Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Ciência da Computação

Modelagem e Verificação Automática de um Protocolo de Controle de Fluxo Adaptativo Usando Traços de Execução

Anne Lorayne Gerônimo Silva Augusto Moreira

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Kyller Costa Gorgônio
Angelo Perkusich
(Orientadores)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

M835m Moreira, Anne Lorayne Gerônimo Silva Augusto.

Modelagem e verificação automática de um protocolo de controle de fluxo adaptativo usando traços de execução / Anne Lorayne Gerônimo Silva Augusto Moreira. – Campina Grande, 2016.

74 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e informática, 2016.

"Orientação: Prof. Dr. Kyller Costa Gorgônio, Prof. Dr. Angelo Perkusich.

Referências.

1. Verificação Formal. Verificação Automática. 3. Sistemas Embarcados. 4. Modelos Formais. 5. NuSMV. 6. Engenharia de Programas de Computador. I. Gorgônio, Kyller Costa. II. Perkusich, Angelo. III.Título.

CDU 004.41(043)

Resumo

O desenvolvimento de sistemas embarcados possibilitou uma forte expansão no número de aplicações dependentes de dispositivos programáveis em áreas tão distintas como automobilística, sistemas financeiros e sistemas médicos. Uma eventual falha em algum desses sistemas pode provocar diferentes graus de danos e prejuízos e, por isso, exige-se um alto grau de confiabilidade em seu funcionamento. O aumento da complexidade dos novos sistemas computacionais e a pressão econômica e busca de novos mercados, concorrem para a busca da redução nos prazos de entrega dos dispositivos programáveis e de seus softwares e sistemas embarcados. Este trabalho apresenta um estudo de caso para a utilização de um método de verificação formal de software aplicado a um sistema computacional de controle de fluxo adaptativo para Gateways Bluetooth Low-Energy utilizados em sistemas de monitoramento remoto de pacientes. Os resultados obtidos neste trabalho confirmam a viabilidade da aplicação do método na verificação formal do software proposto.

Palavras-Chave: Verificação formal; Verificação automática; Sistemas embarcados; Modelos formais; NuSMV.

Abstract

The embedded system development had a positive impact on the expansion of applications dependent on programmable devices inside many areas such as automotive industry, financial services, and medical systems. A failure in any of these systems can cause losses and damages on many levels. Therefore, embedded systems require a high level of reliability while operating. The increasing complexity of these new computational systems, the cost-effective pressure, and the new market demand, contribute to reduce the delivery deadlines of the programmable devices, their softwares, and embedded systems. This research presents a case study in which we evaluated the usage of a formal verification method applied to a computational controlling system, with adaptive flow, for Gateway Bluetooth Low Energy used in patient monitoring systems. The results obtained in this study confirm the application feasibility of the formal verification method of the proposed software.

Keywords: Formal verification; Automatic verification; Embedded systems; Formal models; NuSMV.

Agradecimentos

Agradeço a Deus, em primeiro lugar, pelo dom da vida e pela oportunidade de poder realizar este trabalho, dando-me forças nos momentos difíceis.

Além de dedicar, agradeço aos meus pais Agnaldo e Lourdes, por todo o incentivo, apoio, compreensão, suporte em todos os momentos da minha vida, principalmente neste. Sendo eles os maiores responsáveis por esta conquista.

Agradeço aos professores e orientadores Kyller Gorgônio e Angelo Perkusich, pela orientação necessária ao desenvolvimento desta dissertação, pelos puxões de orelha, pela paciência e pela contribuição em minha formação acadêmica e profissional. Além deles, sou muito grata a Danilo Freire, que me ajudou bastante a conceber e aplicar a ideia central da pesquisa.

Aos meus familiares que sempre reclamaram da minha ausência nos momentos mais especiais.

Agradeço aos meus amigos do Laboratório Embedded e Virtus, em especial aos amigos da sala 108, dos projetos Sony e Motorola e ao meu namorado, pela amizade, carinho, paciência, auxílio e motivação em todos os momentos. Vocês me ajudaram a manter o foco e me incentivaram para que eu concluísse este trabalho.

Aos meus amigos, meus amores em especial, que compreenderam minha ausência e que mesmo assim estavam torcendo por mim.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro para o desenvolvimento desta pesquisa.

Enfim, a todos que contribuíram de forma direta ou indireta para a realização deste trabalho.

Conteúdo

Lista de Abreviaturas

6LoWPAN - *IPv6 over Low Power Wireless Personal Area Network*

ASM - *Abstract State Machine*

BAN - *Body Area Network*

BLE - *Bluetooth Low Energy*

CoAP - *Constrained Application Protocol*

CTL - *Computation Tree Logic*

DPS - *Dispositivos Pessoais de Saúde*

ECG - *Eletrocardiograma*

IETF - *Internet Engineering Task Force*

IoT - *Internet of Things*

IPS - *Informações Pessoais de Saúde*

IPSP - *Internet Protocol Support Profile*

LTL - *Linear Temporal Logic*

L2CAP - *Logical link control and adaptation protocol*

MEF - *Máquina de Estados Finita*

MQTT - *Message Queue Telemetry Transport*

MRP - *Monitoramento Remoto de Pacientes*

NFC - *Near-Field Communication*

NuSMV - *New Symbolic Model Verifier*

PAN - *Personal Area Network*

PMRP - *Processo de Monitoramento Remoto de Paciente*

QoS - *Quality of Service*

SMV - *Symbolic Model Verifier*

TA - *Timed Automata*

UML - *Unified Modeling Language*

WAN - *Wide Area Network*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

O aumento de vendas de dispositivos miniaturizados que tem comunicação pela internet, à exemplo de relógios e pulseiras inteligentes, tem popularizado o termo "Internet das coisas" (IoT). A comunicação entre estes dispositivos e servidores, por meio de sistemas embarcados, contribuem para a coleta e análise de informações de dados. Os resultados destas análises podem auxiliam na melhoria da qualidade de vida de seus usuários [19].

Os sistemas embarcados, ultimamente, expandiram seu uso e foram essenciais no crescimento de uma ampla gama de aplicações como dispositivos portáteis, eletrônicos de consumo, instrumentos médicos e sistemas de controle industriais [12]. O aumento do uso destes sistemas trouxe a necessidade de reduzir o número de falhas eventualmente associadas a tais sistemas. Uma eventual falha em algum desses sistemas pode provocar diferentes graus de danos e prejuízos, como o ocorrido em 2005 que obrigou a Toyota a realizar o *recall* de 625 mil de carros híbridos ao redor do mundo [4], devido à falha que desligava inesperadamente o carro em movimento.

Sistemas que realizam a comunicação de dados do usuário para comandar a ativação ou desativação necessitam uma atenção especial em sua confiabilidade. Um alto grau de confiabilidade é necessário para evitar as falhas e os possíveis problemas associados à elas. Um exemplo é o caso em que um dispositivo coleta a taxa de glicose no sangue, que é monitorada por um sistema de controle *online* que controla um dispositivo que regula a injeção de insulina no paciente monitorado. A ocorrência de falhas na injeção podem causar graves consequências à saúde, gerando inclusive risco de vida ao paciente. Esse caso exemplifica a necessidade da realização de uma verificação destes sistemas de forma rápida e automa-

tizada, com o objetivo de detectar possíveis falhas no seu desenvolvimento e sua correção antes da utilização destes sistemas pelos usuários finais.

Este trabalho apresenta um método de verificação formal de software para protocolos de controle de fluxo de dados, a partir de informações provenientes do núcleo do Gateway Bluetooth Low-Energy.

Foi realizado um estudo de caso onde o sistema computacional proposto por Santos [31], o qual se refere a um sistema de controle de fluxo adaptativo para Gateways Bluetooth Low-Energy aplicado a sistemas de monitoramento remoto de pacientes, será testado pelo método proposto nesta dissertação. O método utilizado foi baseado no método para a verificação de software proposto por Vasconcelos [37]. Foram analisados os resultados para discutir a viabilidade do método. Os resultados obtidos neste trabalho confirmam a viabilidade da aplicação do método na verificação formal do software proposto por Santos [31].

1.1 Objetivos

O objetivo deste trabalho é desenvolver, validar e avaliar a aplicação de um método de verificação automática a protocolos de controle de fluxo de dados, a partir de informações provenientes do núcleo do *Gateway*. O método proposto tem a finalidade de gerar um conjunto de transições de estados para validar a implementação do protocolo a partir da especificação. Por fim, o objetivo principal deste trabalho é validar especificamente protocolos de controle adaptativo utilizados em Gateways BLE aplicados em um contexto de sistema de monitoramento remoto de pacientes.

1.1.1 Objetivos Específicos

Os objetivos específicos deste trabalho são os seguintes:

1. Definir e detalhar um método para verificação automática de protocolos de priorização;

2. Gerar e executar casos de teste a partir da ferramenta NuSMV. Esta etapa tem o propósito de verificar se a implementação do modelo verificador está em conformidade com a especificação;

3. Apresentar um estudo de caso com o objetivo de demonstrar a aplicação do método proposto.

4. Realizar uma avaliação crítica dos resultados obtidos no estudo de caso, de modo a validar que o método proposto realiza a verificação automática de protocolos de controle de fluxo de dados de maneira eficaz.

1.2 Relevância e Contribuições

A relevância do método proposto neste trabalho foi a construção de um modelo reduzido do protocolo de controle de fluxo de forma automática, baseando-se nas informações provenientes do log do núcleo do servidor e a formatação na linguagem da ferramenta NuSMV, permitindo a verificação formal do protocolo de controle pela ferramenta NuSMV. Para utilização do método proposto por outros programas, que já possuem o código instrumentado, necessita-se elaborar um método para obtenção das características do Modelo, o que permitiria a adequação ao modelo proposto.

Desta forma, o método proposto auxilia na garantia da consistência da implementação do protocolo pois, alterações no protocolo podem ter sua verificação formal realizada rapidamente, inclusive facilitando a revisão pois a ferramenta NuSMV localiza e mostra as inconsistências encontradas.

A principal contribuição no contexto deste trabalho é a disponibilização de um método que aumenta a segurança e a confiança, utilizando *model checking*, de protocolos de controle de fluxo adaptativo para Gateways Bluetooth Low-Energy que utilizam bluetooth. Tal tarefa é realizada através da geração de um modelo M utilizado na verificação formal. Com a verificação formal, erros podem ser detectados e corrigidos previamente. A abordagem proposta foi validada com sucesso por meio de um estudo de caso que utilizou o resultado da execução do protocolo proposto por Santos [31].

1.3 Estrutura da Dissertação

Este documento está estruturado da seguinte forma: no Capítulo 2 será apresentada a fundamentação teórica relacionada ao estudo de caso proposto onde serão apresentados os concei-

tos relacionados ao presente trabalho necessários a realização desta pesquisa. Neste capítulo, ainda, serão apresentados o trabalho proposto por Santos [31] e a fundamentação teórica relacionada ao mesmo. No Capítulo 3 serão apresentados os trabalhos relacionados e utilizados na realização desta pesquisa. No Capítulo 4 será apresentado o método proposto, permitindo ao leitor conhecer o método sem o detalhamento necessário empregado nesta pesquisa. No Capítulo 5 será apresentado o estudo de caso realizado, detalhado, assim como uma discussão dos resultados obtidos. Por fim, no Capítulo 6 serão apresentadas as considerações finais, limitações do trabalho e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

O processo de verificação automática de modelos (*Model Checking*) é uma técnica madura e poderosa de verificação de sistemas que vem sendo estudada desde 1980, o que possibilitou o desenvolvimento de lógicas de conhecimento, empregadas em especificações formais e ferramentas de verificação de modelos capazes de contemplar as características dos processos de interação racional, que caracterizam os sistemas multiagentes e reativos [6, 10, 13].

Dessa forma, as lógicas temporais, com operadores modais de conhecimento adicionados, vem sendo amplamente utilizadas para modelar estes sistemas, porém as técnicas de verificação de modelos para lógicas temporais e epistêmicas foram estudadas em proporção bem menor para a checagem de modelos e a representação de conhecimento. A abordagem de verificação de modelos proporciona a oportunidade de modelar evolução do conhecimento no sistema e isto torna-se bastante útil para compreender o comportamento do sistema, bem como seu propósito. Através da modelagem do conhecimento envolvido no sistema, por parte de cada agente, obtém-se a possibilidade de modelar a autonomia e interação, duas características marcantes de sistemas concorrentes [13].

A ferramenta NuSMV, utilizada nesse trabalho, na seção 2.2, baseiam-se em Caldas [6] completando a teoria necessária a modelagem e teste do programa de controle de acesso à rede utilizado no estudo de caso.

2.1 Verificação formal por Model Checking

Verificação de modelo é um método formal para verificar sistemas concorrentes de estado

finito. A partir de um modelo base do sistema, as especificações do modelo serão verificadas nas mudanças de estado do modelo [10]. As especificações do sistema, que será verificado, são escritas em lógica temporal. Ocorre uma verificação, estado por estado, para verificar se as especificações estão sendo satisfeitas no modelo. A verificação formal poderá durar muitos minutos, dependendo do número de estados do modelo especificado.

A aplicação deste método ocorre através de três etapas [11]:

- Modelagem: consiste na etapa de construção do modelo formal do sistema, representado em alguma ferramenta de modelagem. Todas as variações de estado desse sistema serão representadas no modelo;

- Especificação: consiste na etapa de descrição do comportamento do sistema. As variações, dependendo do estado, devem ser descritas. As restrições do sistema também devem ser especificadas.

- Verificação: consiste na etapa de verificar o modelo, de acordo com as especificações descritas na etapa acima. Essa etapa tem como resultado verdadeiro ou false, dependendo se as especificações foram atendidas no modelo ou não. Se as especificações não forem atendidas, a sequência de estados que chegou a falha e as especificações errôneas serão exibidas.

O sistema que foi verificado neste trabalho é composto por um servidor e vários clientes. O servidor realiza o controle de créditos disponibilizados para cada cliente. O controlador desses créditos emite uma **log** para o núcleo do servidor. O **log** gerado após a execução do controlador foi a base utilizada para gerar o modelo do sistema.

Essa técnica foi utilizada neste trabalho devido à vantagem de ser uma técnica com um elevado grau de automatização e à existência de ferramentas de simples utilização. O verificador de modelos utilizado neste trabalho foi o NuSMV devido à facilidade de automatização da conferência do modelo, de acordo com o tipo de modelo utilizado.

2.2 A Ferramenta de Código Livre - NuSMV

O *New Symbolic Model Verifier* - NuSMV [8] é uma ferramenta de verificação de especificações descritas em lógica temporal CTL sobre sistemas de estados finitos. Esta seção

apresenta os princípios dessa linguagem utilizada por esse verificador utilizados neste trabalho.

A lógica temporal linear (LTL) [29] assume o tempo como uma sequência de execução de um sistema onde cada possível caminho de computação é considerado separadamente e descritas como uma única sequencia de execução. De maneira diferente da formulação CTL, ao invés de serem interpretadas sobre árvores de computação, as fórmulas LTL são interpretadas modelando os caminhos individuais de computação, ou seja, a lógica temporal linear expressa propriedades sobre uma sequencia linear de execução do sistema [6].

A ferramenta NuSMV é uma ferramenta de código aberto utilizada para verificação de modelos simbólicos que foi criada a partir da ferramenta SMV, que é o verificador de modelos baseado em BDD desenvolvido na Carnegie Mellon University [6, 25].

O NuSMV provê uma linguagem para descrição do modelo que verifica diretamente a validade das fórmulas em LTL e também em CTL e recebe como entrada um texto que consiste de um programa descrevendo o modelo e algumas especificações descritas como fórmulas em lógica temporal. O resultado da linguagem é a saída "$true$", se a especificação é satisfeita, ou um contraexemplo mostrando porque a especificação representada pelo programa não é satisfeita, ou seja, uma saída "$false$" [6].

O NuSMV foi projetado para atender aos padrões requeridos pela indústria sendo robusto e fácil de manter e modificar, tendo sido escrito em ANSI C e seu código fonte dividido em vários módulos. A linguagem de entrada de NuSMV permite a descrição de máquinas de estado finitas (MEF) que são capazes de descrever processos síncronos e assíncronos e condições de não determinismo [6]. O propósito inicial da entrada do NuSMV é descrever as relações de transição das MEF com as relações de evoluções válidas do estado das MEF (modelo do sistema) e, dessa forma, podem ser identificadas as possíveis configurações futuras de um sistema a partir do seu estado atual. De um modo geral, qualquer expressão no cálculo proposicional pode ser utilizada para definir as relações de transição permitindo uma grande flexibilidade. Essa facilidade, entretanto, requer, ao mesmo tempo, um certo cuidado adicional para evitar inconsistências como a presença de uma contradição lógica, que pode resultar em um $deadlock$ [6, 15].

