemente na Seção 4.1. Por esses sistemas serem produtos reais desenvolvidos e produzidos pela Tecumseh do Brasil, a maior parte de seu código e parte de sua modelagem será preservada por questão de sigilo, bem como os nomes dos produtos que serão aqui chamados de CMVV1 e CMVV2.

4.3 Avaliação CMVV1

O projeto CMVV1 é um projeto de motor de velocidade variável cujo código foi desenvolvido sem nenhuma modelagem prévia, seguindo apenas as especificações do sistema, cujo principal objetivo é o controle de um motor de velocidade variável. Ao longo do seu desenvolvimento, no entanto, foi requisitada a utilização da norma UL1998. Assim deu-se início à pesquisa sobre normas de certificação e aplicação dessas normas em SEs para garantia de confiabilidade e qualidade. Contudo, não dispondo de fundamentação teórica e conhecimentos suficientes de engenharia de software, o projeto CMVV1 começou da implementação e, com ele já desenvolvido, foram inseridos os requisitos presentes na norma UL1998, gerando inconsistências e não atendendo à norma como deveria.

A presente avaliação começa então a partir de um código que não atendeu às especificações da norma de certificação UL 1998, o qual sofreu uma engenharia reversa com intuito de modelar o código incorreto para identificar, no nível de modelo, os defeitos relativos à norma. Posteriormente, um novo código foi gerado a partir do modelo corrigido, sendo que, desta vez, o novo código, atendeu os requisitos de validação da norma UL 1998.

Para descrever a avaliação feita no SE CMVV1, utilizou-se o modelo proposto por Wohlin et al. (2012), apresentado nas avaliações subsequentes e podendo ser resumido na Tabela:

Tabela 4. 1 – Definição da avaliação CMVV1

Analisar	Estratégia UL/ME
Com o propósito de	Avaliar
Em relação a	Visibilidade de identificação de defeitos no nível de modelo, segundo os requisitos UL 1998
Do ponto de vista de	Desenvolvedor de SE
No contexto de	Profissional de desenvolvimento de SE

4.3.1 Seleção do Contexto e dos Participantes

A avaliação feita é parte do trabalho que esta autora, realiza na empresa Tecumseh do Brasil para desenvolvimento e validação de CMVV real. Ressalta-se que a avaliadora é graduada em Engenharia Elétrica com ênfase em Eletrônica e desenvolvedora de software a 8 anos na Tecumseh do Brasil.

4.3.2 Projeto Experimental

A avaliação foi realizada por uma única participante, autora deste trabalho, tendo ela então, realizado todas as atividades propostas para a avaliação da Estratégia UL/ME. As atividades realizadas estão descritas na Tabela 4.1.

4.3.3 Instrumentação

Os instrumentos utilizados na avaliação foram a Estratégia UL/ME, aqui representada pelas Tabelas 3.1, 3.2 e 3.3 e o código CMVV1. Para garantir que o código final não perdeu suas funções em relação ao controle do motor e verificar quantos defeitos a Estratégia UL/ME auxiliou a detectar, foram utilizadas as Tabelas 4.2 e 4.3, respectivamente.

4.3.4 Preparação e execução da avaliação

A avaliação foi realizada foi realizada por um período longo que se estendeu da constatação do código defeituoso, segundo a UL 1998, até a entrega do projeto CMVV1, que se deu quando esse SE foi corrigido com o suporte da Estratégia UL/ME. Vale notar que devido a esta ser uma avaliação em ambiente de trabalho, ela teve que ser conduzida em paralelo com outras atividades de responsabilidade da participante da avaliação. Assim, os tempos descritos na primeira coluna da Tabela 4.1 representam as horas de dedicação exclusiva as etapas de avaliação descrita na coluna 2 da mesma tabela. Essas etapas, e suas respectivas atividades, foram detalhadas na Seção 3.4.1 deste trabalho.

Tabela 4. 2 – Atividades realizadas na Avaliação CMVV1

Horas de atividade	Descrição da atividade
40	Etapa A: Fazer engenharia reversa no código para construção do modelo UML Statechart representativo deste código
10	Etapa B: Identificar os componentes do SE no modelo original
40	Etapa C: Corrigir o modelo do código original
40	Etapa D: Traduzir o modelo correto para código
80	Etapa E: Avaliar se o código final manteve suas funcionalidades de CMVV e agora comtempla os requisitos UL 1998.

Para exemplificar a presente avaliação, temos uma parte do código original, convertida em modelo, na Figura 4.2.

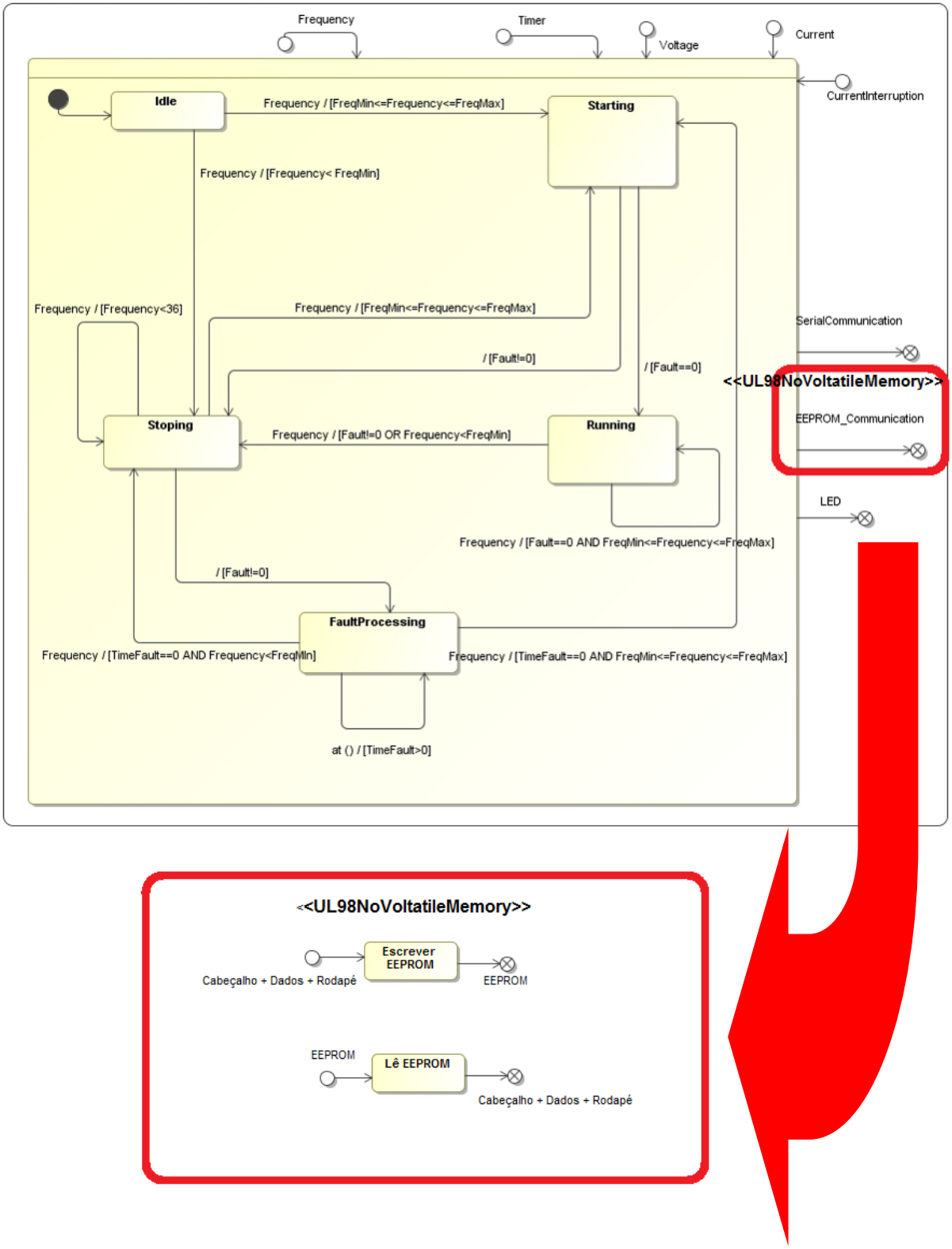


Figura 4. 2 - Modelo UML Statechart com detalhamento da função de comunicação com EEPROM do CMVV1

