TRABALHO DE GRADUAÇÃO

Prototipação e Verificação Formal de Sistema Autônomo com Propriedades Tempo-Real: Um estudo de caso no Body Sensor Network

Ricardo Diniz Caldas

Brasília, Dezembro de 2017

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Prototipação e Verificação Formal de Sistema Autônomo com Propriedades Tempo-Real: Um estudo de caso no Body Sensor Network

Ricardo Diniz Caldas

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Professora Genaina Nunes Rodrigues, CIC/UnB _____
Orientadora

Professora Carla Maria C. C. Koike, CIC/UnB _____
Examinadora interna

Professor Antônio Padilha L. Bo, ENE/UnB _____
Examinador interno

Brasília, Dezembro de 2017

FICHA CATALOGRÁFICA

RICARDO, DINIZ CALDAS
Prototipação e Verificação Formal de Sistema Autônomo com Propriedades Tempo-Real: Um estudo de caso no Body Sensor Network,

[Distrito Federal] 2017.

x, 57p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2017). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Verificação Formal 2.Sistemas autônomos
3. Tempo-real 4.BSN

I. Mecatrônica/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

CALDAS, RICARDO DINIZ, (2017) Prototipação e Verificação Formal de Sistema Autônomo com Propriedades Tempo-Real: Um estudo de caso no Body Sensor Network. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-$n°$025, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 57p.

CESSÃO DE DIREITOS

AUTOR: Ricardo Diniz Caldas

TÍTULO DO TRABALHO DE GRADUAÇÃO: Prototipação e Verificação Formal de Sistema Autônomo com Propriedades Tempo-Real: Um estudo de caso no Body Sensor Network.

GRAU: Engenheiro ANO: 2017

Ricardo Diniz Caldas

Campus Darcy Ribeiro, LES - CIC, Universidade de Brasília.

70910-900 Brasília – DF – Brasil.

Agradecimentos

Agradeço aos meus pais, Lea e Ricardo, e irmãos, Gabriela e Leonardo, que foram fundamentais em todos os âmbitos desta caminhada. Agradeço também à orientadora Profa. Genaina Rodrigues, pelo suporte, paciência e exemplo de comprometimento e integridade durante todo o processo.

Ricardo Diniz Caldas

RESUMO

O desenvolvimento de dispositivos micro eletro-mecânicos em paralelo ao avanço da capacidade de processamento e comunicação em rede traz novas possibilidades de imersão de tecnologias no dia-a-dia do ser humano. Na perspectiva da engenharia, essas possibilidades alavancam novos desafios em cenários que demandam resposta em tempo real, em particular projeto e desenvolvimento de sistemas autônomos com garantias de dependabilidade a nível de *software*. Neste trabalho, foi implementada uma aplicação do Body Sensor Network (BSN), um protótipo de rede de sensores para monitoramento de sinais vitais do corpo humano com o objetivo de detectar situações emergenciais e, a partir da metodologia de controle para automação proposta no artigo seminal *"Software Engineering meets Control Theory"* com utilização da ferramenta UPPAAL para modelagem e validação, um controlador foi desenvolvido e implementado no protótipo com garantias de propriedades de tempo real. Por fim, o protótipo autônomo é avaliado com o intuito de levantar contribuições da aplicação de verificação de modelos formais no projeto de sistemas autônomos com resposta em tempo real.

Palavras Chave: Controle para automação, Engenharia de software, verificação formal de modelos, automatos temporais, BSN, UPPAAL, OpenDaVINCI, sistemas autônomos tempo real

ABSTRACT

The development of microelectro-mechanical devices in parallel with the advance of the capacity of processing and wireless communication brings new possibilities of immersion of pervasive technologies in human daily activities. From the perspective of engineering, these possibilities lead to new challenges in scenarios that demand real-time response, particularly design and development of autonomous systems with software-level dependability guarantees. In this work, an application of the Body Sensor Network (BSN), a sensor network prototype for vital signs monitoring was implemented, aiming emergency detection and, based on the control methodology for automation proposed in the seminal article "Software Engineering meets Control Theory"with UPPAAL tool for modeling and validation, a controller was developed and implemented in the prototype with guarantees of real-time properties. Finally, the autonomous prototype is evaluated with the intention of raising contributions from the use of formal verification in the design of autonomous systems with real-time response.

Keywords: Control Theory, Software Engineering, formal verification, timed automata, BSN, UPPAAL, OpenDaVINCI, real-time autonomous systems

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Nomeclaturas

$\mathbb{T}$	Conjunto de tempos de relógio	
∇	Granularidade calibrada do relógio	
Φ	Fórmula Lógica Proposicional	
φ	Fórmula de Caminho	
SPO2	Medida de oxigenação no sangue	
ECG	Medida de sinais do eletrocardiógrafo	
TEMP	Medida de temperatura corporal	
A	Função Lógica "Para todo"	
E	Função Lógica "Existe"	
imply	Função Lógica de implicação	
τ	Período de ciclo do escalonador	[s]
f	Frequência de execução do escalonador	[Hz]
N_p	Número de módulos da plataforma central	
N_s	Número de módulos do nó sensor	
τ_{exec_p}	Tempo de processamento da plataforma central	[s]
τ_{exece_s}	Tempo de processamento do nó sensor	[s]

Siglas

BSN	Rede de Sensores para o Corpo - *Body Sensor Network*
BAN	Rede de Sensores para Area do Corpo - *Body Area Network*
WBAN	Rede sem fio de Sensores para Aread do Corpo - *Wireless Body Area Network*
TS	Sistema de transição - *Trasition System*
LT	Linear no Tempo - *Linear Temporal*
LTL	Lógica Linear no Tempo - *Linear Temporal Logic*
CTL	Árvore de Computação Lógica - *Computational Tree Logic*
TCTL	Árvore de Computação Lógica Temporal - *Temporal Computational Tree Logic*
CSP	Processos Sequenciais Comunicativos - *Communicating Sequencial Processes*

Capítulo 1

Introdução

1.1 Contextualização

Os avanços tecnológicos das últimas décadas permitiram a fabricação de dispositivos pequenos com sensores e atuadores, alta capacidade de processamento e capacidade de integração em rede por comunicação sem fio. Tais características instigam tanto a academia quanto a indústria a buscarem soluções viáveis para a automação de tarefas complexas com o objetivo de descentralizar a demanda de serviços que limitam a capacidade de entrega.

Por exemplo, um hospital que carece de espaço para abrigar grupos de pessoas que precisam de monitoramento constante dos sinais vitais, mas não são classificadas como casos graves em que há necessidade de ocupar um leito. Nesse sentido, o *Body Sensor Network* (BSN) [5], rede de sensores do corpo humano pode proporcionar o serviço de maneira autônoma com maior conforto para o paciente e reduzir custos do hospital.

Contudo, sistemas dessa natureza, críticos, possuem normas restritas de segurança, confiabilidade e outras qualidades que se não atendidas os tornam inviáveis. Um exemplo é o caso da resposta em tempo real na aplicação do BSN, que em caso de descumprimento pode acarretar em morte ou danos irreparáveis ao paciente. Portanto, é de extrema importância que a equipe de engenharia responsável pelo projeto utilize metodologias que assegurem a qualidade devida em todos os aspectos do produto, tanto a nível de *hardware* quanto *software*.

A qualidade de entrega de serviços está associada ao termo dependabilidade [6], que engloba seis atributos fundamentais na garantia de qualidade de um sistema: disponibilidade, confiabilidade, segurança operacional *(safety)*, integridade, manutenibilidade e confidencialidade. A garantia da dependabilidade pode ser endereçada por meio da verificação formal dos atributos atendidos pelo sistema afinal, as ferramentas de verificação operam sobre linguagens formais de modelagem de sistemas discretos.

Sistemas autônomos complexos preconizam garantias de dependabilidade em tempo de execução. Em 2015, Filieri A. et al. [1] propuseram a aplicação de técnicas da teoria de controle no domínio da engenharia de software por meio do controle em malha fechada com o objetivo de

construir sistemas de software autônomos. Como consequência abriu espaço para concepção de metodologias de projeto de sistemas autônomos com garantias de atributos de dependabilide em tempo real.

1.2 Definição do problema

Assegurar múltiplos atributos de dependabilidade em sistemas de *software* autônomos ainda não possui uma solução trivial [7], já que estes visam responder de maneira satisfatória a variações do ambiente com tempo de resposta aceitável para cada situação específica e portanto de difícil modelagem.

Contudo, sistemas de *software* podem ser modelados com a aplicação de técnicas de linguagens formais como autômatos e, no caso de sistemas com resposta em tempo real autômatos temporais. Os requisitos são formalizados em propriedades e podem ser garantidos por meio de verificadores de modelo. Em contrapartida, a complexidade de execução de algoritmos de verificação de modelo torna desafiadora a realização de verificação contínua em tempo hábil para adaptação automática de sistemas.

Técnicas da Teoria de Controle asseguram comportamento correto de sistemas em resposta a variações no ambiente com garantias de tempo real. Contudo, necessitam de fundamentação matemática bem delimitada por equações do sistema que deve ser controlado, o que não se aplica ao conhecimento comum no âmbito da engenharia de *software*.

1.3 Objetivo

O objetivo deste trabalho é seguir o passo-a-passo da metodologia de projeto de controladores para implementação de sistema de *software* autônomo, com aplicação de técnicas de modelagem formal e verificação de modelos, a partir da implementação do estudo de caso com o *middleware* OpenDaVINCI.

1.4 Apresentação do manuscrito

Os demais capítulos da presente monografia estão organizados da seguinte forma:

No segundo capítulo, o embasamento teórico necessário para fundamentar os conceitos, ferramentas e técnicas utilizadas durante o projeto serão apresentados. O terceiro capítulo descreve em detalhes o desenvolvimento tanto do protótipo quanto a modelagem, concepção do controlador e verificação do modelo criado a partir dos requisitos do projeto. No quarto capítulo a proposta é avaliada por meio de experimentos que verificam a realização dos requisitos levantados. E finalmente, o quinto capítulo apresenta a conclusão do projeto e lista possíveis trabalhos futuros.

Capítulo 2

Referencial Teórico

2.1 Teoria de Controle apliacada a Engenharia de Software

Essa seção é dedicada a discutir os aspectos mais importantes do artigo que motivou este trabalho: Software Engineering meets Control Theory por Filieri A. et al. [1].

Requisitos dinâmicos, ambientes flexíveis e imprevisíveis, condições de operação incertas demandam que sistemas de *software* busquem alternativas robustas para adaptação em tempo de operação. No século passado, a Teoria de Controle estabeleceu uma série de técnicas formalmente bem fundamentadas que são amplamente empregadas na automação de processos com garantias de eficiência e comportamento preciso sob condições de operação em tempo de execução. Apesar das semelhanças entre o controle de plantas industriais e adaptação a nível de *software*, aplicar as técnicas de controle diretamente a automação de sistemas de *software* têm duas principais limitações:

1. A modelagem formal utilizada para descrever o comportamento de sistemas contínuos é dificilmente aplicável a sistemas de *software*,
2. A falta de metodologias de Engenharia de Software que endereçam controlabilidade como propriedades de sistemas.

O artigo então, faz um paralelo entre o processo de concepção de controladores na teoria de controle com o desenvolvimento de *software* com o objetivo de descrever uma metodologia para conceber sistemas de *software* autônomos e assim garantir efetividade, eficiência e robustez. A Figura 2.1 descreve o passo-a-passo da metodologia em que cada nível representa o grau de acoplamento entre a técnica e o sistema. E então cada passo é descrito seguindo a ordem da enumeração desta.

1. Definir os objetivos (*goals*) do controlador. Os objetivos podem ser divididos entre (i) funcionais e (ii) não-funcionais. (i) Os funcionais representam quais características o sistema deve exibir sob certas condições. (ii)Os não-funcionais expressam objetivos de qualidade de entrega do serviço (e.g. desempenho, confiabilidade, consumo de energia).

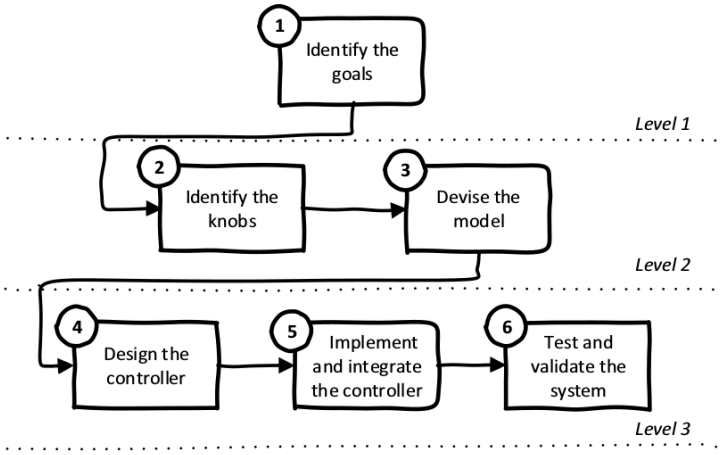


Figura 2.1: Metodologia para controle de softwares autônomos [1]

2. Definir os atuadores (*knobs*) do sistema. Atuadores são as entidades responsáveis por agir no sistema em prol da satisfação dos objetivos levantados para o controlador.

3. Modelar a planta do sistema (*Devise the model*). Em outras palavras, é a formalização matemática do sistema levando em consideração a relação entre objetivo e comportamento dos atuadores e seus possíveis efeitos colaterais sob o sistema.

4. Projetar o controlador (*Design the controller*). Existem diversas técnicas para projeto de controladores no domínio de sistemas contínuos formalizadas na Teoria de Controle, resta ao engenheiro decidir qual aplicar a cada caso em específico.

5. Implementar e integrar o controlador ao sistema modelado (*Implement and integrate the controller*).

6. Validar (*validate*) o comportamento do sistema controlado. O sistema controlado passa por processos de teste de aceitação do controlador implementado para averiguação quantitativa do comportamento do sistema.

O sistema de software prototipado neste trabalho será modelado de acordo com a abstração de sistemas a eventos discretos e a validação será realizada por meio de técnicas de verificação de modelos formais (*model-checking*).

2.2 Body Sensor Network

Body Sensor Networks (BSNs) foi um termo cunhado no início dos anos 2000 para representar todas as aplicações e dispositivos de comunicação que podem estar sobre, dentro ou próximos ao corpo humano [5]. Com o decorrer dos anos, o crescimento percentual da população idosa no mundo e os avanços nas tecnologias de medição alavancaram a quantidade de trabalhos relacionados a aplicações médicas que utilizam BSNs para monitoramento sinais fisiológicos [3].

De acordo com Nadeem et al. [2], as quatro principais áreas de aplicações médicas que utilizam BSNs são: cuidados com a saúde, assistência a deficiências físicas, esportes e monitoramento de atividade humana. Ainda em [2], foi feita uma análise baseada em parâmetros como sub-domínio da aplicação, tipo de sensores usados, roteamento ou técnicas de fusão de dados e a tecnologia *wireless* aplicada das BSNs.

Dentro da classificação de BSNs levantadas por [2], existem as que utilizam meios físicos para comunicação e as que utilizam redes sem fio, as *Wireless Body Sensor Networks* (WBSNs). A comunicação por redes sem fio trouxe vantagens significativas aos sistemas que a adotaram, tanto no quesito de conforto aos usuários quanto modularidade e escalabilidade dos sistemas.

Neste trabalho é abordada a aplicação de sistemas médicos para monitoramento de sinais vitais com detecção de situações de emergência para redes de sensores sem fio.

2.2.1 Características básicas

2.2.1.1 Arquitetura

O panorama geral da arquitetura comum a sistemas de monitoramento de sinais vitais baseado em WBSNs é ilustrado na Figura 2.2, que é composta de três camadas de comunicação distintas pela natureza dos componentes internos e alcance das tecnologias.

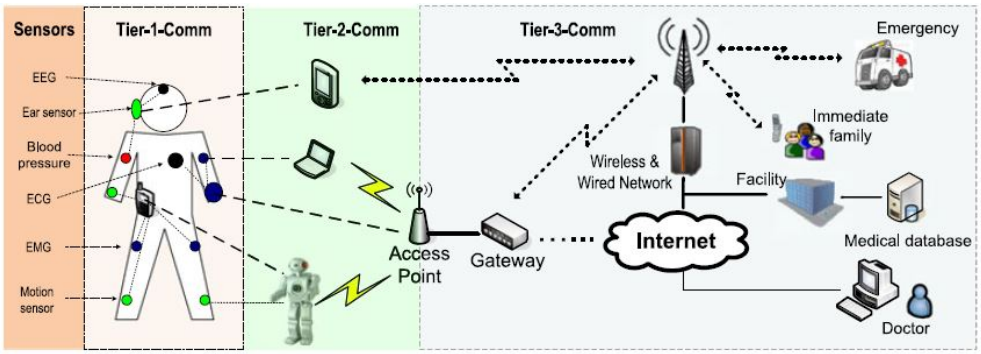


Figura 2.2: Panorama das três camadas da rede de sensores sem fio [2].

As duas primeiras camadas, intra-BSN e inter-BSN, constituem a BSN em si e englobam os módulos que são estudados neste trabalho. A terceira camada vai além da BSN e foge do escopo pelo fato de constituir sistemas que são vistos como serviços oferecidos a partir de informações enviadas.

A primeira camada é chamada intra-BSN. Os dispositivos principais desta são os nós sensores, responsáveis por capturar, pré-processar, e enviar informações coletadas e processadas para a segunda camada. Os módulos da intra-BSN possuem tecnologias de comunicação limitadas a 2 metros de alcance. Cada um dos nós pode ser considerado autônomo e possui mecanismo interno de envio de informações com frequência definida pelo mesmo.

A segunda camada é chamada de inter-BSN. Os dispositivos que a compõe são receptores e centralizadores das informações constantemente enviadas pelos nós sensores, e funcionam como

coordenadores em topologias tipo estrela. Além da comunicação com os nós sensores, os dispositivos que a constituem devem ter capacidade de comunicar entre si ou com redes maiores, como a Internet. Estes dispositivos são responsáveis pelas análises do conjunto de dados e informações recebidos dos nós sensores e requisição de serviços fora da BSN.

