TRABALHO DE GRADUAÇÃO

Verificação, prototipação e análise de estratégias de escalonamento e controle para sistemas auto adaptativos tempo real

Eric Bernd Gil

Brasília, Julho de 2019

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

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

TRABALHO DE GRADUAÇÃO

Verificação, prototipação e análise de estratégias de escalonamento e controle para sistemas auto adaptativos tempo real

Eric Bernd Gil

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do 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 Pedro de Azevedo Berger, CIC/UnB _____
Examinador interno

Brasília, Julho de 2019

FICHA CATALOGRÁFICA

ERIC, BERND GIL
Verificação, prototipação e análise de estratégias de escalonamento e controle para sistemas auto adaptativos tempo real

[Distrito Federal] 2019.

xii, 52p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília, Faculdade de Tecnologia.

1. Sistemas Autoadaptativos 2. Controle
3. Escalonamento

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

GIL, ERIC BERND (2019). Verificação, prototipação e análise de estratégias de escalonamento e controle para sistemas autoadaptativos tempo real. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$07, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 64p.

CESSÃO DE DIREITOS

AUTOR: Eric Bernd Gil

TÍTULO DO TRABALHO DE GRADUAÇÃO: Verificação, prototipação e análise de estratégias de escalonamento e controle para sistemas autoadaptativos tempo real.

GRAU: Engenheiro ANO: 2019

Eric Bernd Gil

SMPW Quadra 28 Conjunto 2 Lote 1 Casa 6.

71745-802 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho aos meus pais, à minha irmã, aos meus amigos da UnB, que me acompanharam nos mais diversos momentos, a todos os outros amigos, que estiveram comigo ao longo dos anos, à minha orientadora Genaina, sempre presente e disponível para me ajudar no que fosse necessário, e à minha namorada Bárbara, pois essa conquista ocorreu graças a ela, que foi meu apoio e incentivo em todos os momentos que passei durante o curso e durante a vida.

Eric Bernd Gil

RESUMO

O avanço tecnológico intenso da última década possibilitou a criação de sistemas cada vez maiores e mais complexos. A tendência é que tais sistemas venham a se expandir a aumentar sua complexidade cada vez mais rápido. Nesse contexto, surge a necessidade de abordagens inovadoras para o projeto e manutenção de tais sistemas, sendo o aumento das suas capacidades de auto-adaptação uma ideia promissora. A Rede de Sensores Corporais, (*Body Sensor Network* (BSN) em inglês), aparece como um exemplo de sistema onde a característica de auto-adaptação pode trazer diversas melhorias em sua qualidade, principalmente no que diz respeito à garantia de dependabilidade. Além disso, o potencial da BSN em uma área crítica como a área médica cria uma necessidade da garantia de requisitos de tempo, fazendo com que ela possua características de tempo real e tornando sua classificação como um sistema auto-adaptativo em tempo real razoável. Ademais, tal classe de sistemas traz consigo diversos desafios de projetos que devem ser solucionados. Dois importantes desafios são: (1) a decisão da política de escalonamento para garantir comportamento tempo real, e (2) a elaboração de um controlador para a garantia da auto-adaptatividade em si. Nesse sentido, o objetivo do presente trabalho é a implementação de um modelo de um escalonador EDF (*Earliest Deadline First*) juntamente com a BSN, que foi o sistmea escolhido para ser utilizado como estudo de caso, o qual é antes formalmente verificado no UPPAAL e depois implementado no middleware OpenDaVINCI. Além disso, este escalonador atua também como controlador da prioridade de escalonamento dos módulos a ele conectados, a partir de uma malha fechada com realimentação negativa. Os resultados mostram que o escalonador EDF com a ação de controle possibilita o escalonamento de módulos com diversas frequências e que respeita às *deadlines* sempre que possível dadas as limitações do *hardware* utilizado, o que não era possível anteriormente, garantindo também um escalonamento justo dos mesmos se assim for desejado.

Palavras chave: *Sistemas tempo real, Sistemas auto-adaptativos, Escalonamento, Earliest deadline first, Controle*

ABSTRACT

The intense technological advances in the last decade made feasible the creation of even bigger and more complex systems. The tendency is that these systems come to expand and increase their complexity each time faster. In this context arises the need for new ways of designing and mantaining them, whereby increasing their capability to self-adapt turns out to be a very promising approach. The Body Sensor Network (BSN) comes out as an example of system where this self-adaptation characteristic can bring several improvements in its quality, mainly concerning dependability guarantees. Additionally, the BSN's potential in a critical area such as the medical one creates the need to comply with time requirements, giving it real-time characteristics and

making reasonable its classification as a real-time self-adaptive system. Furthermore, such class of systems brings with it several design challenges that must be overcome. Two extremely important challenges are: (1) the decision of the scheduling policy, to ensure real-time behaviour, and (2) the elaboration of a controller, to assure self-adaptivity. In this sense, the goal of the present work is the implementation of a formally verified of an Earliest Deadline First (EDF) scheduler together with the BSN, which is the system chosen to be used as a case study, and its posterior implementation in the OpenDaVINCI middleware. Moreover, the scheduler also acts as a controller of the scheduling priority of the modules connected to it, using a closed loop with negative feedback. The results show that the EDF scheduler with the control ability enables the scheduing of modules with different frequencies and that respects the deadlines whenever possible given the utilized hardware limitations, what was not possible before, also guaranteeing a fair scheduling of them if desired.
Keywords: *Real-time systems, Self-adaptive systems, Body Sensor Network, Scheduling, Earliest Deadline First, Control*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Gregos