Na Figura 4.2 tem-se o modelo UML Statechart de um controlador de motor de velocidade variável que resultou da engenharia reversa na Etapa A desta avaliação. É possível observar os estados de acionamento do motor: *Idle, Starting,*

Running, Stoping e *FaultProcessing*. Essas representam, respectivamente, o controlador comando os estados de: espera, partida, funcionamento, parada e processamento de falhas advindas do acionamento do motor pelo SE. Observa-se também iterações do SE como: *SerialCommunication, EEPROM_Communication* e *LED*. No detalhe do desenho, temos a EEPROM, sabidamente um componente do SE, já citado nos Capítulos 2 e 3. Com uma observação mais crítica do modelo, nota-se que, apesar da intenção do desenvolvedor de criar um protocolo para garantir segurança na comunicação com cabeçalho e rodapé pré-fixados, a mesma não estará livre de erros pois não evita que os dados que se seguem depois do cabeçalho e antes do rodapé estejam corretos. A norma UL 1998 pede o uso de um protocolo com *checksum*, pois este teste garante com maior confiabilidade que todos os dados enviados e recebidos estão corretos. Assim o modelo foi corrigido, conforma as Atividades propostas na Etapa C, retirando-se o protocolo anterior e criando-se um modelo com *checksum.* O resultado pode ser observado na Figura 4.3.

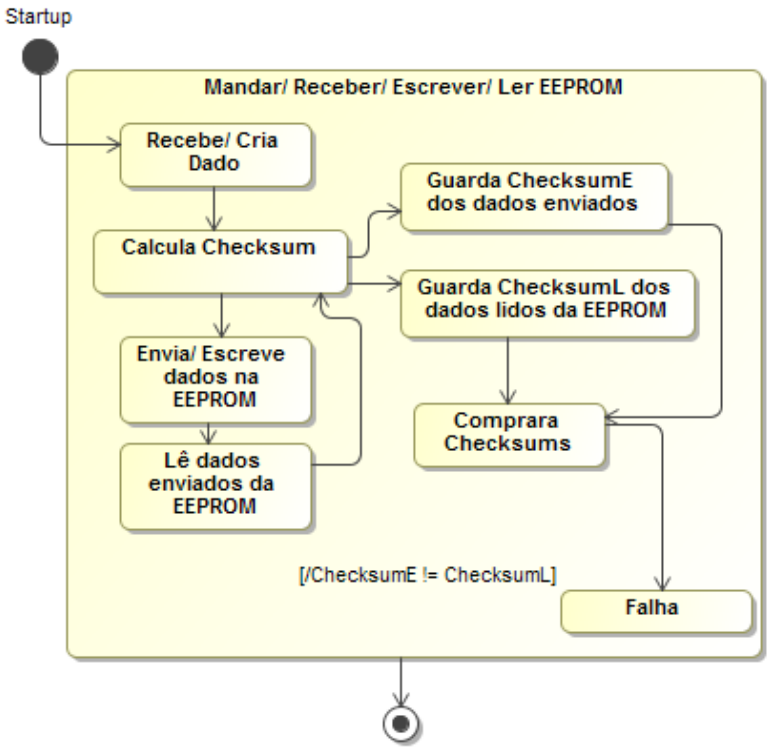


Figura 4. 3 - **Modelo UML Statechart do CMVV1 corrigido**

É possível observar que a Figura 4.3 representa um modelo bem mais complexo que o evidenciado no detalhe da Figura 4.2. O uso de *checksum* em

protocolos de comunicação é realmente mais trabalhoso, pode-se supor, por tudo que já foi dito em relação à modelagem prévia de SEs, que criar um código de protocolo com *checksum* diretamente no código, não seja a alternativa mais assertiva para desenvolver esse tipo de funcionalidade.

Seguindo as etapas da avaliação, o modelo UML Statehart corrigido deveria ser transformado em código embarcado. A seguir, na Figura 4.4, tem-se o código gerado pelo modelo exibido na Figura 4.3.

```
#include <stdio.h>            // Needed for printf()
#include <stdlib.h>           // Needed for rand()

/********************* Type defines*************************/
typedef unsigned char     byte;   // Byte is a char
typedef unsigned short int word16;  // 16-bit word is a short int
typedef unsigned int      word32;  // 32-bit word is an int

/********************* Defines*************************/
#define BUFFER_LEN      256     // Length of buffer

/********************* Prototypes*************************/
word16 checksum(byte *addr, word32 count);

void main(void)
{
 byte      buff[BUFFER_LEN]; // Buffer of packet bytes
 word16    check;          // 16-bit checksum value
 word32    i;            // Loop counter

 // Load buffer with BUFFER_LEN random bytes
 for (i=0; i<BUFFER_LEN; i++)
  buff[i] = (byte) rand();

 // Compute the 16-bit checksum
 check = checksum(buff, BUFFER_LEN);

 // Output the checksum
 printf("checksum = %04X \n", check);
}
```

```
word16 checksum(byte *addr, word32 count)
{
 register word32 sum = 0;

 // Main summing loop
 while(count > 1)
 {
  sum = sum + *((word16 *) addr)++;
  count = count - 2;
 }

 // Add left-over byte, if any
 if (count > 0)
  sum = sum + *((byte *) addr);

 // Fold 32-bit sum to 16 bits
 while (sum>>16)
  sum = (sum & 0xFFFF) + (sum >> 16);

 return(~sum);
}
```

Figura 4. 4 - Código da função de comunicação com EEPROM do CMVV1 corrigido

A Figura 4.4 mostra o código desenvolvido a partir do modelo descrito na Figura 4.3 na Etapa D da proposta de uso da Estratégia UL/ME usada nesta avaliação. A

próxima etapa do uso da estratégia, é testar todo o código desenvolvido a partir dos modelos corrigidos. Para testa-lo foram criadas as Tabelas 4.3 e 4.4, de acordo com o uso da Estratégia UL/ME proposta na seção 3.4.1. Para uma análise mais recorreu-se aos resultados de validação do código incorreto para comparar os resultados.

Tabela 4. 3 - Tabela de validação das funcionalidades de CMVV1

Parâmetro a ser testado	Leitura	Critério/Limites	Código original		Código novo	
			Passou	Falhou	Passou	Falhou
Maior tensão antes de acusar falha		Leitura <= Tensão AC máxima permitida	X		X	
Menor tensão antes de acusar falha		Leitura >= Tensão AC mínima permitida	X		X	
Mínima velocidade antes de parar		Leitura > Velocidade RPM mínima permitida	X		X	
Máxima velocidade		= Velocidade RPM máxima permitida	X		X	

Os testes funcionais tiveram sucesso para o código original e para o código novo. Tabela 4.4, feita de acordo com a Atividade IX da Etapa E, apresentam-se os resultados para os testes segundo os requisitos UL 1998.

Tabela 4. 4 – Validação de CMVV1 segundo os requisitos da norma UL 1998

Componente		Teste	Código original		Código novo	
			Passou	Falhou	Passou	Falhou
1. CPU	Registradores	T1-Teste de memória estático de Registradores de CPU		X	X	
	Contador de Programa	T2- Monitoramento de tempo	X		X	
2. Interrupção		T2- Monitoramento de tempo	X		X	
3. Relógio		T2- Monitoramento de tempo	X		X	

4. Memória	Não volátil	T3- Teste periódico com checksum		**X**	**X**	
	Volátil	T4-Teste de memória estático		**X**	**X**	
	Endereçamento	T3- Teste periódico com checksum		**X**	**X**	
5. Caminhos de dados internos		T3- Teste periódico com checksum		**X**	**X**	
6. Comunicação externa	Dados	T5- Protocolo		**X**	**X**	
	Endereço	T5- Protocolo		**X**	**X**	
	Tempo	T2- Monitoramento de tempo	**x**		**X**	
7. Entradas/Saídas	E/S digitais	T6- Verificação de plausibilidade	**X**		**X**	
	E/S analógica: Conversores A/D e D/A	T6- Verificação de plausibilidade	**X**		**X**	
	E/S Analógica Multiplexadas	T6- Verificação de plausibilidade	**X**		**X**	

Na Tabela 4.4 pode-se notar que o novo código passou em todos os testes. Vale notar a correlação entre as Figuras 4.2, que mostra o modelo de memória não volátil com defeito e, 4.3, que mostra este mesmo modelo corrigido, com o Teste periódico com checksum de memória não volátil na Tabela 4.4. A tabela mostra que

o código original com o modelo defeituoso não passou no teste, enquanto o código original, gerado a partir do modelo corrigido, passou.