A terceira camada vai além da BSN e é específica à implementação do sistema. É geralmente aplicada em áreas metropolitanas com o intuito de conectar uma ou mais redes BSN a hospitais, familiares, centros de atendimento médico e profissionais específicos que prestam serviços.

2.2.2 Dispositivos e tecnologias

As duas primeiras camadas intra-BSN e inter-BSN formam o núcleo das aplicações de monitores de sinais vitais do corpo humano encontradas na literatura. Na intra-BSN é comum encontrarmos os nós sensores e na inter-BSN os nós centrais. Fisicamente o nó sensor é implementado com um circuito elétrico microprocessado, um ou mais sensores da mesma natureza, filtros, conversores AC/DC, memória embutida, alimentação de energia por meio de baterias e dispositivos de comunicação sem fio. A Figura 2.3 representa os módulos típicos encontrados em um nó sensor.

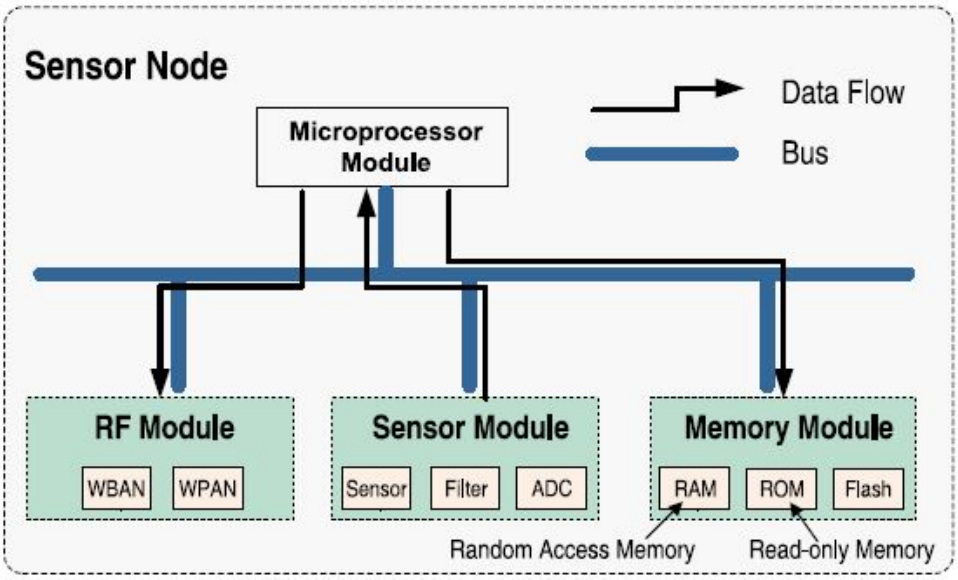


Figura 2.3: Modulos típicos de um nó sensor. [3]

Uma característica fundamental do circuito é possuir baixo consumo de energia e portanto, o dispositivo de comunicação deve ser avaliado com cautela quando da escolha dos componentes no projeto do módulo. Além disso, este deve ser leve, o que limita o tamanho e peso da bateria, reforçando a alta eficiência do circuito em relação ao baixo consumo. Os padrões de comunicação sem fio mais utilizados são IEEE 802.15.4 ou *Bluetooth*, cujo alcance é restrito mas o consumo de energia é baixo.

A plataforma central é a centralizadora da informação e responsável pela comunicação tanto com os nós sensores da intra-BSN quanto com as redes ou dispositivos que estão além da BSN. Portanto, necessita de maior capacidade de processamento, memória, redes de curto e longo alcance,

além de camada de interface com o usuário. É comum utilizar-se de aparelhos celulares, *tablets* ou dispositivos específicos por aplicação na camada inter-BSN.

2.3 Sistemas em tempo real

Requisitos com limite de tempo são necessários em grande parte das aplicações críticas e reativas (e.g. "Em caso de emergência a ambulância deverá ser acionada em no máximo 5 segundos"). Em alguns sistemas, o descumprimento do tempo máximo de execução de uma tarefa pode provocar falhas catastróficas. Sistemas em tempo real demandam, além da corretude lógica do resultado, assegurar a corretude temporal [8].

2.3.1 Classificação

Sistemas em tempo real podem ser classificados de duas formas: *hard realtime* ou *soft realtime* [8]. Em sistemas *hard realtime*, a corretude temporal é crítica e não pode ser sacrificada por outros requisitos. Em alguns sistemas *hard realtime*, principalmente sistemas críticos, a corretude temporal é tão importante que requisitos lógicos podem ser implementados de maneira flexível para garantir que o sistema responderá de acordo com os requisitos temporais.

Em sistemas *soft realtime* a corretude temporal é importante, mas não crítica, ou seja, falhas de tempo real não ocasionam efeitos catastróficos. Nestes, as tarefas o agendamento das tarefas não exige prazos finais rígidos.

Outra classificação de sistemas em tempo real é baseada em eventos e tempos de relógio, que são separados em arquiteturas acionadas por tempo (*time-triggered*) e/ou acionados por eventos (*event-triggered*).

Arquiteturas acionadas por tempo operam de acordo com tempos definidos por relógios integrados independentes ou por leituras do relógio do sistema. Nestas trata-se cada pulso do relógio como uma interrupção que quando alcançada, habilita uma ação para execução. Já as acionadas por eventos operam de acordo com eventos externos que geram interrupções capazes de acionar rotinas a serem executadas.

2.3.2 Relógio

Para fins de formalização e aplicabilidade em sistemas reais, o tempo pode ser visto de duas formas complementares como:

- uma métrica que consiste de um conjunto de valores de tempo;
- um processo único que evolui continuamente.

A primeira, relativa à medição do tempo e a segunda relativa a ordenação temporal de eventos. A capacidade de medir o tempo, implica na possibilidade de comparação entre tempos distintos, ou

seja, um tempo t_1 precede/sucede um tempo t_2. A noção de evolução constante do tempo permite que o *agora* seja designado para definir o tempo atual. E portanto, o relógio pode ser definido como o conjunto:

$$clock = (\mathbb{T}, zero, next, \nabla) \tag{2.1}$$

Em que $\mathbb{T}$ é o conjunto de tempos de relógio, *zero* é um elemento distinto em $\mathbb{T}$, *next* é uma função injetora em $\mathbb{T}$ e $\nabla \in \mathbb{R}^+$. A função *next* é análoga à função de sucessor no domínio $\mathbb{N}$ porém no conjunto definido $\mathbb{T}$. E ∇ é a distância entre duas medições de tempo consecutivas, a granularidade calibrada do relógio.

2.4 Verificação de modelo

Com o passar dos anos e constante evolução de tecnologia (*hardware* e *software*), os sistemas projetados e concebidos por cientistas e engenheiros estão assumindo papéis cada vez mais importantes no dia-a-dia do ser humano. Sistemas cuja falha pode acarretar em danos irreparáveis à vida ou ao meio ambiente são chamados de sistemas críticos [9].

Prever falhas no funcionamento de sistemas complexos e distribuídos na fase de projeto não é uma tarefa trivial sem a ajuda de ferramentas específicas. Técnicas de verificação de modelo buscam garantir, em fase de projeto, o correto funcionamento de sistemas complexos por meio da formalização de requisitos e modelagem do comportamento do sistema.

2.4.1 Modelagem

Transition systems (TS) são comumente utilizados por projetistas para criar modelos que descrevam o comportamento de sistemas cuja abstração de eventos discretos é necessária. Na literatura diversos tipos de TS foram propostos, porém a notação empregada neste trabalho se baseia em Katoen e Baier [4] em que as transições (mudanças de estado) são descritas por nomes de ações e os estados por proposições atômicas (APs). Proposições atômicas são utilizadas para formalizar características temporais que expressam fatos conhecidos sobre os estados do sistema.

Um *transition system* é caracterizado pela tupla $TS = (S, Act, \rightarrow, I, AP, L)$ em que:

- S é o conjunto finito de estados,
- Act é o conjunto finito de ações,
- $\rightarrow \subseteq$ S x Act x S é uma relação de transição,
- I é o conjunto finito de estados iniciais,
- AP é o conjunto de proposições atômicas, e

- $L : S \rightarrow 2^{AP}$ é o conjunto de possíveis palavras (possível conjunto de eventos) formadas pelas proposições atômicas AP.

Para analisar sistemas representados por *transition systems* tanto a abordagem baseada em ações (transições) quanto em estados podem ser empregadas. Neste trabalho são empregadas as proposições atômicas dos estados para formalizar as propriedades. Portanto o algoritmo de verificação de propriedades opera sobre o grafo de estados do TS.

O grafo de estados de um $TS = (S, Act, \rightarrow, I, AP, L)$ é descrito por um dígrafo (V,E) de vértices V = S e arestas $E = \{(s, s') \in S \text{ x } S \mid s' \in Post(s)\}$. Em que $Post(s)$ denota o conjunto de estados sucessores a s.

O grafo de estados de um TS possui um vértice em cada estado de TS e uma aresta para cada par de estados s e s' sempre que s' for um sucessor direto de s no TS para alguma ação a.

O comportamento de um *transition system* é definido por um fragmento de execução, a alternância entre ações e estados. O resultado das execuções de um TS são chamados caminhos. Por sua característica de omitir as ações nos caminhos, o conceito de rastro (*trace*) é cunhado como o conjunto de sequências de proposições atômicas válidas na execução. Os rastros são palavras sobre o alfabeto 2^{AP}, que podem ser finitas em caso de existência de estados terminais ou infinitas caso contrário.

2.4.2 Formalização de requisitos

A fim de verificar formalmente as propriedades do modelo, deve-se formalizar os requisitos elaborados. Neste trabalho, será utilizada uma classe simples de propriedades que combinadas pretendem garantir características fundamentais de sistemas em tempo real e autônomos. A classe abordada é a de propriedades lineares no tempo (LT) [4].

A propriedade linear no tempo (LT) sobre um conjunto de proposições atômicas AP é caracterizada por um subconjunto de $(2^{AP})^{\omega}$, em que $(2^{AP})^{\omega}$ denota um conjunto de palavras obtido na concatenação das 2^{AP} palavras, ou seja, uma propriedade linear no tempo especifica o comportamento desejado do sistema em determinada situação descrita por proposições atômicas. Portanto, satisfazer uma propriedade linear no tempo sobre um TS, significa que a partir de determinado estado todos os caminhos subsequentes ao escolhido, respeitam à propriedade. No contexto deste trabalho utilizamos os conceitos de alcançabilidade e segurança operacional como tipos de propriedades.

Um estado $s \in S$ é dito alcançável em um TS se existe pelo menos um caminho, que começa no estado inicial e atinge o estado s com uma sequência finita de transições α:

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n = s \tag{2.2}$$

Propriedades de segurança afirmam que "nada de ruim pode acontecer", ou seja, buscam garantias no sistema de que estados proibidos não ocorram. *deadlocks* e invariantes são típicos

exemplos de propriedades de segurança. O *deadlock* é caracterizado pela ocorrência de um estado terminal no sistema representado pela composição paralela de todos os processos simultâneos, mesmo que um ou mais componentes não estejam em estado terminal localmente. Invariantes são propriedades LT definidas pela condição que deve ser respeitada por todos os estados alcançáveis.

Uma propriedade LT P_{inv} sobre um conjunto de proposições atômicas AP é invariante se existe uma fórmula lógica proposicional Φ sobre AP,

$$P_{inv} \;=\; \left\{\; A_0 A_1 A_2 \ldots \in (2^{AP})^{\omega} \mid \forall j \geq 0.\; A_j \models \Phi \right\} \tag{2.3}$$

em que Φ é chamada condição invariante de P_{inv} e o operador $\models$ define a satisfação da propriedade Φ sob o conjunto de estados A_j. Invariantes podem ser vistas como propriedades de estado.

2.4.3 Especificação e Validação

Linear Temporal Logic (LTL) foi proposta para especificação e validação de sistemas reativos, de forma que para cada instante de tempo existe somente um possível estado sucessor. Por outro lado, caminhos em um *transition system* trazem a noção de ramificações. Cada estado pode possuir vários sucessores diretos distintos, e por isso, distintas computações podem começar em um estado. Para especificar e validar *transition systems* que consideram tais ramificações, foi introduzida a *Computation Tree Logic* (CTL) [4].

Neste trabalho, por tratarmos de sistemas em tempo real, será utilizada uma extensão do CTL, o *Temporal Computation Tree Logic* (TCTL) que além das capacidadedes da CTL permite a noção de passagem de tempo por meio de restrições de tempo de relógio. A restrição de tempo de relógio sobre um conjunto C de tempos de relógio é formada de acordo com a gramática:

$$g ::=\; x < c \mid x \leq c \mid x > c \mid x \geq c \mid g \wedge g, \tag{2.4}$$

onde $c \in \mathbb{N}$ e $x \in C$.

As fórmulas em TCTL são classificadas entre fórmulas de estado e fórmulas de caminho. As fórmulas de estado sobre um conjunto de proposições atômicas AP são formadas de acordo com a gramática:

$$\Phi ::= \mathit{true} \mid a \mid g \mid \Phi \wedge \Phi \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi, \tag{2.5}$$

onde $a \in AP$, $g \in C$ e φ é uma fórmula de caminho definida por:

$$\varphi ::= \Phi \cup^{J} \Phi, \tag{2.6}$$

onde $J \subseteq \mathbb{R}_{\geq 0}$ é um intervalo cujos limites são números naturais.

A semântica aceita pelas ferramentas que utilizam TCTL para especificação e validação de modelos é baseada nos conceitos de *eventualmente* e *sempre*, definidos como mostrado a seguir.

eventualmente:

$$\exists\Diamond^J\Phi = \exists(true \cup^J \Phi) \tag{2.7}$$

$$\forall\Diamond^J\Phi = \forall(true \cup^J \Phi) \tag{2.8}$$

sempre:

$$\exists\Box^J\Phi = \neg\forall\Diamond^J\neg\Phi \tag{2.9}$$

$$\forall\Box^J\Phi = \neg\exists\Diamond^J\neg\Phi \tag{2.10}$$

J indica um intervalo de tempo em que Φ é satisfeito. No caso das *transition systems* que não possuem restrições de tempo de relógio o intervalo de tempo é assumido como $J = [0, \infty)$.

$\exists\Diamond\Phi$ lê-se "potencialmente satisfaz Φ",

$\forall\Diamond\Phi$ lê-se "Φ é inevitável",

$\exists\Box\Phi$ lê-se "potencialmente sempre Φ é satisfeito"e

$\forall\Box\Phi$ lê-se "invariavelmente Φ é satisfeito".

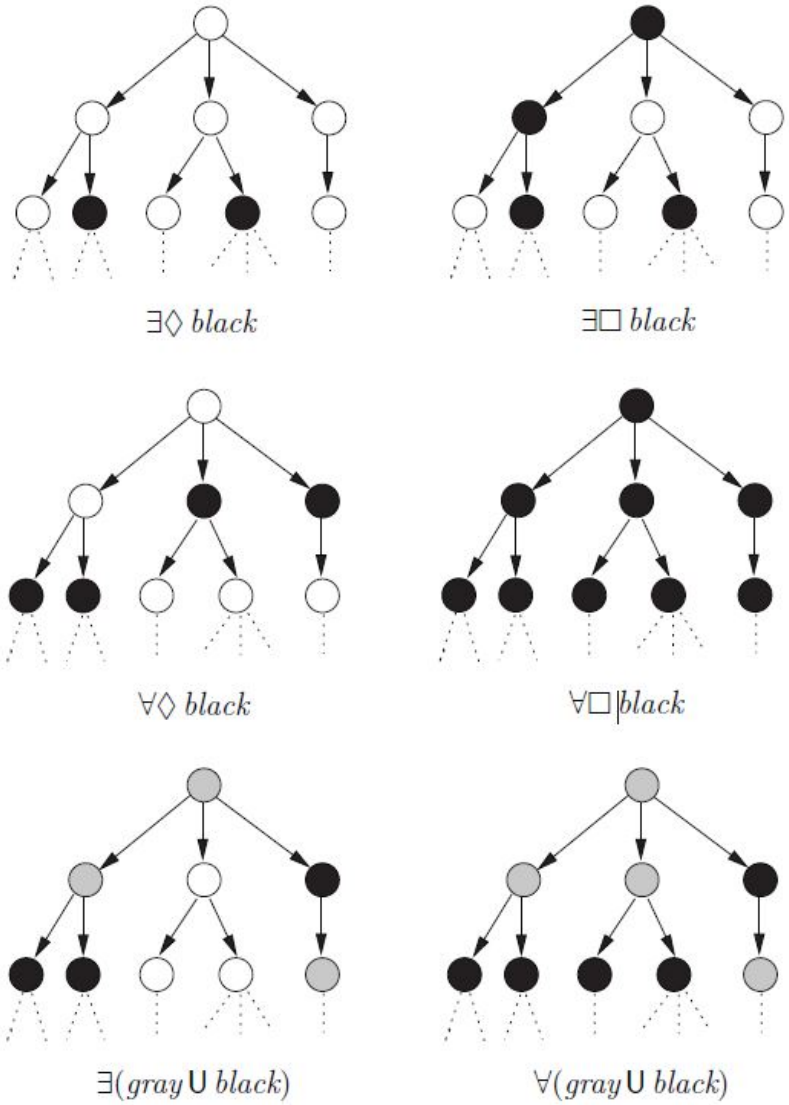


Figura 2.4: Esquemático de fórmulas básicas TCTL [4].

2.5 UPPAAL

Ao longo dos últimos anos diversas ferramentas para modelagem de sistemas em tempo real foram propostas. Entre elas, Redes de Petri estendidas, extensões temporais do *Communicating Sequential Processes* (CSPs), *Real-Time Logic* e UPPAAL [10]. Neste trabalho o UPPAAL, concebido em uma colaboração entre a *Uppsala University*, Suécia, e *Aalborg University*, Dinamarca, foi utilizado para fins de modelagem, simulação e verificação de uma aplicação de rede de sensores do corpo humano.