Φ	Tempo de chegada inicial	[s]
τ	Tarefa	
T	Sequência temporizada	
δ	Restrição de relógio/Condição de guarda	
∇	Granularidade do relógio	
Σ	Alfabeto/Somatório	
σ	Palavra	
ϕ	Fórmula proposicional	
φ	Fórmula de estado	
Δ	Variação	

Grupos Adimensionais

$\mathbb{R}$	Conjunto dos números reais
$\mathbb{Q}$	Conjunto dos números racionais
$\forall$	Quantificador universal
$\in$	Quantificador existencial
$\&$	Operador E lógico
$\neg$	Operador de negação
$\subseteq$	Operador de subconjunto
$\leq$	Operador de comparação menor ou igual que
$\geq$	Operador de comparação maior ou igual que
$>$	Operador de comparação maior que
$<$	Operador de comparação menor que
$\|\|$	Operador OU lógico em linguagem de programação
&&	Operador E lógico em linguagem de programação

Siglas

STR	Sistemas Tempo Real
BSN	Body Sensor Network

SAS	Sistemas Autoadaptativos
FCFS	First Come First Served
EDF	Earliest Deadline First
MAPE-K	Monitor, Analyze, Plan, Execute, Knowledge
BAN	Body Area Network
OMS	Organização Mundial da Saúde
IEE	Institute of Electrical and Electronics Engineers
TS	Transition System
WCET	Worst Case Execution Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
CPU	Central Processing Unit
POSIX	Portable Operating System Interface
PID	Proporcional, Integral e Derivativo

Capítulo 1

Introdução

1.1 Contextualização

No decorrer dos próximos anos, a tendência é que os sistemas projetados se tornem cada vez maiores e mais complexos. Nesse contexto, buscam-se abordagens inovadoras para o projeto dos mesmos, que facilitem sua construção e manutenção. Além disso, diversos comportamentos são esperados desses sistemas, necessitando de uma classificação eficiente e uma maneira efetiva de abordar as complexidades inerentes a cada um.

Sistemas Tempo Real (STR) são uma classe de sistemas em que a corretude do seu comportamento depende não só dos resultados lógicos obtidos mas também do tempo físico em que esses resultados são produzidos [8]. *Sistemas Tempo Real Distribuídos* são uma subclasse dos STR que consistem em um conjunto de nós interconectados por uma rede local que se comunicam apenas por troca de mensagens, tendo cada nó um relógio tempo real [9] e capacidade de processamento individual.

Além do que foi previamente comentado, a busca pela melhoria de requisitos funcionais e não funcionais de sistemas tempo real e tempo real distribuídos, como por exemplo sistemas utilizados no ambiente médico [10][11], é também algo de crescente interesse. Essa melhoria visa aumentar a qualidade dos serviços oferecidos, sendo esta última muitas vezes associada ao conceito de dependabilidade, que engloba seis requisitos fundamentais de um sistema: disponibilidade, confiabilidade, segurança operacional, integridade, manutenibilidade e confidencialidade [12]. Neste sentido, surgiu a ideia de *Sistemas Auto-Adaptativos* (SAS), *Self-Adaptive Systems* em inglês, que, de acordo com Weyns [13], são sistemas computacionais que conseguem se adptar autonomamente para atingir metas baseados em seus objetivos de alto nível.

Um exemplo de sistema de que tem tido um interesse crescente de pesquisa é a *Body Sensor Network* (BSN), que pertence a um grupo de sistemas com a denominação *Wireless Sensor Networks* (WSNs). Sua aplicação no ambiente médico se dá na forma de nós sensores miniaturizados, vestíveis e implantáveis, que medem informações fisiológicas do corpo humano e as transmitem via comunicação sem fio para um dispositivo de controle no próprio corpo ou em uma localização acessível [14]. Por atuar em um ambiente onde muitas vezes se está lidando com pacientes em

estado grave, a BSN acaba se tornando um sistema que necessita de garantias de tempo real, em que há a necessidade de que a situação do paciente seja detectada em tempo hábil para a tomada das medidas necessárias por parte da equipe médica responsável. Neste caso, a classificação deste tipo de sistema como um *Sistema Tempo Real Distribuído* é necessária.

Dessa maneira, a engenharia desta classe de sistemas, que visam a garantia de requisitos de dependabilidade em tempo real, é um grande desafio que busca unir o conceito de auto adaptação com uma resposta temporalmente correta, alcançando a melhor qualidade de serviço possível.

1.2 Definição do problema

Em trabalho anterior foi implementado um modelo de uma BSN no *software* de verificação de modelo (*model checking*, em inglês) *UPPAAL*, que utiliza a teoria de autômatos temporais [15][16]. Este modelo, entretanto, utiliza o algoritmo de escalonamento *First Come First Served* (FCFS) para garantia das propriedades tempo real do sistema. Por sua vez, o protótipo implementado seguiu utilizando o mesmo algoritmo de escalonamento, dado que o *middleware* utilizado, o OpenDaVINCI, só fornecia esta opção, sendo que este algoritmo não atribui prioridades de execução no caso de um estado crítico e não leva em consideração possíveis *deadlines* a serem cumpridas. Além disso, se trata de um algoritmo estático, que acaba por não atender da melhor maneira possível às requisições do sistema. Ainda no que diz respeito à simulação, existia um entrave para a definição de módulos com diversas frequências de operação, o que era uma característica desejável do sistema prototipado.