4.3.5 Análise e Discussão dos Resultados

De acordo com a Tabela 4.4, o uso da Estratégia UL/ME corrigiu dos defeitos do SE, segundo os requisitos da norma UL 1998. Ressalta-se que os requisitos que foram implementados com sucesso no código original, dependem de implementações de hardware, o que pode ter facilitado sua previsão em software. Vale ressaltar também que os demais defeitos encontrados no código defeituosos têm implementação muito específicas segundo a própria, podendo ser citados os requisitos de protocolo, os quais possuem particularidades, como inclusão de endereços de memória no checksum, como pode ser observado na Figura 4.3.

4.3.6 Ameaças à Validade

As ameaças à validade podem ser classificadas em interna, externa, de construção e de conclusão (WOHLIN , RUNESON , *et al.*, 2012), de acordo com a abaixo:

- Validade interna: validade dentro do ambiente e confiabilidade dos resultados;
- Validade externa: quão genéricos são os resultados;
- Validade de construção: construção do experimento reflete a realidade;
- Validade de conclusão: relação entre a conclusão do experimento e a forma como este foi desenvolvido.

No caso desta avaliação incorreram ameaças das quatro classificações, e todos os seus vieses giram em torno dos seguintes fatores:

- Participantes: só houve uma participante, tornando o espaço amostral muito restrito. A participante é a própria desenvolvedora da estratégia e do SE, podendo o domínio destes ter facilitado a avaliação.
- Treinamento: como dito acima, a própria desenvolvedora da Estratégia UL/ME é a avaliadora, portanto é fácil para a mesma utilizar a estratégia.
- SE: o sistema pode ser muito simples, ainda mais por ter sido desenvolvido pela própria avaliadora, o que pode ter facilitado a

compreensão do SE e a aplicação da estratégia. Além disso o SE é de um domínio de aplicação específico.

Quanto à validade de construção destaca-se:

As ameaças a validade da presente avaliação não puderam ser minimizadas, pois num contexto real não foi possível que outro participante realizasse a avaliação nem que a participante avaliasse outro domínio.

4.3.7 Conclusões da avaliação CMVV1

A presente seção teve como objetivo apresentar os passos da avaliação da Estratégia UL/ME, mostrando sua viabilidade em detectar defeitos e mostrando como eles podem ser evitados no código. De um SE. Comparando-se com o número de defeitos na aplicação sem e com o uso da estratégia, pode-se observar que uso da Estratégia UL/ME auxiliou na identificação e correção de todos os defeitos do código original.

4.4 Avaliação CMVV2

Esta avaliação foi feita com o SE CMVV2 e seu principal foi avaliar o uso da Estratégia UL/ME no modelo UML Statechart CMVV2, o qual foi, posteriormente, transformado em código que, por sua vez, foi submetido a validação segundo a norma UL 1998.

A avaliação realizada com o SE CMVV2, foi definida de acordo com a Tabela 4.5.

Tabela 4.5 - Definição da avaliação CMVV2

Analisar	Estratégia UL/ME
Com o propósito de	Avaliar
Em relação à	Adequação do código em relação aos requisitos da norma UL 1998
Do ponto de vista do	Desenvolvedor de SE
No contexto de	Profissional de desenvolvimento de SE

4.4.1 Seleção do Contexto e dos Participantes

A avaliação feita é parte do trabalho que esta autora realiza na empresa Tecumseh do Brasil para desenvolvimento e validação de CMVV real. Ressalta-se que a avaliadora é graduada em Engenharia Elétrica com ênfase em Eletrônica e desenvolvedora de software a 8 anos na Tecumseh do Brasil.

4.4.2 Projeto Experimental

A avaliação foi realizada por uma única participante, autora do presente trabalho, tendo ela, realizado todas as atividades desta avaliação segundo as etapas e atividades da Seção 3.4.2, descritas na Tabela 4.5.

4.4.3 Instrumentação

Os instrumentos utilizados na avaliação foram a Estratégia UL/ME, descrita na Seção 3.4.2. Para garantir que o código final não perdeu suas funções em relação ao controle do motor e verificar quantos defeitos a Estratégia UL/ME auxiliou a detectar, foram utilizadas as Tabelas 4.2 e 4.3, respectivamente.

4.4.4 Preparação e execução da avaliação

A avaliação foi realizada durante o desenvolvimento do SE CMVV2. Como esta também foi uma avaliação em ambiente de trabalho, ela teve que ser conduzida em paralelo com outras atividades de responsabilidade desta autora. Assim os tempos descritos na primeira coluna da Tabela 4.6 representam as horas de dedicação exclusivas à atividade de avaliação descrita na coluna 2 da mesma tabela.

Tabela 4. 6 – Atividades realizadas na Avaliação CMVV2

Horas de atividade	Descrição da atividade
40	Etapa A': Modelagem do SE CMVV2 utilizando Estratégia UL/ME.
40	Etapa B': Engenharia avante para traduzir o modelo correto em código
80	Etapa C': Avaliar se o código final manteve suas funcionalidades de CMVV e o número de defeitos segundo os requisitos da norma UL 1998.

Por motivo de sigilo empresarial, os modelos não podem ser apresentados em sua totalidade, porém, para exemplificar a avaliação, na Figura 4.3 apresenta-se o modelo da função de comunicação do componente do SE de memória volátil:

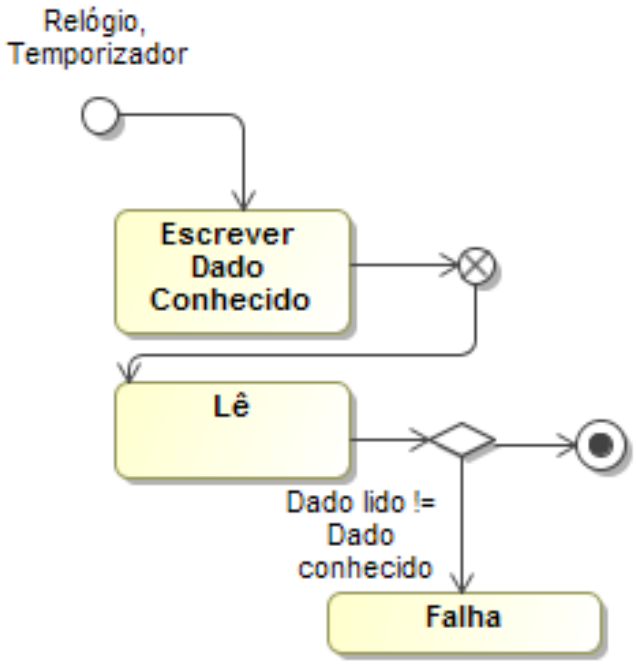


Figura 4.5 - Modelo UML Statechart CMVV2

Feitos os modelos UML Statechars, estes foram convertidos em código. Na Figura 4.4 apresenta-se o exemplo do código gerado pelo modelo da Figura 4.3:

```
#define  OPTION_BYTE (*(volatile unsigned char *)
0x0080)  /* Option bytes */
void main(void) {
  OPTION_BYTE = 0xFF;
```

```
void SendCommand (volatile MyData * bus, int command, int  data)

{
  // wait while the bus is busy:
  while (bus->isbusy)
  {
    // do nothing here.
  }
  // set data first:
  bus->data    = data;
  // writing the command starts the action:
  gadget->command = command;
}
```

Figura 4.6 - Código de CMVV2

Para avaliar os defeitos de código presentes no SE criou-se telas semelhantes à da avaliação anterior e de acordo com a proposta de uso da Estratégia UL/ME para desenvolvimento de SEs do Capítulo 3.