UPPAAL é uma ferramenta de modelagem, simulação e verificação de sistemas com requisitos de tempo real. É apropriada para sistemas que podem ser modelados como uma coleção de processos com estrutura de controle finita e relógios de valores compreendidos no conjunto dos reais, que se comunicam por canais e/ou variáveis compartilhadas [10].

2.5.1 Modelagem

A modelagem dos processos segue uma linguagem descritiva definida a partir de *transition systems* típicos do UPPAAL, as redes de autômatos temporais. O estado do sistema é definido pelos locais de todos os autômatos, e arestas são os elementos que ligam os locais de maneira unidimensional e definem o fluxo de passagem de estados. Cada autômato pode acionar uma aresta separadamente ou em sincronia com outro estado da mesma rede. A passagem do tempo de relógio associados ao tempo de processamento do local são modelados como atrasos no acionamento de arestas.

Cada aresta possui quatro atributos:

- **Seleção**: declaração de variáveis locais,
- **Guarda**: expressão booleana que impede acionamento da aresta em caso de falsa ou permite em caso de verdadeira,
- **Sincronização**: expressão de sincronização entre acionamento de arestas por meio de canais, e
- **Atualização**: expressão de atribuição.

Cada local possui dois atributos:

- **Nome**: string descritiva do estado e
- **Invariante**: expressão booleana que obriga o acionamento das arestas adjacentes ocorra antes de tornar-se falsa.

Um exemplo de dois autômatos com sincronização é mostrado na Figura 2.5.

Figura 2.5: Exemplo UPPAAL

O autômato da esquerda possui um só estado *init* de invariante $x \leq 20$, e uma aresta com guarda $x \geq 10$ e sincronização pelo canal *reset*, onde x foi declarado do tipo *clock* e *reset* do tipo *channel*. Já o autômato da direita possui dos estados a e b e as arestas, em que a direcionada a b possui um elemento de sincronização pelo canal *reset* e a direcionada a a possui atributo de atualização que incrementa a variável inteira *timer*.

O funcionamento da rede de autômatos do exemplo ocorre pelo acionamento do autômato da esquerda quando $10 \leq x \leq 20$ ativa a aresta do autômato da direita o obrigando a somar um à variável *timer*. A letra *C* no estado *b* indica que aquele é um estado *commited*, que impede passagem de tempo durante o processamento daquele estado é dito imediato.

2.5.2 Verificação de modelo

O principal objetivo da verificação formal é averiguar se o modelo atende os requisitos especificados. Da mesma forma que o modelo, os requisitos devem ser formalmente definidos. O UPPAAL utiliza uma versão simplificada do TCTL. Como no TCTL, a linguagem de assertivas do UPPAAL utiliza fórmulas de estado e fórmulas de caminho. As propriedades verificadas pelo UPPAAL são apresentadas na Tabela 2.1. Em que p e q são fórmulas de estado que indicam a propriedade a ser verificada.

Nome	**Assertiva**	**Equivalente a**
Possivelmente	$E\Diamond\ p$	-
Invariavelmente	$A\Box\ p$	$not\ E\Diamond\ not\ p$
Potencialmente sempre	$E\Box\ p$	-
Eventualmente	$A\Diamond\ p$	$not\ E\Box\ not\ p$
Leva a	$p\ \rightarrow\ q$	$A\Box\ (p\ imply\ A\Diamond\ q)$

Tabela 2.1: Propriedades no UPPAAL

2.6 OpenDaVINCI

OpenDaVINCI é um *framework* código aberto totalmente escrito em C++ que permite o desenvolvimento de estruturas para sistemas virtuais, em rede e ciber-físicos. Desenvolvido na *Chalmers University*, Suécia, o *software* é capaz de desempenhar papel de *middleware* para sistemas distribuídos em contextos de tempo real. Para isso ele disponibiliza os seguintes recursos:

- Módulos de *software* distribuídos com capacidade de processar dados em ambientes tempo real,
- Acionamento de módulos por interrupções de dados ou tempo,
- Manutenção centralizada dos módulos seguindo o protocolo de configuração de sistemas dinâmicos e modulares,
- Princípio de troca de dados baseia-se em *Publish/Subscribe* ou comunicação direta,
- Comunicação por protocolos TCP/UDP ou memória compartilhada e porta serial,
- Monitor de comunicação por *logs*,

- Estrutura de dados tratadas em tempo de compilação e *random-access serialization*,
- Filas, pilhas e *hash maps*, *thread-safe*, e
- Capacidade de execução supervisionada com controle de comunicação, tempo e agendamento de módulos.

É importante notar que essas características garantem integração com dispositivos físicos (sensores e atuadores). Neste trabalho foi utilizado o módulo *odsupercomponent* desenvolvido pelos criadores do projeto, que tem características de escalonador com algorítmo *first comes first served* (FCFS) e barramento para comunicação *Publish/Subscribe* entre os módulos.

2.6.1 Componente odsupercomponent

Desenvolvido em código aberto pelos próprios criadores do OpenDaVINCI, o *middleware odsupercomponent* é responsável pela gerência e execução em tempo real dos módulos, atuando como escalonador. Também funciona como barramento da arquitetura de comunicação *publish/subscribe* que redistribui aos receptores as mensagens encapsuladas como *containers* enviadas pelos publicadores no ciclo de execução anterior.

O escalonamento de módulos consiste no envio do sinal de *pulse* em forma de mensagem encapsulada em *container* que habilita o ciclo de execução do módulo, colocando-o em uma espécie de semáforo temporizado que espera o *pulse_ack* que indica final da execução. Em seguida, envia outro *pulse* ao módulo seguinte ou espera o período do ciclo terminar.

A configuração de execução do *odsupercomponent* é feita por linha de comando no momento da execução do módulo. As configurações das variáveis são:

- *cid* - identificador da conferência para agrupar os módulos,
- *configuration* - caminho do arquivo de configuração estática,
- *freq* - frequência base de execução dos módulos em Hz,
- *managed* - opção utilizada para política determinística do escalonador,
- *logFile* - nome do arquivo no qual serão salvas as mensagens de log,
- *logLevel <none, info, warn, debug>* - tipos de mensagens de log que devem ser salvas no arquivo de log,
- *realtime* - prioridade de processo no sistema operacional, e
- *verbose* - nível de verbosidade para impressão de mensagens na tela.

Os módulos plataforma central e nó sensor também podem ser configurados por linha de comando, porém limitados a definir o identificador da conferência (obrigatório) e a frequência de execução relativa à frequência configurada na execução do *odsupercomponent*.

A configuração da frequência do *odsupercomponent* tem papel fundamental, já que esta irá determinar o período de ciclo do escalonador.

A configuração *realtime* define um valor de [0,42], onde zero é mínimo e quarenta e dois máximo, para indicar a prioridade de processamento do programa ao sistema operacional. Essa opção só é possível em sistemas cujo *kernel* permite execução preemptiva e tempo real, e portanto necessita que a execução do módulo seja feita com permissão de administrador do sistema.

O *managed* indica que o escalonamento dos módulos dos protótipos será feito de maneira determinística, em que os módulos são executados sem interrupções e sequencialmente na ordem em que foram instanciados. Essa configuração permite duas opções: *simulation* e *simulation_rt*. O *simulation* define que após a execução do último módulo do ciclo, o primeiro irá começar. Já o *simulation_rt* coloca o protótipo em estado de espera até que o tempo de ciclo seja concluído, garantindo que os ciclos sejam executados sempre com mesmo período.

O *configuration* indica o caminho do arquivo de configuração estática carregado pelo *odsupercomponent* no momento em que é executado. Este é importante pelo fato de que pode-se colocar diferentes padrões de configurações utilizados pelos módulos da aplicação.

Capítulo 3

Desenvolvimento

3.1 Introdução

O capítulo de desenvolvimento explora cada detalhe da metodologia de controle para automação de sistemas de *software* com reposta em tempo real. Além das nuâncias da prototipação de sistemas autônomos no OpenDaVINCI e como a verificação formal de modelos contribui para o processo de validação da metodologia.

Na seção de pré-projeto, os requisitos formalizados a partir da descrição do estudo de caso são ilustrados pela árvore de objetivos. Então, a lógica do protótipo da BSN desenvolvido é descrita e exemplificada para todos os módulos implementados. Nesta seção são discutidos os procedimentos mais relevantes a cada uma das lógicas implementadas. E na seção de projeto do controlador, cada um dos passos para automação do protótipo da BSN é descrito com motivação das escolhas das tecnologias utilizadas e resultados obtidos nas validações tanto do modelo apresentado quanto da análise do protótipo.

Tendo em vista possíveis necessidades de automação de sistema para fins de otimização, qualidade, segurança, o presente capítulo exercita uma metodologia de controle para automação com um protótipo da BSN desenvolvido na fase de pré projeto.

3.1.1 Componentes da BSN

A BSN tem como principal objetivo detectar situações de emergência para reportar a unidades externas de modo que algum serviço de atendimento ao paciente seja disponibilizado. Por lidar com seres humanos com risco de saúde e cujo tempo de resposta é fator determinante na satisfação do objetivo principal, a BSN é classificada no conjunto de sistemas *hard realtime*. É importante ressaltar que o estudo de caso em questão considera somente a porção do *software* da aplicação, portanto questões relativas ao meio físico e *hardware* são abstraídas sem perda de generalidade para a solução.

Como desenhado na Figura 3.1, o sistema possui dois módulos: a plataforma central e os nós

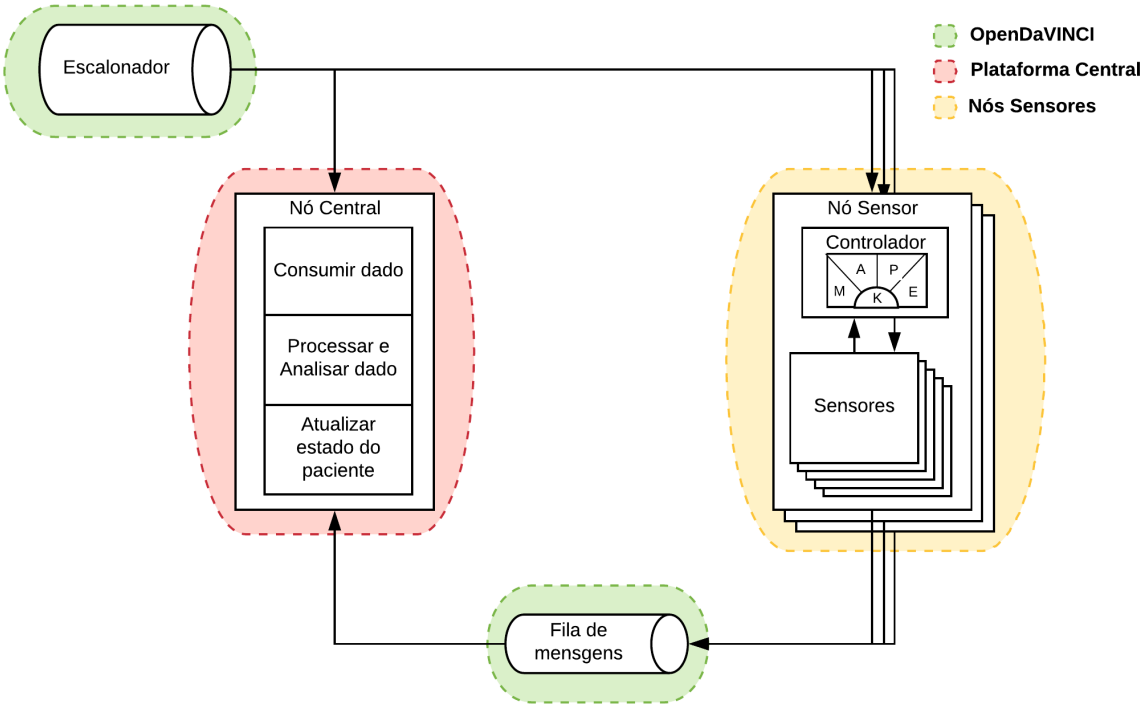


Figura 3.1: Arquitetura de implementação da BSN

sensores. Cada nó sensor possui um dispositivo sensor para captar os sinais vitais do paciente, um processador para analisar e classificar os sinais obtidos e uma unidade de comunicação para enviar dados na rede. Os tipos de sensores da BSN são:

- Termômetro,
- Eletrocardiógrafo, e
- Oxímetro de Pulso.

A plataforma central é o centralizador das informações enviadas pelos sensores. Portanto possui um registro de todos os sensores da BSN, um processador para estimar o estado do paciente a partir dos dados recebidos e um módulo de comunicação com a rede para receber informações. Além disso, ela possui um módulo de comunicação com o ambiente externo à BSN para enviar sinais em caso de estado de emergência detectado, e por fim um log com os dados relevantes do recebimento de dados dos sensores.

3.2 Pré-projeto

O Pré-projeto constitui-se de duas tarefas: (i)formalização dos requisitos e (ii) prototipação da BSN. A formalização utiliza o conceito de árvore de objetivos da Engenharia de Requisitos

Orientada a Objeitvos e a prototipação considera o desenvolvimento de um protótipo da BSN em cima do *framework* OpenDaVINCI.

3.2.1 Requisitos

O primeiro passo no desenvolvimento de um sistema fundamenta-se no levantamento de requisitos que representam seu comportamento. Dentre as várias técnicas de formalização de requisitos, escolhemos a árvore de objetivos pelo fato de que esta é diretamente mapeável na fase de escolha de objetivos do processo de controle para automação de sistemas.

Árvores de objetivos retratam um ator (*actor*), objetivos funcionais (*hard goals*), objetivos não-funcionais (*soft goals*), e tarefas (*tasks*). O objetivo raiz é o principal objetivo a ser atingido pelo ator. Estas representam hierarquicamente os objetivos que devem ser atingidos para satisfação do objetivo raiz, portanto na perspectiva *top-down* quanto mais profundos são os nós menos abstratos e mais próximos da operacionalização de fato estão. As tarefas representam a operacionalização dos objetivos superiores.

O comportamento da BSN é descrito pela árvore de objetivos da Figura 3.2 cujo nó raiz G1 representa o principal objetivo a ser cumprido: detecção de emergência. Para satisfazê-lo é necessário satisfazer um objetivo não-funcional G0 (Tempo de resposta menor que 250ms) e o objetivo funcional G2 (Estado de saúde do paciente detectado), ou seja, uma emergência somente é de fato detectada caso o tempo de processamento seja menor que 250 milisegundos e o sistema esteja ciente atual do estado do paciente.

Requisitos não-funcionais por natureza, não são diretamente operacionalizáveis. A satisfação de requisitos funcionais contudo, como o G2, necessitam da satisfação de seus refinamentos, neste caso o G4 (Sinais vitais processados) que por sua vez depende de G5 (Sinais vitais monitorados) e G6 (Sinais vitais analisados), ou seja, os sinais vitais só podem ser processados pela aplicação caso eles passem por processo de monitoramento e análise.

O requisito funcional G5 é operacionalizável pela tarefa T1 (Monitorar sinais vitais), que é de fato operacionalizada por meio das sub-tarefas T1.1 (Coletar dados dos sensores), T1.2 (Transferir dados coletados) e T1.3 (Persistir dados). Desta maneira, para monitorar os dados dos sensores deve-se primeiro coletar os dados dos sensores, transferí-los e finalmente persistí-los em registros.

Diferentemente das outras tarefas a T1.1, possui decomposições *ou*, que determina a necessidade de somente uma sub-tarefa ser operacionalizada para que esta seja também. Portanto T1.1 é operacionalizada por meio de T1.11 (Capturar dados de SPO2) ou T1.12 (Capturar dados de ECG) ou T1.13 (Capturar dados de TEMP). Em suma, diz-se que os dados foram coletados quando os dados do oxímetro, eletrocardiógrafo ou termômetro forem de fato coletados. Por outro lado, o requisito G6 é operacionalizado pela tarefa T2 (Analisar sinais vitais), que por sua vez é operacionalizada pelas sub-tarefas folhas T2.1 (Processar dados dos sensores) e T2.2 (Detectar estado do paciente).

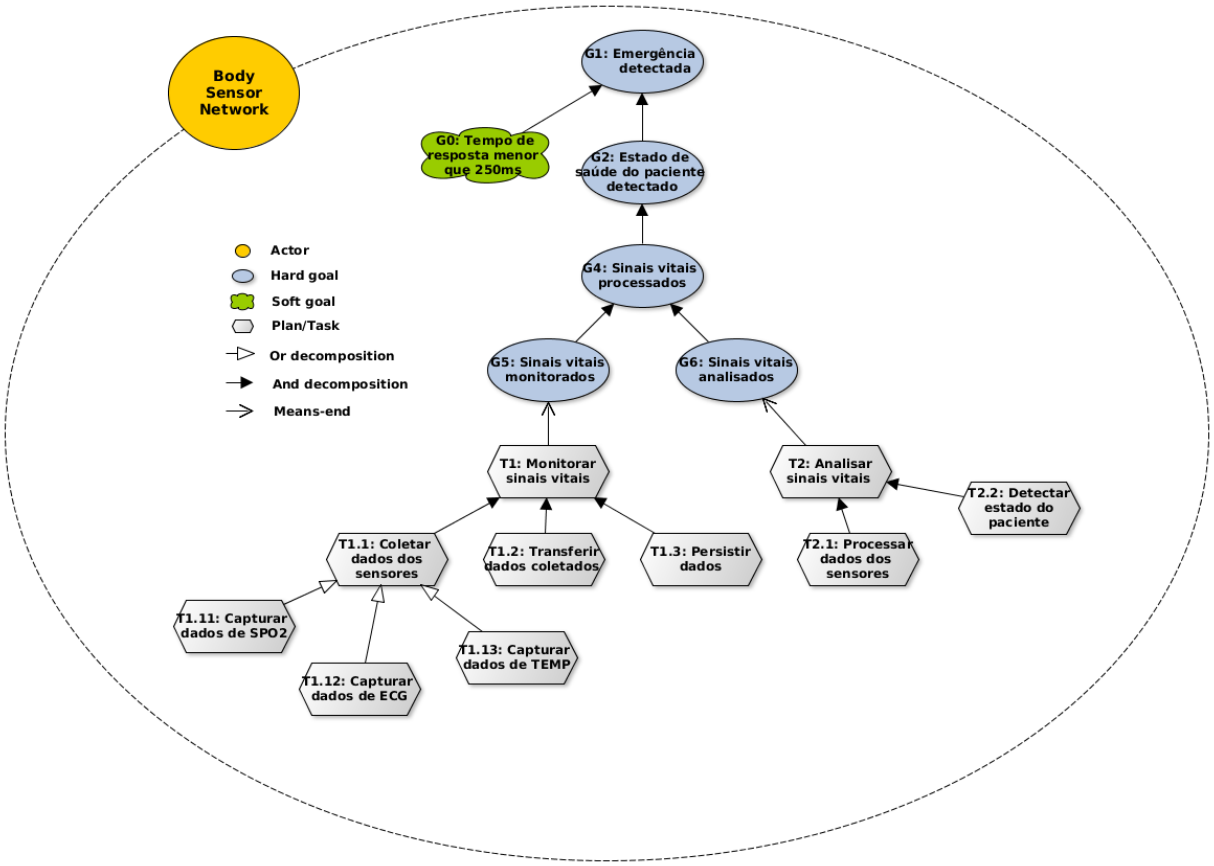


Figura 3.2: Árvore de objetivos

3.2.2 Protótipo

O protótipo da BSN foi desenvolvido baseado nos requisitos da árvore de objetivos a fim de aplicar e validar a metodologia de projeto de controlador para automação do *software* em questão. Em vista das características de sistema de *software* distribuído e requisitos de tempo real, o *framework* OpenDaVINCI foi escolhido como tecnologia para implementar o protótipo da BSN. Codificado em C++ o OpenDaVINCI implementa um *middleware* que escalona módulos de *software*, acionados por interrupções temporais ou de dados, com configurações de execução de tempo real. Além disso, o *middleware* é capaz de atuar como barramento da arquitetura orientada a serviços *publish/subscribe* para comunicação entre os módulos.