Neste trabalho foram solucionados os problemas acima descritos a partir da modelagem e implementação de um escalonador EDF para o OpenDaVINCI. Com isso surgiu outro problema a ser solucionado: a garantia de um escalonamento justo quando assim fosse desejado, ou seja, que o escalonamento não permitisse que para módulos de mesma importância dentro do sistema existisse uma grande diferença na taxa de perda de *deadlines*, caso houvesse alguma perda no sistema por limitações do mesmo. Deste desafio surge a ideia da utilização da teoria de controle para garantir características de auto-adaptabilidade ao escalonador, de maneira que este possa garantir esta característica de justiça no escalonamento do sistema.

1.3 Objetivos do projeto

O objetivo deste trabalho é sanar os problemas descritos, a partir da modelagem e implementação de um escalonador que utilize o algoritmo de escalonamento *Earliest Deadline First* (EDF) não-preemptivo para o OpenDaVINCI. Além disso, realiza-se a inserção da teoria de controle no sistema por meio da atuação deste escalonador como um controlador, utilizando como base a ideia proposta por Stankovic et al [4], visando otimizar o desempenho e viabilizar a auto-adaptabilidade.

1.4 Resultados preliminares

Os resultados preliminares tratam primeiramente da verificação formal do modelo da BSN no *software UPPAL*, onde este possui um escalonador que segue a política de escalonamento EDF. A verificação foi realizada com sucesso, mostrando que o novo escalonador é uma alternativa robusta ao escalonador FCFS previamente utilizado.

Além disso, os resultados preliminares também tratam da implementação do escalonador EDF, o qual teve a adição da funcionalidade de um controlador para melhoria de desempenho e viabilização de auto-adaptabilidade. Os testes realizados mostraram que o sistema utilizando o novo escalonador teve o comportamento dentro do esperado.

1.5 Estrutura do manuscrito

Esta monografia tem seus demais capítulos organizados da seguinte maneira:

O Capítulo 2 trata do referencial teórico, que é a fundamentação dos conceitos utilizados no trabalho. O Capítulo 3 trata do desenvolvimento do trabalho, onde serão apresentados os métodos e o que foi realizado até o presente momento. O Capítulo 4 trata dos resultados preliminares obtidos. Por fim, o Capítulo apresenta conclusões acerca do projeto e lista possíveis trabalhos futuros a serem realizados visando a continuação do que aqui foi feito.

Capítulo 2

Fundamentação Teórica

2.1 Sistemas Tempo Real

Sistemas Tempo Real (STR) [1] são uma classe de sistemas em que a corretude do seu comportamento, sendo este sua sequência de saídas ou respostas no tempo, depende não só dos resultados lógicos obtidos mas também do tempo físico em que esses resultados são produzidos. Exemplos de sistemas dessa natureza são diversos e das mais diversas áreas, indo desde carros autônomos até plantas nucleares, aviões ou aeronaves espaciais.

Existem duas classificações de sistemas tempo real, podendo um sistema dessa natureza ser *hard realtime* ou *soft realtime*. Um sistema *hard realtime* é um sistema capaz de lidar com tarefas denominadas como *hard tasks*, que são tarefas em que o não cumprimento de seu prazo, denominado *deadline* da tarefa, podem gerar resultados catastróficos para o sistema. Já um sistema *soft realtime* é um sistema que não é capaz de lidar com as *hard tasks*, sendo este capaz de lidar apenas com tarefas denominadas *firm tasks*, que são tarefas em que o não cumprimento de seus prazos não gera consequências catastróficas porém o resultado gerado não é útil para o sistema, e tarefas denominadas *soft tasks*, que são tarefas em que o não cumprimento de seus prazos não gera consequências catastróficas e ainda gera um resultado útil, porém causa perda de performance do sistema [1].

2.1.1 Escalonamento de tarefas

Para a garantia de propriedades tempo real nos STR utiliza-se a técnica denominada de escalonamento das tarefas do sistema. Essas tarefas são as atividades que um processo deverá realizar para o término de seu processamento na CPU, podendo um processo consistir de uma tarefa ou múltiplas tarefas concorrentes. Essas tarefas podem ocorrer de forma aperiódica ou periódica, a depender do sistema, sendo cada instância de uma tarefa denominada como *job*.

O escalonamento é responsável por determinar a ordem com que as tarefas irão executar no processador, visando garantir que nenhuma perca sua *deadline*, caso isso seja possível, o que garante uma maior previsibilidade do sistema. Esse escalonamento só se faz necessário caso as

tarefas sejam concorrentes, ou seja, se duas ou mais tarefas estiverem requisitando o processador ao mesmo tempo. De acordo com Buttazo [1] as tarefas que estão aptas a utilizar o processador se dividem em três classificações: tarefa ativa, que está apta a executar no processador independente de sua disponibilidade, tarefa pronta, que está disponível e esperando pelo processador, e tarefa em execução, que está sendo executada pelo processador. Todas as tarefas prontas são armazenadas em fila onde a ordem de execução dependerá da política de escalonamento implementada.