Tabela 4.7 – Tabela de validação das funcionalidades de CVV2

Parâmetro a ser testado	Leitura	Critério/Limites	Código CVV2	
			Passou	Falhou
Maior tensão antes de acusar falha		Leitura <= Tensão AC máxima permitida	X	
Menor tensão antes de acusar falha		Leitura >= Tensão AC mínima permitida	X	
Mínima velocidade antes de parar		Leitura > Velocidade RPM mínima permitida	X	
Máxima velocidade		= Velocidade RPM máxima permitida	X	

A Tabela 4.7 apresenta os resultados dos testes de funcionalidade do CMVV2, segundo sua especificação, lembrando que esta tabela foi criada de acordo com a proposta de uso da Estratégia UL/ME descrita na Seção 3.4.2, na Etapa C', Atividade XIII De acordo com a tabela, o código do SE CMVV2 foi desenvolvido atendo as funcionalidades da especificação deste SE.

Seguindo as etapas desta avaliação, foi criada a Tabela 4.8 para validar a presença dos requisitos UL 1998 no código do CMVV2.

Tabela 4.8 - Validação de CMVV2 segundo os requisitos da norma UL 1998

Componente		Teste	Código CMVV2	
			Passou	Falhou
1. CPU	Registradores	T1-Teste de memória estático de Registradores de CPU	**X**	
	Contador de Programa	T2- Monitoramento de tempo	**X**	
2. Interrupção		T2- Monitoramento de tempo	**X**	
3. Relógio		T2- Monitoramento de tempo	**X**	
4. Memória	Não volátil	T3- Teste periódico com checksum	**X**	
	Volátil	T4-Teste de memória estático	**X**	
	Endereçamento	T3- Teste periódico com checksum	**X**	
5. Caminhos de dados internos		T3- Teste periódico com checksum	**X**	
6. Comunicação externa	Dados	T5- Protocolo	**X**	
	Endereço	T5- Protocolo	**X**	
	Tempo	T2- Monitoramento de tempo	**X**	

7. Entradas/Saídas	E/S digitais	T6- Verificação de plausibilidade	**X**	
	E/S analógica: Conversores A/D e D/A	T6- Verificação de plausibilidade	**X**	
	E/S Analógica Multiplexadas	T6- Verificação de plausibilidade	**X**	

Na Tabela 4.8 pode-se notar que o código passou em todos os testes. Vale notar que a Figura 4.5 apresenta um modelo de memória volátil de SE de acordo com a Estratégia UL/ME (Tabela 3.3) e que o código que representa este modelo, apresentado na Figura 4.6, passou teste de memória estático na Tabela 4.8.

4.4.5 Análise e Discussão dos Resultados

De acordo com a Tabela 4.8, o uso da Estratégia UL/ME na modelagem auxiliou no desenvolvimento de modelos para SEs em desenvolvimento livres dos defeitos em código segunda a norma UL 1998.

4.4.6 Ameaças à Validade

As ameaças à validade serão classificadas da mesma forma da avaliação anterior. Nesta segunda avaliação, também incorreram ameaças internas, externas, de construção e de conclusão. Todos os seus vieses das ameaças giram em torno dos seguintes fatores:

- Participantes: só houve uma participante, tornando o espaço amostral muito restrito. A participante é a própria desenvolvedora da estratégia e do SE, podendo o domínio destes ter facilitado a avaliação.
- Treinamento: como dito acima, a própria desenvolvedora da Estratégia UL/ME é a avaliadora, portanto é fácil para a mesma utilizar a estratégia.
- SE: o sistema pode ser muito simples, ainda mais por ter sido desenvolvido pela própria avaliadora, o que pode ter facilitado a

compreensão do SE e a aplicação da estratégia. Além disso o SE é de um domínio de aplicação específico.

Quanto à validade de construção destaca-se:

As ameaças a validade da presente avaliação não puderam ser minimizadas devido à dificuldade de encontrar outro desenvolvedor da área disposto a repetir a avaliação.

4.4.7 Conclusões da avaliação CMVV2

A presente seção teve como objetivo mostrar os passos da avaliação da Estratégia UL/ME para desenvolvimento de SE em conformidade com os requisitos da norma UL 1998. Como não foi detectado defeitos em código segundo os requisitos da norma, pode-se concluir que o uso da estratégia auxilia no desenvolvimento de SEs que atendam aos requisitos da norma.

4.5 Considerações finais

Neste capítulo foram apresentadas duas avaliações do uso da Estratégia UL/ME de acordo com o que foi proposto na Seção 3.4 do Capítulo 3. Na primeira avaliação o uso da estratégia auxiliou a correção um código de SE já implementado e que possuía 50% de seus componentes com defeitos segundo os requisitos da norma UL 1998. Com o uso da estratégia nessa primeira avaliação, foi possível identificar todos os defeitos do código original no modelo construído por meio de engenharia reversa e, após o modelo ter sido corrigido, o código gerado com base neste modelo passou a atender os requisitos da norma UL 1998.

Na segunda avaliação a Estratégia UL/ME foi utilizada para auxiliar a modelagem de um SE. O código gerado com base neste modelo desenvolvido foi validado e contemplou todos os requisitos da norma UL 1998.

É possível ainda observar que a correção de defeitos em códigos implementados usando a Estratégia UL/ME é um procedimento que leva 50 horas a mais para ser concluído em relação ao uso da estratégia já no início do

desenvolvimento. Contudo o uso da estratégia mostrou-se vantajoso em ambos os casos.

Capítulo 5

CONCLUSÃO

Dada a intensa presença de SEs no cotidiano atual, a preocupação com a segurança do usuário tem se tornado cada vez mais importante para diferentes tipos de instituições. Em decorrência disso, há várias normas de certificação que aplicadas atualmente no mercado, cujo objetivo é justamente avaliar se os aspectos de segurança e qualidade dos SEs estão sendo cumpridos.

Neste trabalho, mostrou-se a similaridade dessas normas e estabeleceu-se como referência a norma UL 1998 que, assim como as outras, é aplicada no nível de código, o que faz com que os defeitos sejam identificados depois que o SE está implementado.

Sabendo-se que o desenvolvimento de software seguindo os preceitos da engenharia de software leva a uma aplicação de maior qualidade, o objetivo deste trabalho foi possibilitar a identificação antecipada dos requisitos da norma UL 1998 no nível de modelo. Assim, neste trabalho foram apresentados o estudo e o desenvolvimento de modelos de componentes de SEs, utilizando UML Statechart, com o objetivo de contemplar os requisitos da Norma de Certificação UL 1998 e assim, possibilitar o uso dessa norma no nível de modelagem, anteriormente à codificação do SE.

Esses modelos podem ser utilizados tanto no caso do SE já estar implementado e haver necessidade de adequá-lo segundo a UL 1998, ou no caso de um SE que está sendo desenvolvido de acordo com um processo avante, em que a aplicação é primeiramente modelada para depois ser implementada. Para cada uma dessas possibilidades de uso dos modelos foram criados passos que compõem o que foi denominado de Estratégia UL/ME, uma vez que o propósito é aplicar os

requisitos da norma UL 1998 em modelos de estado, mais especificamente, utilizando o modelo UML Statechart.

A estratégia UL/ME foi avaliada pela autora deste trabalho no contexto de dois SEs reais desenvolvidos na Tecumseh São Carlos, empresa em que a autora é funcionária. Nessa avaliação teve-se a oportunidade de explorar a estratégia nos dois casos citados anteriormente, isto é, em um SE que já estava implementado e não cumpria os requisitos da UL 1998 e em outro SE cujo desenvolvimento seguiu o processo de engenharia avante. Em especial no segundo caso, pode-se observar que, de fato, é possível inserir os requisitos da norma UL 1998 na modelagem dos componentes de SEs e assim, garantir que esses requisitos façam parte do projeto antes do desenvolvimento do código. Com os exemplos do Capítulo 4 é possível dizer também que a codificação seguindo os requisitos da norma UL 1998 é facilitada pelo uso dos modelos de componentes desenvolvidos no presente trabalho.

5.1 Contribuições e Limitações

Como contribuições deste trabalho pode-se citar o estudo que foi realizado sobre as normas de certificação, cujas similaridades estão apresentadas na Seção 2.5 do Capítulo 2 pelas Tabelas 2.1 e 2.2. Apesar desse estudo poder ser muito mais aprofundado, tem-se aqui uma visão geral dessas similaridades.