Na sequência serão descritos os elementos da especificação de um sistema utilizando a linguagem de entrada para a ferramenta NuSMV [6]:

- ***MODULE*** – Encapsula as declarações de todas as variáveis, inclusive as de estado e os eventos que são declarados ***BOOLEAN***. Cada módulo pode conter a regra de transição dos estados e a especificações das propriedades, sendo possível declarar um módulo com parâmetros para realizar a reutilização de código e a passagem ocorre por referência;

- ***VAR*** – As declarações das variáveis são realizadas no bloco ***VAR*** e podem ser do tipo enumerado, intervalar, booleano, além de instâncias de outros módulos;

- ***ASSIGN*** – Nesta seção são declaradas as regras que determinam a inicialização e transições para o próximo valor de uma variável. A atribuição direta estabelece o valor inicial da variável, por meio da comando ***init***(variável). O comando ***next***(variável) fornece o valor da variável no próximo estado, a partir do corrente, sendo que a atribuição em ***next***(variable) pode ser feita utilizando a regra ***case***, onde é possível determinar o próximo valor da variável em função de várias condições;

- ***DEFINE*** – A seção ***DEFINE*** é utilizada para associar uma variável a uma expressão, assim uma variável em ***DEFINE*** é sempre substituída pela sua definição quando é encontrada na especificação;

- ***MODULE main*****(módulo principal)** - Toda especificação NuSMV deve possuir um módulo principal, sem parâmetros, que representam o sistema;

- ***CTLSPEC*** – As propriedades CTL são especificadas precedidas da palavra chave ***CTLSPEC*** (SPEC anteriormente). Em caso de utilizar propriedades formuladas em LTL o comando utilizado será o ***LTLSPEC***.

Código 2.2.1 *Modelo do Sistema na Linguagem NuSMV*

MODULE main
VAR
    request : boolean;
    state : $\{$*ready, busy*$\}$*;*
ASSIGN

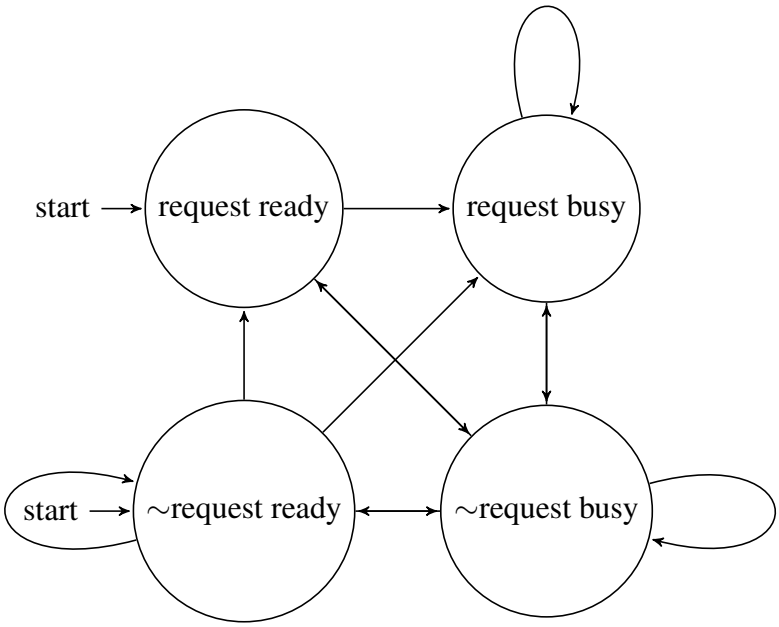
```
    init(state) := ready;
    next(state) :=
        case
            state = ready & request: busy;
            1 : {ready, busy};
        esac;
LTLSPEC
    G(request -> F state=busy)
```

Caldas apresenta o Código 2.2.1 como um modelo do sistema descrito ou programado na linguagem NuSMV. Um programa em NuSMV pode ter um ou vários módulos e, tal como algumas linguagens de programação, um dos módulos deve ser chamado "$main$", sendo nos módulos declaradas as variáveis e seus respectivos valores. As atribuições normalmente são feitas com um valor inicial para cada variável e em seguida uma especificação do próximo valor por meio de uma expressão formada pelos valores correntes das variáveis, cuja expressão pode ser não-determinística [6].

O Código 2.2.1 consiste de duas variáveis request do tipo booleana e state do tipo enumerado $\{ready, busy\}$, onde 0 denota "$false$" e 1 representa "$true$". Os valores iniciais e subsequentes da variável $request$ não são definidos no programa e, desta forma, os valores de $request$ serão definidos pelo ambiente externo durante a execução do programa e, com isso, a variável $state$ fica parcialmente definida, inicialmente ela é "$ready$" e pode ficar "$busy$" se a variável $request$ for "$true$", contudo, se a variável $request$ for "$false$" o valor de state fica indeterminado. A avaliação da expressão formada com o "$case$" é feita de cima para baixo, sendo que a primeira avaliação do lado esquerdo do sinal ":" (dois pontos) que tiver seu valor verdadeiro, atribuirá o valor do lado direito para a variável declarada. Desta forma o valor "1:" permanece como avaliação padrão, caso nenhuma das avaliações anteriores seja verdadeira [6].

O programa denota o sistema de transição da Figura 2.1, onde existem quatro estados e cada estado é definido em função dos valores possíveis para as duas variáveis. No sistema o nome "$\sim request$" significa um valor falso para a variável "$request$". O programa e o sistema de transição são não-determinísticos, isto é, o próximo estado não é unicamente de-

Figura 2.1: Máquina de Estados do Sistema.



finido. Qualquer transição de estado baseado no comportamento da variável "$state$" vem em pares e o resultado é a transição para um estado sucessor onde "$request$" pode ser falso ou verdadeiro. Na Figura 2.1 verifica-se que a partir do estado $\{\sim request, busy\}$ o sistema pode caminhar para quatro estados destinos, ele mesmo e mais três outros estado. As especificações em LTL são introduzidas pela palavra chave LTLSPEC e são simples fórmulas LTL. No programa exemplo a especificação está declarando que para qualquer estado, se o valor de "$request$" é verdadeiro, então eventualmente ocorrerá um estado "$busy$" [6].

E finalmente, optou-se pela ferramenta NuSMV pelos seguintes motivos:

• é uma ferramenta código livre e pode ser gratuitamente adquirida através da URL *http://nusmv.irst.itc.it*;

• possui uma boa documentação que auxilia na construção dos modelos e das especificações em LTL e CTL;

• permite trabalhar com arquivos de entrada contendo a descrição do modelo.

2.3 Sistemas de Monitoramento Remoto

Nesta seção apresenta-se a proposta de trabalho de Santos [31] e a fundamentação teórica para o entendimento de tal proposta. O trabalho proposto pelo autor foi um controlador de

fluxo de dados baseado em créditos para o Bluetooth Low-Energy que foi utilizado no estudo de caso deste trabalho.

O número de dispositivos conectados em rede cresce rapidamente com a aumento em escala da produção e consequente redução do preço dos dispositivos e, são disponíveis em áreas como vigilância e controle de tráfego, residências, indústrias, comércio entre outros. Já os dispositivos embarcados, sejam eles sensores ou atuadores, tornam-se capaz de se conectar entre si e com a Internet, viabilizando a chamada Internet das Coisas (do inglês *Internet of Things* – IoT) [5].

A IoT possibilita a criação de uma rede de dispositivos conectada à rede humana habilitando a interação desses dispositivos com o mundo físico. Um exemplo das inúmeras possibilidades seria a de um dispositivo automático que monitore a temperatura do ambiente, e sendo informado da chegada das pessoas pela rede, ligue o ar condicionado ou aquecedor para aumentar o conforto humano dos moradores na chegada a residência. Nesse caso, um dispositivo coleta informações do ambiente e da rede humana e interage diretamente com o mundo físico. Dessa forma, o cenário onde sensores, atuadores e unidades computacionais interagem com o mundo físico é viável através de sistemas físico-cibernéticos concretizando-se um tendência de integração dos sistemas físico-cibernéticos com a internet [23, 38].

Dessa forma, um dos desafios da IoT para permitir a integração de dispositivos heterogêneos à internet está na possibilidade destes dispositivos se comunicarem de maneira eficaz respeitando suas restrições no que diz respeito ao consumo de energia, processamento e armazenamento. Com isso, o uso de tecnologias de comunicação eficientes e de baixo consumo torna-se essencial, o que permitiu o desenvolvimento e aprimoramento de tecnologias de comunicação e sensoriamento de tecnologias de baixo custo, manutenção e consumo de energia como o Bluetooth [1], *Near-Field Communication* (NFC) [2], ZigBee [3] e o recente *Bluetooth Low Energy* (BLE), apenas citando algumas tecnologias disponíveis.

Adicionalmente, a cada dia aumenta o crescente número de dispositivos conectados, como *smartphones* e *smartwatches* que permitem o desenvolvimento de novos serviços e aplicações que integram a IoT com tecnologias de uso pessoal. Nessa linha de pesquisa e desenvolvimento, novos protocolos estão sendo desenvolvidos e avaliados em diversas camadas como o *Constrained Application Protocol* (CoAP) [32] e o *Message Queue Telemetry Trans-port*(MQTT) [24], que apresentam soluções na camada de aplicação que podem ser

utilizadas por dispositivos embarcados com poucos recursos computacionais e de armazenamento energia, como sensores.

Esses protocolos são executados sobre o Protocolo de Internet (do inglês, *Internet Protocol* - IP), viabilizando a integração da Internet com dispositivos pessoais. No uso pessoal dessas tecnologias em dispositivos para o consumidor final, a utilização do BLE para comunicação com a Internet se torna bastante promissora dada sua ampla adoção em dispositivos como *smartphone*, *smart-tv*, entre outros, além do seu baixo consumo de energia [26].

Nessa mesma linha, novos perfis ou configurações permitem que novos sensores e dispositivos possam transmitir dados IP utilizando suas interfaces de transmissão. É o caso do perfil IP Service Profile (IPSP) do BLE [36] em conjunto com novos mecanismos de controle de fluxo na camada de enlace, os quais criam novos tipos de canais para comunicação para transportar dados IP. Além disso, uma especificação para o uso do protocolo 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) [27, 36] sobre BLE foi definida pelo IETF.

Esse conjunto de tecnologias e dispositivos permite ao dispositivo de uso pessoal, como *smartphone*, tornar-se Gateways de Internet para dispositivos periféricos de um usuário utilizando o BLE. Esse cenário permite que Gateways possam ser considerados serviços pessoais, ou seja, são utilizados por um único usuário através de seu dispositivo pessoal. Nesse contexto, esses Gateways criam redes do tipo PAN BLE (do inglês *Personal Area Network* - PAN) e se conectam com diversos dispositivos e serviços. Com essa configuração, uma mesma PAN pode ser compartilhada por serviços de áudio, vídeo, acessibilidade, e interação humano-máquina.

Esse tipo de compartilhamento de rede pode levar a um decréscimo na Qualidade de Serviço (QoS) de rede dos dispositivos periféricos conectados, como uma redução na taxa de transmissão média, por exemplo [30, 39]. Considerando redes com tecnologia Bluetooth Low-Energy, por exemplo, o dispositivo mestre da rede é o responsável pelo gerenciamento do fluxo de dados entre os dispositivos clientes. Dessa forma, nesse tipo de rede, fica a critério do mestre distribuir recursos na rede, ou seja, esse dispositivo acaba distribuindo os recursos de maneira igualitária entre todos os clientes. Nesse sentido, requisitos de QoS de rede para BLE podem ser definidos a partir de alguns parâmetros, os quais são utilizados pela camada de enlace da especificação Bluetooth [33] para o controle de canais:

- Taxa de transmissão máxima: define o quão rápido pacotes devem ser enviados bidirecionalmente por aplicações BLE.

- Latência de acesso: define o atraso máximo aceitável para um pacote ser enviado a interface a área após sair da camada de enlace.

- Variação de atraso: define os valores máximos e mínimos de atraso de envio de pacotes entre as camadas de enlace de dois nós.

Analisando esses requisitos de QoS em relação a sua percepção para aplicações BLE, a taxa máxima de transmissão torna-se o principal fator de avaliação. Redes PAN BLE, portanto, devem garantir valores de taxa de transmissão para aplicações com diferentes requisitos de QoS. Essas aplicações podem pertencer a diferentes áreas, tais como a área de saúde em sistemas de Monitoramento Remoto de Pacientes.

2.3.1 Sistemas de Monitoramento Remoto de Pacientes

O aumento de custos com a saúde pessoal e a demanda crescente por novos serviços para o tratamento de doenças crônicas fez com que fossem criados novos desafios e oportunidades para os serviços de saúde [34]. Dessa maneira o uso da tecnologia para o monitoramento de pacientes vem expandindo-se e o interesse por tecnologias de monitoramento de saúde impulsiona o desenvolvimento de novos Dispositivos Pessoais de Saúde (DPS) com interfaces de comunicação embutidas como medidores de pressão arterial, oxímetros, glicosímetros, termômetros, entre outros.

Através dos DPS os dados da saúde do indivíduo monitorado são coletados e enviados para a Internet através de suas interfaces de comunicação, permitindo o seu acesso por profissionais da área de saúde que podem monitorar a evolução do estado de saúde dos pacientes remotamente, aumentando a capacidade de prever complicações no estado de saúde e tomar ações com antecedência, evitando complicações futuras. Este processo, em conjunto com suas tecnologias envolvidas, é conhecido como Saúde Conectada [7].

Um sistema de Saúde Conectada viabiliza um cenário onde DPS se conectam com a Internet para exportar seus dados, portanto, construindo a Internet das Coisas para a área de saúde. É possível casos de uso onde um DPS faz a coleta dos dados de saúde do paciente, e

estes dados são compartilhados automaticamente com o seu médico transparentemente. Um sistema de Saúde Conectada é composto por múltiplos componentes, desde o DPS até o serviço de Monitoramento Remoto de Pacientes (MRP) na Internet, sendo a base desse sistema a coleta de Informações Pessoais de Saúde (IPS) através de um DPS, e o seu compartilhamento pela Internet através de uma interface de comunicação.

A partir dessa coleta e compartilhamento de dados, um dispositivo agregador ou um Gateway recebe essas IPS e as encaminha para o serviço de MRP na Internet. Esses Gateways podem ser dispositivos pessoais portáteis como *smartphones* ou computadores pessoais, permitindo desde o DPS até o serviço em nuvem, onde a informação é transportada por diversos meios de comunicação ao seu destino a partir da rede pessoal ou corporal (do inglês *Body Area Network* - BAN) até uma rede de larga escala (do inglês *Wide Area Network* - WAN) como a Internet.

Alguns desafios devem ser considerados na implantação de sistemas de Saúde Conectada para o MRP, dependendo do público alvo diferentes tipos de dispositivos poderão ser utilizados. Por exemplo, se o caso de uso for a realização de um MRP contínuo em diferentes localizações, o uso de Gateways portáteis e pervasivos, como *smartphones*, torna-se obrigatório em conjunto com esses Gateways, sensores corporais podem ser utilizados como DPS, como por exemplo, uma pulseira que faz aferições de frequência cardíaca continuamente.

Além de utilizar diferentes tecnologias de transmissão, como BLE, esses dispositivos e Gateways precisam definir protocolos para a troca de dados de saúde. Em relação a esse ponto, boa parte das soluções e fabricantes definem seus próprios protocolos, criando soluções verticais onde seus DPS conversam apenas com seus Gateways e serviços de saúde. Dessa forma, associações e grupos de trabalhos definiram padrões de interoperabilidade para diversos níveis da cadeia de comunicação de um sistema de Saúde Conectada, no nível de DPS, na família de padrões ISO/IEEE 11073.

Essas normas definem como esses dispositivos devem trocar dados com outras entidades (o padrão ISO/IEEE 11073:20601 [20]), e como as informações de saúde devem ser representadas (o padrão ISO/IEEE 11073:10101 [21]). Um detalhe do ISO/IEEE 11073 é sua independência da camada de transporte, viabilizando seu uso sobre qualquer tipo de protocolo ou tecnologia de transporte.

Quanto ao compartilhamento de dados de saúde com serviços na Internet, o Continua

Health Alliance[1], apresenta recomendações para o compartilhamento de dados de saúde entre DPS, agregadores de dados e serviços de armazenamento de dados de saúde [7]. Essas recomendações têm como objetivo viabilizar um cenário onde DPS compartilham dados com a Internet através de Gateways e agregadores padronizados.

Em um cenário mais amplo da Saúde Conectada, pacientes são monitorados constantemente, e processos de MRP (PMRP) podem ser executados a qualquer momento. Um PRMP se caracteriza como uma sequência de ações onde dispositivos e sensores enviam dados a serviços na Internet, os quais tomam decisões sobre a saúde do paciente. Já no caso do MRP, na maioria dos casos esse decréscimo nas características de QoS entregues não influencia nas aplicações como o caso de uma aferição eventual de pressão arterial ser entregue com alguns segundos de atraso.

Entretanto, durante situações de urgência, quando as informações de um DPS são essenciais para um diagnóstico, torna-se necessário elevar os requisitos de QoS desse DPS durante o processo de monitoramento como em um processo de MRP onde diversos dispositivos podem ser utilizados, um monitor de ECG portátil, um medidor contínuo de glicose implantado sob a pele, e bombas de insulina, e esses dispositivos devem ter seus requisitos de QoS de rede satisfeitos em prol do bem-estar do paciente.

2.3.2 Controlador de fluxo de dados baseado em créditos para o Bluetooth Low-Energy

Um controlador de fluxo de dados baseado em créditos para o Bluetooth Low-Energy foi proposto por Santos [31]. Esse controlador faz uso de um novo mecanismo de controle de fluxo baseado em créditos introduzido no Bluetooth 4.2. O controlador tem como objetivo evitar que o controlador do dispositivo mestre entre em um estado de funcionamento instável e pare de funcionar, deixando inativo alguns serviços que podem ser prioritários. Santos [31] definiu alguns requisitos básicos para o desenvolvimento do novo controlador, são eles:

- O dispositivo mestre conhece a priori a limitação de banda de seu controlador Bluetooth.