O protótipo da BSN foi codificado em C++ de acordo com a estrutura de módulos distribuídos do OpenDaVINCI. Com o intuito de operacionalizar as tarefas da árvore de objetivos, dois módulos e uma biblioteca foram implementados: o nó sensor, a plataforma central e a biblioteca *libopenbasn*. Os módulos foram implementados como herdeiros da classe *TimeTriggeredConferenceModule*, que representa os módulos acionados por tempo. A biblioteca contém uma única classe que define a mensagem que carrega informações do sensores e é utilizada na comunicação entre os módulos.

O diagrama da Figura 3.3 ilustra o ciclo de vida básico do módulo acionado por interrupções de tempo no OpenDaVINCI. No qual o círculo verde representa a execução do módulo e o ver-

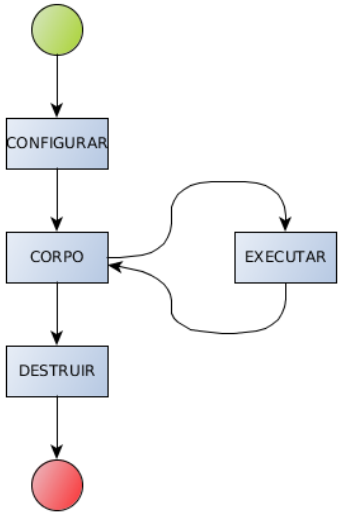


Figura 3.3: Ciclo de vida de módulos acionados por interrupções temporais

melho seu encerramento. A partir da execução, o módulo acionado por temporização passa por um método *setup* onde podem ser feitas configurações estáticas do módulo. Em seguida, o corpo do módulo é executado, onde ocorre o processamento do algoritmo implementado com o comportamento do mesmo, representado pelo bloco "Executar", que fica em loop infinito recebendo periódicamente sinais do escalonador para acionar o código interno. Quando um sinal de parada é enviado, o módulo passa pelo processo de *teardown*, para destruição de prováveis alocações durante o programa, e então o módulo é encerrado.

O módulo escalonador da BSN permite a configuração da política de escalonamento e frequência ($freq$) de execução. Na BSN a política de escalonamento utilizada é não-preemptiva, estática e offline. Não-preemptiva indica que o escalonador não interrompe a execução do módulo, estática por atribuir os parâmetros antes da execução dos módulos e offline por planejar o escalonamento dos módulos antes da execução. A frequência $freq$ utilizada é de 10Hz com intuito de satisfazer o requisito não-funcional G0 da árvore de objetivo. A Figura 3.4 ilustra o escalonamento da BSN com três sensores instanciados e uma plataforma central.

A Figura 3.5 ilustra uma visão alto nível dos componentes da BSN na perspectiva da arquitetura de comunicação do OpenDaVINCI, onde os nós sensores são os publicadores de mensagens e a plataforma central é o consumidor das mensagens.

3.2.2.1 Nó sensor

Os nós sensores são responsáveis por captar dos dados dos sensores, filtrá-los, processá-los, classificá-los, analisá-los e por fim enviá-los pela rede. Para fins de simulação, foi implementada uma cadeia de Markov de três locais ("baixo", "moderado"e "alto") que dado o estado de risco atual do nó sensor, o próximo estado é gerado pelas probabilidades indicadas pelas arestas de

[1]O escalonamento não representa os tempos computacionais dos módulos. Estes estão representados qualitativamente.

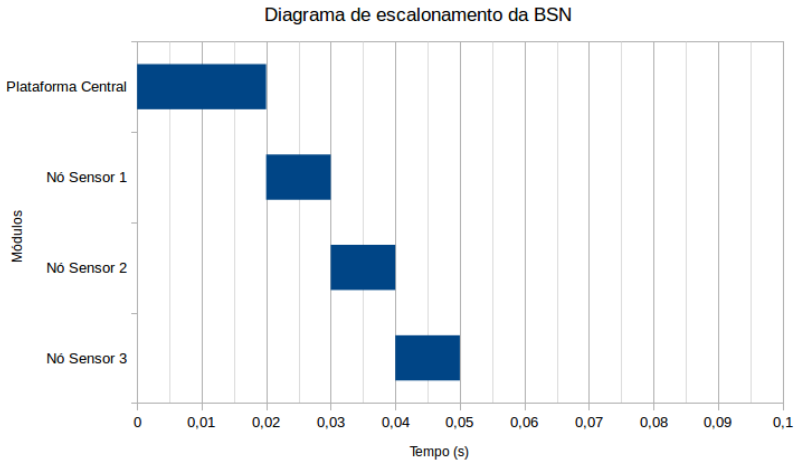


Figura 3.4: Diagrama de escalonamento dos módulos na BSN[1]

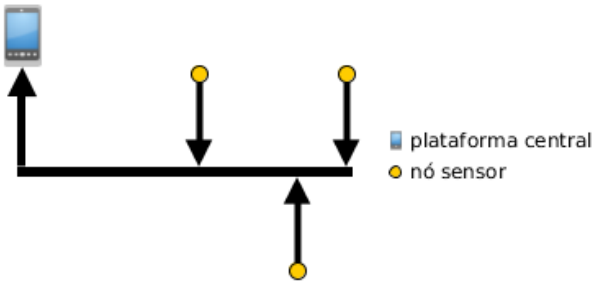


Figura 3.5: Componentes da BSN arquitetura de comunicação

transição. Portanto, a contribuição dos nós sensores na operacionalização das tarefas da árvore de requisitos, dá-se via T1.11, T1.12 ou T1.13, dependendo somente do tipo de sensor atribuído ao nó sensor em questão. A classe do módulo nó sensor possui quatro atributos:

1. Identificador,
2. Tipo do sensor,
3. Estado do nó sensor, e
4. Fila de dados.

O *identificador* carrega o valor inteiro para identificação do módulo. O *tipo de sensor* associado carrega um valor inteiro correspondente aos sensores: (1) Termômetro, (2) Eletrocardiógrafo e (3) Oxímetro de pulso. O *estado* é do tipo *string* e representa o estado de risco que o dado atual do sensor representa ao paciente: "risco baixo", "risco moderado"ou "risco alto". E a fila de dados que é uma estrutura do tipo *deque<string>* onde são armazenados os cinco últimos estados do nó sensor.

A Figura 3.6 ilustra o diagrama de fluxo das atividades codificadas no módulo do nó sensor, cujo funcionamento respeita o ciclo de vida básico do módulo acionado por tempo. A etapa de

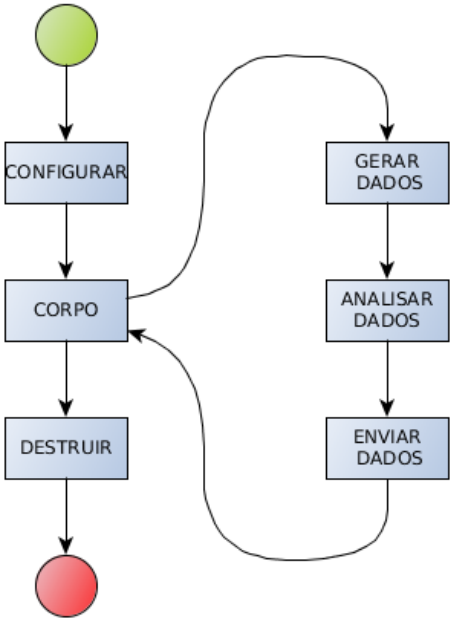


Figura 3.6: Ciclo de vida do módulo nó sensor.

configuração do módulo consiste em atribuir o tipo de sensor que está associado a ele. A regra de associação do tipo do sensor ao módulo nó sensor segue o padrão:

- Identificador for igual a zero o tipo do sensor é termômetro,
- Identificador for igual a um o tipo do sensor é eletrocardiógrafo, e
- Identificador for igual a dois o tipo do sensor é oxímetro de pulso.

A etapa destruir neste caso não é utilizada. O corpo do módulo nó sensor espera constantemente pelo sinal de autorização do escalonador, que ocorre a uma frequência ($freq$), e o algoritmo interno do módulo possui três procedimentos sequênciais: (i) gerar dados, (ii) analisar dados e (iii) enviar dados. O procedimento de geração de dados obedece à cadeia de Markov geradora de novos estados de risco para o nó sensor, baseado no estado atual e nas probabilidades de transição definidas, como ilustrado pela Figura 3.7. Por exemplo, caso o estado de risco atual do nó sensor seja "baixo", o algorítmo gera um número inteiro de um a cem e a chance de retornar "baixo"é de 70%, "moderado"é 25% e "alto"é 5%.

A etapa seguinte consiste na análise do estado gerado na etapa anterior e possui três procedimentos sequenciais:

1. inserção do novo estado risco na fila de dados,
2. contabilização do número de estados de risco de cada classificação, e
3. atribuição do estado de risco com três ou mais ocorrências ao estado de risco atual do nó sensor.

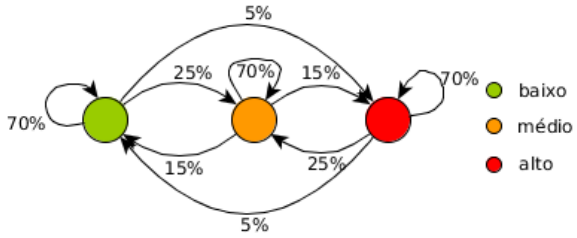


Figura 3.7: Autômato probabilístico gerador de estados do sensor

Em suma, o novo estado gerado pela cadeia de Markov é inserido na primeira posição da fila de dados (o elemento do final da fila é perdido em consequência) e o procedimento contabiliza os estados de risco presentes na fila de dados. Caso algum elemento se repita por três vezes ou mais o estadod e risco do nó sensor receberá seu valor. Como ilustra a Figura 3.8.

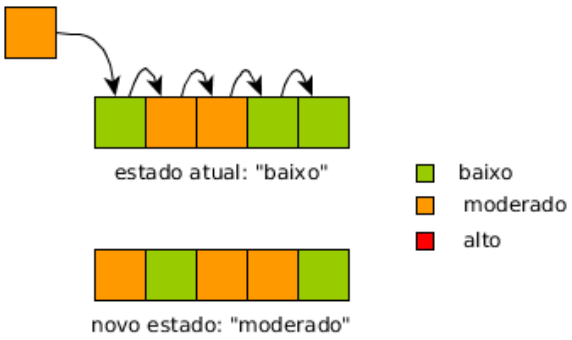


Figura 3.8: Análise do estado de risco do nó sensor

E finalmente prepara-se a mensagem que contém: (i) identificador do nó sensor, (ii) tipo de sensor do nó sensor, (iii) estado do nó sensor atualizado, e (iv) a *timestamp* do instante em que os dados foram gerados. A mensagem é então encapsulada na estrutura de *container* do OpenDaVINCI e enviada com o método *Send*.

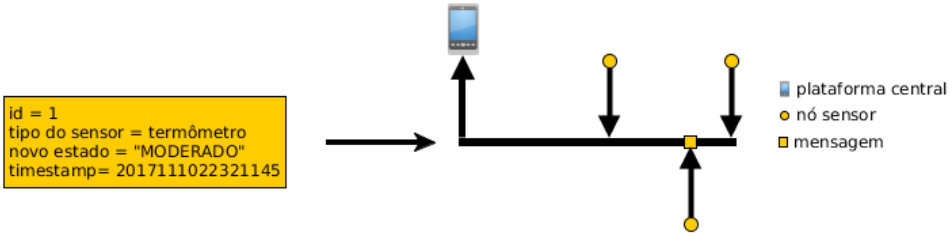


Figura 3.9: Comportamento de envio de dados do nó sensor

A Figura 3.9 ilustra o comportamento do envio de mensagens do nó sensor na arquitetura do OpenDaVINCI cuja caixa amarela à esquerda exemplifica uma mensasgem, e à direita a mensagem em processo de envio na arquitetura.

3.2.2.2 Plataforma central

A plataforma central é responsável pelo consumo das mensagens enviadas pelos nós sensores. Em posse de novos dados, esta os atribui ao registro do nó sensor remetente, recalcula o estado de saúde do paciente baseado nas informações registradas para, enfim persistir os dados atuais salvos na plataforma. Portanto, a contribuição da plataforma central na operacionalização das tarefas da árvore de objetivos dá-se via tarefas T1.3, T2.1 e T2.2. A classe da plataforma central possui cinco atributos:

1. Identificador,
2. Fila de mensagens,
3. Estado do paciente,
4. Registro de sensores e estados do nó sensor, e
5. Arquivo.

O *identificador* é único e carrega o valor inteiro correspondente ao módulo. A *fila de mensagens* é uma fila do tipo *FIFOQueue* própria do OpenDaVINCI e que armazena as mensagens (*containers*) recebidos naquele ciclo. O *estado do paciente* é do tipo *string* e pode assumir valores: "ruim", "médio"ou "bom". O registro de sensores é uma estrutura do tipo *map<uint32_t, string>* que guarda os tipos de sensores (e.g. termômetro, electrocardiógrafo, oxímetro) associado ao estado de risco atual do respectivo nó sensor. E o *arquivo* guarda um ponteiro do arquivo que vai ser preenchido com dados das informações registradas em cada ciclo.

A Figura 3.10 ilustra o diagrama de fluxo das atividades codificadas no módulo plataforma central, cujo funcionamento respeita o ciclo de vida básico do módulo acionado por tempo. Na configuração do módulo duas operações são efetuadas: (i) indicado à fila de mensagens quais tipos de mensagens podem ser recebidas e (ii) o arquivo para persistência dos dados é aberto. Na etapa de destruição o arquivo é fechado e salvo. O corpo do módulo espera constantemente pelo sinal de autorização (*release*) do escalonador que, ocorre na frequência configurada ($freq$). O módulo verifica então a existência de mensagens para serem consumidas, em caso positivo o algoritmo interno do módulo executa os quatro procedimentos sequenciais: (i) consumir dado, (i) atualizar registros, (iii) detectar estado crítico, e (iv) persistir os dados. Em caso negativo a execução do módulo o coloca em estado de espera novamente.

O primeiro procedimento consiste simplesmente de consumação da mensagem, pela atribuição do conteúdo em um objeto de tipo *SensorNodeData*definido na biblioteca *libopenbasn*. Em posse do objeto com as informações da mensagem, a plataforma central calcula e atualiza o estado de saúde do paciente. O cálculo do estado do paciente fundamenta-se na atualização do registro de sensores com o estado de risco do nó sensor recebido, e em seguida varredura deste para efetuar o somatório dos pesos relativos descritos na Tabela 3.1, e finalmente avaliar o estado de saúde do paciente aplicando a regra para avaliação da Tabela 3.2, onde x representa o somatório dos pesos dos estados dos sensores registrados. Como ilustrado na Figura 3.11.

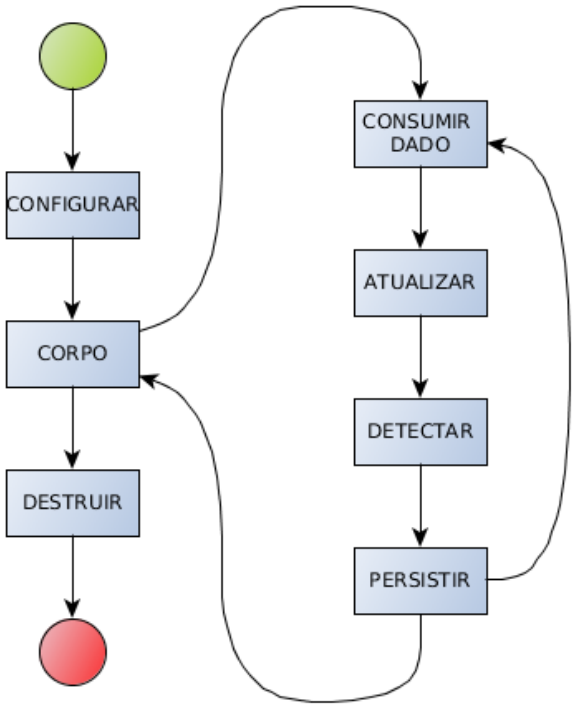


Figura 3.10: Ciclo de vida do módulo plataforma central

Estado do sensor	Baixo	Moderado	Alto
Peso	0,1	1	5

Tabela 3.1: Estados dos sensores e pesos

Regra de classificação	$0 \leq x < 1$	$1 \leq x < 5$	$5 \leq x < 20$
Estado do paciente	Ruim	Médio	Bom

Tabela 3.2: Regra para avaliação do estado do paciente

Então, o estado do paciente avaliado é atribuído à respectiva variável. Como mostra o texto da parte (2) da Figura 3.11. A etapa seguinte avalia o valor atual do estado do paciente, caso seja igual a "ruim"uma mensagem é impressa na tela indicando que uma situação de emergência foi alcançada. A implementação simplória desse procedimento é motivada pela ideia de prova de conceito, procedimentos complexos de envio de mensagens à serviços externos está fora do escopo do projeto.