Outra contribuição são os modelos em UML Statechart que representam os requisitos da UL 1998, bem como as palavras chave que foram criadas para facilitar a identificação de situações incorretas em SEs já desenvolvidos, as quais também foram modeladas a fim de facilitar sua correção. Neste caso, também pode-se considerar uma contribuição o relacionamento estabelecido entre as situações incorretas e os modelos que as corrigem.

As limitações do presente trabalho residem principalmente no grande número de domínios de SEs, resultando, igualmente, em um grande número de possibilidades no desenvolvimento desses.

Há também um grande número de normas para cada domínio de aplicação. Neste trabalho foi evidenciado, no Capítulo 2, que as normas que se prestam à

certificação de SEs, mesmo que de domínios de aplicação diferentes, são bastante similares; contudo, elas têm suas diferenças e esse é outro ponto a ser explorado com mais detalhe em trabalhos futuros.

Ainda em se tratando de normas de certificação, vale ressaltar que as consideradas neste trabalho estão restritas ao contexto de SEs. Eventualmente, normas de certificação para outros domínios poderiam levar à identificação de outros tipos de defeitos nos modelos aqui considerados.

Também se pode citar como limitação deste trabalho o fato da avaliação ter sido realizada pela própria autora. Isso traz vários vieses para o estudo realizado, mas, por se tratar de SEs reais (aos quais se deram preferência), com questões de sigilo envolvidas, não havia possibilidade do estudo ser realizado por pessoas externas.

As Contudo, pode-se dizer que o presente trabalho é um início de investigação que pode ser promissora no uso de requisitos de certificação de SEs em nível de modelo, para auxiliar tanto na correção de SEs ou na modelagem de SEs, antes da codificação.

5.2 Lições aprendidas

Com base no aprendizado adquirido durante este trabalho, algumas lições podem ser destacadas:

- Um dos grandes desafios de se trabalhar com SEs são suas inúmeras possibilidades e aplicações. Portanto, para um desenvolvimento genérico, é preciso criar uma referência aplicável a qualquer SE que, no caso do presente trabalho, tal referência foram os componentes que formam um SE;
- Ratificando os resultados de Freire (2011), mais uma vez a modelagem feita anteriormente à codificação mostrou-se uma boa abordagem no desenvolvimento de SEs;
- Por mais que o presente trabalho tenha tentado facilitar o uso da norma de certificação UL 1998 no desenvolvimento de SEs, a utilização dos modelos

aqui desenvolvidos requer o mínimo de conhecimento na área de embarcados;

- É possível criar modelos genéricos que representem o mesmo componente do SE para diversos domínios de aplicação e a utilização e estudo continuado desses modelos só tende a melhorá-los.

5.3 Trabalhos Futuros

Como trabalhos futuros destacam-se os seguintes pontos:

- Aplicar os modelos desenvolvidos neste trabalho em vários domínios de aplicação de SEs;
- Criar modelos de componentes de SEs que contemplem todos os possíveis testes enunciados na norma de certificação UL 1998;
- Fazer o exercício de desenvolver modelos de componentes baseando-se totalmente numa única norma e, posteriormente, em uma segunda norma, e verificar se a diferença entre esses modelos é significativa;
- Desenvolver ferramentas que possam auxiliar a construção dos modelos UML Statechart de acordo com a norma de certificação UL 1998 e, posteriormente, generalizar essa ferramenta para contemplar outras normas às quais se deseja submeter o SE à aprovação.

REFERÊNCIAS

ANTÔNIO, E. A. RTSS: Uma Familia de Técnicas de Leitura Para Suporte à Inspeção de Modelos SYSML e SIMULINK. **Universidade Federal de São Carlos**, São Carlos - SP, 2014.

BARR, M. **Porgramming Embeded Systems in C and C++**. 1. ed. Sebastopol: O'Reilly, 1999.

BARRY , R. **Using the FreeRTOS real time kernel:** a practical guide. [S.l.]: Real Time Engineers, 2010.

BELMAN-FLORES, J. M. . E. A. Enhancements in domestic refrigeration, approaching a sustainable refrigerator – A review. **Renewable and Sustainable Energy Reviews**, v. 51, 2015. 955 -968.

CAFFERY, F. M.; DORLING, A. Medi SPICE development. **Journal of Software Maintenance and Evolution: Research and Practice**, v. 22, 2010. p. 255-268.

CARRO, L.; WAGNER, F. R. Sistemas computacionais embarcados. **Jornadas de atualização em informática. Campinas: UNICAMP**, 2003.

DAVIS, R. I.; GROLLEAU, E. Special issue on scheduling and timing analysis for advanced real-time systems. **Real-Time Systems**, v. 51, 2015. p. 125-127.

DESAI, M. **Software in Programmable Components. Underwriters Laboratories Inc.** Underwriters Laboratories Inc. Illinois. 1998.

EBERT, C.; JONES, C. Embedded software: Facts, figures, and future Computer. **Computer**, v. 42, n. 4 , p. 42-52, 2009.

FORCE. **UML Revision Task. OMG unified modeling language: Superstructure**. Object Management Group (OMG). [S.l.]. 2010.

FREIRE, G. Uma investigação sobre o uso da UML/Statechart para representar o comportamento de aplicações modeladas em Matlab/Simulink. **Universidade Federal de São Carlos**, São Carlos - SP, 2011.

GALL, H. Functional safety IEC 61508 / IEC 61511 the impact to certification and the user. **Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on**, 2008. p. 1027-1031.

GIZOPOULOS, D.; PASCHALIS, A.; ZORIAN, Y. Embedded processor-based self-test. **Springer Science & Business Media**, 2013.

HAREL, D. A visual formalism for complex systems. **Science of Computer Programming**, 8, n. 3, Jun 1987. 231-274.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. **In: Proceedings of the ACM International Conference on Multimedia**, ACM, 2014. p. 675-678.

KANDT, R. K. Experiences in improving flight software development processes. **Software, IEEE**, v. 26, n. n. 3, 2009. p. 58-64.

LI, Q.; YAO, C. **Real-time concepts for embedded systems**. [S.l.]: CRC Press, 2003.

LIANG , K. et al. Fault localization in embedded control system software. In. **Proceedings of the First International Workshop on Software Engineering for Smart Cyber-Physical Systems**, IEEE Press, 2015. 8-14.

LINDGREN, E.; MÜNCH, J. Software Development as an Experiment System: A Qualitative Survey on the State of the Practice. **Agile Processes, in Software Engineering, and Extreme Programming**, 2015. p. 117-128.

MARWENDEL, P. Embedded System Design:Embedded Systems Foundations of Cyber-Physical Systems. **Springer Science & Business Media**, n. 2, 2011.

MYERS, G. J. et al. The Art of Software Testing. **Business Data Processing: a Wiley Series**, 2004.

NAIR, S. et al. Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. **Information and Software Technology**, v. 60, p. 1-16, 2015.

NAUGHTON, M.; MCGRATH, J.; HEFFERNAN, D. Real-time software modelling using statecharts and timed automata approaches. In. **Irish Signals and Systems Conference, 2006. IET.**, IET, 2006, 2006. p. 129-134.

PFALLER, C. et al. On the integration of design and test: a model-based approach for embedded systems. In. **Proceedings of the 2006 international workshop on Automation of software test**, ACM, 2006. p. 15-21.

PRESSMAN, R. S. **Engenharia de software**. [S.l.]: AMGH Editora, 2009.

SIMON, D. E. **An Embedded Software Primer**. 1st. ed. Boston, MA, USA: Addison-Wesley Professional, 1999.

UMM-E-HABIBA , A. K.; JAVED , M. Y. Gap Analysis in Software Engineering Process Adoption in Implementing High End Embedded System Design. **J. Appl. Environ. Biol. Sci**, v. 4, n. 75, p. 495-503, 2014.

WALIA, S.; CARVER, J. C. A systematic literature review to identify and classify software requirement errors. **Information and Software Technology**, v. 51, n. 7, p. 1087-1109, 2009.

WASSYNG, A. et al. Software certification: Is there a case against safety cases? **Foundations of computer software. Modeling, development, and verification of adaptive systems**, Springer Berlin Heidelberg, 2010. 206-227.

WOHLIN , C. et al. Systematic Literature Reviews. In. **Experimentation in Software Engineering**, Springer Berlin Heidelberg, 2012. p. 45-54.