[1] http://www.continuaalliance.org

- O dispositivo mestre deve controlar o fluxo de dados originado por suas aplicações e serviços.

- O dispositivo mestre da rede deve controlar o fluxo de dados de seus clientes (slaves), quando necessário através da alocação dinâmica de créditos para os mesmos.

O funcionamento de um típico controlador baseado em créditos para o Bluetooth Low-Energy é apresentado na Figura 2.3.2. O controle de créditos é executado toda vez que o cliente envia um novo pacote *LE-Frame* ao mestre. Ao receber um novo *LE-Frame*, o dispositivo mestre decrementa a contagem de créditos daquele cliente e verifica qual a contagem atual do mesmo. Caso essa contagem fique menor do que um valor X, esse valor atribuído a X é definido no controlador, o dispositivo mestre decide enviar mais Y créditos ao dispositivo cliente, fazendo com que o mesmo fique com X + Y créditos disponíveis para envio, onde X e Y são parâmetros de implementação do *Logical link control and adaptation protocol* - L2CAP do mestre.

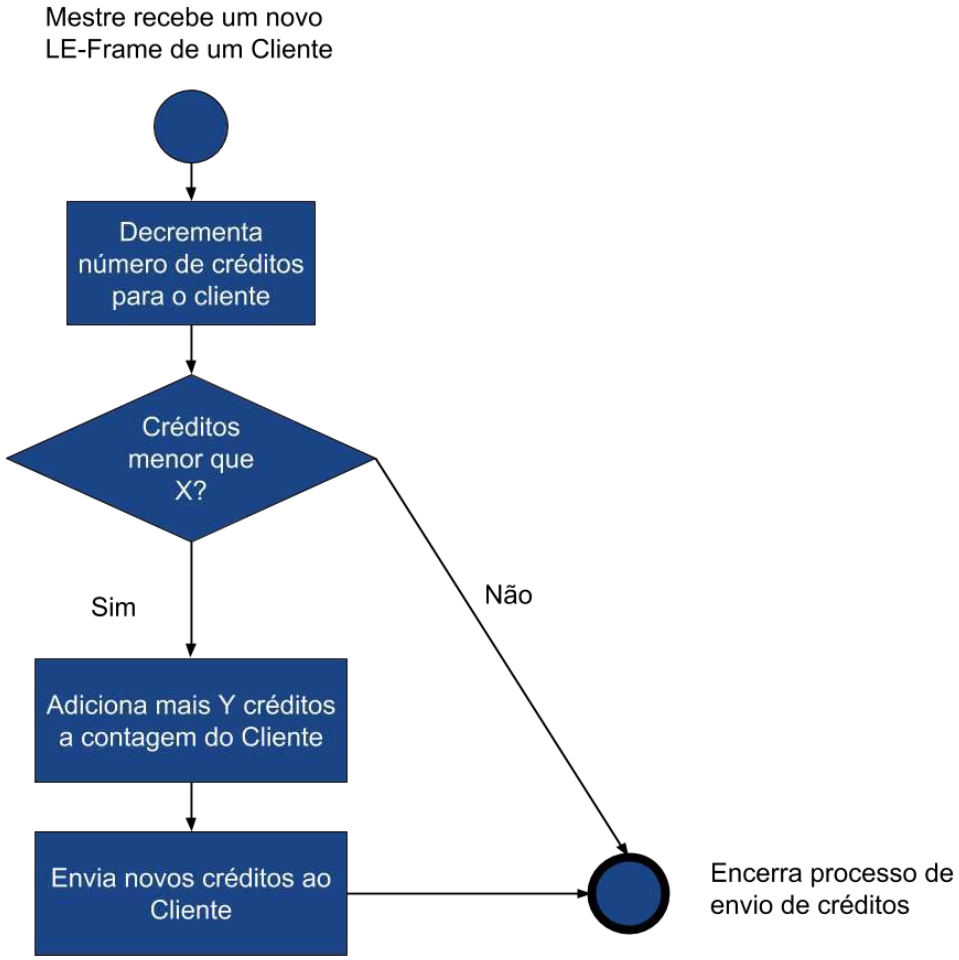
Nesse modelo padrão de controle de distribuição de créditos, todos os clientes sempre têm créditos disponíveis para envio. Deste modo, esses dispositivos tem liberdade para enviar dados a todo o momento enquanto os *slots* de tempo, na camada de enlace, estiverem disponíveis. O controlador Bluetooth não implementa controle de acesso sofisticado na camada de enlace, com isso o modelo de distribuição homogênea de créditos na camada L2CAP faz com que todos os clientes tenham o mesmo direito de acesso ao canal. Portanto, todos os clientes têm a seu dispor canais sem garantia de QoS, ou seja, canais *best-efforts*.

Após os fatos mencionados acima, Santos propôs um novo modelo de controle de fluxo baseado em créditos, o qual faz uso de regras de controle através de um *Configurador Adaptativo*, e de *Agentes* independentes de distribuição de créditos. O novo modelo de controle proposto pelo autor é apresentado na Figura 2.3.2.

De maneira semelhante ao controlador padrão descrito na Figura 2.3.2, ao receber um novo *LE-Frame* de um cliente o dispositivo mestre decrementa a contagem de créditos daquele nó. Nesse momento, o novo controlador verifica se a contagem de créditos do cliente alcançou o valor $zero$ ou não. O valor $zero$ foi escolhido para dar oportunidade ao dispositivo mestre de "segurar" o envio de novos *LE-Frames* por parte daquele cliente. Permitindo que o dispositivo mestre controle o envio de novos pacotes por parte do cliente por meio

Figura 2.2: Fluxo de controle baseado em créditos padrão.

Fonte: Santos [31], 2016, p. 76

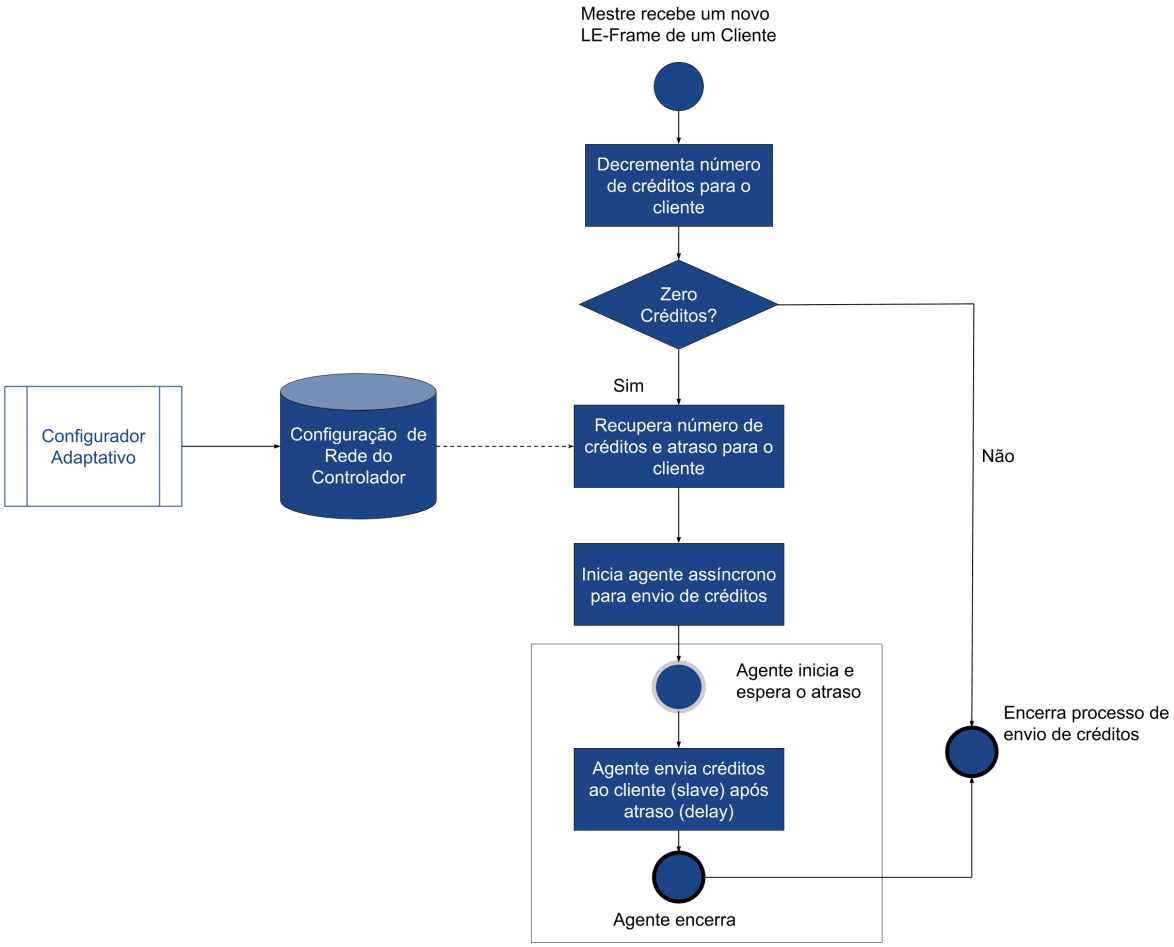


de uma estratégia de "pressão reversa" ou *backpressure*. Essa estratégia faz com que as aplicações e serviços nos clientes recebam alertas informando que o meio não está disponível naquele momento, com isso, permitindo que as mesmas sejam capazes de lidar com tal situação antes do envio de novos dados ao meio.

Por meio dessa definição de estratégia de controle, foi proposto um novo controlador por Santos [31] que faz uso de regras de controle geradas e armazenadas por um *Configurador Adaptativo*.

Esse modelo generalizado para controle de fluxo baseado em créditos torna-se base para a aplicação de diferentes modelos de controle executados pelo *Configurador Adaptativo*. Foi

Figura 2.3: Fluxo de controle baseado em créditos com regras e agentes de distribuição.
Fonte: Santos [31], 2016, p. 77



utilizado para o estudo de caso deste trabalho o Modelo de controle com uma distribuição simples de créditos entre os clientes.

Controle com Distribuição Simples de Créditos

Com o objetivo de evitar que a soma da taxa de transmissão utilizada por todos os clientes de um mestre ultrapasse o limite $maxTx$, que foi definido como parâmetro de entrada no *Configurador Adaptativo*, o modo de controle mais simples é dividir $maxTx$ igualmente entre todos os dispositivos conectados.

O processo de controle com distribuição simples é executado e atualizado toda vez que um evento de conexão ou desconexão de um novo dispositivo cliente é identificado. Ao receber esse evento, o *Configurador Adaptativo* executa o algoritmo de distribuição simples de

créditos (SCD). Esse algoritmo tem como entrada os parâmetros $maxTx$[2], MTU[3] e $xCrs$[4], além da lista de dispositivos conectados $AllNodes$. A saída do algoritmo é uma lista para cada cliente n com o valor da taxa de transmissão nTx, período de atraso para o envio de créditos $Del[n]$ e o número de créditos para enviar a cada interação $Crs[n]$. Esses valores são armazenados no controlador de rede, e utilizados pelo controlador de fluxo apresentado na Figura 2.3.2.

Neste trabalho foi utilizado o protocolo de controle de fluxo de dados com distribuição simples de créditos para o Bluetooth Low-Energy proposto por Santos [31] para realizar o estudo de caso do método proposto nesta dissertação.

[2]A taxa máxima de dados que o dispositivo controlador BLE suporta.

[3]O tamanho máximo em *bytes* de cada *LE-Frame* utilizado pelo nó mestre.

[4]O número mínimo de créditos que devem ser enviados a um cliente por interação.

Capítulo 3

Trabalhos Relacionados

Esse capítulo apresenta os principais trabalhos que aplicam a técnica de Simulação e Verificação Formal por *Model Checking*, que se enquadram no objetivo deste trabalho. Alguns dos trabalhos relacionados com a técnica de simulação e verificação formal por *model checking* são apresentados a seguir.

Galvão [18] que, na sua dissertação, propõe um conjunto de modelos de alguns blocos funcionais da norma IEC 61.131-3 visando a simulação e a verificação formal da especificação de comando de um sistema mecatrônico. A modelagem destes blocos funcionais é feita utilizando TA (*Timed Automata*) e o software de simulação e verificação (UPPAAL), que permite simular e realizar *model checking* sobre modelos em TA. O foco proposto pelo autor foi a verificação formal por *Model Checking* utilizando formalismos que suportam considerações de tempo, além de perseguir a verificação de controladores lógicos programáveis industriais e, dessa forma, a proposta teórica não se adéqua ao fim proposto nesse trabalho, que não utiliza o tempo.

Stefani [35], em sua dissertação, escreve sobre o modelo de máquina de estado abstrata (ASM - *Abstract State Machine*), utilizado para especificação formal de sistemas com alto grau de abstração e rigor matemático, podendo utilizar uma linguagem baseada no modelo ASM para escrever uma especificação em alto nível ou modelo básico. Esse modelo pode ser especificado para a ferramenta NuSMV possibilitando a verificação formal e automática da implementação. Este trabalho, embora utilize outro tipo de linguagem baseada no modelo ASM, apresenta exemplos do uso da ferramenta NuSMV que auxilia no embasamento utilizado nesta dissertação.

Fernandes [16], em sua dissertação, descreve um método, utilizando a *Unified Modeling Language* (UML), para realizar a verificação automática dos diagramas de atividade, estado e sequência da UML. O arcabouço proposto tem como objetivo fornecer uma abordagem para realizar a tradução dos diagramas de atividade, estado e sequência para linguagem formal de entrada do verificador formal NuSMV, automatizando o processo de tradução dos diagramas para a linguagem formal capacitando-se a fornecer um conjunto de validações pré-definidas que são utilizadas para verificar os diagramas. Esse trabalho, na medida que se propõe a explanar sobre a tradução da linguagem UML permitindo a alimentação do verificador formal NuSMV, sem utilizar uma modelagem formal, contribuiu para entender alguns caminhos que podem ser trilhados para conseguir esse propósito.

Ferreira [17], em sua dissertação, realiza um pesquisa bibliográfica sobre as principais técnicas para automatizar a verificação de sistemas que podem ser modelados máquinas de estado finitas (MEF), particularmente as que se enquadram na denominação de verificação de modelos (*Model Checking*). Este trabalho apresenta a parte teórica muito semelhante ao apresentado por Vasconcelos [37] e, dessa forma, foi utilizado com o objetivo de complementar o trabalho proposto por Vasconcelos.

Vasconcelos [37], em sua dissertação, descreve um método para a verificação formal automática de programas a partir de modelos reduzidos gerados pela análise de **múltiplos traços** de execuções do código. Este método teve por objetivo auxiliar os programadores na detecção de erros introduzidos durante a implementação do código fonte do programa. O método proposto pelo autor consiste em seis etapas. A primeira etapa do método consiste na instrumentação do código fonte do programa que será verificado, com a finalidade de coletar e armazenar a transição de estados das variáveis durante execuções do programa em um arquivo de histórico de execução, denominado **log**. A segunda etapa do método consiste na execução do código fonte de um sistema para limpeza do para-brisa instrumentado, para a coleta dos traços e criação do arquivo de **log**. A terceira etapa do processo é a aplicação de um algoritmo para o tratamento dos traços e a geração de um modelo reduzido do programa. Na quarta etapa, o modelo reduzido obtido é formatado para a semântica da ferramenta de verificação formal NuSMV. Na quinta etapa, o verificador de modelos é executado para confrontar as especificações e o modelo reduzido formatado. O verificador de modelos é executado por meio de uma ferramenta de verificação de modelos que implementa uma

varredura exaustiva. Essa varredura tem por objetivo encontrar um contraexemplo que negue a especificação. Caso a especificação não seja satisfeita, o verificador de modelos apresentará um contraexemplo, ou seja, uma sequência de transição de estados (**traços**) que levou a falha. Por fim, na última etapa, a falha deve ser analisada e rastreada no código fonte, o que auxiliará na correção do código. O processo deve ser reiniciado para a verificação de novas falhas. A validação desse método de verificação automática de programas é apresentado em um estudo de caso. O método proposto por Vasconcelos se destaca por apresentar uma automatização da implementação, visto que os traços das execuções, a construção do modelo reduzido do programa e a formatação para a linguagem da ferramenta NuSMV ocorrem de forma automática. Desta forma, a complexidade de construção do modelo para a ferramenta NuSMV é ocultada, sendo necessário apenas a descrição das especificações do programa em LTL ou CTL de acordo com a semântica do NuSMV. O método proposto pelo autor é a base do método proposto neste trabalho, modificando-se as etapas de geração do modelo, a instrumentação do código e a geração das especificações do sistema. O método proposto neste trabalho diferencia-se, também, por criar automaticamente os invariantes e as especificações do sistema na linguagem CTL de forma automática.

Capítulo 4

Apresentação do Método Proposto

Neste capítulo, introduz-se o método para a verificação formal automática de protocolos de controle de fluxo a partir do modelo gerado com as informações provenientes dos traços de execução. De acordo com o que foi apresentado no Capítulo 2, neste trabalho será aplicada uma técnica de verificação formal, o *model checking*, utilizando o NuSMV.

Conforme apresentado na fundamentação teórica, o processo de verificação por *model checking* utiliza o modelo formal gerado do sistema que será verificado. O sistema que foi verificado neste trabalho é composto por um servidor e vários clientes. O servidor realiza o controle de créditos disponibilizados para cada cliente. O controlador desses créditos emite uma **log** para o núcleo do servidor. O **log** gerado após a execução do controlador foi a base utilizada para gerar o modelo do sistema.