O último procedimento antes da plataforma voltar ao estado de espera é a persistência dos dados em formato de tabela .xls cujo cabeçalho consiste de:

[2]A medida dos valores de tempo é relativa ao *timestamp* obtido no momento do procedimento de configuração da plataforma central

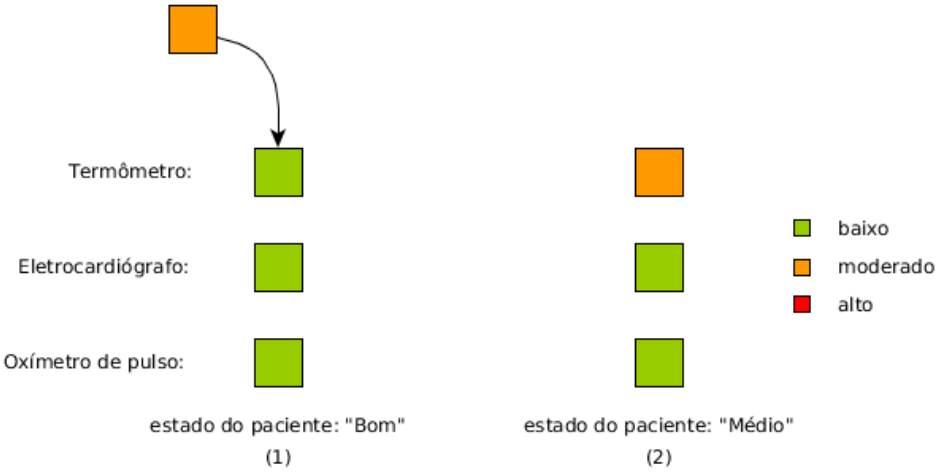


Figura 3.11: Cálculo e avaliação do estado do paciente

Cabeçalho	Possíveis valores	Descrição
ID do nó sensor	0 a 2	Identificação do nó sensor que enviou o dado
Estado do Termômetro	{"baixo","moderado","alto"}	Representa o estado de risco do nó sensor do termômetro
Estado do ECG	{"baixo","moderado","alto"}	Representa o estado de risco do nó sensor do eletrocardiógrafo
Estado do Oxímetro	{"baixo","moderado","alto"}	Representa o estado de risco do nó sensor do oxímetro de pulso
Estado do Paciente	{"bom","médio","ruim"}	Representa o estado atual do paciente
Enviado às (s)	$\mathbb{R}$	Tempo em segundos do instante que a mensagem é enviada pelo nó sensor
Recebido às (s)	$\mathbb{R}$	Tempo em segundos do instante que a mensagem é recebida pela plataforma central
Processado às (s)	$\mathbb{R}$	Tempo em segundos do instante que a mensagem é processada pela plataforma central

Tabela 3.3: Cabeçalho e possíveis valores dos dados persistidos[2]

3.3 Projeto

Até esse ponto, a aplicação da BSN funciona como um sistema de *software* de monitoramento de sinais vitais para detecção de emergência com tempo de resposta menor que 250 milisegundos. Nesta seção a metodologia de controle para automação de processos é aplicada com respeito às

peculiaridades do tipo de sistema que se deseja controlar: sistema de *software* com requisitos de tempo real [1]. A sequência de passos resume-se a:

1. Escolha do(s) objetivo(es),
2. Escolha do(s) atuador(es),
3. Modelagem formal do sistema,
4. Projeto do controlador,
5. Implementação e teste do controlador no modelo, e
6. Implementação e teste do controlador na planta.

Em que (1) e (2) estão necessariamente associados às escolhas de projeto baseadas nos requisitos de automação propostos pelos projetistas, cargos superiores na hieraquia corporativa ou consumidores do produto. Em seguida (3), o sistema é modelado por técnicas convenientes ao tipo de sistema, natureza dos objetivos e atuadores levantados na etapa anterior. O passo (4) é o projeto de fato do controlador a automatizar o processo, a técnica escolhida neste passo é fortemente dependente dos itens anteriores. Os dois últimos (5) e (6) resumem-se à implementação do controlador no modelo e validação, para enfim, implementar e validar o comportamento final da planta controlada.

3.3.1 Objetivos e atuadores

A automação de processos pode ter vários motivos: otimização, robustez, qualidade, segurança, tempo mínimo. No caso da BSN, os nós sensores enviam mensagens constantemente à plataforma central com periodicidade fixa e igual ao período de ciclo configurado do escalonador, mesmo em casos cujo estado de risco do sensor é baixo, e portanto pode provocar perdas por *overhead* e taxas altas de perdas de pacotes. Decidiu-se então, que o processo deve ser otimizado para alterar a frequência de captação de dados em virtude do estado atual do nó sensor. Dois requisitos devem ser adicionados à árvore de objetivos, como ilustrado pelos objetivo G3 (Frequência de envio de dados alterada) e sua respectiva tarefa T3 (Alterar frequência de execução) na Figura 3.12.

Entretanto, a escolha do atuador é fundamentalmente associada à do objetivo, já que este é a entidade que fará possível a automação do processo para atingi-lo. Em ambientes de *software* a escolha do atuador é uma tarefa menos complexa quando comparada a outros sistemas. Afinal, desde que o objetivo não dependa de configurações estáticas efetuadas na execução do programa (e.g. *release time*) pode-se codificar uma variável de guarda específica para atuar no comportamento do sistema controlado. No caso da automação da BSN definimos:

> Objetivo: Otimizar o consumo de bateria pelo processo de captura de dados baseado no estado do nó sensor, e
>
> Atuador: Variável de guarda que permite ou não a execução do módulo naquele ciclo de escalonamento.

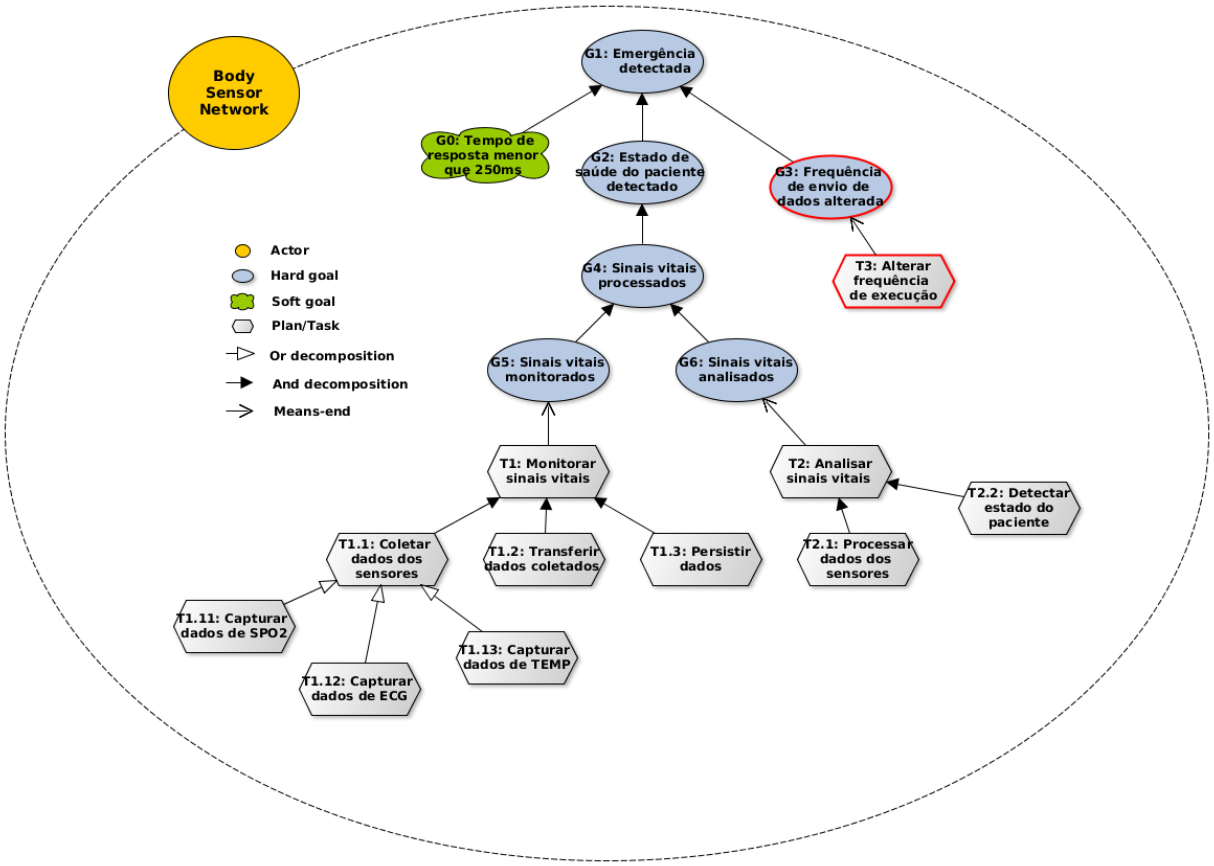


Figura 3.12: Árvore de objetivos para BSN autônoma

3.3.2 Modelagem Formal do sistema no UPPAAL

Em projetos de controladores a fase de modelagem do sistema é fundamental para correto funcionamento do controlador quando implementado na aplicação real. A modelagem de sistemas de *software* pode ser complexa quando não impossíveis no domínio contínuo, porque estes operam em ambientes variados e flexíveis que em grande parte das aplicações possuem características não-determinísticas. Portanto, abstraimos do comportamento mais baixo nível do funcionamento de um *software* no sistema computadorizado e tratamos do problema na perspectiva de sistemas a eventos discretos.

Utilizou-se neste trabalho linguagens formais (e.g. autômatos) para modelar o sistema em questão. Todavia, a característica de tempo real da BSN limita as possíveis linguagens e assim nos remete a escolha entre modelos temporizados. Em vista dos processos subsequentes de verificação de modelos, a escolha da ferramenta na qual serão modelados o sistema deve ser adiantada. Para este trabalho escolhemos a ferramenta UPPAAL para modelagem e verificação de autômatos temporais, tanto pela riqueza da sintaxe de implementação quanto pelas capacidades do próprio verificador.

O modelo da BSN deve levar em consideração a capacidade de relacionar o objetivo da automação ao atuador, analogamente ao processo de obtenção das funções de transferência no domínio

contínuo. Portanto características intrínsecas ao comportamento interno dos módulos do protótipo devem ser levadas em consideração no processo de modelagem. Abstrações do escalonador de módulos do OpenDaVINCI também devem ser endereçadas visto que a frequência de execução do mesmo é fundamental para a satisfação do requisito de tempo de resposta menor que 250ms (G0). A arquitetura de comunicação do protótipo também deve ser endereçada como operacionalização da tarefa (T1.2) de transferência dos dados, e indiretamente objetivo de detecção de emergência (G1).

3.3.2.1 Escalonador de módulos

A modelagem do escalonador de módulos depende da política de escalonamento do OpenDaVINCI, como mencionando anteriormente: não-preemptiva, estática e offline. Ou seja, em um ciclo de execução todos os módulos serão executados completamente, em ordem pré-estabelecida com parâmetros atribuídos antes da execução. O comportamento do escalonador é descrito pelo autômato temporal da Figura 3.13 e as variáveis descritas na Tabela 3.4.

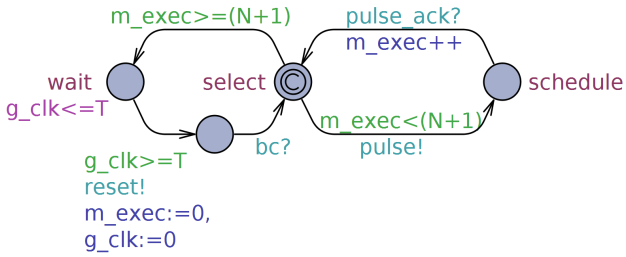


Figura 3.13: Autômato temporal modelo do estalonador

O escalonador executa sequencialmente, pelo envio e recebimento de sinais nos canais *pulse* e *pulse_ ack*, respectivamente, a todos os módulos ativos e então fica em estado de espera (*wait*) até que o tempo de ciclo seja atingido. Quando este é atingido, o escalonador envia um sinal *reset* a todos os módulos para prepará-los para o início de outro ciclo, e novamente entra em estado de espera. Deste, sairá somente quando receber o sinal *bc* indicando que as mensagens do ciclo passado foram redistribuídas aos módulos. Atingindo o estado inicial pronto para um novo ciclo.

3.3.2.2 Buffer

O processo de envio e replicação de mensagens do OpenDaVINCI foi modelado pela estrutura de fila de tamanho finito, o *buffer*. O *buffer* recebe, armazena e redistribui as mensagens enviadas por publicadores aos módulos receptores. O comportamento do *buffer* é descrito pelo atômato temporal da Figura 3.14 e as variáveis descritas na Tabela 3.5.

Durante a execução dos módulos, o buffer fica em estado de recebimento de mensagens (*receive*), onde a cada mensagem recebida a função *insert(sd)* insere o dado recebido no array (*m_ puffer*). Mediante recebimento do sinal *reset* o *buffer* entra em modo de *bcast*, nesse estado ele remove os itens armazenados pelo início da fila e envia via canal *broadcast* a todos os módulos do sistema até que a fila fique vazia. O estado *shiftdown* aplica a correção ao array utilizado para representar a

Nome	Tipo	Descrição
T	int	Período em (μs) de ciclo do escalonador.
g_clk	clock	Relógio que determina o tempo do ciclo do escalonador.
N	int	Número de nós sensores ativos.
m_exec	int	Número de módulos executados no ciclo.
pulse	chan	Canal ponto-a-ponto de ativação dos módulos.
pulse_ack	chan	Canal ponto-a-ponto que informa fim de execução do módulo.
reset	broadcast chan	Canal *broadcast* coloca os módulos em estado de pronto para execução.
bc	broadcast chan	Canal *broadcast* que informa fim de redistribuição de mensagens.

Tabela 3.4: Variáveis do modelo do escalonador

fila. Ao finalizar o processo de *bcast* o buffer envia um sinal *bc*indicando ao escalonador que todas as mensagens foram redistribuídas e estepode recomeçar a executar os módulos no novo ciclo.

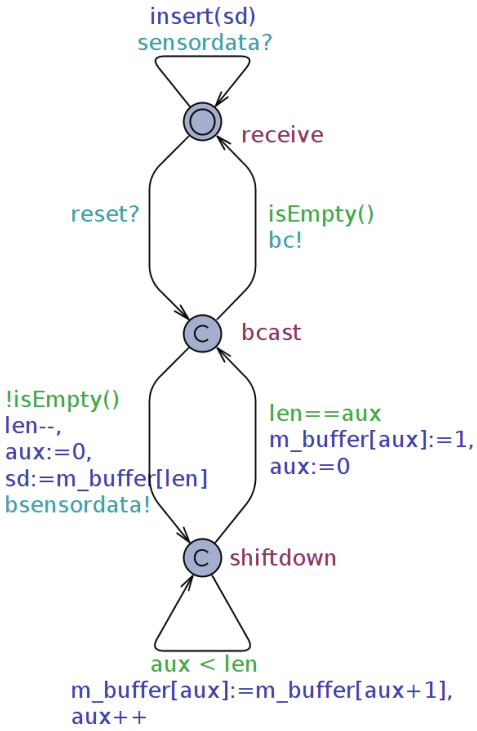


Figura 3.14: Autômato do modelo do buffer

3.3.2.3 Módulos

Os módulos são os agentes operacionalizadores das tarefas (T1.1, T1.3, T2.1 e T2.2) e por isso a modelagem de características fundamentais e únicas de cada um devem ser modeladas para garantir correta implementação do controlador. No entanto, ambos são acionados pelo escalonador e deve-se modelar também a porção da sincronização entre eles. O comportamento básico entre os módulos é então modelado de acordo com o autômato temporal da Figura 3.15 e respectiva tabela de variáveis Tabela 3.6.

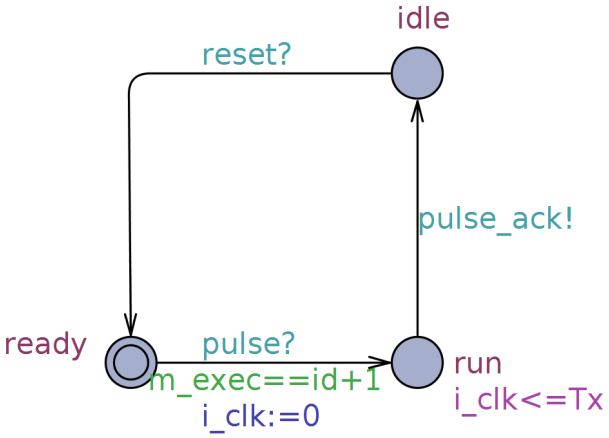


Figura 3.15: Autômato do modelo básico do módulo

O ciclo de vida básico de um módulo na perspectiva do escalonador consiste de três estados: *ready*, *run* e *idle*. O *ready* indica que o módulo está aguardando o sinal *pulse* do escalonador

Nome	Tipo	Descrição
m_puffer	int[N]	Array de inteiros de tamanho N que será utilizado no armazenamento de mensagens.
len	int	Variável utilizada para guardar o número de espaços ocupados no array.
aux	int	Variável para auxiliar das operações no array.
sd	int	Variável para representar o dado enviado.
sensordata	chan	Canal ponto-a-ponto que indica envio de dados *sd*.
reset	broadcast chan	Canal *broadcast* coloca os módulos em estado de pronto para execução.
bc	broadcast chan	Canal *broadcast* que informa fim de redistribuição de mensagens.
bsensordata	broadcast chan	Canal *broadcast* que redistribui uma mensagem ao sistema.