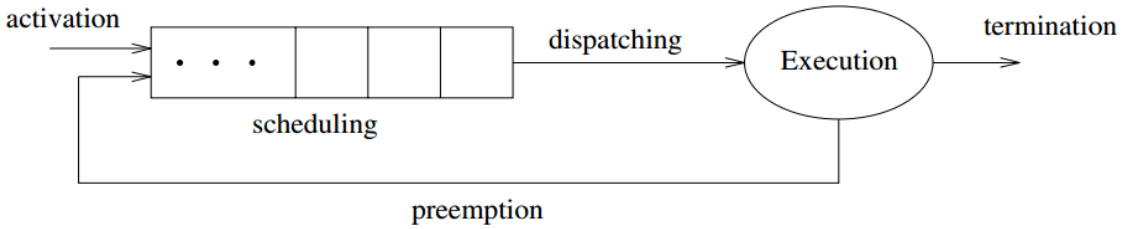

Figura 2.1: Fila de tarefas prontas [1]

Os algoritmos de escalonamento possuem diversas classificações com relação às mais diversas características que estes possuem, as quais serão listadas a seguir:

- Preemptivo ou Não-preemptivo
  - Escalonamentos preemptivos são aqueles em que tarefas em execução podem ser interrompidas e para que outra tarefa seja executada, sendo que esta última tem maior prioridade dada a política de escalonamento escolhida.
  - Escalonamentos não-preemptivos são aqueles em que tarefas em execução serão processadas até o final da sua execução, independentemente de haverem tarefas com maior prioridade na fila de execução.
- Online ou Offline
  - Escalonamentos online são aqueles em que as decisões a respeito do escalonamento são feitas em tempo de execução do sistema, na medida em que novas tarefas forem chegando no sistema ou em que tarefas terminem sua execução.
  - Escalonamentos offline são aqueles em que as decisões a respeito do escalonamento são realizadas em todo o conjunto de tarefas antes da ativação das mesmas, ou seja, antes das tarefas estarem prontas para execução.
- Dinâmico ou Estático
  - Escalonamentos dinâmicos são aqueles em que os parâmetros sobre os quais a política de escalonamento toma suas decisões podem ser alterados em tempo de execução.
  - Escalonamentos estáticos são aqueles em que as decisões de escalonamento são tomadas sobre parâmetros que não podem ser alterados em tempo de execução, tendo estes seu valor definido antes da ativação da tarefa.

A política de escalonamento analisa diversos parâmetros relacionados às tarefas para a tomada de decisões, sendo que cada política prioriza os parâmetros que julgar necessários visando um escalonamento ótimo, se este for possível. Os principais parâmetros estão listados a seguir:

- Tempo de chegada (r_i) é o tempo físico em que a instância da tarefa se torna pronta;
- Tempo computacional (C_i) é o tempo que a instância da tarefa precisa ficar no processador para sua completa execução;
- Deadline absoluta (d_i) é o tempo físico, com relação ao tempo de chegada da primeira instância da tarefa, em que uma instância deve terminar sua execução;
- Deadline relativa (D_i) é o tempo físico, com relação ao tempo de chegada da instância da tarefa que está sendo executada, em que uma determinada instância deve terminar sua execução. Calculada como: $D_i = d_i - r_i$;
- Tempo de início (s_i) é o tempo no qual a instância da tarefa começa sua execução;
- Tempo de término (f_i) é o tempo no qual a instância tarefa termina sua execução;
- Tempo de resposta (R_i) é a diferença entre o tempo de término e o tempo de início.
- Tempo de folga (X_i) é o tempo máximo que uma instância da tarefa pode ficar na fila de tarefas prontas sem ser executada. Calculado por: $X_i = d_i - r_i - C_i$.

2.1.1.1 Earliest Deadline First

Earliest Deadline First (EDF) é uma política de escalonamento que prioriza tarefas com a menor deadline absoluta d_i. O algoritmo é dinâmico, online e pode ser preemptivo ou não-preemptivo. Por considerar a deadline absoluta, para um escalonamento EDF deve-se armazenar o tempo de chegada r_i de cada tarefa, para o cálculo da deadline absoluta no instante de tempo em que o algoritmo for tomar a decisão de escalonamento. O cálculo realizado é simples: suponha um instante de tempo em que o relógio é igual a *clk*, dado um tempo de chegada da instância inicial Φ_i, um tempo de chegada da instância atual j r_{ij} e uma *deadline* relativa D_i, a *deadline* absoluta d_i para uma tarefa i qualquer é dada pela equação 2.1