A modelagem do programa do sistema de créditos será realizada utilizando a metodologia proposta por Vasconcelos [37], pois a obtenção do modelo requer um conhecimento específico na linguagem da ferramenta de modelagem e nos fundamentos teóricos da verificação de modelos e, mesmo com o conhecimento necessário a criação e manutenção de modelos não é uma tarefa trivial [14, 28].

A metodologia proposta por Vasconcelos [37] consiste em cinco etapas: (i) instrumentação do sistema controlador de fluxo que será verificado; (ii) execução do controlador de fluxo instrumentado; (iii) tratamento dos traços e geração de um modelo reduzido do programa; (iv) formatação do modelo para a semântica de verificação formal NuSMV; e (v) verificação formal do Modelo gerado.

Este capítulo apresenta o método proposto. O detalhamento do mesmo e suas etapas

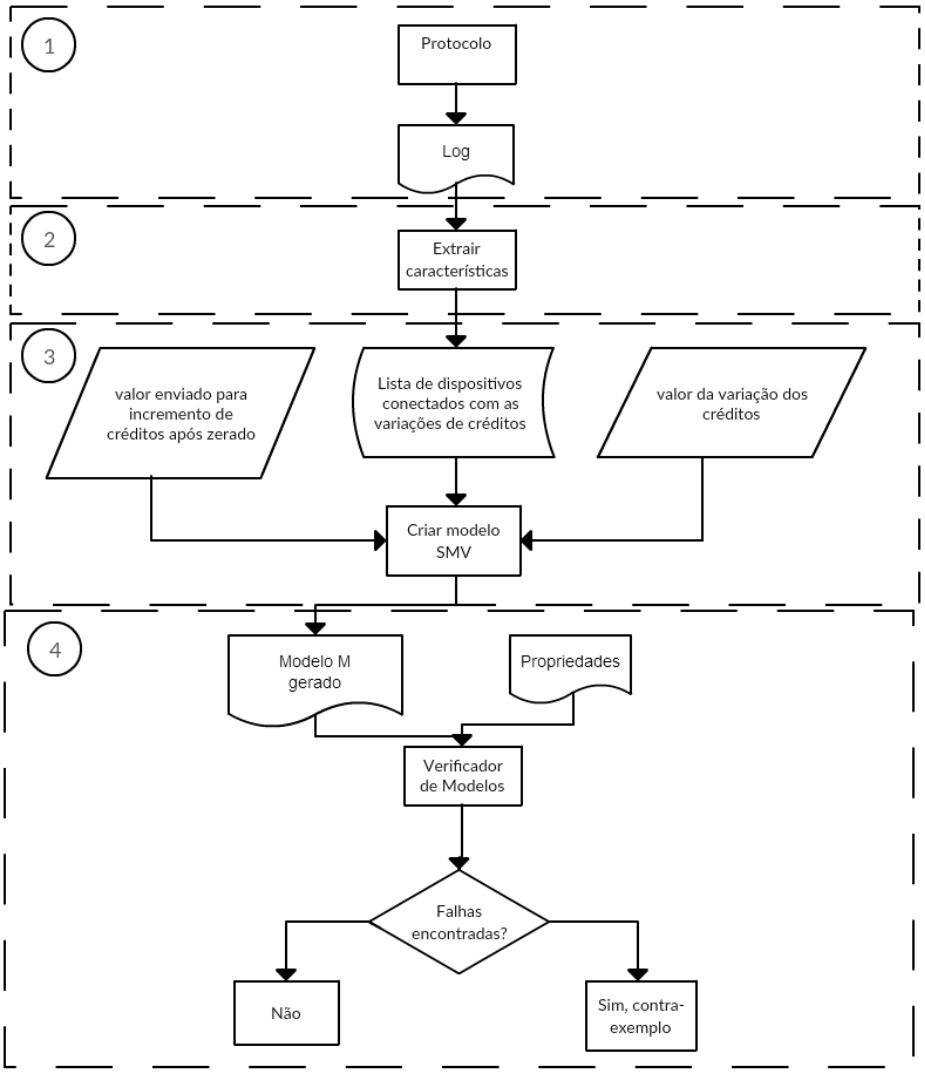
serão descritos detalhadamente no próximo Capítulo.

Por fim, serão apresentados os resultados obtidos neste método proposto na Seção 5.7.

4.1 Descrição do Método Proposto

O método proposto no presente trabalho é ilustrado na Figura 4.1 e detalhado nas Subseções subsequentes. A ideia do método é prover suporte à verificação formal de protocolos de controle de fluxo adaptativo para gateways bluetooth low-energy.

Figura 4.1: Método Proposto.



O método proposto pode ser dividido em quatro macro etapas: (i) obtenção do **log** de execução do protocolo; (ii) extração das características do protocolo; (iii) criação do Modelo

SMV do protocolo; e (iv) verificação formal do Modelo gerado. Estas etapas serão descritas a seguir.

4.1.1 Obtenção do log de execução do protocolo

Esta etapa do método, refere-se ao item 1 na Figura 4.1. Para obtenção do **log** ou **traços** de execução se faz necessária a execução do protocolo para a coleta dos traços provenientes do núcleo do *Gateway*. Os **traços** possuem as transições dos dados necessárias para a geração das especificações que serão checadas no modelo. Os **traços** coletados são enviados para um arquivo de **log** para serem utilizados na próxima etapa.

4.1.2 Extração das características do protocolo

Esta etapa do método, refere-se ao item 2 na Figura 4.1. Para a extração das características do protocolo se faz necessária a verificação do arquivo de log para verificar quais traços detêm as transições dos dados necessárias para a geração das especificações para a verificação formal. Após isso, as informações são armazenadas em uma lista que armazena as informações pertinentes da classe *Device*. Após as características coletadas e agrupadas, a etapa seguinte será a criação do modelo SMV.

4.1.3 Criação do Modelo SMV do protocolo

Esta etapa do método, refere-se ao item 3 na Figura 4.1. Nesta etapa do método, após a extração dos valores que serão checados na verificação formal será realizada a geração do modelo SMV, baseada no modelo Cliente/Servidor ensinado por Clarke [9], com os valores das características passadas como parâmetro. Esta etapa tem como resultado um Modelo M escrito em **.smv* que possui o modelo de Kripke [22] especificado em SMV e as especificações que serão checadas na próxima etapa.

4.1.4 Verificação formal do Modelo gerado

Esta etapa do método, refere-se ao item 4 na Figura 4.1. Nesta etapa do método, após criação do Modelo, o mesmo é executado na ferramenta *NuSMV* para a verificação formal do

modelo. Dois tipos de saídas podem ser esperadas: uma saída com todas as especificações "*true*"ou uma saída com uma ou mais especificações "*false*"e seus contraexemplos associados.

Essas são as macro etapas realizadas no método proposto neste trabalho.

4.2 Conclusão

Neste capítulo foi descrito o método proposto, bem como suas etapas de funcionamento. O método proposto foi dividido em quatro macro etapas: obtenção do **log** de execução do protocolo, extração das características do protocolo, criação do Modelo SMV do protocolo e verificação formal do Modelo gerado. No método proposto foram apresentadas uma automatização da construção do modelo reduzido de um protocolo de controle de fluxo de dados e uma formatação para a linguagem da ferramenta NuSMV, gerando um suporte à verificação formal destes protocolos.

No capítulo seguinte é apresentado um caso de estudo com a utilização do método e é descrito detalhadamente todos os passos para a aplicação do método proposto utilizando o protocolo de controle de fluxo adaptativo para gateways bluetooth low-energy.

Capítulo 5

Estudo de Caso – Aplicação do Método Proposto utilizando Protocolo de controle de fluxo adaptativo para gateways bluetooth low-energy

Neste capítulo um estudo de caso é usado para validar o método proposto. O protocolo utilizado foi proposto por Santos [31] cuja a finalidade é o controle de fluxo de dados, a partir de informações provenientes do núcleo do *Gateway*.

Seguindo a metodologia proposta por Vasconcelos [37], a primeira etapa do método consiste na instrumentação do sistema controlador de fluxo que será verificado, com a finalidade de coletar e armazenar a transição de estados das variáveis durante execuções do programa em um arquivo de histórico de execução, ao qual denominaremos de **log**. Este conjunto de transições em uma execução, traços ou caminhos, serão denominados **traços** a partir de agora. Esta etapa foi executada por Santos [31] em seu trabalho e não foi utilizada no método proposto.

A segunda etapa do processo consiste na execução do controlador de fluxo instrumentado, para a coleta dos **traços** e criação do arquivo de **log** que é apresentado na Seção 5.1. Uma etapa subsequente a esta, prevista no modelo de Vasconcelos [37], seria aplicação de um algoritmo para o tratamento dos **traços** e a geração de um modelo reduzido do programa. No caso deste trabalho não foi necessária pois a quantidade de **traços** associada é pequena.

Os **traços** foram coletados automaticamente e as **propriedades** do modelo foram identificados manualmente disponibilizando-as para a próxima etapa. Essa etapa é descrita em mais detalhes na Seção 5.2.

Na terceira etapa, que contém o núcleo de nossa proposta do estudo de caso, o modelo obtido na etapa anterior é formatado para a semântica de verificação formal NuSMV, que foi descrita na fundamentação teórica. Essa terceira etapa é descrita em mais detalhes na Seção 5.3.

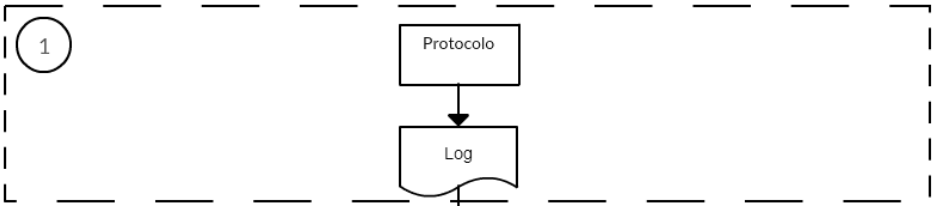
Na quarta etapa, o verificador de modelos é executado para confrontar as especificações e o modelo formatado. Em caso de verificar que o modelo formatado não satisfaz as especificações, o verificador de modelos apresentará um contraexemplo, ou seja, uma sequência de transição (**traços**) que levou a falha. Ainda nesta etapa, foram acrescentadas mudanças nas características do modelo em sua geração, para induzir falhas e provar a consistência do método na geração de método. Essa quarta etapa é descrita em mais detalhes na Seção 5.4.

Uma última etapa foi acrescentada para uma verificação fim-a-fim do método proposto. Uma propriedade foi modificada desde o controlador de fluxo até a geração do modelo para comprovar que após essa mudança o método manterá a consistência. Esta quarta etapa é descrita em mais detalhes na Seção 5.5.

5.1 Execução do Sistema Computacional de Controle de Fluxo Adaptativo para Gateways Bluetooth Low-Energy Instrumentado

Esta primeira etapa se refere a etapa 1 do método proposto, obtenção do log de execução do protocolo.

Figura 5.1: Etapa 1 - Método Proposto



Como descrito anteriormente, no capítulo 2, para a execução do controlador de fluxo

instrumentado faz-se necessário realizar a inicialização de todos os dispositivos bluetooth conectados ao servidor, desconectando-os e reconectando-os, além de garantir a supressão do conteúdo anteriormente armazenado no *dmesg*[1]. Essas ações permitem, com a reinicialização do arquivo **log** e dispositivos conectados ao servidor, que o arquivo log.txt receba os **traços** de todos os passos realizados pelos dispositivos clientes, inclusive a parte relativa à inicialização dos dispositivos. Os **traços** são provenientes do núcleo do *Gateway* e possuem as transições dos dados que serão verificadas formalmente.

Dessa forma, procedeu-se a desconexão de todos os dispositivos bluetooth conectados ao servidor através do comando **echo "disconnect A4:02:B9:01:AE:8B 1» /sys/kernel/debug/bluetooth/6lowpan_control** para todos os dispositivos conectados ao servidor. Posteriormente foi realizada a ativação do módulo de bluetooth_6lowpan do servidor.

Em seguida, realizou-se a conexão de todos os dispositivos bluetooth, que serão clientes ou *slaves*, ao servidor através do comando **echo "connect A4:02:B9:01:AE:8B 1» /sys/kernel/debug/bluetooth/6lowpan_control**, para cada cliente.

O procedimento, anteriormente descrito, permitiu que o arquivo **log** recebesse todos os **traços** do programa fonte decorrentes de sua lógica de inicialização de conexão com os dispositivos bluetooth clientes e, também, os **traços** da lógica de execução normal, que controlam o acesso dos dispositivos ao servidor para realizar a transmissão de dados.

O passo seguinte foi a coleta dos **traços** de execução ou **log**, onde se utilizou o dmesg. De forma a garantir que todos os registros anteriormente armazenados fossem apagados executou-se o comando **dmesg –clear**, e em seguida procedeu-se o armazenamento dos **traços** no arquivo log.txt por meio do comando **dmesg –w > log.txt**.

Anteriormente foram preparados os dispositivos bluetooth e o arquivo log.txt para receber todos os **traços** decorrentes da execução deste programa.

O próximo passo constitui-se na preparação para execução do programa de controle de acesso de dispositivos bluetooth, de forma que cada passo ou estado da lógica contida no programa de controle de acesso pudesse ser colhido e armazenado no arquivo log.txt.

Assim, procedeu-se a execução do controlador de fluxo de dados, proposto por Santos [31] e detalhado na Seção 2.3.2, para ativar o envio dos **traços** que serão utilizados a

[1]É um comando que é utilizado para examinar ou controlar todas as mensagens do núcleo, por padrão ele exibe todas as mensagens do núcleo do sistema operacional.

posteriori.

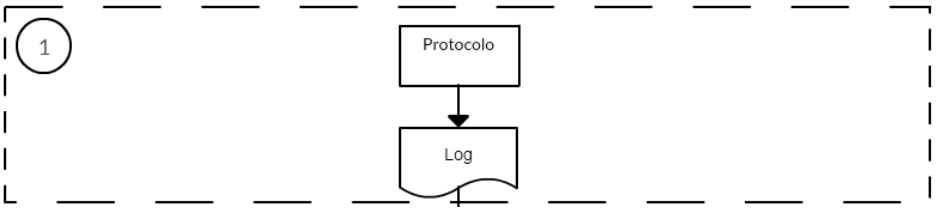
De forma a permitir o início do envio de dados dos dispositivos bluetooth cliente para o servidor, será necessário carregar e habilitar o uso do módulo **bluetooth_6lowpan**. Logo após a ativação é modificado estado da conexão bluetooth para que outros dispositivos possam visualizá-lo e criar uma conexão com o mesmo, utilizando o comando **hciconfig hci0 leadv**.

Depois que os clientes ativam seus módulos de bluetooth, o servidor irá abrir conexão bluetooth com os mesmos. Utilizando a conexão ipv6, anteriormente aberta, inicia-se o envio de dados ao servidor, permitindo ao protocolo instrumentado a realização do tratamento do envio de dados dos dispositivos clientes através do controle de créditos associados a cada dispositivo. Para ativar a variação dos créditos foi executado no cliente o comando **ping6**[2] **-I bt0**[3] **-s 210**[4] **fe80::21a:7dff:feda:7125**[5]. Dessa forma, o **log** terá as variações nos estados necessárias para a verificação formal.

5.2 Verificação do log gerado após a execução

Esta etapa está contida, também, na etapa 1 do método proposto, obtenção do log de execução do protocolo.

Figura 5.2: Etapa 1 - Método Proposto



Após a execução do protocolo, o **log** gerado será analisado para identificar os **traços** que devemos coletar para realizar a verificação formal do modelo. Com a captura e armazenamento, no arquivo log.txt, dos **traços** (estados) associados a execução do controlador de fluxo de dados, descritos anteriormente, nesta seção delineia-se os critérios de seleção dos

[2]ping6 é a versão IPv6 do ping, que é um programa que possibilita o usuário verificar se um endereço de IP existe e pode aceitar requisições.

[3]Nome da conexão bluetooth aberta com o servidor.

[4]Tamanho específico do pacote que será enviado na requisição.

[5]Ipv6 do servidor.

traços colhidos que serão submetidos à verificação formal. De forma a tornar mais didática a discussão, gerou-se um arquivo log.txt com apenas dois clientes que será descrito a seguir, Código Fonte 5.1.

No Código fonte 5.1 apresenta-se um exemplo de **log**, salvo em um arquivo, gerado após a execução do servidor com dois clientes. Nesse trecho do arquivo de **log** observa-se que a mensagem enviada pelo núcleo mostra a variação nos créditos do dispositivo de MAC 00:1a:7d:da:71:0b conectado ao servidor: "**,DF01,cflow,addr 00:1a:7d:da:71:0b,rx_credits 15->14**". Os créditos são decrementados um a um, e quando os créditos do dispositivo são zerados o núcleo envia uma outra mensagem informando que o dispositivo de MAC 00:1a:7d:da:71:0b estava com créditos zerados e, de acordo com a lógica, receberá mais 2 créditos para continuar executando suas ações. Visualiza-se, também, na linha 6 no trecho de Código 5.2 a mensagem de novos créditos enviada pelo núcleo: "**,DF01,newcredit,addr 00:1a:7d:da:71:0b,returning 2,total 2**".