Tabela 3.5: Variáveis do modelo do buffer

para começar a execução. O *run* indica que o módulo está em execução. E o *idle* indica que o módulo já foi executado naquele ciclo e espera por um sinal *reset* para voltar ao estado *ready*. É importante notar que a passagem do tempo no estado *run* está limitada à invariante $i\_clk \leq Tx$, utilizada para impedir que o relógio interno ultrapasse o limite Tx. E esta, representa para fins de verificação do modelo, o tempo de processamento do módulo.

O comportamento específico de cada módulo é modelado pelo processo *run*. De modo que quando relevante ao processo de projeto do controle ou de verificação do modelo, pode-se implementar a lógica da execução do módulo por meio de um apêndice ao estado *run* com fim apontando para o próximo estado *idle* e enviando um sinal *pulse_ ack*. No caso do protótipo do OpenDaVINCI implementado, dois módulos devem ser modelados: o nó sensor e a plataforma central.

O modelo dos nós sensores é uma especificação do modelo básico da Figura 3.15 que operacionaliza a tarefa (T1.1) da árvore de objetivos, Figura 3.2. Seu comportamento é modelado pelo autômato temporal da Figura 3.16 e respectiva tabela de variáveis Tabela 3.7.

Dito que o estado *run* modela o tempo de processamento de todos os processos representados pelos estados que representam o comportamento da execução do nó sensor juntos, é necessário colocar estes com a bandeira *commited*, que impede passagem de tempo naquele estado. NO

Nome	Tipo	Descrição
Tx	int	Período em (μs) estimado como tempo de processamento do módulo.
i_clk	clock	Relógio interno do módulo.
m_exec	int	Número de módulos executados no ciclo.
pulse	chan	Canal ponto-a-ponto de ativação dos módulos.
pulse_ack	chan	Canal ponto-a-ponto que informa fim de execução do módulo.
reset	broadcast chan	Canal *broadcast* coloca os módulos em estado de pronto para execução.

Tabela 3.6: Variáveis do modelo básico do módulo

Nome	Tipo	Descrição
Texecs	int	Período em (μs) estimado como tempo de processamento do módulo.
m_state	int	Variável que guarda o estado do nó sensor.
sd	int	Dado enviado pelo módulo.
sensordata	chan	Canal ponto-a-ponto que indica envio de dados.

Tabela 3.7: Variáveis do modelo do nó sensor

entanto, no caso do estado *send* o modo *urgent* foi utilizado no lugar de *commited* porque todas as transições que levam a ele, enviam um sinal ponto-a-ponto com a mensagem do estado do sensor, e no estado de recebimento do buffer a prioridade deve ser dele para que nenhum dado seja perdido. Então, optou-se por deixar *urgent* nesse estado.

Quando executado, o nó sensor passa por três estados: *generate*, *analysis* e *send*. No primeiro, a diretiva $rand\_category : int[1, 3]$ randomiza um número de um a três, e atribui à variável que marca o estado do nó. A segunda possui sentido bem demarcado somente no protótipo e a terceira é precedida pelo envio do dado randomizado no estado *generate* ao buffer.

É importante ressaltar que o envio de dados no UPPAAL pode ser feito por meio de variável compartilhada. A variável *sd* foi declarada em escopo global e por isso pode ser acessada por qualquer módulo, então no envio do sinal *sensordata*, o módulo que o recebe é capaz de obter o valor de *sd*.

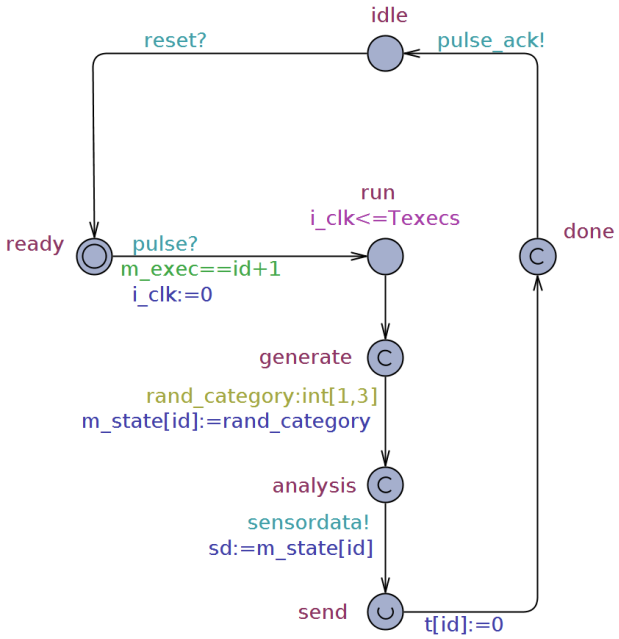


Figura 3.16: Autômato do modelo nó sensor

O modelo da plataforma central é uma especificação do modelo básico da Figura 3.15 que operacionaliza as tarefas T1.3, T2.1 e T2.2 da árvore de objetivos, Figura 3.2. Além de representar os estados pertinentes ao comportamento do código no OpenDaVINCI, este é estendido com dois outros estados utilizados na verificação de modelos: o *detected* e o *receive*, pela alcançabilidade de estados. O comportamento do módulo é descrito autômato temporal da Figura 3.17 e as variáveis descritas na Tabela 3.8.

Nome	**Tipo**	**Descrição**
Texecb	int	Período em (μs) estimado como tempo de processamento do módulo.
m_puffer	int[]	Array de inteiros que armazena mensagens.
m_health_status	int	Variável que guarda o estado do de saúde do paciente.
sd	int	Dado enviado pelo módulo.
bsensordata	broadcast chan	Canal *broadcast* que indica distribuição de mensagens pelo sistema.

Tabela 3.8: Variáveis do modelo da plataforma central

Durante o estado de preparação para executar (*ready*) o modelo pode receber dados distribuídos pelo *buffer* geral e armazená-los no seu *buffer* interno por meio da função *insert*. Para consumir os dados, a função *dequeue()* é utilizada. O comportamento da plataforma central consiste em

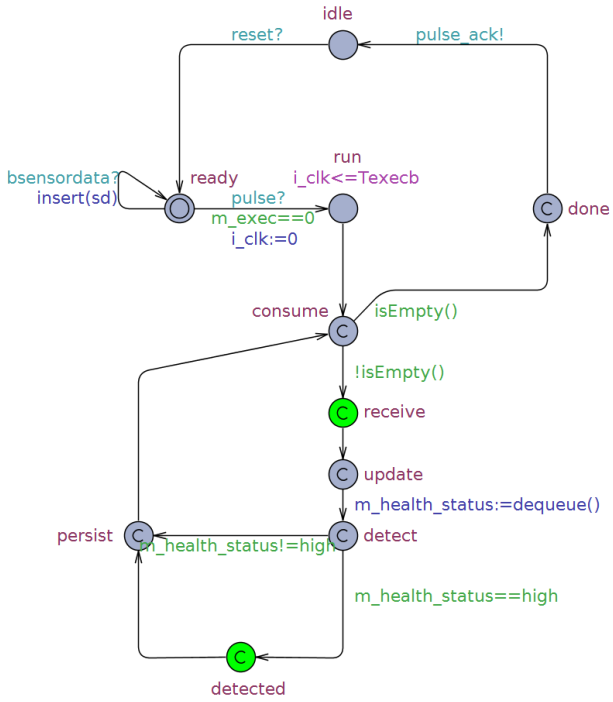


Figura 3.17: Autômato do modelo da plataforma central

consumir dado (*receive*), atualizar o estado do paciente, passar por procedimento de detecção de situações emergenciais e persistir dados relevantes. Até que todas as mensagens recebidas naquele ciclo sejam consumidas.

3.3.3 Projeto do controlador

O projeto do controlador em si consiste na aplicação de técnicas que resultam no controle do atuador e que leva o sistema à resposta esperada. Existem diversas técnicas de projeto de controle, cada uma com suas vantagens de desvantagens, a escolha entre essas técnicas depende de uma série de fatores relativos ao sistema que se deseja controlar: o objetivo de automação, o atuador e suas limitações, a linguagem formal utilizada para modelagem, a sintaxe do modelo, viabilidade de implementação no sistema real.

No caso da BSN com requisitos de tempo real, modelada por autômatos temporais no UPPAAL codificada no *framework* OpenDaVINCI, escolhemos projetar o controlador como um supervisório que atua diretamente sobre a variável de guarda da execução do módulo nó sensor baseada no estado atual do mesmo. A concepção de controle supervisório em sistemas a eventos discretos pode ser executada por especificação do comportamento do sistema controlado, em termos do desenvolvimento de um modelo local que representa os estado e transições que geram o comportamento desejado, e então faz-se a composição paralela da especificação geral (composição entre especificações locais) com o modelo da planta para obter o sistema controlado.

O UPPAAL permite que a técnica seja executada por meio de canais que ativam o controlador assim que a transição para o local desejado é ativada e então a especificação define o comportamento

desejado por envio de sinais por canais da mesma forma que foi ativada. Esta técnica é fácilmente implementada em código com o uso de máquinas de estados finitos. Especificação do controlador:

1. Caso o estado do nó sensor atual esteja em baixo o algoritmo interno deve ser executado a cada dez ciclos,
2. Caso o estado do nó sensor esteja em moderado o algoritmo interno deve ser executado a cada cinco ciclos,
3. Caso o estado do nó sensor esteja em baixo o algoritmo interno deve ser executado a cada ciclo.

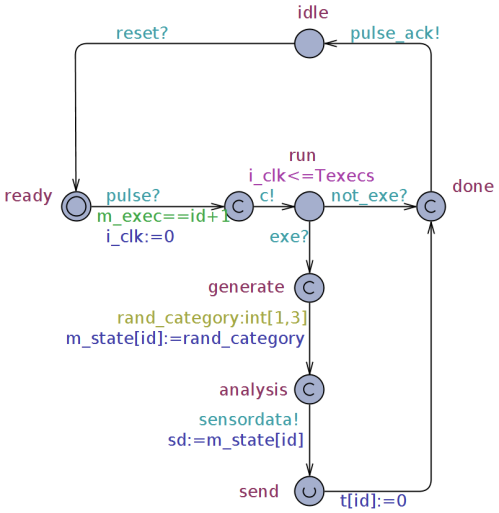
3.3.4 Implementação e Validação

O último passo da metodologia propõe que o controlador seja implementado e o comportamento do sistema automatizado validados, primeiro contudo o fazemos no modelo e uma verificação formal no UPPAAL são efetuados com o objetivo de certificar a satisfação dos requisitos propostos. Para então, implementar e validar o código do controlador no protótipo.

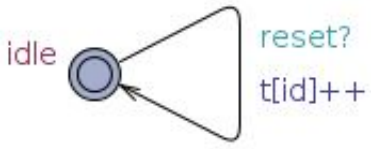
3.3.4.1 Implementação no Modelo

A implementação do controlador no modelo do UPPAAL é feita conforme descrito na etapa de projeto do controlador. Primeiro concebe-se um autômato temporal que representa o comportamento levantado como especificação do controlador. Em seguida é feito o acoplamento do controlador à planta por meio de canais pono-a-ponto que ativam o comportamento do controlador assim que a planta atinge e o estado esperado, e o controlador envia sinais distintos dependendo do estado das variáveis de controle naquele estado (i.e. estado atual do nó sensor e número de ciclos).

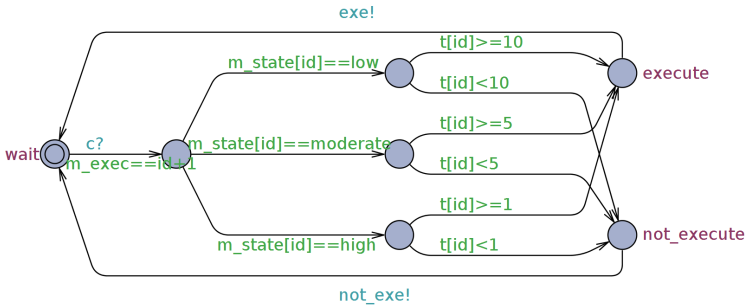
Para contabilizar a quantidade de ciclos do escalonador um relógio que incrementa um à variável t, usada para controle do número de ciclos, foi desenvolvido para operar em paralelo aos outros módulos do modelo. A Figura 3.18a ilustra o módulo do nó sensor após o processo de acoplamento do controlador, a Figura 3.18b ilustra o relógio de controle do número de ciclos de escalonamento computados desde a última execução do módulo nó sensor e a Figura 3.18c ilustra o autômato temporal da especificação do controlador. E a Tabela 3.9 descreve os canais de controle adicionados ao modelo.



(a) Modelo do nó sensor autônomo



(b) Modelo do relógio do controlador



(c) Modelo do controlador

Figura 3.18: Modelo completo do nó sensor autônomo

Nome	Tipo	Descrição
c	chan	Canal ponto-a-ponto ativa o comportamento do controlador.
exe!	chan	Canal ponto-a-ponto que comanda a execução do módulo nó sensor.
not_exe	chan	Canal ponto-a-ponto que impede a execução do módulo nó sensor.

Tabela 3.9: Canais de controle ponto-a-ponto

3.3.4.2 Validação do Modelo

O processo de verificação formal do modelo busca validar atributos que representam requisitos funcionais ou não-funcionais do sistema, em outras palavras, tenta garantir que o comportamento do sistema expresso no modelo, está de acordo com as expectativas. Portanto, se assumirmos que

a modelagem de sistema foi bem feita e o modelo puder ser considerado bem próximo ao que se espera do sistema, a verificação formal tem capacidade de assegurar que o sistema irá se comportar conforme os requisitos especificados pelos projetistas.

A ideia de utilizar a verificação de modelos na metodologia de projeto de controle para sistemas de *software* é garantir que o sistema controlado apresente o comportamento esperado pelos objetivos levantados e além disso garantir que nenhuma outra propriedade tenha sido violada por efeitos inesperados da implementação deste. A validação do comportamento do modelo reside na verificação formal das propriedades formalizadas dos requisitos da BSN. A sintaxe das propriedades segue à gramática de especificação de propriedades do UPPAAL, Tabela 2.1. A verificação do modelo resume-se a três tipos de verificação:

1. Comportamento do sistema não-controlado,
2. Comportamento do controlador, e
3. Comportamento do sistema controlado.

O sistema não-controlado tem como objetivo raiz G1 (Emergência detectada) cujas decomposições são G2 (Estado do paciente detectado) e G0 (Tempo resposta menos que 250ms) portanto, necessita-se verificar: (i)a coleta de dados dos sensores (T1.1), (ii) a transferência de dados coletados (T1.2), (iii) apersistência dados (T1.3), (iv)o processo dos dados dos sensores (T2.1) e (v) detecção do estado do paciente(T2.2), de acordo com a respectiva árvore de objetivos. Satisfeitos esses requisitos podemos assumir que não existem efeitos colaterais prejudiciais do controlador no sistema anterior à implementação deste.

Com a adição do controlador ao sistema, além dos objetivos G0 e G2 deve-se verificar o (G3) Frequência de envio de dados alterada a (T3) Alterar frequência de execução deve ser verificada, grantindo assim satisfação G3. A Tabela 3.10 descreve as propriedades utilizadas para satisfazer os comportamentos requisitados para o sistema não-controlado, o controlador denotado por "C"e o sistema controlado. As tarefas satisfeitas por cada propriedade estão devidamente mapeadas na tabela. A Tabela 3.11 indica o resultado da verificação.

Os resultados representados na Tabela 3.11 indicam satisfação de todas as propriedades levantadas nos três comportamentos do modelo: não-controlado, controlador e controlado. E implica na satisfação de todos os requisitos descritos pela árvore de objetivos do protótipo da BSN autônomo, Figura 3.12.

3.3.4.3 Implementação no Protótipo

A implementação do controlador no protótipo é diretamente mapeável em uma máquina de estados finitos, que em C++ pode ser codificada como uma série de condicionais que averiguam as regras da especificação. Da mesma maneira que na implementação no modelo o código deve ser modificado com uma condicional que verifica a condição de guarda, atuador escolhido, e assim permitir ou não a execução do módulo naquele ciclo. O procedimento que efetua a lógica mediante

#	Tarefa	Propriedade	Descrição
I	G1	$E\Diamond$ *bodyhub.detected*	A plataforma central possivelmente detectará o estado de emergência.
II - IV	T1.1, T1.2	$sensornode(k).send \rightarrow bodyhub.receive$	Sempre que dados forem enviados pelo nó sensor, a plataforma central eventualmente os receberá.
V	T1.3	$A\Diamond$ *bodyhub.persist*	Eventualmente a plataforma central irá persistir dados.
VI	T2	$A\Diamond$ *bodyhub.update*	Eventualmente a plataforma central irá atualizar os dados.
VII	C	$A\Box\,(controller(k).not\_execute)\,imply$ $((m\_state[k] == low\,and\,t[0] < 10)$ $or\,(m\_state[0] == moderate\,and\,t[0] < 5)$ $or\,(m\_state[0] == high\,and\,t[0] < 1))$	Todo estado em que o contr0lador não permitiu a execução do módulo implica que as condições temporais não foram suficientes para tal.
VIII	C	$A\Box\,(controller(k).execute)\,imply$ $((m\_state[k] == low\,and\,t[0] \geq 10)$ $or\,(m\_state[0] == moderate\,and\,t[0] \geq 5)$ $or\,(m\_state[0] == high\,and\,t[0] \geq 1))$	Todo estado em que o controlador permitiu a execução do módulo implica que as condições temporais foram suficientes para tal.
IX	T3	$A\Box\,(m\_state[0] == low)\,and$ $(sensornode(0).generate)\quad imply$ $(t[0] == 10)$	Sempre que o estado do nó for *baixo* e o nó sensor gerar novos dados, o relógio deve marcar 10 ticks.
X	T3	$A\Box\,(m\_state[0] == moderate)\,and$ $(sensornode(0).generate)\,imply$ $(t[0] == 5)$	Sempre que o estado do nó for *moderado* e o nó sensor gerar novos dados, o relógio deve marcar 5 ticks.
XI	T3	$A\Box\,(m\_state[0] == high)\,and$ $(sensornode(0).generate)\,imply$ $(t[0] == 1)$	Sempre que o estado do nó for *alto* e o nó sensor gerar novos dados, o relógio deve marcar 1 tick.