$$d_i = D_i + (r_{ij} - \Phi_i) \tag{2.1}$$

Uma ilustração do EDF preemptivo é mostrada na Figura 2.2

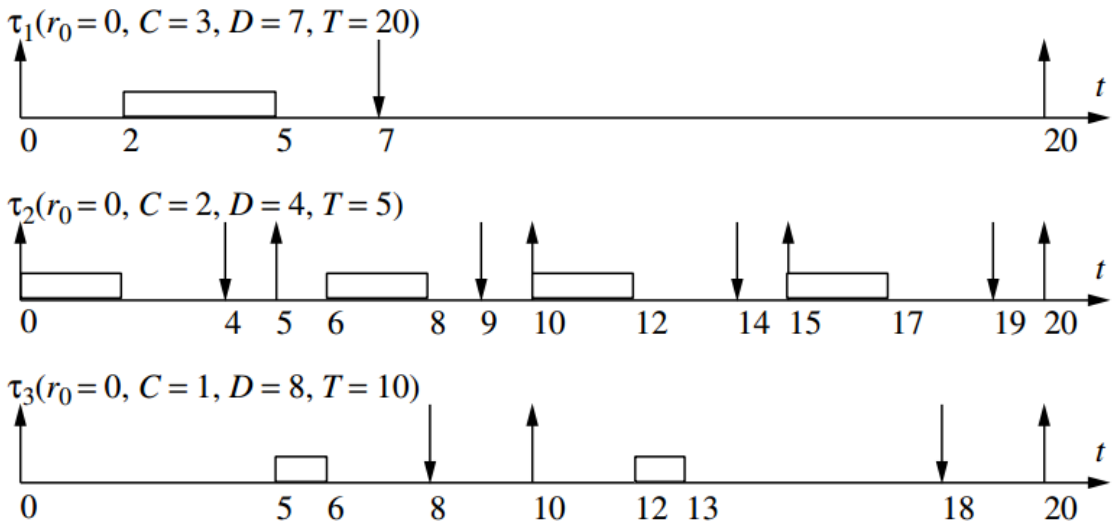

Figura 2.2: Ilustração do EDF preemptivo [2]

Pode-se perceber a mudança da prioridade entre as tarefas τ_i no instante t = 5, por exemplo, onde a tarefa τ_3 tem prioridade sobre a tarefa τ_2, sendo que inicialmente se tinha o contrário. Outra nota a ser feita é o fato da preemptividade não ter sido necessária, uma vez que nenhuma tarefa que chega enquanto o processador está sendo utilizado tem deadline menor do que a tarefa em execução, sendo assim esse exemplo serve também para ilustrar o EDF não preemptivo.

2.1.2 Sistemas tempo real distribuídos

Como definido na seção 1.1, um sistema tempo real distruibuído consiste em um conjunto de nós interconectados por uma rede local que se comunicam apenas por troca de mensagens, tendo cada nó um relógio tempo real e capacidade de processamento individual. A principal diferença ao tratar de um sistema dessa natureza está no fato de que não há memória compartilhada e,

portanto, toda a troca de informações tem de ser feito por mensagens, devendo dessa maneira haver sincronização dos clocks locais de cada nó a fim de evitar erros de temporização no sistema, enquanto nos sistemas tempo real não distribuídos a memória é compartilhada e o processamento ocorre em um nó individual. Percebe-se assim que o principal desafio em um sistema tempo real distribuído está na temporização do sistema globalmente, sendo o desafio do escalonamento de tarefas um trabalho local, que ocorre em cada nó de processamento.

2.1.3 Modelagem matemática dos sistemas tempo real

O desenvolvimento de sistemas tempo real traz consigo a necessidade da compreensão do tempo: como medi-lo, como representá-lo e como pensar acerca dele. As representações que permitem a formalização matemática do tempo da maneira mais conveniente são os relógios. *Nissanke*, em [17], definiu duas maneiras para a visualização matemática do tempo:

- Uma métrica consistindo de um conjunto de valores, dentre outras coisas;
- Um processo único que evolui continuamente.

Além disso, ambas as representações devem ser vistas como complementares, sendo a primeira relativa à medição do tempo e a segunda relativa à referência temporal de eventos. Após definida uma métrica para o tempo tem-se a capacidade de fazer comparações entre tempos distintos, dado que espera-se que o conjunto de valores temporais seja ordenado por relações de $\leq$, $<$, $\geq$ e $>$, e, sendo assim, dados dois instantes t_1 e t_2 pode-se dizer que um tempo t_2 precede ou sucede um tempo t_1. Além desses operadores, inclui-se no conjunto de valores temporais os operadores $+$ e $-$, para operações de adição e subtração de instantes de tempo, tendo assim a noção de intervalos temporais. Tais noções permitem a formalização dos relógios ou, mais precisamente, do tempo de relógio [17]:

$$clock = (\mathbb{T}, zero, next, \nabla) \tag{2.2}$$

Onde $\mathbb{T}$ é o conjunto dos valores de tempo, *zero* é um elemento distinto de $\mathbb{T}$, *next* é uma função injetora em $\mathbb{T}$ e $\nabla \in \mathbb{R}^+$. Informalmente, pode-se dizer que a função *next* é análoga à função de sucessor no domínio $\mathbb{N}$, retornando o valor imediatamente depois de um determinado t em $\mathbb{T}$. Além disso, valores de $\mathbb{T}$ são da forma $next(zero)$, $next(next(zero))$, etc, e ∇ é a granularidade do relógio, ou seja, a distância entre dois valores de $\mathbb{T}$ consecutivos, sendo esta calibrada.