Código Fonte 5.1: Exemplo do **log** gerado pelo servidor

```
1  [ 1356.951904] ,DF01,cflow,addr 00:1a:7d:da:71:0b,rx_credits 16->15
2  [ 1356.951905] ,DF01,skblen,addr 00:1a:7d:da:71:0b,skb->len 42
3  [ 1356.951907] ,DF01,llparam,addr 00:1a:7d:da:71:0b,interval 56,latency
       0,timeout 42
4  [ 1356.951908] Start of new SDU. sdu_len 40 skb->len 40 imtu 65535
5  [ 1357.300768] conn df22bd00 len 27 flags 0x2
6  [ 1357.300772] Start: total len 66, frag len 27
7  [ 1357.301321] conn df22bd00 len 27 flags 0x1
8  [ 1357.301323] Cont: frag len 27 (expecting 39)
9  [ 1357.301849] conn df22bd00 len 12 flags 0x1
10 [ 1357.301851] Cont: frag len 12 (expecting 12)
11 [ 1357.301853] len 62, cid 0x0040
12 [ 1357.301870] chan df357000, len 62
13 [ 1357.301872] ,DF01,cflow,addr 00:1a:7d:da:71:0b,rx_credits 15->14
14 [ 1357.301873] ,DF01,skblen,addr 00:1a:7d:da:71:0b,skb->len 62
```

Após a verificação do **log** foram identificados os **traços** e as propriedades necessárias à verificação formal. Os **traços** selecionados foram: o de mudança de créditos e o de recebimento de novos créditos, descritos anteriormente.

Código Fonte 5.2: Trecho de Código que mostra o recebimento de novos créditos

```
1 [ 1476.187966] ,DF01,cflow,addr 00:1a:7d:da:71:0b,rx_credits 1->0
2 [ 1476.187967] ,DF01,skblen,addr 00:1a:7d:da:71:0b,skb->len 223
3 [ 1476.187968] ,DF01,llparam,addr 00:1a:7d:da:71:0b,interval 56,latency
    0,timeout 42
4 ...
5 [ 1476.835282] ,DF01,llparam,addr 00:1a:7d:da:71:0b,interval 56,latency
    0,timeout 42
6 [ 1476.835285] ,DF01,newcredit,addr 00:1a:7d:da:71:0b,returning 2,total 2
```

Os critérios de escolha dos traços, que estavam relacionados aos objetivos deste trabalho, foram a escolha dos **traços** que indicavam a mudança de créditos nos dispositivos bluetooth conectados, bem como o envio de novos créditos a estes dispositivos. A realização dessa escolha foi possível pela pequena quantidade de dispositivos associados, facilitando a visualização dos **traços** importantes para a verificação formal.

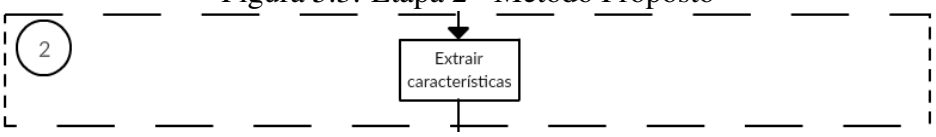
5.3 Geração do Modelo M

Nesta etapa do método, após a extração dos valores que serão checados na verificação formal será realizada a geração do modelo SMV, baseada no modelo Cliente/Servidor ensinado por Clarke [10], utilizando as características coletadas na etapa anterior.

5.3.1 Extração das características

Esta etapa se refere a etapa 2 do método proposto, extração das características do protocolo.

Figura 5.3: Etapa 2 - Método Proposto



Na fundamentação teórica, na seção 2.2, realizou-se uma breve exposição de como deve ser a preparação (modelagem) do programa que será verificado formalmente, de acordo com o padrão proposto pela ferramenta NuSMV. Assim, foi preparada uma entrada do NuSMV descrevendo as relações de transição com as relações de evoluções válidas no modelo do

sistema e, dessa forma, podem ser identificadas as possíveis configurações futuras do sistema a partir do seu estado inicial.

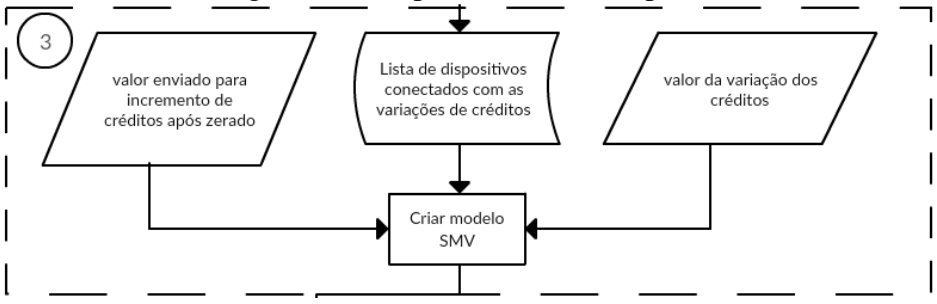
Para realizar a extração das características do modelo utilizou-se as informações contidas no arquivo de **log**, gerado pela execução do controlador de fluxo instrumentado. Para extrair as características foi utilizado um método, escrito em Java, que permitiu a identificação e memorização dos dispositivos conectados ao servidor e, para cada dispositivo conectado, foram identificadas e coletadas as variações dos créditos. O método pode ser visto no Apêndice A.1.

O método que realiza a extração das características do modelo, inicialmente, realiza a leitura do arquivo de **log**, linha 5. Após a leitura do **log**, todas as linhas foram verificadas. Em cada linha do **log** foi verificado se a linha possui o texto: **00:00:00:00:00:00 ->**, que indica que uma conexão foi aberta com algum dispositivo, na linha 9. Se a linha possui tal texto, é criada uma instância da classe *Device* utilizando o identificador(MAC) do dispositivo encontrado na linha checada. Foi verificado, também, se a linha possui o texto: **rx_credits**, que indica a variação nos créditos dos dispositivos conectados, na linha 17. Se a linha possui esse texto, foi verificado se já existe um dispositivo conectado com o identificador presente na linha, se não ele será adicionado na lista de dispositivos conectados. Após isso as variações de créditos são adicionadas na lista de variações na instância da classe *Device*.

5.3.2 Gerador do Modelo M

Esta etapa do estudo de caso se refere a etapa 3 do método proposto, criação do Modelo SMV do protocolo.

Figura 5.4: Etapa 3 - Método Proposto



Os dados coletados, pelo método anteriormente exposto, geram entradas para outro método, o criadorDeModelosSMV também em Java, que é o ModeloM. As entradas são: a lista

dos dispositivos conectados, o número de créditos que o modelo enviará após zerados os créditos e o número que será decrementado de acordo com o uso dos créditos, que já foram coletados pelo método anterior. Dessa forma o programa retornará o nome do arquivo gerado no formato *.smv que representa o modelo reduzido do controlador de fluxo instrumentado no formato da ferramenta NuSMV.

Conforme visto na seção 2.2, o NuSMV utiliza a declaração MODULE que permite a criação de divisões onde é permitido a declaração de todas as variáveis, inclusive as de estado e os eventos que são declarados booleanos e, cada módulo, pode conter a regra de transição dos estados e as especificações das propriedades. Desta forma, foram criadas três seções MODULE: Uma modelando os clientes; uma que modela o servidor; uma que modela o programa controlador da comunicação dos bluetooth, a main. Além disso, foram determinadas as regras invariantes que devem ser verificadas e as especificações que verificam as variações de créditos de cada dispositivo.

A parte inicial do programa coleta os dados oriundos do procedimento anteriormente executado e inicia a escrita de dados no arquivo modelo.smv, modelo que alimenta a ferramenta NuSMV. Em seguida detalha-se todas as seções.

Um modelo base foi gerado em smv, previamente, para verificar quais mudanças são necessárias de acordo com cada tipo de execução. Foi observado que são necessárias mudanças apenas no MODULE do Servidor, nas variáveis internas creditosParaEnviar e valorDoIncrementoDeCreditos. O modelo base é apresentado no Código 5.3. Dessa forma, a geração do modelo se realiza a partir de um modelo base, variando apenas estas variáveis. As linhas que serão alteradas estão marcadas, linhas 34 e 35. O MODULE main e as especificações sempre serão diferentes, quando varia-se o tipo de execução do protocolo.

Código Fonte 5.3: Modelo base para criar o Modelo SMV

```
1
2 MODULE cliente (autorizado, creditoNovo, valorDoIncrementoDeCreditos,
      idCliente)
3 VAR
4    estadoDosCreditos:{Up, Down};
5    creditos : 0 .. 100;
6    conectados : boolean;
7    id : 0 .. 100;
```

```
8     state : {esperando, conectado};
9
10  ASSIGN
11    init(state) := esperando;
12    init(creditos) := 0;
13    init(estadoDosCreditos) := Up;
14    id := idCliente;
15    next(state) :=
16     case
17       state=esperando & autorizado : conectado;
18       state=conectado & !autorizado : esperando;
19       TRUE : state;
20     esac;
21
22  TRANS
23    (estadoDosCreditos = Up & creditos != 0  & next(creditos) = creditos +
          valorDoIncrementoDeCreditos) |
24    (estadoDosCreditos = Down & next(creditos) = creditos -
          valorDoIncrementoDeCreditos) |
25    (estadoDosCreditos = Up & creditos = 0 & next(creditos) = creditoNovo)
26
27  MODULE servidor (req)
28  VAR
29    state : {ocioso, ativo};
30    autorizando : boolean;
31    creditosParaEnviar : 0 .. 100;
32    valorDoIncrementoDeCreditos : 0 .. 100;
33
34  ASSIGN
35    creditosParaEnviar := 5;
36    valorDoIncrementoDeCreditos := 1;
37    next(state) :=
38     case
39       state=ocioso & req : ativo;
40       state=ativo & !req : ocioso;
41       TRUE : state;
42     esac;
```

```
43    autorizando := (state = ativo);
```

A primeira seção, a modelagem base do modelo, escreve no modelo base as características: creditosParaEnviar e incrementoDeCreditos. Esta parte, pode ser vista no Apêndice no Código A.2 , resultou na criação do MODULE cliente e servidor no arquivo modeloBase.smv.

No Código 5.4 é mostrado o Código gerado por essa etapa no modelo SMV, sendo esta uma parte da entrada da ferramenta NuSMV.

Código Fonte 5.4: Entrada da ferramenta NuSMV - Parte I

```
1  MODULE cliente (autorizado, creditoNovo, valorDoIncrementoDeCreditos,
       idCliente)
2  VAR
3    estadoDosCreditos:{Up, Down};
4    creditos : 0 .. 100;
5    conectados : boolean;
6    id : 0 .. 100;
7    state : {esperando, conectado};
8
9  ASSIGN
10   init(state) := esperando;
11   init(creditos) := 0;
12   init(estadoDosCreditos) := Up;
13   id := idCliente;
14   next(state) :=
15    case
16     state=esperando & autorizado : conectado;
17     state=conectado & !autorizado : esperando;
18     TRUE : state;
19    esac;
20
21 TRANS
22   (estadoDosCreditos = Up & creditos != 0  & next(creditos) = creditos +
       valorDoIncrementoDeCreditos) |
23   (estadoDosCreditos = Down & next(creditos) = creditos -
       valorDoIncrementoDeCreditos) |
24   (estadoDosCreditos = Up & creditos = 0 & next(creditos) = creditoNovo)
```

```
25
26 MODULE servidor (req)
27 VAR
28    state : {ocioso, ativo};
29    autorizando : boolean;
30    creditosParaEnviar : 0 .. 100;
31    valorDoIncrementoDeCreditos : 0 .. 100;
32
33 ASSIGN
34    creditosParaEnviar := 5;
35    valorDoIncrementoDeCreditos := 1;
36    next(state) :=
37     case
38       state=ocioso & req : ativo;
39       state=ativo & !req : ocioso;
40      TRUE : state;
41     esac;
42    autorizando := (state = ativo);
```

Na última seção é tratado o módulo main, que modela o programa que controla o acesso dos dispositivos ao servidor e adiciona as especificações que serão checadas pelo método. As variáveis são c1 (cliente 1), c2 (cliente 2) e s (servidor). Seguindo-se ao último modelo, especifica-se todas as invariantes. Esta parte é apresentada no Código A.3, no Apêndice, que gerou a parte final do arquivo modeloBase.smv, Código 5.5.

O resultado da etapa anterior é apresentado no Código 5.5. As especificações adicionadas se referem aos invariantes do modelo e às variações nos créditos. Para realizar a checagem do modelo foram adicionados invariantes e algumas especificações:

- INVARSPEC c1.creditos >= 0

  Esta especificação é para garantir que o valor dos créditos sempre serão acima de zero, representada nas linhas 7 e 10.

- INVARSPEC (s.state = ocioso -> (c1.state = esperando))

  Esta especificação é para garantir que quando o servidor estiver ocioso os clientes sempre ficarão com o estado esperando, representada nas linhas 8 e 11.

- SPEC AG(s.state = ativo -> AF(c1.state = conectado))

  Esta especificação é para garantir que quando o servidor estiver ativo os clientes sempre ficarão com o estado conectado para poder ocorrer o envio dos dados, representada nas linhas 9 e 12.

- INVARSPEC s.creditosParaEnviar >= 0

  Esta especificação é para garantir que o servidor sempre enviará créditos positivos para os clientes, representada na linha 13.

- INVARSPEC (c1.id != c2.id)

  Esta especificação é para garantir que os clientes sempre terão identificadores diferentes, representada na linha 14.

- SPEC AG((c1.creditos = 20 & c1.estadoDosCreditos = Down) -> AX ((c1.creditos = 19)))

  Esta especificação é para garantir que quando um cliente estiver com os créditos em 20, por exemplo, e o estadosDosCreditos estiver em *Down*, obrigatoriamente o próximo valor será 19 (mediante a especificação na variação de créditos), representada na linha 15.

- SPEC AG((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX ((c2.creditos = 5)))

  Esta especificação é para garantir que quando um cliente estiver com os créditos zerados e o estadosDosCreditos estiver em *Up*, obrigatoriamente o próximo valor dos créditos será 5 (mediante a especificação de incremento de créditos do modelo), representada na linha 24.

Código Fonte 5.5: Entrada da ferramenta NuSMV - Parte II

```
1 MODULE main
2 VAR
3    c1 : cliente(s.autorizando, s.creditosParaEnviar, s.
         valorDoIncrementoDeCreditos, 1);
4    c2 : cliente(s.autorizando, s.creditosParaEnviar, s.
         valorDoIncrementoDeCreditos, 2);
```

```
5    s : servidor(TRUE);
6
7  INVARSPEC c1.creditos >= 0
8  INVARSPEC (s.state = ocioso -> (c1.state = esperando))
9  SPEC AG(s.state = ativo -> AF(c1.state = conectado))
10 INVARSPEC c2.creditos >= 0
11 INVARSPEC (s.state = ocioso -> (c2.state = esperando))
12 SPEC AG(s.state = ativo -> AF(c2.state = conectado))
13 INVARSPEC s.creditosParaEnviar >= 0
14 INVARSPEC (c1.id != c2.id)
15 SPEC AG((c1.creditos = 20 & c1.estadoDosCreditos = Down)
16 -> AX ((c1.creditos = 19)))
17 SPEC AG((c1.creditos = 19 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 18)))
18 ...
19 SPEC AG((c1.creditos = 18 & c1.estadoDosCreditos = Down) ->
20 SPEC AG((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX ((c1.creditos
       = 5)))
21 ...
22 SPEC AG((c2.creditos = 9 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 8)))
23 SPEC AG((c2.creditos = 8 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 7)))
24 ...
25 SPEC AG((c2.creditos = 0 & c2.estadoDosCreditos = Up)
26 -> AX ((c2.creditos = 5)))
```

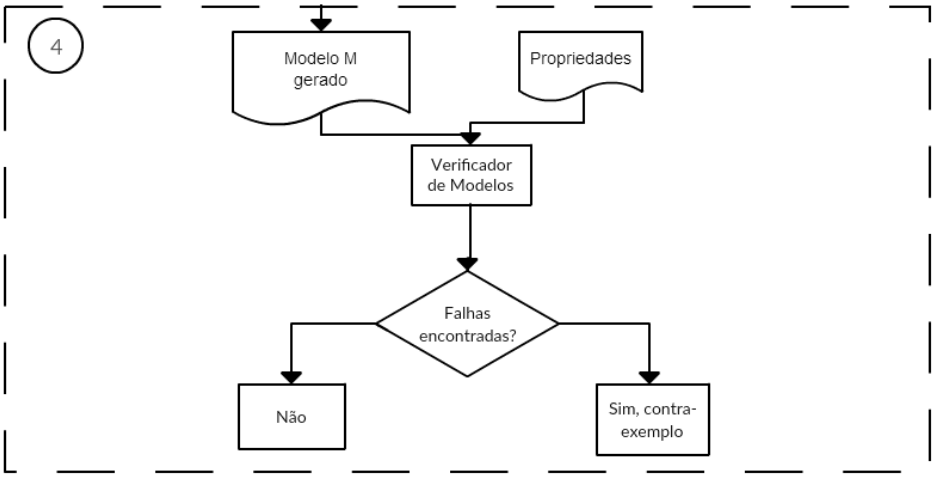
Após esta etapa, o modelo estará completo e pronto para servir de entrada no verificador de modelos.

5.4 Verificação do modelo

Nesta seção foram realizadas várias verificações de modelos, com variações no protocolo e na geração do modelo para garantir a consistência do método proposto neste trabalho.

Esta etapa do estudo de caso se refere a etapa 4 do método proposto, verificação formal do Modelo gerado.