WOLF, M. **Computer as Components:** principles of embedded computing system desing. [S.l.]: Elsevier, 2012.

YU, Y. et al. Reverse engineering goal models from legacy code. In. **Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on**, IEEE, 2005. 363-372.

Apêndice A

DEFEITOS DE HARDWARE E SOFTWARE E TESTES

Neste apêndice são tratadas com detalhes os defeitos de software e hardware que norma UL 1998 busca evitar através de seus requisitos. São melhor descritos também os testes que a norma propõe para identificar esses defeitos.

Descrição das falhas potenciais:

H1: Circuito aberto ou um nível de sinal não variável.

H2: Defeito no oscilador.

H3: Como o nome sugere erros de um único bit ocorrem quando um único bit é alterado durante a transmissão de dados devido a interferência na comunicação de rede. Erros corrigíveis são geralmente erros de bit único. Erros incorrigíveis são sempre os erros de memória de múltiplos bits.

H4: Falha de curto circuito entre linhas de sinal.

S1: Erro de configuração de prioridades.

S2: Erro na configuração do relógio.

S3: Erro de design de software.

Descrição detalhada dos testes

T1- Teste de memória estático de Registradores de CPU

Os registradores da CPU são testados periodicamente (ou somente no boot) escrevendo sucessivamente as sequências binárias (Comprimento depende da

arquitetura), seguido de 010101 101010 nos registos, e em seguida, lendo os valores a partir desses registros para verificação. Esse teste é utilizado para determinar se qualquer bit da memória está preso em '1' ou '0'.

T2. Monitoramento de tempo

Verifica a confiabilidade da base de tempo (relógio) do sistema, isto é, o relógio do sistema não está nem muito rápido nem muito lento dependendo de uma segunda base de tempo de referência.

Pode ser utilizado:

• Oscilador Secundário (SOSC)

• A própria frequência (50 Hz, 60 Hz) da linha de alimentação de tensão AC.

T3-T5 Teste periódico com checksum

Checksum de uma mensagem é uma soma aritmética modular de palavras de código mensagem de um comprimento de palavra fixa (por exemplo, valores de bytes) que podem eventualmente conter o endereço, no caso de dados que estejam sendo enviados a uma memória não volátil. A soma de verificação é sempre calculada antes de envio e faz parte do pacote de dador a ser enviado e também logo após o recebimento, quando a soma é então confrontada com o dado presente no endereço determinado para o checksum afim de confirmar se o pacote de dados foi enviado e recebido sem corrupção de dados.

T4. Teste de memória estático periódico

O teste de memória estático periódico envia sequências de “0” e “1” conhecida para endereços aleatórios e depois lê esses dados para detectar falhas de um único bit na memória variável. O objetivo é verificar se algum endereço está preso em '1' ou '0'. Este teste deve ser feito sempre na inicialização antes da rotina principal.

T6. Protocolo

Implementações protocolo com uma soma de verificação para evitar escritas e leituras erradas.

T7. Cheque de Plausibilidade

O teste *Plausibility Check* em entradas analógicas, digitais e multiplexadas é feito aplicando-se uma tensão conhecida e verificando se o microprocessador lê corretamente o valor aplicado. Para as saídas analógicas, digitais e multiplexadas é forçado um estimulo que produza uma saída conhecida. Logo este é um teste funcional que envolve hardware e não pode ser modelado.

Anexo A

MODELOS UML STATECHART DE SE DE DIFERENTES DO MÍNIOS DE APLICAÇÃO

Nesse anexo serão expostos diversos modelos UML Statechart de SEs de diferentes domínios para familiarizar o leitor a esse tipo de modelo e também para a partir do estudo dos mesmos elaborar um conjunto de palavras chaves que serão utilizadas para facilitar a detecção dos componentes de qualquer SEs representado em modelo. A detecção dos componentes de SEs em nível de modelo é importante no contexto do presente trabalho, pois irá viabilizar a aplicação dos testes previstos em norma em nível de modelo, objetivo do presente trabalho.

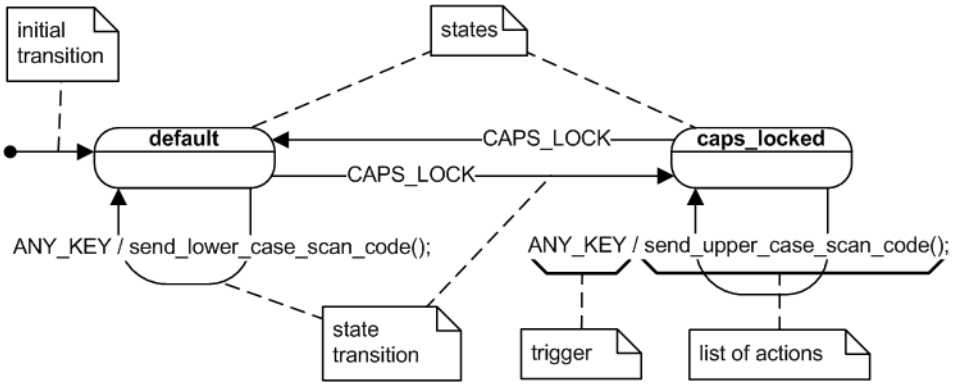


Figura 5.1 - Teclado de Computador (http://www.uml-diagrams.org/activity-diagrams-examples.html).

Figura de domínio de aplicação doméstico. Pode-se considerar: *ANY_KEY* um componente de interrupção.

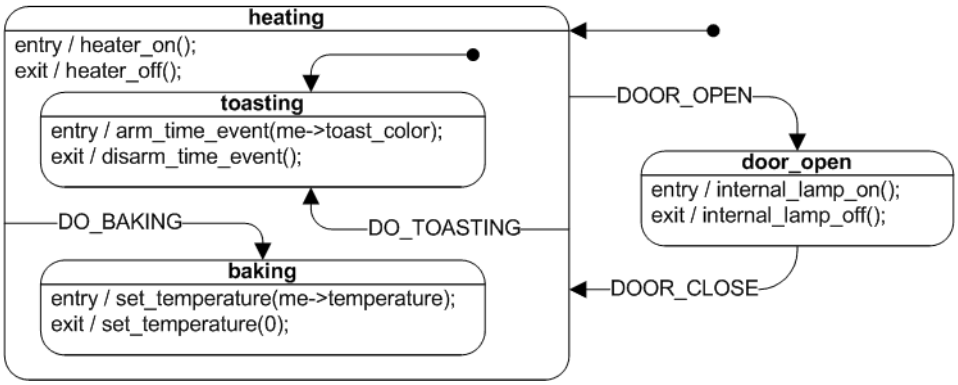


Figura 5.2 - Torradeira (http://www.uml-diagrams.org/activity-diagrams-examples.html).

Figura de domínio de aplicação doméstico. Pode-se considerar: *arm_time_event,* um componente de relógio (timer, clock), *internal_lamp_on, internal_lamp_off* é uma saída digital.