2.2 UPPAAL

UPPAAL é uma ferramenta de verificação de modelo (model checking em inglês) criada no ano de 1995 em uma parceria de pesquisadores das instituições *Uppsala Universitet*, Suécia, e *Aalborg University*, Dinamarca, que utiliza autômatos temporais para modelagem, simulação e verificação de sistemas tempo real. A modelagem de tais sistemas utiliza ainda variáveis inteiras, tipos de

dados estruturados, funções e canais para sincronização [18]. Nesta seção é feito um breve resumo sobre a teoria por trás da verificação de modelo, autômatos temporais e também sobre como é feita a modelagem dentro do UPAAL.

2.2.1 Verificação de modelo

A maior parte do tempo e esforço no projeto de sistemas complexos é utilizada em testes para garantir que eles apresentem comportamento correto antes de seu lançamento. É nesse contexto que entram os métodos de verificação formal, que, além de serem técnicas mais efetivas de verificação, permitem a verificação em todas as etapas do processo de projeto, reduzindo o tempo e esforço gastos nesta atividade. Esses métodos permitem se ter garantia com rigor matemático do funcionamento do sistema, ao invés de depender de testes empíricos realizados exaustivamente, os quais não dão nenhuma garantia da corretude do sistema.

A verificação de modelo é um método formal cuja verificação se baseia, como o nome já diz, em modelos de estados finitos que descrevem matematicamente e sem ambiguidade o comportamento dos sistemas. Essa técnica explora todos os possíveis estados do sistema em questão, analisando todos os possíveis cenário de maneira sistemática, dando garantia de que o sistema realmente satisfaz determinada propriedade formal que se deseje para ele.

2.2.1.1 Sistemas de transição de estado (Transition systems)

Um *Transition System* (TS) é o tipo de modelo padrão que verificadores de modelo utilizam para representação de sistemas. Esse modelo descreve o comportamento do sistema e consiste basicamente em um grafo direcionado onde os nós representam estados e as arestas modelam transições, que são mudanças de estado, podendo ser definido por uma tupla TS da seguinte maneira [19]:

$$TS = (S, Act, \rightarrow, I, AP, L) \tag{2.3}$$

onde

- S é o conjunto de estados;
- Act é um conjunto de ações, sendo essas realizadas nas transições, podendo representar comunicação entre processos ou alteração de variáveis;
- $\rightarrow \subseteq S \times Act \times S$ é uma relação de transição;
- I é o conjunto dos estados iniciais;
- AP é o conjunto das proposições atômicas, que expressam fatos conhecidos sobre os estados do sistema e são utilizadas para formalização das características temporais do mesmo;
- $L : S \rightarrow 2^{AP}$ é o conjunto das palavras que são possíveis de se formar a partir das proposições atômicas do sistema.

2.2.1.2 Formalização de Propriedades

Para a verificação de propriedades desejadas no *Transition System* que modela um sistema, deve-se ter uma maneira de especificar as mesmas. Uma classe de propriedades utilizada para garantir a verificação formal de propriedades em um sistema é a de propriedades lineares no tempo. Uma propriedade linear no tempo sobre o conjunto de proposições atômicas AP é um subconjunto de $(2^{AP})^{\omega}$ [19], snedo que $(2^{AP})^{\omega}$ é o alfabeto de palavras dado pela concatenação infinita de palavras em 2^{AP}, sendo uma propriedade linear no tempo uma linguagem de palavras infinitas sobre esse alfabeto, uma vez que se considera aqui sistemas sem estados terminais, como no caso de sistemas paralelos. Senso assim, uma propriedade linear no tempo é uma especificação de um determinado comportamento que se espera que o sistema possua e satisfazê-la dentro do TS garante que a partir de um determinado estado escolhido todos os caminhos tem o comportamento desejado.

As propriedades de maior importância para esse trabalho são de dois tipos e visam atender um requisito de dependabilidade: segurança operacional, sendo essas propriedades de segurança operacional e de alcançabilidade. O primeiro tipo se trata de garantir que nada de ruim acontecerá, ou seja, que o comportamento de sistema não levará a um resultado catastrófico, como um acidente. Já o segundo tipo trata de garantir que os estados do sistema são alcançáveis dentro do TS que modela o sistema, sendo que um estado é alcançável se a partir do estado inicial existe pelo menos um caminho que leva o TS do estado inicial s_0 para este com uma sequência finita de ações a.

A propriedade mais importante no contexto da segurança operacional é a propriedade de *deadlock*. Essa propriedade diz respeito ao fato de o sistema como um todo estar em um estado terminal mesmo que haja um componente que localmente não esteja e se mostra como um problema de projeto em sistemas paralelos, que são os sistemas aqui considerados, uma vez que, além do fato de um componente ainda poder continuar a operar, não se deseja estados terminais pois o processamento não termina e é sempre necessário que este esteja ocorrendo para o bom funcionamento do sistema. Não menos importante é a propriedade de alcançabilidade, uma vez que se deseja que os estados do sistema sejam alcançáveis para evitar erros no processamento ou comportamentos indevidos, uma vez que cada estado normalmente representa um processamento diferente que está sendo realizado.