Figura 5.5: Etapa 4 - Método Proposto



5.4.1 Verificação inicial

O verificador de modelos *NuSMV* foi executado para a verificação dos requisitos da especificação, utilizando o modelo gerado anteriormente. Conforme resultado apresentado no Código B.1 no Apêndice, o modelo gerado não indicou ocorrência de falha já que, todas as linhas checadas apresentaram o resultado *true*.

5.4.2 Verificação com falhas

Para testar as propriedades do Modelo gerado, foram realizadas algumas mudanças no método de geração do modelo *.smv* para garantir que o verificador encontrará falhas e com isso, garantir a veracidade da especificação gerada pelo método proposto.

Verificação no acréscimo no número de créditos

Primeiro, foram inseridas modificações na geração do modelo modificando o número de créditos que serão enviados para o cliente quando os seus créditos forem zerados. Inicialmente, no protocolo, o número utilizado é **2**, foi modificado na geração do modelo para que seja **3** para comprovar que o contraexemplo informará **2** como valor esperado.

Para alterar a geração do modelo é necessário, apenas, modificar o parâmetro do método *criadorDeModelosSMV*. O Código utilizado inicialmente pode ser visto no Código 5.6 e o alterado para a gerar a falha pode ser visto no Código 5.7.

Código Fonte 5.6: Método para criar o Modelo SMV inicial

```
1
2 public static void main (String[] args) throws java.lang.Exception {
3     List<Device> connectedDevices = extratorDeCaracteristicas(new File("
          log.txt"));
4     String filePath = criadorDeModelosSMV(connectedDevices, 2, 1);
5     System.out.println(filePath);
6 }
```

Código Fonte 5.7: Método para criar o Modelo SMV alterado

```
1
2 public static void main (String[] args) throws java.lang.Exception {
3     List<Device> connectedDevices = extratorDeCaracteristicas(new File("
          log.txt"));
4     String filePath = criadorDeModelosSMV(connectedDevices, 3, 1);
5     System.out.println(filePath);
6 }
```

O verificador de modelos *NuSMV* foi executado para verificar os requisitos da especificação que foram alterados, utilizando o modelo gerado após a alteração. Conforme o resultado apresentado no Código 5.8 que pode ser visto no Apêndice, o modelo gerado apresentou falhas.

Código Fonte 5.8: Verificação do modelo gerado após alteração do modelo

```
1
2 -- specification AG ((c1.creditos = 0 &
3 c1.estadoDosCreditos = Up) -> AX c1.creditos = 2)
4 is false
5 -- as demonstrated by the following execution sequence
6 Trace Description: CTL Counterexample
7 Trace Type: Counterexample
8   -> State: 1.1 <-
9     c1.estadoDosCreditos = Up
10    c1.creditos = 0
11    c1.conectados = FALSE
12    c1.id = 1
13    c1.state = esperando
```

```
14      c2.estadoDosCreditos = Up
15      c2.creditos = 0
16      c2.conectados = FALSE
17      c2.id = 2
18      c2.state = esperando
19      s.state = ativo
20      s.autorizando = TRUE
21      s.creditosParaEnviar = 3
22      s.valorDoIncrementoDeCreditos = 1
23    -> State: 1.2 <-
24      c1.creditos = 3
25      c1.state = conectado
26      c2.creditos = 3
27      c2.state = conectado
```

Podemos observar no resultado do verificador que a variação oriunda da execução do controlador de fluxo era de 0->2, com a modificação o modelo esperava que a variação no Código de créditos fosse de 0->3, o que não ocorreu. Podemos observar que no estado 1.1 o valor dos créditos de c1 estava zerado **c1.creditos = 0** e no estado 1.2 esperava 3 **c1.creditos = 3**, visto que o valor dos créditos para enviar no servidor estava 3 na especificação **s.creditosParaEnviar = 3**. Como a especificação **specification AG ((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX c1.creditos = 2)** foi gerada a partir da execução do Código original podemos comprovar que esta modificação no gerador ocasionou falha e ao voltarmos para a especificação original, o verificador não encontrou falhas.

Verificação da variação dos créditos

Primeiro foram inseridas mudanças na geração do método modificando o valor do decremento de créditos. Inicialmente, no protocolo o número utilizado é **1**, foi modificado o modelo para que seja **2** e foi comprovado que o contraexemplo informará que o valor esperado é **1**.

Para modificar a geração do modelo é necessário, apenas, modificar o parâmetro do método *criadorDeModelosSMV*. O Código utilizado inicialmente pode ser visto no Código 5.9 e o alterado para a gerar a falha pode ser visto no Código 5.10.

Código Fonte 5.9: Método para criar o Modelo SMV inicial

```
1
2 public static void main (String[] args) throws java.lang.Exception {
3     List<Device> connectedDevices = extratorDeCaracteristicas(new File("
          log.txt"));
4     String filePath = criadorDeModelosSMV(connectedDevices, 2, 1);
5     System.out.println(filePath);
6 }
```

Código Fonte 5.10: Método para criar o Modelo SMV alterado

```
1
2 public static void main (String[] args) throws java.lang.Exception {
3     List<Device> connectedDevices = extratorDeCaracteristicas(new File("
          log.txt"));
4     String filePath = criadorDeModelosSMV(connectedDevices, 2, 2);
5     System.out.println(filePath);
6 }
```

O verificador de modelos *NuSMV* foi executado para a verificação dos requisitos alterados, utilizando o modelo gerado após a alteração. Conforme resultado apresentado no Código 5.11, o modelo gerado apresentou falhas.

Código Fonte 5.11: Verificação do modelo gerado após alteração do modelo

```
1 -- specification AG ((c1.creditos = 8 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 7)  is
2  false
3 -- as demonstrated by the following execution sequence
4 Trace Description: CTL Counterexample
5 Trace Type: Counterexample
6   -> State: 1.1 <-
7     c1.estadoDosCreditos = Up
8     c1.creditos = 0
9     c1.conectados = FALSE
10    c1.id = 1
11    c1.state = esperando
12    c2.estadoDosCreditos = Up
13    c2.creditos = 0
```

```
14      c2.conectados = FALSE
15      c2.id = 2
16      c2.state = esperando
17      s.state = ativo
18      s.autorizando = TRUE
19      s.creditosParaEnviar = 2
20      s.valorDoIncrementoDeCreditos = 2
21    -> State: 1.2 <-
22      c1.creditos = 2
23      c1.state = conectado
24      c2.creditos = 2
25      c2.state = conectado
26    -> State: 1.3 <-
27      c1.creditos = 4
28      c2.creditos = 4
29    -> State: 1.4 <-
30      c1.creditos = 6
31      c2.estadoDosCreditos = Down
32      c2.creditos = 6
33    -> State: 1.5 <-
34      c1.estadoDosCreditos = Down
35      c1.creditos = 8
36      c2.estadoDosCreditos = Up
37      c2.creditos = 4
38    -> State: 1.6 <-
39      c1.estadoDosCreditos = Up
40      c1.creditos = 6
41      c2.creditos = 6
```

Podemos observar no resultado do verificador que a variação do decremento de créditos oriunda da execução do controlador de fluxo era de 1 em 1, como pode ser visto na especificação gerada pelo **log** na linha 1 que era 8 e depois foi para 7. Com a modificação o modelo esperava-se que a variação do decremento fosse de 2 em 2, pode-se observar na linha 20 o valor especificado no modelo para incremento de créditos, o que não ocorreu. Podemos observar que no estado 1.5, linha 33, o valor dos créditos de c1 estava em **8 c1.creditos = 8** e no estado 1.6, linha 40, esperava 6 **c1.creditos = 6**, uma vez que o valor do decremento

dos créditos estava 2 na especificação **s.valorDoIncrementoDeCreditos = 2**. Como a especificação **specification AG ((c1.creditos = 8 & c1.estadoDosCreditos = Down) -> AX c1.creditos = 7)** foi gerada a partir da execução do Código original podemos comprovar que esta modificação no gerador ocasionou falha e ao voltarmos para a especificação original, o verificador não encontrou falhas.

5.5 Verificação no acréscimo no número de créditos no Modelo e no Servidor

A modificação objetivou comprovar que ao modificar a especificação no servidor e na geração do modelo, o verificador não encontrará falhas, pois servidor e modelo atenderão as mesmas propriedades.

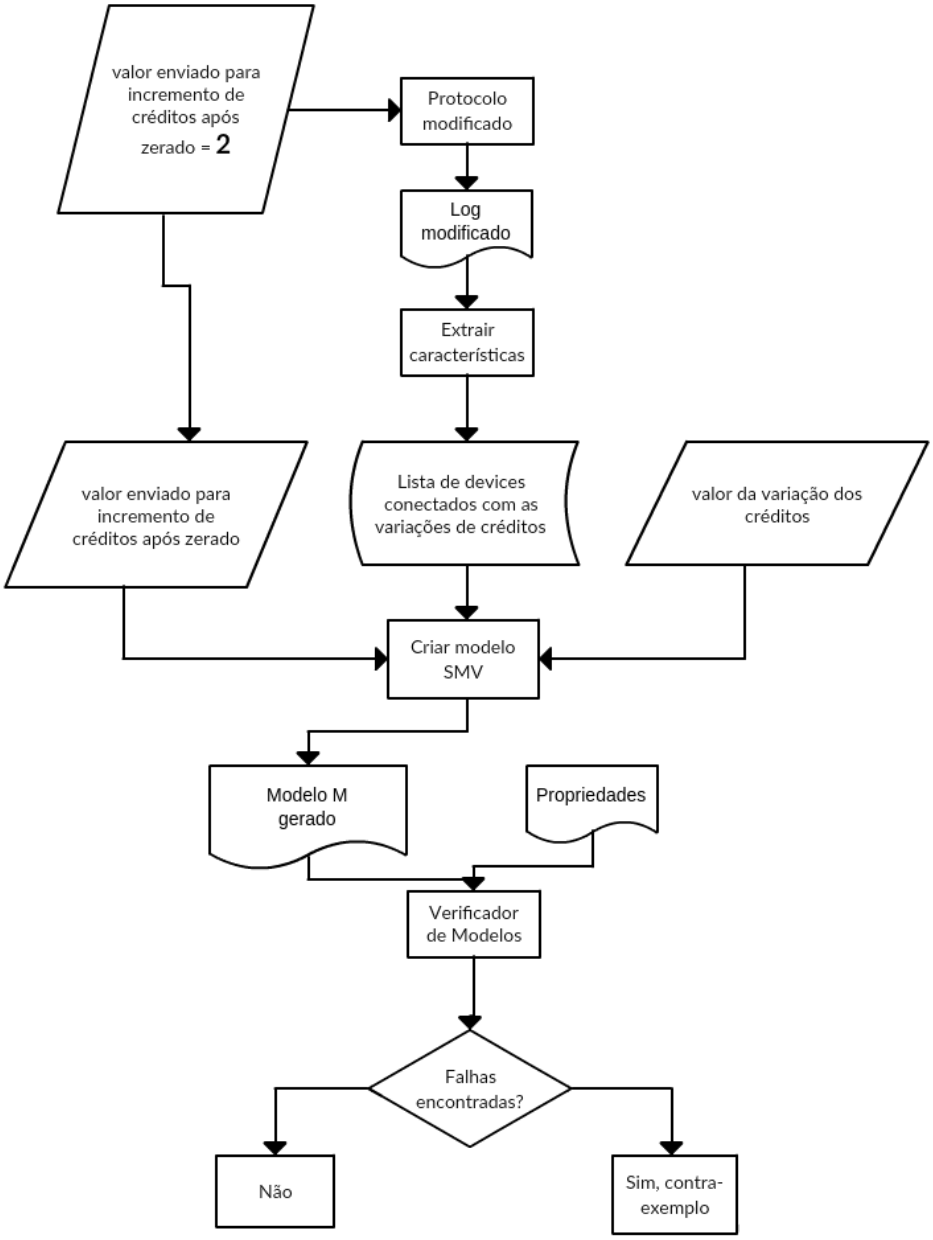
Inicialmente o protocolo foi definido para retornar **2** créditos quando um dispositivo, conectado, estivesse com zero créditos. O fluxo do método proposto nesta dissertação, utilizando o protocolo original, pode ser visto na Figura 5.6.

Para provar a aplicabilidade do método proposto nesta dissertação, o protocolo foi modificado para conseguir executar uma verificação formal com mudanças nas propriedades desde o controlador de fluxo. O fluxo do método, utilizando o protocolo modificado, pode ser visto na figura 5.7.

O valor na renovação de créditos no arquivo *blelistener.py* anteriormente era **2**, como pode ser visto na Figura 5.8. Este valor foi modificado para **5**, como pode ser visto na Figura 5.9, para gerar um novo arquivo de coleta de **traços** de execução do controlador de fluxo.

Após a modificação, o **log** apresentará o retorno de 5 créditos no traço, um exemplo de traço que contém essa propriedade é: **,DF01,newcredit,addr a4:02:b9:01:ae:8b,returning 5,total 5**, como pode ser visto na linha 5 no Código 5.12.

Figura 5.6: Fluxo do Método com o Protocolo Original



5.5.1 Verificação do log gerado após a execução

Após a execução do protocolo modificado, podemos verificar o **log** gerado com a característica modificada. O trecho do **log** que demonstra a mudança na característica pode ser visto no Código 5.12.

Figura 5.7: Fluxo do Método com o Protocolo Modificado

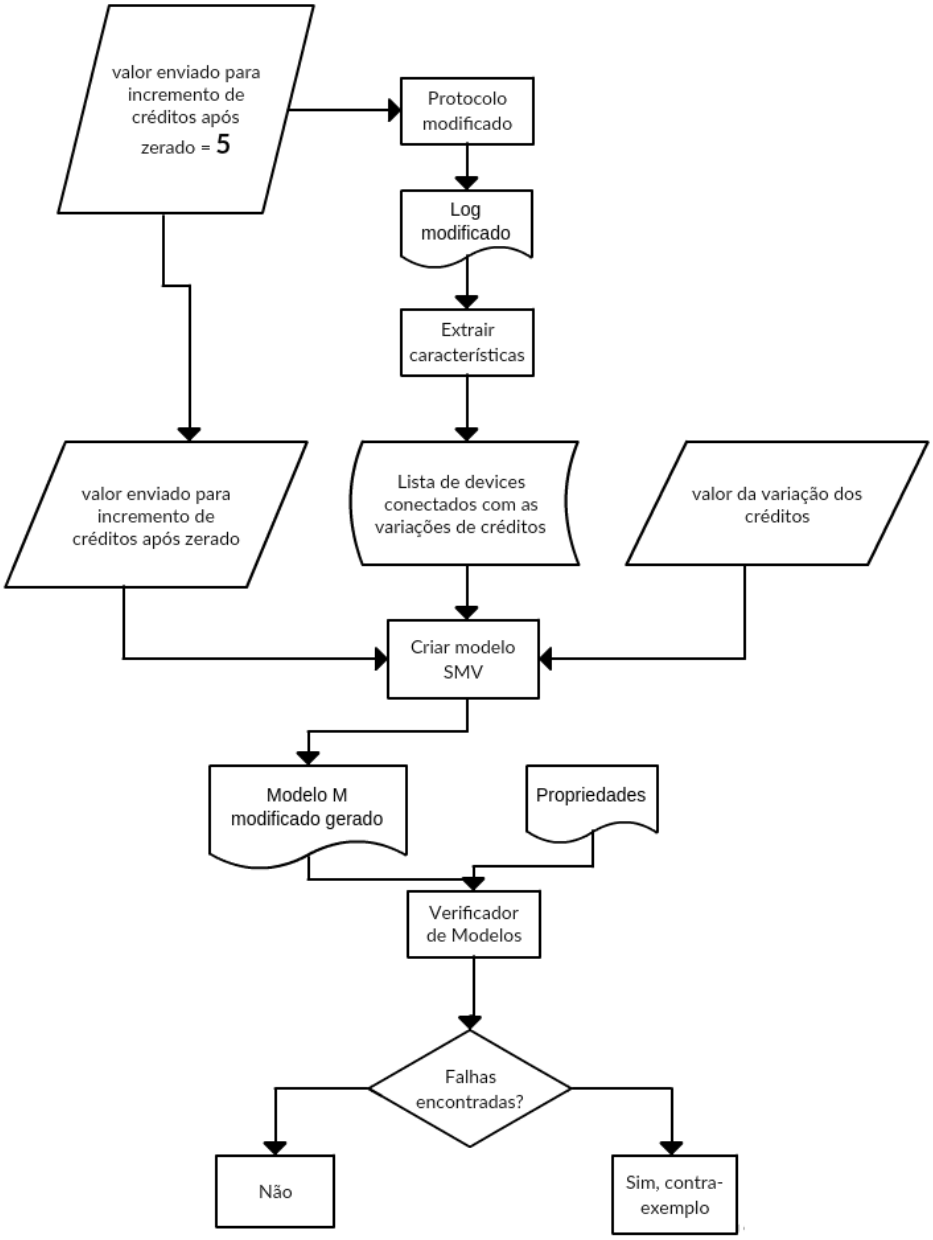


Figura 5.8: Característica inicial do protocolo

```
ubuntu@ubuntu-VirtualBox:~/PycharmProjects/blecontroller$ cat blelistener.py
import subprocess
import bleflowcontroller
import datetime
import _thread
import time

#this is the default number of returned credits. It is used to tune the controller
crs = 2
```

Código Fonte 5.12: Exemplo do **log** gerado pelo servidor após a modificação da característica no servidor

Figura 5.9: Característica modificada do protocolo

```
ubuntu@ubuntu-VirtualBox:~/PycharmProjects/blecontroller$ cat blelistener.py
import subprocess
import bleflowcontroller
import datetime
import _thread
import time

#this is the default number of returned credits. It is used to tune the controller
crs = 5
```

```
1 [ 1576.928107] chan dc9e3000, msg dd035b28, len 221
2 [ 1576.928108] chan dc9e3000 len 221
3 [ 1576.928112] chan dc9e3000, skb db720cc0 len 227 priority 0
4 [ 1577.732366] ,DF01,llparam,addr a4:02:b9:01:ae:8b,interval 56,latency
    0,timeout 42
5 [ 1577.732369] ,DF01,newcredit,addr a4:02:b9:01:ae:8b,returning 5,total 5
6 [ 1577.732371] conn c01be700, code 0x16, ident 0x03, len 4
7 [ 1577.732373] code 0x16
```

Após a característica modificada no controlador de fluxo e comprovada no **log** modificado, outro modelo foi gerado utilizando o método proposto para uma próxima verificação formal utilizando as propriedades modificadas.