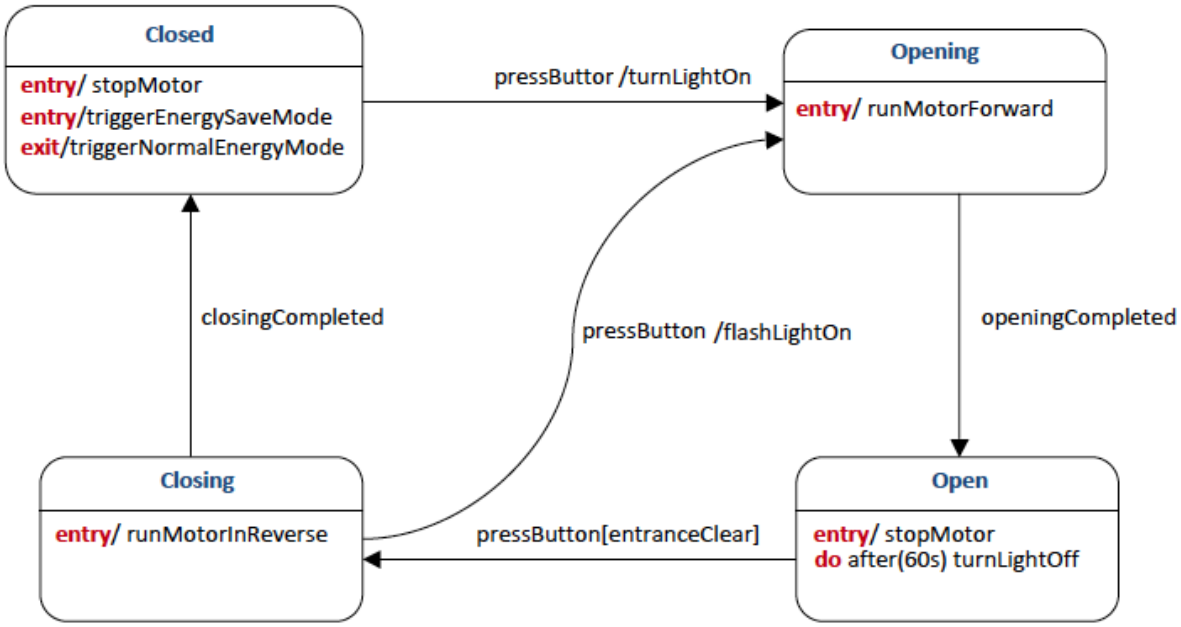


Figura 5.3 - Portão de garagem (http://tims-ideas.blogspot.com.br/2011/04/umple-totorial-2-basic-state-machines.html)

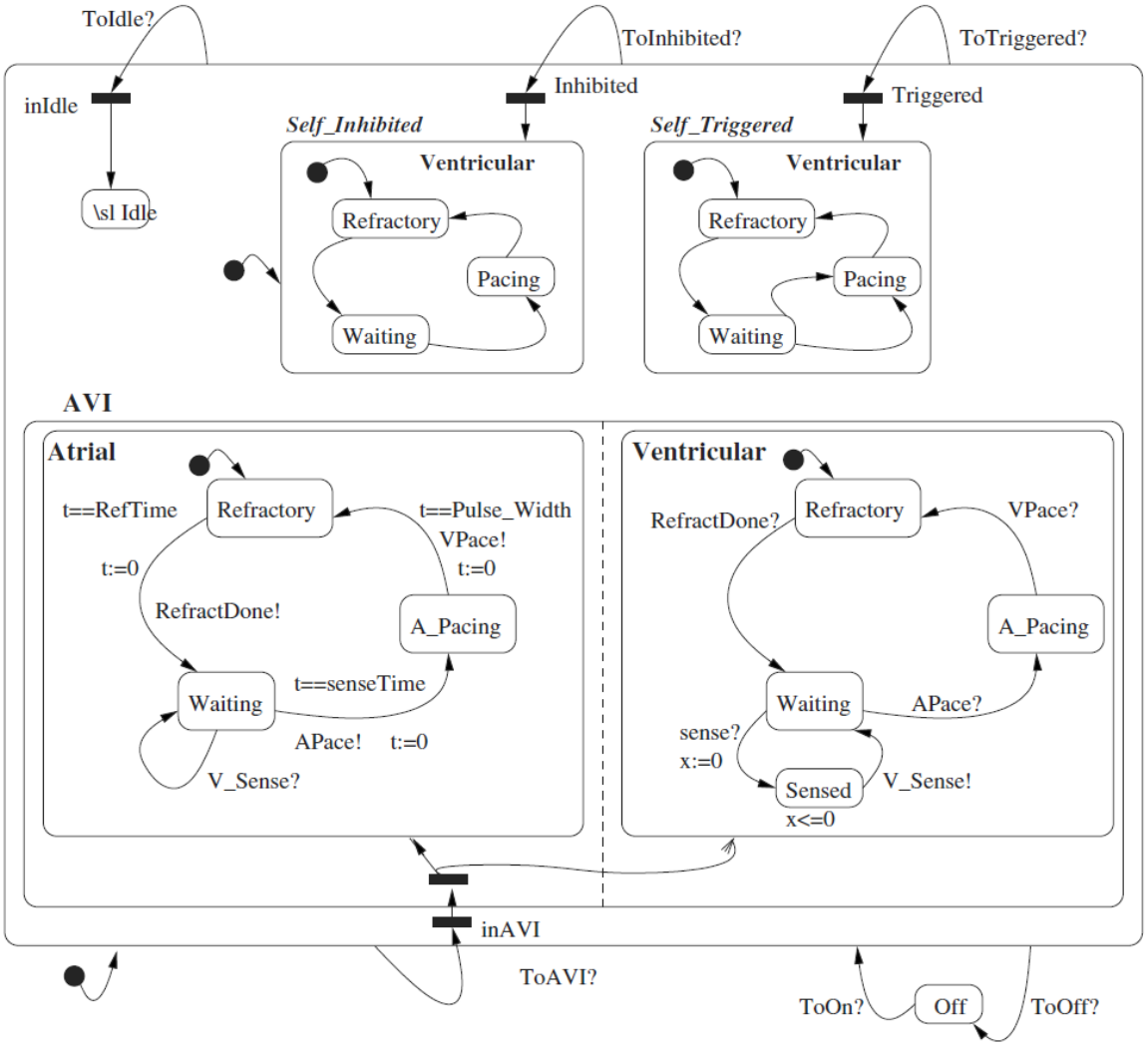


Figura 5.4 - Marca-passo (http://www.uml.org.cn/oobject/slides-nwpt-2001.pdf).

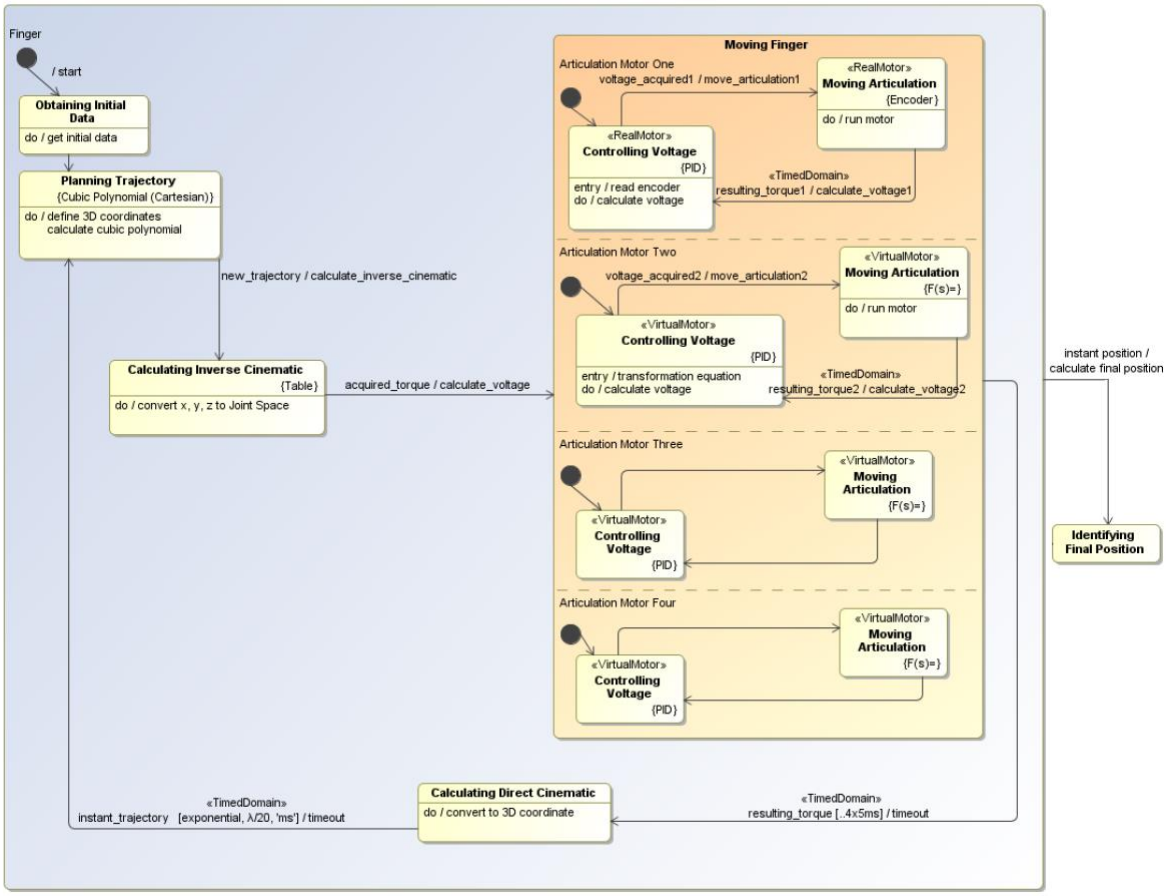


Figura 5.5 - Mão Kanguera (FREIRE, 2011).