Além disso, em sistemas tempo real é importante considerar as propriedades denominadas invariantes. Essas propriedades são dadas por uma condição ϕ para cada estado e se requerem que essa condição seja garantida para cada estado alcançável, uma vez que seu não cumprimento pode levar a um mau funcionamento do sistema. Formalmente, como definido em [19], uma propriedade linear no tempo P_{inv} é invariante sobre um conjunto de proposições atômicas AP se existe uma fórmula proposicional ϕ sobre AP tal que

$$P_{inv} = \{A_0 A_1 A_2 \ldots \in (2^{AP}) | \forall j \geq 0 . A_j \models \phi\} \tag{2.4}$$

sendo ϕ denominada condição invariante de P_{inv}, ou ainda condição de estado de P_{inv}.

2.2.2 Autômatos temporais

Antes de definir o que são autômatos temporais, deve-se considerar as características do principal aspecto que os diferenciam de outros autômatos: a noção de tempo. Escolhe-se o conjunto dos números reais positivos, ou não negativos, $\mathbb{R}$ como o domínio do tempo nestes sistemas. Além disso, explicita-se aqui a abordagem utilizada por *Alur* e *Dill* em [20] para um melhor entendimento das definições dentro dessa teoria. Na teoria de autômatos temos para um determinado alfabeto Σ um conjunto de palavras σ_i, $i \geq 1$, onde cada palavra denota a ocorrência de um evento e pode-se realizar uma analogia entre essas palavras e as ações dentro de um *Transition System*. Nos autômatos temporais cada palavra σ é unida com uma sequência temporal T formando uma palavra temporizada (σ,T) dentro do alfabeto Σ, sendo essa sequência temporal $T = T_1 T_2 \ldots$ uma sequência infinita de valores de tempo $T_\mathrm{i} \in \mathbb{R}$, $T_\mathrm{i} > 0$, que é monotônica, i.e, $T_\mathrm{i} < T_{\mathrm{i}+1}$ $\forall i \geq 1$, e que $\forall t \in \mathbb{R}$ existe algum $T_\mathrm{i} > t$. Com a tupla (σ,T) definida, temos a noção de que o evento σ_i ocorre no tempo T_i.

Outra definição importante é a de restrições de relógio, também chamadas de condições de guarda, ou seja, o tipo de condições que podem ser impostas nas arestas de um autômato temporal com relação às suas variáveis de relógio dentro de um conjunto $x \in X$. Uma restrição δ dentro do conjunto de restrições $\Phi(X)$ é definida da seguinte maneira:

$$\delta \;:=\; x \leq c \quad | \quad c \leq x \quad | \quad \neg\, \delta \quad | \quad \delta_1 \wedge \delta_2 \tag{2.5}$$

onde $c \in \mathbb{Q}$, sendo $\mathbb{Q}$ o conjunto dos racionais. Além disso, uma última definição importante é a de tabela de transição temporizada, representada por $\mathscr{A}$ e definida como mostrado a seguir:

$$\mathscr{A} \;:=\; (\Sigma, S, S_0, C, E) \tag{2.6}$$

onde

- Σ é um alfabeto finito;
- S é um conjunto de locais finitos;
- $S_0 \subseteq S$ é o conjunto dos locais iniciais;
- C é um conjunto finito de relógios;
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ é o conjunto de transições. Uma aresta $(s, s', a, \lambda, \delta)$ representa uma transição do estado s para o estado s' para um evento a. O conjunto $\lambda \subseteq C$ representa os clocks a serem resetados com a transição e δ é uma restrição de relógio, como definida na equação 2.5.

Dadas essas definições, é possível definir os autômatos temporais. Um autômato temporal $\mathcal{A}$ pode ser definido da seguinte maneira [21]:

$$\mathcal{A} \;:=\; (\Sigma, S, S_0, C, E, Inv) \tag{2.7}$$

Pode-se perceber que a tupla de um autômato é a tupla de uma tabela de transição temporizada com a adição da função $Inv : L \to C\_\prec(X)$, que associa invariantes a um local. Por fim, um estado de um autômato temporal é um par $(s, v) \in S \times \mathbb{R}^C$ onde s é a localização atual e $v : X \to \mathbb{R}$ é uma função que associa uma variável de relógio a um valor em $\mathbb{R}$, sendo a semântica do autômato dada por um *Transition System Temporizado* com transições que possuem rótulos pertencentes a Σ e transições de atraso rotuladas com valores pertencentes a $\mathbb{R}$ que representam o atraso, como definido a seguir [21]:

$$\mathcal{S}_{\mathcal{A}} := (L, L_0, \to, \Sigma) \tag{2.8}$$

onde

- $L = S \times \mathbb{R}^X$;
- $L_0 = S_0, v_0$ onde $v_0(x) = 0 \ \forall x \in X$;
- $\to$: Relação de transição composta por
  - Transições de ação: $(s, v) \xrightarrow{a} (s', v')$ se e somente se existir $s \xrightarrow{a,\delta,\lambda} s'$ tal que $v \models \delta$, $v' = [r \leftarrow 0]v$ e $v' \models Inv(s')$.
  - Transições de delay: se $d \in \mathbb{R}$, $(s, v) \xrightarrow{d} (s, v + d)$ se $v + d \models Inv(s)$.

2.2.3 Modelagem e funcionamento básico

A modelagem de sistemas no UPPAAL se dá por meio de uma interface gráfica onde o usuário cria autômatos diversos em *templates*, colocando condições de guarda, canais de sincronização e atualizações de váriaveis, onde tudo deve estar declarado na seção *Declarations* do template para variáveis e canais locais, ou na seção *Declarations* global para variáveis e canais globais. A interface gráfica e dois autômatos de exemplo são mostrados nas Figuras 2.3 e 2.4.