5.5.2 Geração do Modelo com as propriedades modificadas

Foi inserida uma mudança na geração do método modificando a mesma propriedade que foi modificada no controlador de fluxo do protocolo, o número de créditos que serão enviados para o cliente quando os seus créditos forem zerados. Inicialmente, no protocolo o número utilizado é **2**, modificaremos o modelo para que seja **5** e comprovaremos que não será encontrado contraexemplo para o modelo.

Para alterar a geração do modelo é necessário, apenas, modificar o parâmetro do método *criadorDeModelosSMV*. O Código alterado para gerar o modelo pode ser visto no Código 5.13.

Código Fonte 5.13: Método para criar o Modelo SMV Modificado

```
1 public static void main (String[] args) throws java.lang.Exception {
2     List<Device> connectedDevices = extratorDeCaracteristicas(new File("
          logModificado.txt"));
3     String filePath = criadorDeModelosSMV(connectedDevices, 5, 1);
```

```
4       System.out.println(filePath);
5   }
```

5.5.3 Modelo M Gerado

No Código C.1, apresenta-se a representação do modelo M do programa no formato do NuSMV, construído a partir do **log** gerado e apresentado no Código fonte 5.1.

Verificação formal do modelo modificado

O verificador de modelos *NuSMV* foi executado, novamente, para a verificação das propriedades modificadas, utilizando o modelo gerado anteriormente. Conforme resultado apresentado no Código B.2, o modelo gerado não apresentou falhas.

Diante do resultado visto anteriormente, pode-se afirmar que o método utilizado mantém a consistência das propriedades, quando modificadas no controlador de fluxo e no gerador do modelo.

5.6 Verificação formal com Variação no Número de Clientes

Nesta seção foram executados vários testes para analisar o tempo de execução do protocolo de controle de fluxo, juntamente com a geração de seu modelo reduzido e a verificação formal do mesmo. Para realizar este experimento, inicialmente o protocolo foi executado com dois, com três e quatro clientes permitindo a análise da variação do tempo de execução do protocolo com a mudança do número de clientes. Cumprida essa etapa, foram realizadas as gerações de modelo, associadas a essas execuções do protocolo, onde foram medidos os tempos de execução para geração do modelo reduzido. Por fim, foram realizadas as verificações formais dos modelos gerados para medição do tempo de execução dos mesmos.

Em todas as execuções do protocolo foram enviados dados para o servidor com a mesma quantidade de pacotes, garantindo que esse parâmetro não influenciaria o experimento. Em todos os clientes foi executado o comando **ping6 -I bt0 -s 210 -c 10**

fe80::202:72ff:fed6:9836 para garantir o mesmo tamanho de pacote e quantidade enviados para o servidor.

Número de clientes	Tempo de execução do controlador	Tempo de geração do modelo	Tempo de verificação formal	Número de especificações verificadas
2	1m39,719s	0m0,190s	0m0,176s	49
3	4m33,422s	0m0,284s	0m0,684s	75
4	5m4,299s	0m0,371s	0m53,980s	102

Tabela 5.2: Tempos de Execução do Experimento.

Após análise dos resultados de execução apresentados na Tabela 5.2 foi detectado que quanto maior o número de clientes conectados ao servidor, enviando dados para o mesmo, maior será o tempo de execução do controlador, de geração do modelo e de verificação formal. Observou-se, ainda, que a variação de três para quatro clientes teve grande influência no tempo de verificação formal, setenta e oito vezes maior que o tempo de verificação formal com três clientes. É importante lembrar que foi realizado um tratamento para remover as especificações duplicadas, as mesmas não são adicionadas no modelo para garantir o melhor desempenho na etapa de verificação formal.

Pode-se afirmar, diante dos dados aprensentados anteriormente, que mesmo variando o número de clientes conectados ao servidor no momento de execução do controlador o método proposto neste trabalho garante a geração automática do modelo e a verificação formal do mesmo, embora com o tempo de verificação formal destas verificações alterado. Verificou-se, ainda, que com o aumento do número de especificações a serem verificadas o tempo de verificação formal também é acrescido e, que mesmo com a variação de tempo de verificação formal, que depende do número de clientes conectados ao servidor, as verificações foram realizadas com sucesso.

5.7 Discussão dos Resultados Obtidos

Os resultados gerados pela execução da ferramenta NuSMV confirmaram os pressupostos do método apresentado e utilizado neste estudo de caso.

Inicialmente, os traços, gerados pelo método proposto por Vasconcelos [37] aplicado ao *software* de controle proposto por Santos [31], juntamente com o modelo informal do *software* de controle foram executados na ferramenta NuSMV que não apresentou como saída indicação de falhas, conforme explicado na seção 5.4.1.

A comprovação das especificações do modelo foi verificada com a realização de modificações na geração do modelo, onde foram introduzidas modificações nas características. Nestas ocasiões houveram indicações de falhas no resultado da execução da ferramenta NuSMV utilizando os **traços** do *software* de controle original e o modelo informal gerado após as mudanças, conforme indicado nas seções 5.4.2 e 5.4.2.

Para verificar a aplicabilidade do método proposto a outras situações, na seção 5.5, realizou-se uma modificação do *software* de controle proposto por Santos [31]. Nesta ocasião foram introduzidas modificações na lógica do *software* de controle e, utilizando-se como entrada o modelo informal modificado do *software* de controle e os **traços** resultantes da execução deste *software* modificado, executou-se novamente a ferramenta NuSMV que não localizou falhas no *software* modificado.

Por fim, pode-se afirmar que o objetivo principal deste trabalho, desenvolver e validar um método para verificação automática de um protocolo de controle de fluxo adaptativo para *Gateways Bluetooth Low-Energy* aplicado a sistemas de monitoramento remoto de pacientes que utilizam *bluetooth* a partir da verificação formal do modelo gerado, foi alcançado. Pode-se afirmar, ainda, que os objetivos específicos também foram alcançados.

Capítulo 6

Conclusões

Neste capítulo serão apresentadas as conclusões desta dissertação, ressaltando suas contribuições e trabalhos futuros que fornecem a direção para auxiliar futuras pesquisas. O trabalho aqui apresentado introduziu um método de verificação automática a protocolos de controle de fluxo de dados, a partir de informações provenientes do núcleo do *Gateway*. A abordagem proposta nesta pesquisa pode ser dividida em quatro macro etapas: (i) obtenção do log de execução do protocolo; (ii) extração das características do protocolo; (iii) criação do Modelo SMV do protocolo; e (iv) verificação formal do Modelo gerado. Ao fim dessas etapas, teremos como resultado um protocolo de controle de fluxo adaptativo para gateways bluetooth low-energy verificado formalmente, de acordo com as características selecionadas.

O método se destaca por apresentar uma automatização da construção do modelo reduzido de um protocolo de controle de fluxo de dados e uma formatação para a linguagem da ferramenta NuSMV, gerando um suporte à verificação formal destes protocolos. Outro destaque para tal proposta é o auxílio na garantia de consistência de implementação do protocolo, já que alterações no método podem ser realizadas e verificadas formalmente com rapidez. O método proposto pode ser utilizado por outros programas com o código instrumentado e que possuem variações em suas características descritas no resultado da execução. Seria necessário apenas uma modificação no método de obtenção das características do Modelo que deve adequar-se ao modelo utilizado e modificar a escrita das especificações para testar o modelo. O método possibilita a aplicação de testes nas características do protocolo de forma paralela ao processo de desenvolvimento facilitando o encontro de falhas na implementação do protocolo. Por fim, pode-se afirmar que o objetivo principal deste trabalho que é validar

especificamente protocolos de controle adaptativo utilizados em *Gateways BLE* aplicados em um contexto de sistema de monitoramento remoto de pacientes foi atingido.

As principais dificuldades encontradas neste trabalho foram relacionadas à escolha dos traços utilizados para a criação das especificações do modelo, já que tais traços são escolhidos manualmente. Esta etapa poderia ser "automatizada", baseada na escolha do usuário. Uma interface com uma lista das especificações e *checkboxes* poderiam auxiliar no momento da escolha das mesmas. Retirando a necessidade de visualizar o **log** linha por linha e retirar as especificações duplicadas.

Outra dificuldade encontrada foi na execução do protocolo devido à limitação de processamento das máquinas utilizadas no estudo de caso. Quando utilizado mais de dois clientes o *gateway*, núcleo do processamento, não suportava o excesso de requisições e gerava um travamento no mesmo.

Em relação aos possíveis trabalhos futuros derivados deste trabalho de pesquisa, destacam-se os seguintes itens:

- Alterar o método proposto para verificar o protocolo utilizando autômatos temporizados;
- Desenvolver uma ferramenta *Open Source* que realiza o fim-a-fim do método proposto nesta dissertação. Esta ferramenta teria um interface para colocar o valor das características, selecionar as características que seriam utilizadas na verificação formal e o resultado da verificação seria exibido. Retirando a necessidade de realizar o passo a passo do método proposto nesta dissertação;
- Desenvolver um tradutor que escreva automaticamente as características passadas do modelo;
- Aplicar o método proposto em outros estudos de caso, para aumentar a confiança na abordagem proposta;
- Realizar testes de verificação com mais clientes associados ao servidor;
- Adicionar novos verificadores, no método proposto, para aumentar a confiança na abordagem.

Bibliografia

[1] Bluetooth technology website. https://www.bluetooth.com/. Accessed: 2016-07-21.

[2] Nfc forum. http://nfc-forum.org/. Accessed: 2016-07-21.

[3] The zigbee alliance. http://www.zigbee.org. Accessed: 2016-07-21.

[4] Toyota recalls 625,000 hybrids over software glitch, July 2015. [Online; posted 15-July-2015].

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805, October 2010.

[6] Ruiter Braga Caldas. Modelagem, verificação formal e codificação de sistemas reativos autônomos. 2009.

[7] Randy Carroll, Rick Cnossen, Mark Schnell, and David Simons. Continua: An interoperable personal healthcare ecosystem. *IEEE Pervasive Computing*, 6(4):90–94, 2007.

[8] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: A new symbolic model verifier. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 495–499, London, UK, UK, 1999. Springer-Verlag.

[9] Edmund Clarke. Introduction to model checking - smv tools. Carnegie Mellon University - School of Computer Science, Jan-28 to Apr-29, 2005. 47 p. Lecture Notes, available at www.cs.cmu.edu/~emc/15817-s05/.

[10] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[11] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[12] S. Das, W. Zhang, and Y. Liu. A fine-grained control flow integrity approach against runtime memory attacks for embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(11):3193–3207, Nov 2016.

[13] Carla Amor Divino Moreira Delgado. *Modelagem e verificação de propriedades epistêmicas em sistemas multi-agentes*. PhD thesis, UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2007. Avaliable at `http://ls13-www.cs.tu-dortmund.de/homepage/publications/delgado/publications_pdf/DelgadoThesisDSc.pdf`.

[14] Saulo Oliveira Dornellas Luiz, Genildo de Moura Vasconcelos, and Leandro Dias da Silva. Formal specification of dsp gateway for data transmission between processor cores of omap platform. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1545–1549. ACM, 2008.

[15] Jiping Fan, Jian Jiao, Wenbo Wu, and Tingdi Zhao. A model-checking oriented modeling method for safety critical system. In *Reliability Systems Engineering (ICRSE), 2015 First International Conference on*, pages 1–6. IEEE, 2015.

[16] Flávio Gonçalves Fernandes. Um arcabouço para verificação automática de modelos uml. Master's thesis, PUC MG, 2011.

[17] Nelson França Guimarães Ferreira. Verificação formal de sistemas modelados em estados finitos. Master's thesis, Universidade de São Paulo, 2005.

[18] Joel Maurício Rocha Galvão. Conversão sistemática do comportamento definido nos blocos funcionais da norma iec 61 131-3 para autómatos finitos temporizados, 2015.

[19] I. Horvat, N. Lukac, R. Pavlovic, and D. Starcevic. Smart plug solution based on bluetooth low energy. In *Consumer Electronics - Berlin (ICCE-Berlin), 2015 IEEE 5th International Conference on*, pages 435–437, Sept 2015.

[20] IEEE. ISO/IEEE 11073-20601: Health informatics - Point-of-care medical device communication - Part 20601:Optimized exchange protocol Standards. Technical report, 2000.

[21] IEEE. ISO/IEEE 11073-10101: Health informatics - Point-of-care medical device communication - Part 10101:Nomenclature. Technical report, 2008.

[22] Saul A Kripke. A completeness theorem in modal logic. *The journal of symbolic logic*, 24(01):1–14, 1959.

[23] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems : a cyber physical systems approach*. Electrical Engineering & Computer Sciences. First Edition, s. l., 2011. Bibliogr. Index.

[24] D Locke and MQ Telemetry Transport. Mq telemetry transport (mqtt) v3.1 protocol specification. *IBM developer Works Technical Library*, 2010. Available at `http://www.ibm.com/developerworks/webservices/library/ws-mqtt`.

[25] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[26] J. Nieminen, C. Gomez, M. Isomaki, T. Savolainen, B. Patil, Z. Shelby, M. Xi, and J. Oller. Networking solutions for connecting bluetooth low energy enabled machines to the internet of things. *IEEE network*, 28(6):83–90, Nov 2014.

[27] J Nieminen, T Savolainen, M Isomaki, B Patil, Z Shelby, and C Gomez. Ipv6 over bluetooth (r) low energy. Technical report, 2015. Avaliable at `https://www.rfc-editor.org/info/rfc7668`.

[28] da Silva Leandro Dias Vasconcelos Genildo de Moura Perkusich, Angelo. Verificação automática de programas a partir da monitoração de múltiplas execuções de código. In *Anais do XIX Congresso Brasileiro de Automática*, pages 4901–4908. CBA 2012, 2012.

[29] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[30] M Asim Minhas Kashif Saghar and Tariq Farooq. Application of formal methods for validation and verification of embedded system communication protocol. In *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 385–390. IEEE, 2016.

[31] Danilo Freire de Souza Santos. *Controle de fluxo adaptativo para Gateways Bluetooth Low-Energy aplicado a sistemas de monitoramento remoto de pacientes*. PhD thesis, Universidade Federal do Campina Grande, 2016.

[32] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014. Available at `https://www.rfc-editor.org/info/rfc7252`.

[33] Bluetooh SIG. Bluetooth specification version 4.2. Technical report, 2014.

[34] D. P. Simons. Consumer electronics opportunities in remote and home healthcare. In *2008 Digest of Technical Papers - International Conference on Consumer Electronics*, page 1–2. IEEE, 2008.

[35] Italo Giovani A. Stefani. Método de refinamento machina. Master's thesis, UFMG, 2007.

[36] F. Berntsen J. Decuir R. Heydon V. Zhodzishsky T. Savolainen, K. Kerai and E. Callaway. Internet protocol support profile. Technical report, 2014.

[37] Genildo de Moura Vasconcelos. Verificação automática de programas a partir da monitoração de múltiplas execuções de código. Master's thesis, UFCG, 2012.

[38] Fang-Jing Wu, Yu-Fen Kao, and Yu-Chee Tseng. From wireless sensor networks towards cyber physical systems. *Pervasive and Mobile Computing*, 7(4):397–413, 2011.

[39] Daniyal Yasin, Kashif Saghar, and Shahzad Younis. Formal modeling and verification of rumor routing protocol. In *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 318–323. IEEE, 2016.