Tabela 3.10: Propriedades verificadas no modelo controlado

passagem do estado atual do nó sensor e número de ciclos desde a última execução do módulo atribui à condição de guarda do atuador o resultado booleano da avaliação.

A Figura 3.19 ilustra o diagrama de fluxo do nó sensor após o acoplamento da máquina de estados finitos que faz o controle da execução do módulo mediante estado do nó sensor e número de ciclos passados desde a última execução do módulo.

3.3.4.4 Validação do Protótipo

A validação do protótipo da BSN com resposta em tempo real e nós sensores autônomos pode ser feita de diversas maneira, já que consiste em basicamente testar o comportamento do código antes e depois da implementação do controlador. Com intuito de fazer uma análise criteriosa do

I	II	III	IV	V	VI	VII	VIII	IX	X	XI
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Tabela 3.11: Resultado da verificação das propriedades no modelo controlado

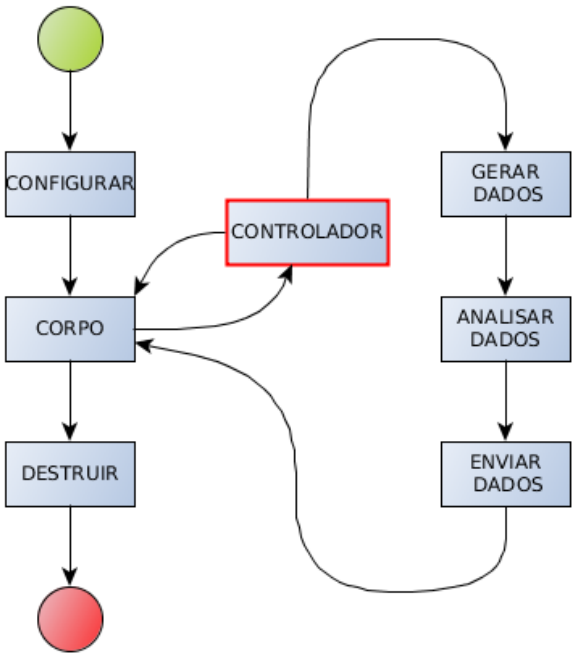


Figura 3.19: Diagrama de fluxo do nó sensor autônoma

estado do nó sensor e do tempo de fato passado entre uma execução e a seguinte, instrumentou-se o código do nó sensor com a função:

$$clock\_gettime(CLOCK\_REALTIME,\ ts)$$

que atribui uma *timestamp* do horário do processador do ambiente de execução à variável ts de tipo *timespec*, e a cada execução do mesmo é impressa uma linha no arquivo .xls, com os dados: *Tempo passado(s)*, *Estado atual do nó sensor*, e *Tempo desde a última execução (s)*. O tempo passado indica a quantidade de segundos que passaram desde a inicialização do módulo em questão, usado como referência. O *estado atual do nó sensor* é o valor da variável estado do nó sensor no momento da execução, e o *tempo passado desde a última execução (s)* é monitorado e registrado em arquivo de log.

O experimento foi exercitado em dois cenários por meio da manipulação da variável de atuação do controlador, como descrito na Tabela 3.13. A configuração do escalonador do OpenDaVINCI se dá pelo cenário descrito na Tabela 3.12.

O cenário (I) simula a execução do protótipo com características de anterior à implementação

[3] "exe"é de fato o nome da variável no código.

[4] A explicação para tal cenário não cabe a essa seção e é descrita no capítulo seguinte.

Cenário	N_p	N_s	**Política**	**Frequência(Hz)**	**Prioridade**
1	1	3	*simulation_rt*	10	20

Tabela 3.12: Cenário determinístico utilizado na execução do protótipo[3]

Cenário	**Variável de atuação**[4]	**Valor**
I	*exe*	true
II	*exe*	resultado do MEF do controlador

Tabela 3.13: Cenários para validação do protótipo

do controlador e (II) simula a execução do protótipo após o acoplamento do controlador. Os resultados obtidos estão expressos na forma de dois gráficos: estado do sensor por tempo e tempo decorrido entre duas execuções consecutivas por tempo, Figuras 3.21 e 3.20. O eixo Y do gráfico de estado do sensor por tempo representa os estados do sensor 1, 2 e 3 como baixo, moderado e alto respectivamente.

Para o mesmo período de 168 segundos de execução, o protótipo controlado gerou um total de 264 dados contra 2002 do protótipo não controlado, naturalmente haverá uma maior variação entre os estados do sensor no mesmo período de tempo. O que é evidenciado pela diferença de tempo entre a geração consecutiva de dados, que no protótipo não controlado fica em 0,1 segundo independente do estado do nó sensor, ao contrário do comportamento do protótipo controlado que altera a frequência de geração de novos dados baseado no estado atual do sensor.

Porém, o mais importante dos resultados é que o comportamento do protótipo controlado, gráficos das Figuras 3.21, corresponde ao comportamento verificado pelas das propriedades (IX), (X) e (XI) da Tabela 3.11 de resultados da verificação de modelo do protótipo controlado certifica que o protótipo irá: em caso do estado do nó ser *baixo*, gerar um novo dado somente após dez ciclos, em caso do estado do nó sensor ser *moderado*, gerar um novo dado somente após 5 ciclos e em caso do estado do nó sensor ser *alto* gerar um novo dado após 1 ciclo.

Além disso, corrobora com o requisito não-funcional G0, que determina tempo de resposta para detecção de emergências menor que 0,250 segundos, já que o período de ciclo do nó sensor no momento em que está em estado *alto* é de 0,1 segundo.

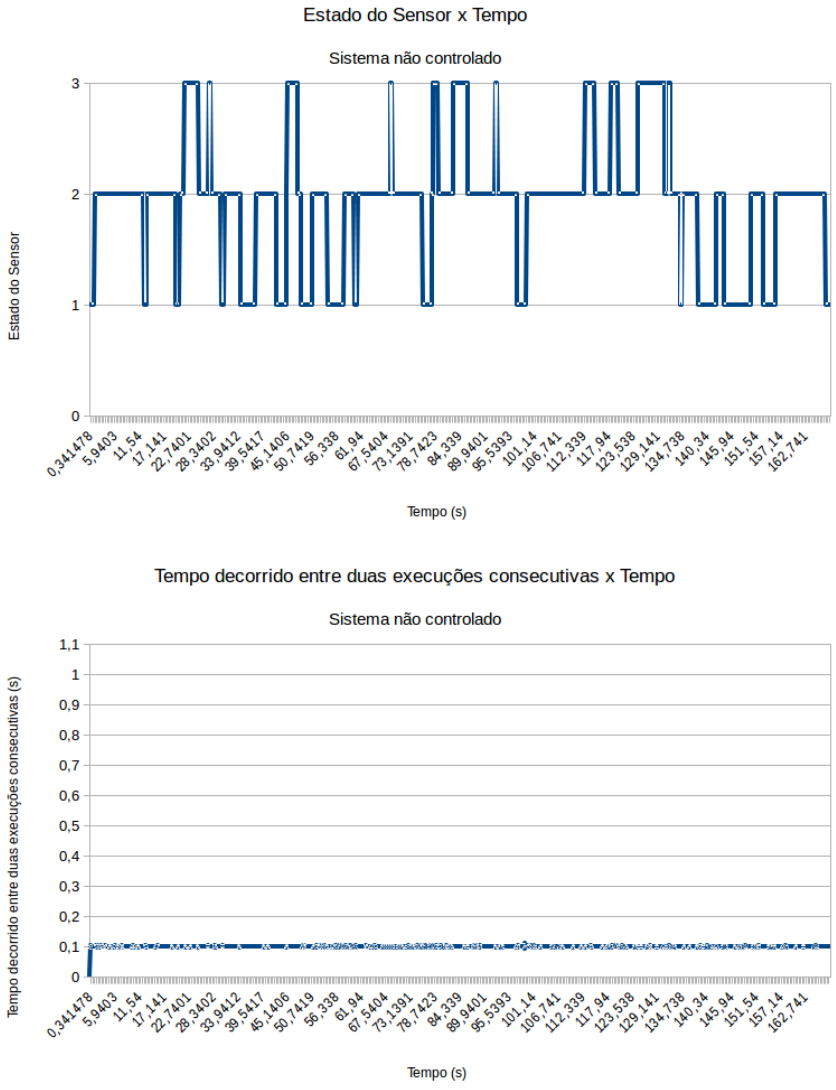


Figura 3.20: Gráficos do comportamento do protótipo sem controlador

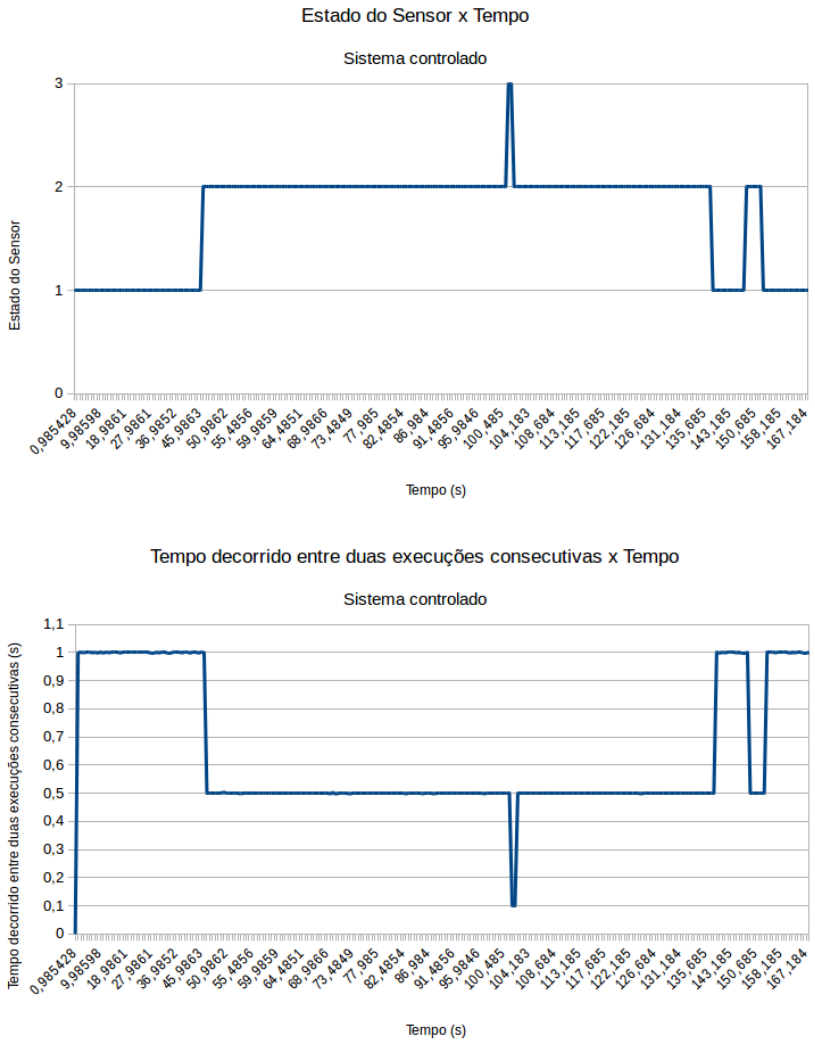


Figura 3.21: Gráficos do comportamento do protótipo com controlador

Capítulo 4

Avaliação da Proposta

4.1 Introdução

A avaliação da proposta deve tecer referencial para a comparação do funcionamento do protótipo antes e depois do processo de automação, e avaliar se os objetivos do controle foram atingidos para assim poder discutir a validade da utilização da verificação formal de modelo no projeto de automação de sistemas de *software*. A validação da proposta deve se basear em métricas de referência que determinam se os resultados obtidos no processo são aceitáveis ou não.

Neste capítulo é feita uma avaliação das capacidade de execução do protótipo[1] em tempo real com intuito de validar as métricas utilizadas na avaliação e análise do protótipo. Em seguida, comparar os resultados da validação do modelo no UPPAAL com a validação do protótipo autônomo no OpenDaVINCI. E finalmente discutir vantagens e desvantagens da verificação formal como técnica de garantia de propriedades tempo real no processo de concepção de sistemas de *software* autônomos.

4.2 Avaliação do OpenDaVINCI

Para avaliar a capacidade de execução em tempo real do OpenDaVINCI empregada no processo de validação do protótipo da BSN com sensores autônomos, deve-se definir as métricas para execução do escalonador do OpenDaVINCI baseado nas capacidades do mesmo no ambiente onde foram realizados os testes.

Para tanto, configurações importantes relativas aos componentes do ambiente de testes é apresentado. Em seguida, os experimentos relativos ao tempo de execução dos módulos e impacto no tempo de ciclo do escalonador, limite inferior de frequências de operação do escalonador e experimento para verificar configurações com execução ótima em termos de tempo de resposta com determinismo do escalonador são efetuadas.

[1] Repositório do Projeto Desenvolvido Disponível em https://github.com/rdinizcal/OpenBASN/tree/tcc

4.2.1 Métricas

Por consistir de um sistema tempo-real cujo objetivo de automação está intrínsecamente ligado ao tempo de resposta do sistema, é fundamental que definamos os parâmetros de comparação de passagem do tempo durante o processo de execução do protótipo. Porém, quando fala-se de sistemas distribuídos e computadorizados, a variável tempo não é trivialmente definida por não possuir um referêncial claro como outras variáveis. Nesse sentido, todas as medidas realizadas neste capítulo foram obtidas com base na captura de *timestamps* do processador no momento da execução do código por meio da utilização de uma função da biblioteca *sys/time.h*. As configurações da máquina onde os experimentos foram executados estão descritas na Tabela 4.1.

Recurso	**Especificação**	**Observação**
Processador	4x Intel(R) Core(TM) i7-5500U CPU @2.40GHz	
Memória	8.075MB	
Sistema Operacional	Ubuntu 16.04.3 LTS	
Kernel	Linux 4.4.86-rt99 (x86_64)	SMP PREEMPT RT
Compilador	GNU C Compiler version 5.4.0 20160609	
Disco Rígido	ATA Corsair Force LE	

Tabela 4.1: Configurações relevantes do sistema

4.2.2 Experimentos

A garantia do funcionamento correto do sistema depende da execução determinística dos módulos por parte do escalonador. Portanto, é necessário entender as variáveis que influem diretamente no tempo de ciclo e limitações de configurações do OpenDaVINCI para poder assegurar que o comportamento do sistema obedeça a requisitos de tempo real. Por isso, três análises de execução do protótipo foram feitas: (1) quanto aos tempos de execução dos módulos instanciados, e (2) quanto aos limites de configuração da frequência de operação do escalonador do OpenDaVINCI.

4.2.2.1 Tempos de Execução dos Módulos

A inequação 4.1 determina os fatores que influenciam diretamente no tempo de execução do ciclo e que devem ser garantidos para execução determinística do escalonador de módulos e por consequência do sistema.

$$\tau > (N_p \text{ x } \tau_{exec_p}) + (N_s \text{ x } \tau_{exec_s}), \tag{4.1}$$

onde τ é o período de escalonamento dos módulos, τ_{exec_p} é o tempo médio que o algoritmo do módulo da plataforma central leva para ser completamente executado, τ_{exec_s} é o tempo médio que o algoritmo do módulodo nó sensor leva para ser completamente executado, e N_p e N_s são

os números de instâncias dos módulos. Ou seja, a soma total do tempo de execução de todos os módulos não deve exceder o período do escalonador.

A variável τ é configurada no momento da execução do escalonador por meio da frequência em *hertz*. A quantidade de sensores e plataformas instanciadas dependem da aplicação e são também manipuláveis, contudo não faz sentido instanciar mais de uma plataforma central. Os tempos de processamento τ_{exec_p} e τ_{exec_s} dependem de uma série de fatores relacionados tanto ao *hardware* quanto ao *software*.

Para obter tempos de execução dos módulos (τ_{exec_p} e τ_{exec_s}) utilizados na avaliação da proposta, o código do protótipo foi instrumentado com cronômetros que recuperam o *timestamp* do processador no momento da execução, e registram em arquivo a diferença entre começo e final da execução do módulo no ciclo. O protótipo foi configurado de acordo com os dados dos parâmetros na Tabela 4.2.

$f(Hz)$	N_p	N_s
20	1	3

Tabela 4.2: Configuração do protótipo para obtenção de τ_{exec_p} e τ_{exec_s}

A partir da execução do módulo instrumentado, obteve-se os valores da média e desvio padrão dos tempos de execução dos módulos apresentado na Tabela 4.3.

$\tau_{exec_p}(\mu s)$	$\tau_{exec_{s0}}(\mu s)$	$\tau_{exec_{s1}}(\mu s)$	$\tau_{exec_{s2}}(\mu s)$
98,525 ± 20,803	34,932 ± 12,971	32,138 ± 9,250	32,298 ± 11,513

Tabela 4.3: τ_{exec_p} e $\tau_{exec_{sn}}$ obtidos na execução dos módulos

4.2.2.2 Frequência de operação do OpenDaVINCI

Testes de escalabilidade da configuração inicial do escalonador do OpenDaVINCI foram realizados com o intuito de avaliar cenários que testam o comportamento esperado do protótipo. Notou-se que existem limites inferiores que impossibilitam o componente central de executar os módulos com frequência maior que 25Hz, pois, a partir desta, o período medido não acompanha alterações na frequência, estabilizando em 0,044s. A Figura 4.1 detalha as medições realizadas e o comportamento da curva obtida.