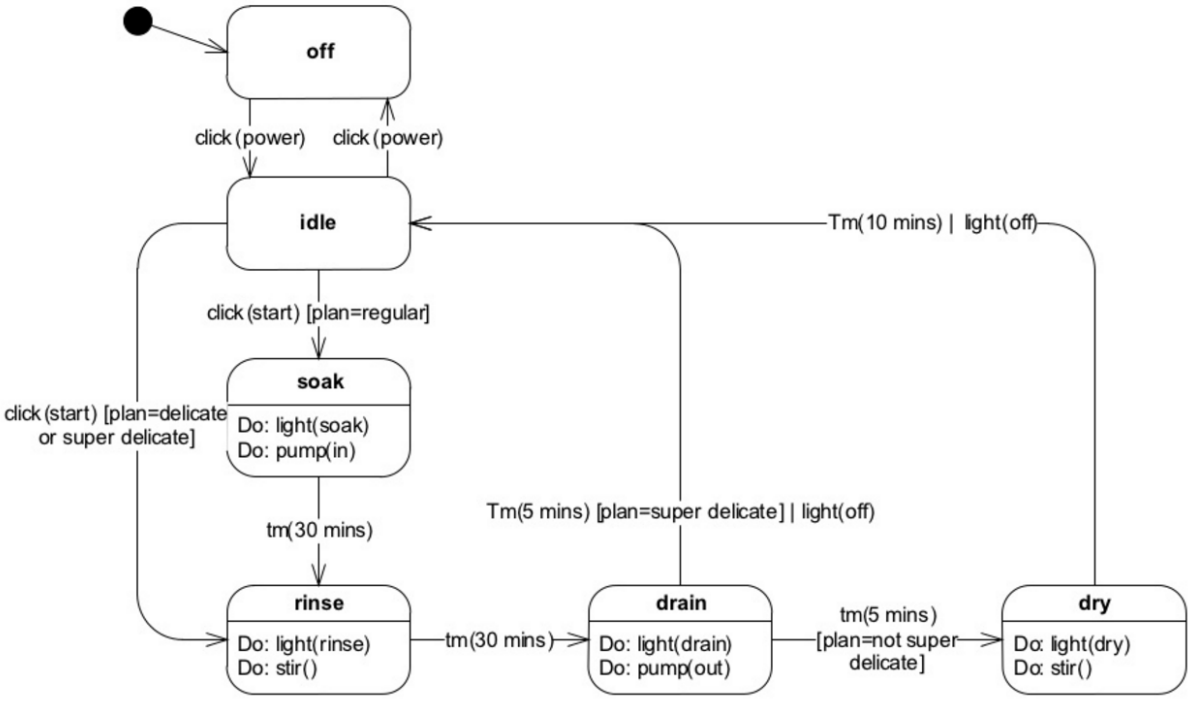


Figura 5.6 - Máquina de lavar (http://pt.slideshare.net/erant/uml-statechart-diagrams).

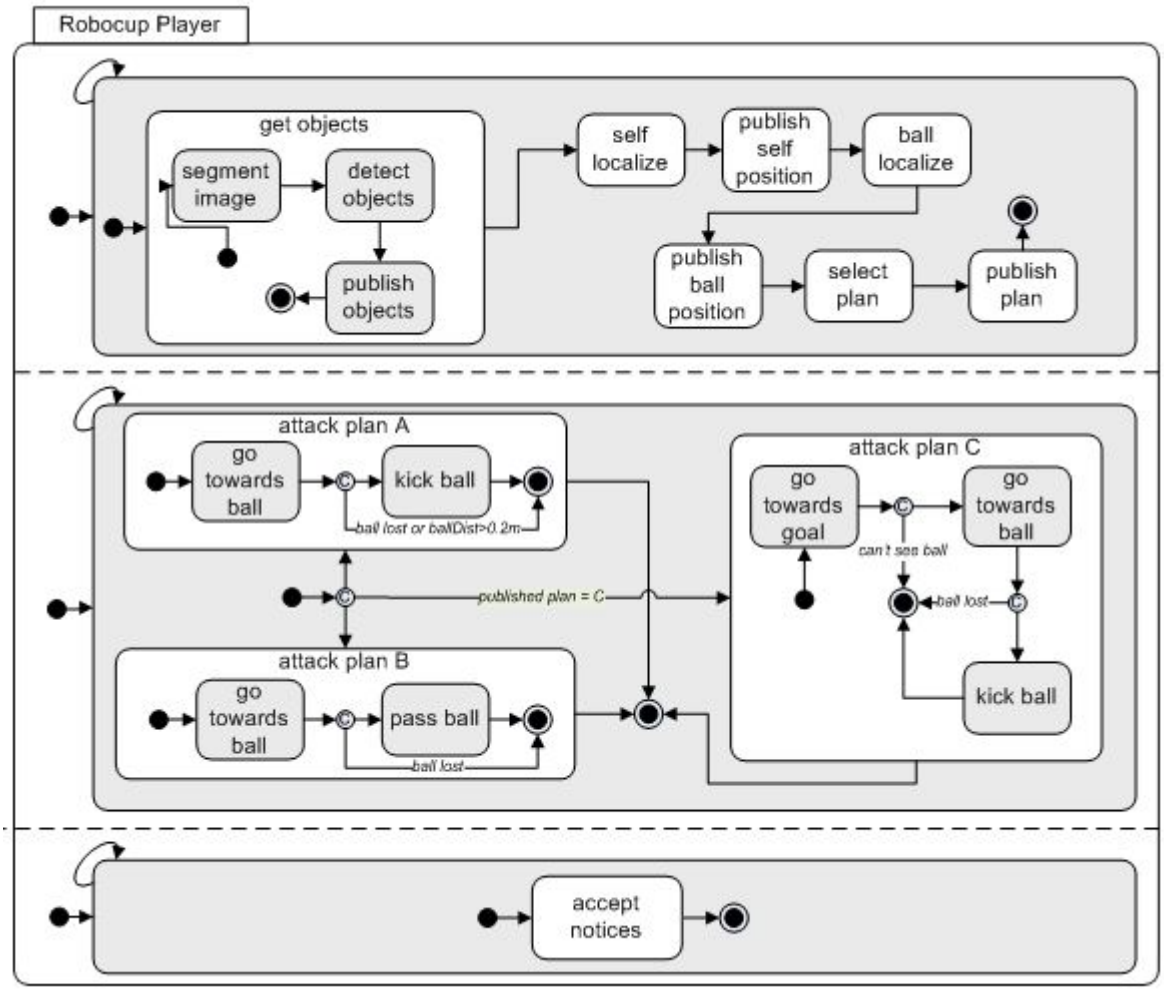


Figura 5.7 – Robô jogador de futebol (http://slideplayer.com/slide/4907526/).

Os modelos das Figuras 5.1 até 5.7 mostram como a modelagem UML Statechart pode ser aplicada em diferentes domínios de aplicação. Esses modelos auxiliaram na observação de como os desenvolvedores modelam o mesmo componente para um domínio diferente. Por exemplo, *Any key*, na Figura 5.1, *Press button*, na Figura 5.3, *ToOn* e *ToOff*, na Figura 5.4, *click (power)*, na Figura 5.6 e *click_start* na Figura 5.7, são todos exemplos de uma interrupção de um SE, ou seja, uma entrada assíncrona que obriga o SE a tomar uma decisão, no caso do modelo a interrupção gera uma mudança de estado. Nas Figuras 5.2, 5.5 e 5.6, *arm_time_event, TimeDomain* e *Tm*, respectivamente, são exemplos de relógios do SE. A observação dessas palavras que denominam componentes do SE em nível de modelo será útil criação de palavras chaves, apresentada no Capítulo 3.