Apêndice A

Métodos utilizados no método proposto

Código Fonte A.1: Método para extração das características

```
1
2  private static List<Device> extratorDeCaracteristicas(File arquivoDeLog)
3          throws IOException {
4      List<Device> connectedDevices = new ArrayList();
5      BufferedReader br = new BufferedReader(new FileReader(arquivoDeLog));
6      String line = null;
7      String lines = "";
8      while ((line = br.readLine()) != null) {
9          if (line.contains("00:00:00:00:00:00 ->")) {
10             String[] lineSp = line.split("00:00:00:00:00:00 ->")[1]
11                     .split(" ");
12             Device device = new Device();
13             device.setAddr(lineSp[1]);
14             if (!connectedDevices.contains(device)) {
15                 connectedDevices.add(device);
16             }
17         } else if (line.contains("rx_credits")) {
18             // ,DF01,cflow,addr a4:02:b9:01:ae:8b,rx_credits 20->19
19             String[] addrAndCredits = line.split("addr ");
20             String[] addr = addrAndCredits[1].split(",rx_credits ");
21             String addrValue = addr[0];
22             String[] credits = addr[1].split("->");
23             Device device = new Device();
24             device.setAddr(addrValue);
```

```
25              if (!connectedDevices.contains(device)) {
26                  connectedDevices.add(device);
27              }
28              for (int i = 0; i < connectedDevices.size(); i++) {
29                  if (connectedDevices.get(i).equals(device)) {
30                      Device currentDevice = connectedDevices.get(i);
31                      List<String> creditsList = currentDevice.getCredits()
                            ;
32                      creditsList.add(credits[0]);
33                      creditsList.add(credits[1]);
34                      currentDevice.setCredits(creditsList);
35                  }
36              }
37          }
38          lines = lines + line + "\n";
39      }
40      br.close();
41      return connectedDevices;
42  }
```

Código Fonte A.2: Método para criar o Modelo SMV Parte I

```
1  private static String criadorDeModelosSMV(List<Device> connectedDevices,
       int creditosParaEnviar, int incrementoDeCreditos) throws
       FileNotFoundException, UnsupportedEncodingException {
2          try {
3              String filename= "modeloBase.smv";
4              FileWriter fw = new FileWriter(filename,true);
5              fw.write("  creditosParaEnviar := " + String.valueOf(
                   creditosParaEnviar) +";\n");
6              fw.write("  valorDoIncrementoDeCreditos := " + String.valueOf(
                   incrementoDeCreditos) + ";\n");
7              FileReader fr = new FileReader("/home/anne/Area de Trabalho/
                   modelBase2.smv");
8              int c;
9              while((c = fr.read())!=-1) {
10                 fw.write(c);
11             }
```

Código Fonte A.3: Método para criar o Modelo SMV Parte II

```
1         fw.write("MODULE main\n");
2         fw.write("VAR\n");
3         for (int i = 1; i <= connectedDevices.size(); i++) {
4             fw.write("  c" + i + " : cliente(s.autorizando, s.
                  creditosParaEnviar, s.valorDoIncrementoDeCreditos, " + i +
                  ");\n");
5         }
6         fw.write("  s : servidor(TRUE);\n\n");
7         for (int i = 1; i <= connectedDevices.size(); i++) {
8             fw.write("INVARSPEC c"+i+".creditos >= 0\n");
9             fw.write("INVARSPEC (s.state = ocioso -> (c"+i+".state =
                  esperando))\n");
10            fw.write("SPEC AG(s.state = ativo -> AF(c" + i + ".state =
                  conectado))\n");
11        }
12        fw.write("INVARSPEC s.creditosParaEnviar >= 0\n");
13        for (int i = 1; i <= connectedDevices.size(); i++) {
14            for (int j = i+1; j <= connectedDevices.size(); j++) {
15                fw.write("INVARSPEC (c" + i + ".id != c" + j + ".id)\n");
16            }
17        }
18        List<String> especificacoes = new ArrayList<>();
19         for (int j = 1; j <= connectedDevices.size(); j++) {
20             int i = 0;
21             Device d = connectedDevices.get(j-1);
22             int creditsSize = d.getCredits().size();
23             while(i < creditsSize) {
24                 String lineCheckComecoDosCreditos = "";
25                 String creditBefore = d.getCredits().get(i);
26                 String creditAfter = d.getCredits().get(i + 1);
27                 String bandState = "";
28                 if (Integer.valueOf(creditBefore) < Integer.valueOf(
                       creditAfter)) {
29                     bandState = "Up";
30                 } else if (Integer.valueOf(creditBefore) > Integer.
                       valueOf(creditAfter)) {
```

```
31                      bandState = "Down";
32                  }
33                  if (Integer.valueOf(creditBefore) == incrementoDeCreditos
                        && i - 1 >= 0 && Integer.valueOf(d.getCredits().get(i
                        - 1)) == 0) {
34                      lineCheckComecoDosCreditos = String.format("SPEC AG((
                            c" + j + ".creditos = \%s &\%s) -> AX ((c"+j+".
                            creditos = \%s)))",
35                              d.getCredits().get(i - 1), "Up", creditBefore
                                ) + "\n";
36                  }
37                  if (i - 1 >= 0 && Integer.valueOf(d.getCredits().get(i -
                        1)) == 0) {
38                      lineCheckComecoDosCreditos = String.format("SPEC AG((
                            c"+j+".creditos = \%s & c"+j+".estadoDosCreditos =
                             \%s) -> AX ((c"+j+".creditos = \%s)))",
39                              d.getCredits().get(i - 1), "Up", creditBefore
                                ) + "\n";
40                  }
41                  i = i + 2;
42                  String line = String.format("SPEC AG((c"+j+".creditos =
                        \%s & c"+j+".estadoDosCreditos = \%s) -> AX ((c"+j+".
                        creditos = \%s)))",
43                          creditBefore, bandState, creditAfter) + "\n";
44                  if (!especificacoes.contains(line)) {
45                      especificacoes.add(line);
46                  } else if (!especificacoes.contains(
                        lineCheckComecoDosCreditos)) {
47                      especificacoes.add(lineCheckComecoDosCreditos);
48                  }
49              }
50          }
51          for (final String especificacao : especificacoes) {
52              fw.write(especificacao);
53          }
54          fr.close();
55          fw.close();
```

```
56          return filename;
57      } catch(IOException ioe) {
58          return "Erro ao criar o modelo";
59      }
60  }
```

Apêndice B

Resultados de verificação formal

Código Fonte B.1: Verificação do modelo gerado

```
1
2  *** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
3  ...
4  *** Copyright (c) 2007-2010, Niklas Sorensson
5
6  -- specification AG (s.state = ativo -> AF c1.state = conectado)  is true
7  -- specification AG (s.state = ativo -> AF c2.state = conectado)  is true
8  -- specification AG ((c1.creditos = 8 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 7)  is true
9  -- specification AG ((c1.creditos = 7 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 6)  is true
10 -- specification AG ((c1.creditos = 6 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 5)  is true
11 -- specification AG ((c1.creditos = 5 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 4)  is true
12 -- specification AG ((c1.creditos = 4 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 3)  is true
13 -- specification AG ((c1.creditos = 3 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 2)  is true
14 -- specification AG ((c1.creditos = 2 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 1)  is true
15 -- specification AG ((c1.creditos = 1 & c1.estadoDosCreditos = Down) ->
       AX c1.creditos = 0)  is true
```

```
16 -- specification AG ((c1.creditos = 2 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 1)  is true
17 -- specification AG ((c1.creditos = 1 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 0)  is true
18 -- specification AG ((c1.creditos = 2 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 1)  is true
19 -- specification AG ((c1.creditos = 1 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 0)  is true
20 -- specification AG ((c1.creditos = 2 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 1)  is true
21 -- specification AG ((c1.creditos = 1 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 0)  is true
22 -- specification AG ((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX
      c1.creditos = 2)  is true
23 -- specification AG ((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX
      c1.creditos = 2)  is true
24 -- specification AG ((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX
      c1.creditos = 2)  is true
25 -- specification AG ((c2.creditos = 4 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 3)  is true
26 -- specification AG ((c2.creditos = 3 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 2)  is true
27 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
28 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
29 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
30 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
31 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
32 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
33 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
```

```
34 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
35 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
36 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
37 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
38 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
39 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
40 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 2)  is true
41 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 2)  is true
42 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 2)  is true
43 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 2)  is true
44 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 2)  is true
45 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 2)  is true
46 -- invariant c1.id != c2.id  is true
47 -- invariant (s.state = ocioso -> c1.state = esperando)  is true
48 -- invariant (s.state = ocioso -> c2.state = esperando)  is true
49 -- invariant c1.creditos >= 0  is true
50 -- invariant c2.creditos >= 0  is true
51 -- invariant s.creditosParaEnviar >= 0  is true
```

Código Fonte B.2: Verificação do modelo modificado gerado

```
1
2 *** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
3 *** Enabled addons are: compass
4 *** For more information on NuSMV see <http://nusmv.fbk.eu>
5 *** or email to <nusmv-users@list.fbk.eu>.
```

```
6  *** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>
7
8  *** Copyright (c) 2010-2014, Fondazione Bruno Kessler
9
10 *** This version of NuSMV is linked to the CUDD library version 2.4.1
11 *** Copyright (c) 1995-2004, Regents of the University of Colorado
12
13 *** This version of NuSMV is linked to the MiniSat SAT solver.
14 *** See http://minisat.se/MiniSat.html
15 *** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
16 *** Copyright (c) 2007-2010, Niklas Sorensson
17
18 -- specification AG (s.state = ativo -> AF c1.state = conectado)  is true
19 -- specification AG (s.state = ativo -> AF c2.state = conectado)  is true
20 -- specification AG ((c1.creditos = 20 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 19)  is true
21 -- specification AG ((c1.creditos = 19 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 18)  is true
22 -- specification AG ((c1.creditos = 18 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 17)  is true
23 -- specification AG ((c1.creditos = 17 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 16)  is true
24 -- specification AG ((c1.creditos = 16 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 15)  is true
25 -- specification AG ((c1.creditos = 15 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 14)  is true
26 -- specification AG ((c1.creditos = 14 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 13)  is true
27 -- specification AG ((c1.creditos = 13 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 12)  is true
28 -- specification AG ((c1.creditos = 12 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 11)  is true
29 -- specification AG ((c1.creditos = 11 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 10)  is true
30 -- specification AG ((c1.creditos = 10 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 9)  is true
31 -- specification AG ((c1.creditos = 9 & c1.estadoDosCreditos = Down) ->
```

```
      AX c1.creditos = 8)  is true
32 -- specification AG ((c1.creditos = 8 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 7)  is true
33 -- specification AG ((c1.creditos = 7 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 6)  is true
34 -- specification AG ((c1.creditos = 6 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 5)  is true
35 -- specification AG ((c1.creditos = 5 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 4)  is true
36 -- specification AG ((c1.creditos = 4 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 3)  is true
37 -- specification AG ((c1.creditos = 3 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 2)  is true
38 -- specification AG ((c1.creditos = 2 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 1)  is true
39 -- specification AG ((c1.creditos = 1 & c1.estadoDosCreditos = Down) ->
      AX c1.creditos = 0)  is true
40 -- specification AG ((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX
      c1.creditos = 5)  is true
41 -- specification AG ((c2.creditos = 20 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 19)  is true
42 -- specification AG ((c2.creditos = 19 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 18)  is true
43 -- specification AG ((c2.creditos = 18 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 17)  is true
44 -- specification AG ((c2.creditos = 17 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 16)  is true
45 -- specification AG ((c2.creditos = 16 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 15)  is true
46 -- specification AG ((c2.creditos = 15 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 14)  is true
47 -- specification AG ((c2.creditos = 14 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 13)  is true
48 -- specification AG ((c2.creditos = 13 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 12)  is true
49 -- specification AG ((c2.creditos = 12 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 11)  is true
```

```
50 -- specification AG ((c2.creditos = 11 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 10)  is true
51 -- specification AG ((c2.creditos = 10 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 9)  is true
52 -- specification AG ((c2.creditos = 9 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 8)  is true
53 -- specification AG ((c2.creditos = 8 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 7)  is true
54 -- specification AG ((c2.creditos = 7 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 6)  is true
55 -- specification AG ((c2.creditos = 6 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 5)  is true
56 -- specification AG ((c2.creditos = 5 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 4)  is true
57 -- specification AG ((c2.creditos = 4 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 3)  is true
58 -- specification AG ((c2.creditos = 3 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 2)  is true
59 -- specification AG ((c2.creditos = 2 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 1)  is true
60 -- specification AG ((c2.creditos = 1 & c2.estadoDosCreditos = Down) ->
      AX c2.creditos = 0)  is true
61 -- specification AG ((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX
      c2.creditos = 5)  is true
62 -- invariant c1.id != c2.id  is true
63 -- invariant (s.state = ocioso -> c1.state = esperando)  is true
64 -- invariant (s.state = ocioso -> c2.state = esperando)  is true
65 -- invariant c1.creditos >= 0  is true
66 -- invariant c2.creditos >= 0  is true
67 -- invariant s.creditosParaEnviar >= 0  is true
```

Apêndice C

Modelo gerado pelo método proposto

Código Fonte C.1: Representação do Modelo modificado gerado

```
1  MODULE cliente (autorizado, creditoNovo, valorDoIncrementoDeCreditos,
       idCliente)
2  VAR
3     estadoDosCreditos:{Up, Down};
4     creditos : 0 .. 100;
5     conectados : boolean;
6     id : 0 .. 100;
7     state : {esperando, conectado};
8
9  ASSIGN
10    init(state) := esperando;
11    init(creditos) := 0;
12    init(estadoDosCreditos) := Up;
13    id := idCliente;
14    next(state) :=
15     case
16       state=esperando & autorizado : conectado;
17       state=conectado & !autorizado : esperando;
18      TRUE : state;
19     esac;
20
21 TRANS
22    (estadoDosCreditos = Up & creditos != 0  & next(creditos) = creditos +
         valorDoIncrementoDeCreditos) |
```

```
23    (estadoDosCreditos = Down & next(creditos) = creditos -
          valorDoIncrementoDeCreditos) |
24    (estadoDosCreditos = Up & creditos = 0 & next(creditos) = creditoNovo)
25
26  MODULE servidor (req)
27  VAR
28    state : {ocioso, ativo};
29    autorizando : boolean;
30    creditosParaEnviar : 0 .. 100;
31    valorDoIncrementoDeCreditos : 0 .. 100;
32
33  ASSIGN
34    creditosParaEnviar := 5;
35    valorDoIncrementoDeCreditos := 1;
36    next(state) :=
37     case
38       state=ocioso & req : ativo;
39       state=ativo & !req : ocioso;
40      TRUE : state;
41     esac;
42    autorizando := (state = ativo);
43  MODULE main
44  VAR
45    c1 : cliente(s.autorizando, s.creditosParaEnviar, s.
          valorDoIncrementoDeCreditos, 1);
46    c2 : cliente(s.autorizando, s.creditosParaEnviar, s.
          valorDoIncrementoDeCreditos, 2);
47    s : servidor(TRUE);
48
49  INVARSPEC c1.creditos >= 0
50  INVARSPEC (s.state = ocioso -> (c1.state = esperando))
51  SPEC AG(s.state = ativo -> AF(c1.state = conectado))
52  INVARSPEC c2.creditos >= 0
53  INVARSPEC (s.state = ocioso -> (c2.state = esperando))
54  SPEC AG(s.state = ativo -> AF(c2.state = conectado))
55  INVARSPEC s.creditosParaEnviar >= 0
56  INVARSPEC (c1.id != c2.id)
```

```
57 SPEC AG((c1.creditos = 20 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 19)))
58 SPEC AG((c1.creditos = 19 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 18)))
59 SPEC AG((c1.creditos = 18 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 17)))
60 SPEC AG((c1.creditos = 17 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 16)))
61 SPEC AG((c1.creditos = 16 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 15)))
62 SPEC AG((c1.creditos = 15 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 14)))
63 SPEC AG((c1.creditos = 14 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 13)))
64 SPEC AG((c1.creditos = 13 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 12)))
65 SPEC AG((c1.creditos = 12 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 11)))
66 SPEC AG((c1.creditos = 11 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 10)))
67 SPEC AG((c1.creditos = 10 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 9)))
68 SPEC AG((c1.creditos = 9 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 8)))
69 SPEC AG((c1.creditos = 8 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 7)))
70 SPEC AG((c1.creditos = 7 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 6)))
71 SPEC AG((c1.creditos = 6 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 5)))
72 SPEC AG((c1.creditos = 5 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 4)))
73 SPEC AG((c1.creditos = 4 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 3)))
74 SPEC AG((c1.creditos = 3 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 2)))
```

```
75 SPEC AG((c1.creditos = 2 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 1)))
76 SPEC AG((c1.creditos = 1 & c1.estadoDosCreditos = Down) -> AX ((c1.
       creditos = 0)))
77 SPEC AG((c1.creditos = 0 & c1.estadoDosCreditos = Up) -> AX ((c1.creditos
        = 5)))
78 SPEC AG((c2.creditos = 20 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 19)))
79 SPEC AG((c2.creditos = 19 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 18)))
80 SPEC AG((c2.creditos = 18 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 17)))
81 SPEC AG((c2.creditos = 17 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 16)))
82 SPEC AG((c2.creditos = 16 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 15)))
83 SPEC AG((c2.creditos = 15 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 14)))
84 SPEC AG((c2.creditos = 14 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 13)))
85 SPEC AG((c2.creditos = 13 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 12)))
86 SPEC AG((c2.creditos = 12 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 11)))
87 SPEC AG((c2.creditos = 11 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 10)))
88 SPEC AG((c2.creditos = 10 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 9)))
89 SPEC AG((c2.creditos = 9 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 8)))
90 SPEC AG((c2.creditos = 8 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 7)))
91 SPEC AG((c2.creditos = 7 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 6)))
92 SPEC AG((c2.creditos = 6 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 5)))
```

```
93 SPEC AG((c2.creditos = 5 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 4)))
94 SPEC AG((c2.creditos = 4 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 3)))
95 SPEC AG((c2.creditos = 3 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 2)))
96 SPEC AG((c2.creditos = 2 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 1)))
97 SPEC AG((c2.creditos = 1 & c2.estadoDosCreditos = Down) -> AX ((c2.
       creditos = 0)))
98 SPEC AG((c2.creditos = 0 & c2.estadoDosCreditos = Up) -> AX ((c2.creditos
        = 5)))
```