Assume-se então, frequência máxima de execução dos módulos é de 20Hz ou período mínimo de $0,05s$. Novos experimentos foram realizados para avaliar o comportamento do escalonador na busca por configurações ótimas que garantem que o sistema vai, de fato, responder dentro do limite de 250ms. Portanto, as configurações de frequência de execução e prioridade do processo são variadas de acordo com a Tabela 4.4.

[2]Política de escalonamento de módulos por parte do escalonador do OpenDaVINCI - first-come first-served (FCFS) = não preempiva, estática e offline.

Prioridade de escalonamento em tempo-real do escalonador de tarefas do kernel - limitada de [0,42].

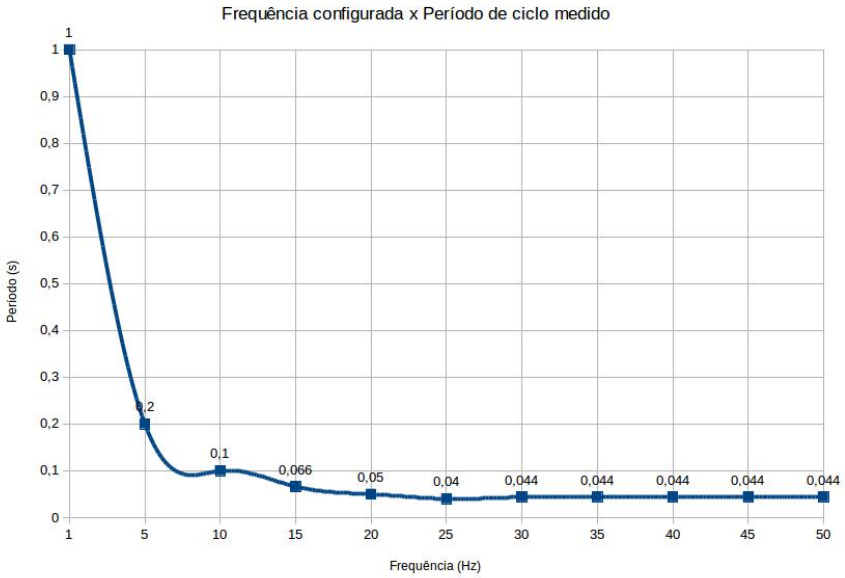
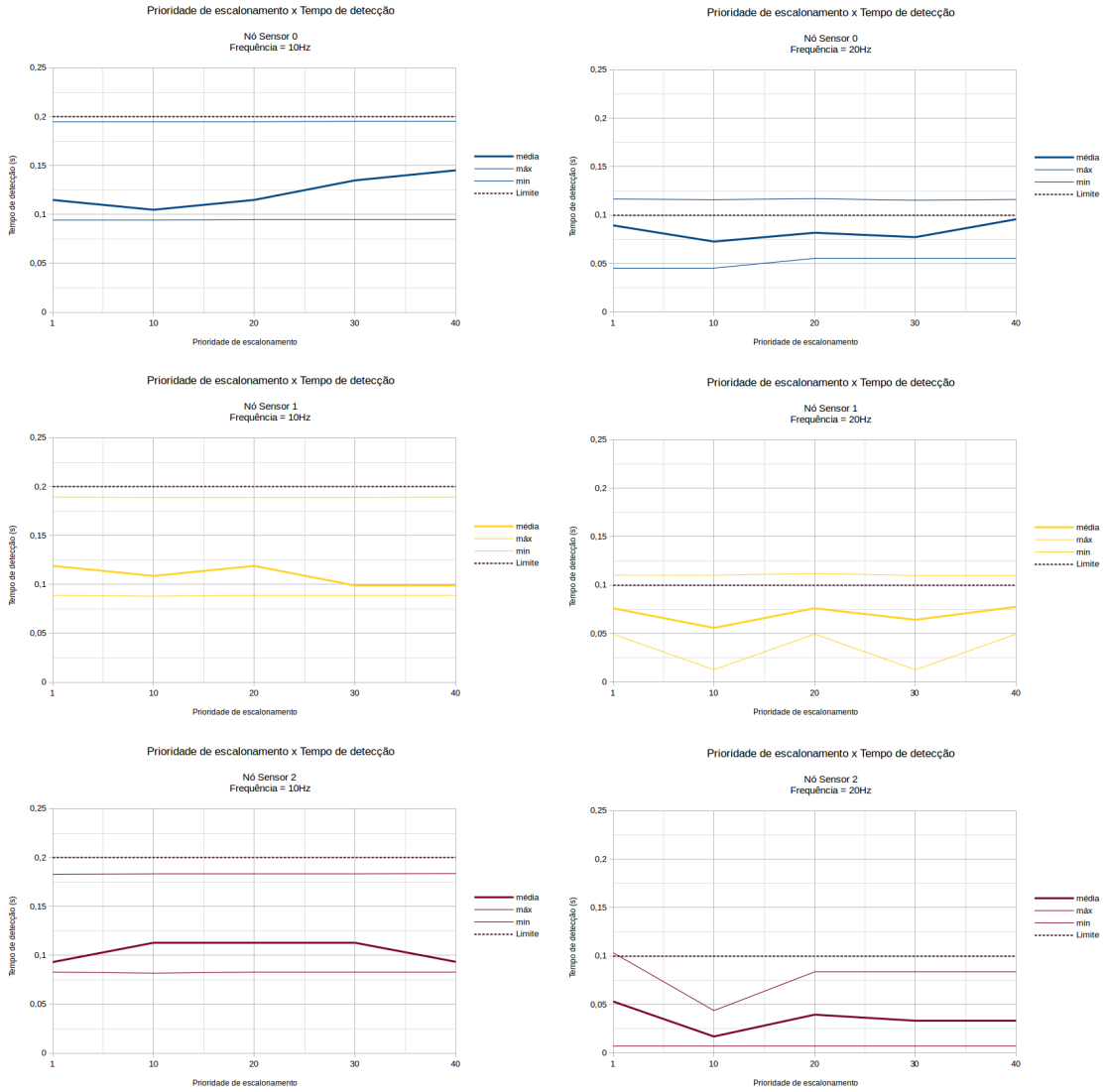


Figura 4.1: Curva de frequência por período medido

Cenário	N_p	N_s	**Política de Escalonamento**	**Frequência(Hz)**	**Prioridade**
1	1	3	*first-come first-served (FCFS)*	10	1
2	1	3	*first-come first-served (FCFS)*	10	10
3	1	3	*first-come first-served (FCFS)*	10	20
4	1	3	*first-come first-served (FCFS)*	10	30
5	1	3	*first-come first-served (FCFS)*	10	40
6	1	3	*first-come first-served (FCFS)*	20	1
7	1	3	*first-come first-served (FCFS)*	20	10
8	1	3	*first-come first-served (FCFS)*	20	20
9	1	3	*first-come first-served (FCFS)*	20	30
10	1	3	*first-come first-served (FCFS)*	20	40

Tabela 4.4: Cenários para verificação da configuração ótima do escalonador[2]

Desta vez o experimento consistiu em medir a diferença entre o instante em que um dado é gerado no nó sensor e o momento em que ele passa pelo processo de detecção de emergência na plataforma central. Nos diferentes cenários expressos na Tabela 4.4. O resultados das execuções do protótipo instrumentado são dispostos em gráficos de tempo de detecção de alteração no sinal fisiológico em segundos, por prioridade de escalonamento configurada na execução. Foram plotadas, a média, caso de tempo máximo (pior caso) e caso de tempo mínimo (melhor caso) identificados para cada prioridade escalonamento.



(a) 10hz (b) 20hz

Figura 4.2: Gráficos de tempo de detecção por prioridade de escalonamento

4.2.3 Análise dos Resultados

Os tempos de execução dos módulos da plataforma central (τ_{exec_p}) e dos nós sensores (τ_{exec_s}) expostos na Tabela 4.3, em comparação com a frequência máxima de operação do escalonador do OpendDaVINCI, Figura 4.1 permite afirmar que estes não representam ameaças à validade ao escalonamento determinístico dos módulos, já que o tempo de execução destes representam 0,1% do menor tempo de ciclo permitido.

A comparação entre os gráficos do sistema executado com frequência de 10Hz (4.2a) e 20Hz (4.2b) permite notar que há maior estabilidade e determinismo quando este é configurado para executar com menor frequência, 10Hz, já que os tempos máximos e mínimos neste caso são melhor demarcados e correspondem ao comportamento esperado. Já com 20Hz, nota-se que nos casos dos três sensores há extrapolação do limite máximo de detecção, indicando que a métrica 4.1 foi

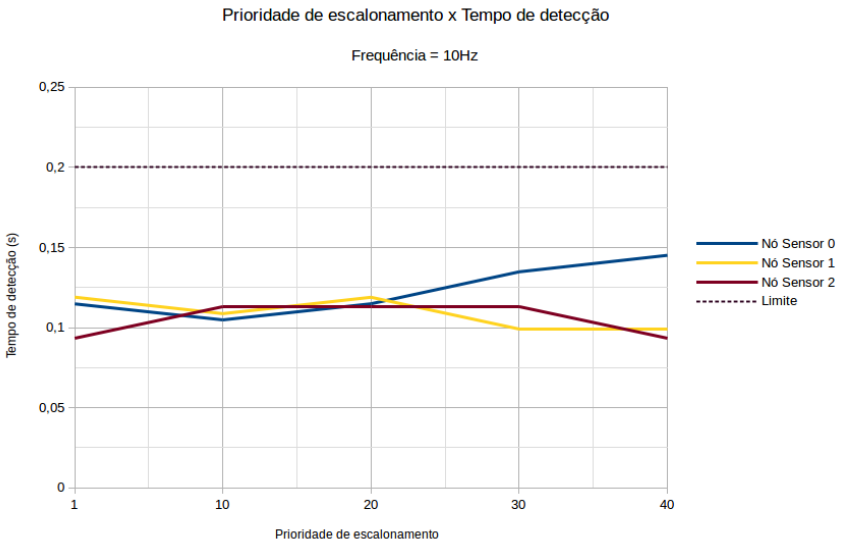


Figura 4.3: Gráfico comparativo entre médias de tempo de detecção com frequência de 10 Hz.

extrapolada.

Os gráficos comparativos entre as médias dos tempos de detecção dos sensores com frequência de 10Hz (4.3) e 20Hz (4.4) reiteram o maior determinismo da configuração com 10Hz pelo fato da média dos tempos de detecção ocorrer nos primeiros 25 milésimos de segundo (um quarto de ciclo) subsequentes ao término do ciclo no qual as mensagens foram enviadas. O que não pode ser afirmado no segundo caso, em que os tempos de detecção estão dispersos para cada um dos nós.

Observando todos os gráficos obtidos, é possível perceber que não há correlação, ou é muito baixa, entre os tempos de detecção de alteração de estado do paciente e a prioridade de escalonamento configurada para cada um dos casos. Esse resultado pode ser explicado pelo ambiente de execução dos testes, que possui recursos (memória, processadores) excedentes para o tipo de aplicação executada, então a concorrência se torna um problema bem resolvido em termos de quantidade de recursos disponíveis contra o requisitado.

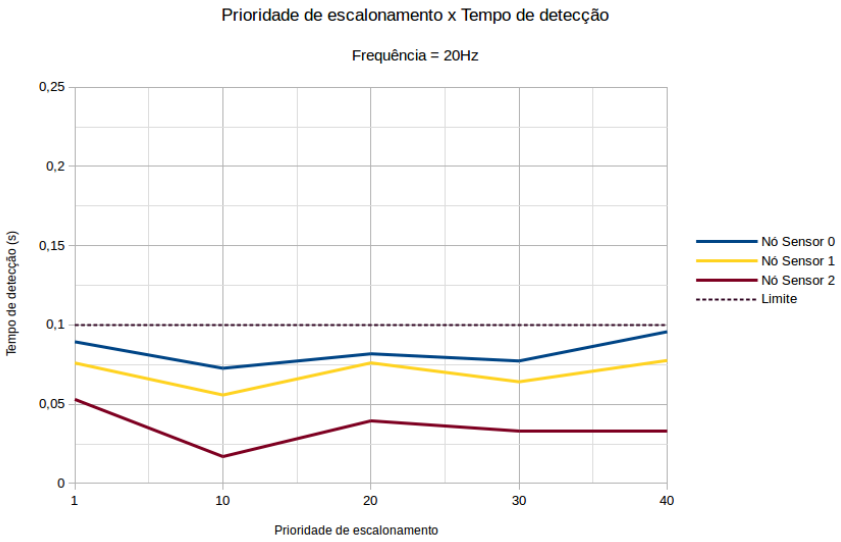


Figura 4.4: Gráfico comparativo entre médias de tempo de detecção com frequência de 20 Hz.

4.3 Avaliação da metodologia

Os bons resultados obtidos na análise de dados do protótipo autônomo correlatos aos da verificação do modelo no UPPAL, evidenciam boas escolhas de objetivo e atuador, corretude da modelagem do sistema não controlado, eficiência da técnica utilizada no projeto do controlador, implementação do controlador no código do protótipo, e por fim aceitação da metodologia utilizada para automatizar um sistema de *software* com requisitos de tempo real.

Apesar das limitações de execução, tanto do limite inferior de frequência de operação como da indiferença na configuração da prioridade do processo, que são afetados diretamente pelas configurações do ambiente de execução 4.1, o OpenDaVINCI se mostrou satisfatório quanto ao escalonamento dos módulos em modo de *first-come first-served (FCFS)* empregado na execução do protótipo com precisão de tempo real de execução dos módulos para frequências de 10Hz.

O UPPAAL foi fundamental na correta modelagem do sistema e verificação do modelo. A sintaxe dos autômatos temporais concorrentes utilizados para modelagem no UPPAAL é completa e permite modelar uma infiniade de diferentes cenários de modelos tempo real.

E finalmente, apesar da simplicidade do estudo de caso, a verificação formal de modelos se mostrou bastante útil no contexto da metodologia de concepção de controle para automação de sistemas de *software* em tempo real. A verificaçao de modelos é uma ferramenta poderosa e amplamente utilizada no projeto de sistemas para garantias dependabilidade. E levanta o questionamento da capacidade da empregabilidade de verificadores de modelo para garantia de múltiplos atributos de dependabilidade para sistemas autônomos.

Capítulo 5

Conclusões

A essência da metodologia abordada neste trabalho guia o processo de projeto de controladores para sistemas lineares e invariantes no tempo com modelos matemáticos bem definidos em certo ponto de operação às variáveis que deseja-se controlar. No caso específico de sistema de *software* há uma grande dificuldade de aplicá-lo pelo fato de que o ambiente que os circunda são em sua maioria não determinísticos ou de difícil modelamento matemático.

Contudo, a perspectiva de sistemas discretos e por consequência modeláveis por autômatos, abre precedentes para aplicação dessa metodologia com capacidade de garantias de atributos de dependabilidade, que trazem garantias fundamentais à realização de sistemas em ambientes voláteis com requisitos de tempo-real.

A simplicidade do estudo de caso, tem como objetivo demarcar bem todas as etapas do processo, o que pode ser visto como ameaça à validade do projeto que não prova escalabilidade tanto para sistemas autônomos por multi-objetivos quanto para obejtivos mais complexos.

É importante notar também que existem diversas ferramentas que poderiam ter sido utilizadas tanto para modelagem de sistemas com requisitos de tempo real quanto a para verificação destes, como redes de petri temporizadas, ou CSPs. A escolha do UPPAAL se deu por além da fácil sintaxe e permissão de codificação de funções acessório em linguagem similar ao C, as facilidades oferecidas pela ferramenta fora simulação de tempo de execução, auxílio a identificação de *livelocks* em estruturas complexas, facilidade de *debug* de situações de estado terminal não previsto e suporte a análise de propriedades tempo-real, além de extenso uso de estado-da-arte em verificação de modelos para sistemas autônomos [11].

5.1 Trabalhos Futuras

Algumas linhas de estudo são apresentadas para dar continuidade ao presente trabalho:

- Empregabilidade de outras ferramentas para modelagem discreta de sistemas de *software*,
- Aplicação do recurso de verificação de modelos não determinísticos por meio da sintaxe

probabilística permitida pelo UPPAAL para verificação de cenários não modelados,

- Estudos de caso de sistema com integração de fato a sistemas físicos e garantias de propriedades e seu impacto,
- estudo da capacidade de verificação contínua de modelos para múltiplos atributos de dependabilidade no contexto de sistemas autônomos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FILIERI, A. et al. Software engineering meets control theory. In: *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. [S.l.]: IEEE, 2015. (SEAMS '15), p. 71–82.

[2] NADEEM, A. et al. Application specific study, analysis and classification of body area wireless sensor network applications. *Computer Networks*, v. 83, n. Supplement C, p. 363 – 380, 2015.

[3] CHEN, M. et al. Body area networks: A survey. *Mobile Networks and Applications*, v. 16, n. 2, p. 171–193, Apr 2011.

[4] BAIER, C.; KATOEN, J.-P. *Principles of Model Checking (Representation and Mind Series)*. [S.l.]: The MIT Press, 2008.

[5] YUCE, M. R. Implementation of wireless body area networks for healthcare systems. *Sensors and Actuators A: Physical*, v. 162, n. 1, p. 116 – 129, 2010.

[6] AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 1, n. 1, p. 11–33, jan. 2004. ISSN 1545-5971.

[7] MAHDAVI-HEZAVEHI, S. et al. A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems. *Information and Software Technology*, v. 90, n. Supplement C, p. 1 – 26, 2017.

[8] NISSANKE, N. *Realtime Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[9] KNIGHT, J. C. Safety critical systems: Challenges and directions. In: *Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002. (ICSE '02), p. 547–550. ISBN 1-58113-472-X.

[10] LARSEN, K. G.; PETTERSSON, P.; YI, W. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, Springer–Verlag, v. 1, n. 1–2, p. 134–152, out. 1997.

[11] NORMAN, G.; PARKER, D.; SPROSTON, J. Model checking for probabilistic timed automata. *Formal Methods in System Design*, v. 43, n. 2, p. 164–190, Oct 2013.

ANEXOS

I. DESCRIÇÃO DO CONTEÚDO DO CD

Descrever CD.

II. PROGRAMAS UTILIZADOS