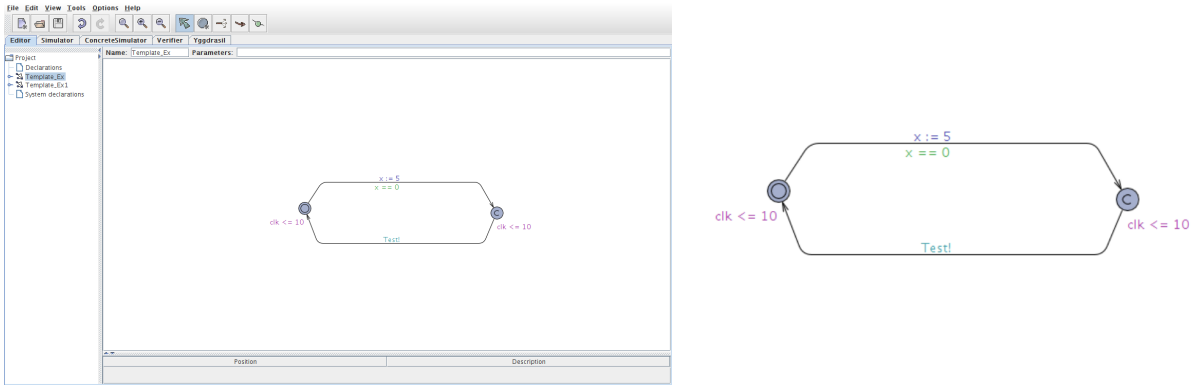

Figura 2.3: Exemplo de modelagem de autômato no UPPAAL

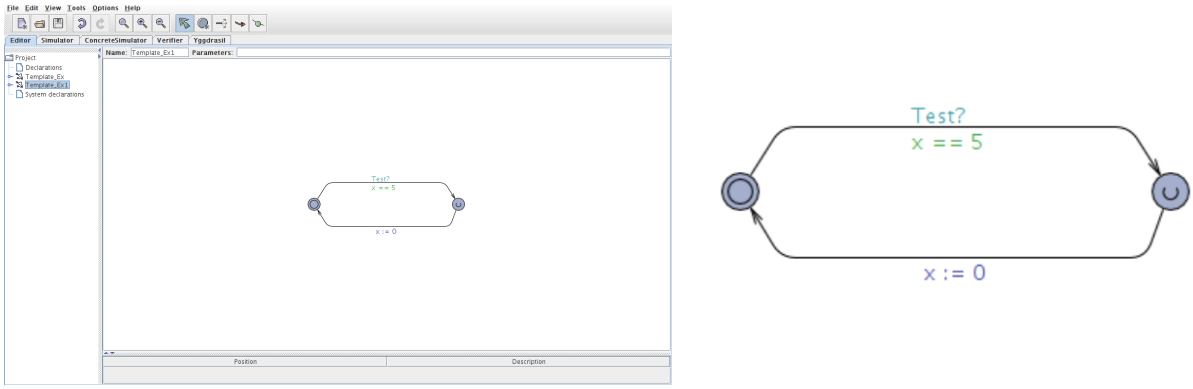

Figura 2.4: Exemplo de modelagem de autômato no UPPAAL

Os locais em um modelo são representados utilizando a representação convencional de autômatos, que são os círculos, sendo o círculo que possui um outro de menor diâmetro em seu interior a representação de local inicial, o círculo com 'C' em seu interior um local denominado como *commited*, que representa um local em que não há passagem de tempo e a próxima transição a ser realizada deverá envolver ele ou outro de mesma natureza, e o círculo com 'U' em seu interior um local denominado como *urgent*, que representa um local em que não há passagem de tempo sem a necessidade da próxima transição o envolver.

No modelo utilizado como exemplo existem duas variáveis globais: `x` e `Test`. `x` é uma variável inteira declarada como `int x` e `Test` é um canal declarado como `chan Test`. Além disso em `Template_Ex` foi declarada uma variável local: `clk`. `clk` é uma variável do tipo relógio declarada como `clock clk`. As condições de guarda são expressas na cor verde como em `x == 0`, as atualizações de variáveis são expressas na cor azul escuro como em `x := 5`, os invariantes de local são expressos na cor vinho como em `clk <= 10`, as sincronizações por canais são expressas na cor azul claro como em `Test!` e `Test?`, sendo que a sentença com '!' representa o envio do sinal e a sentença com '?' representa o recebimento do sinal. Finalmente, os templates devem ser declarados em *System declarations* utilizando a sintaxe `system Template1, Template2, ...;`, sendo no caso do exemplo `system Template_Ex, Template_Ex1;`.

Modelado o sistema na aba *Editor*, pode-se realizar uma simulação na aba *Simulation* ou ainda uma verificação na aba *Verifier*. Com relação a esta última, temos que tal verificação utiliza fórmulas lógicas expressas em *Timed Computation Tree Logic* (TCTL) [22], que é a *Computation Tree Logic* [23] extendida com condições de relógio para expressar condições tempo real.

Em TCTL, as fórmulas podem ser fórmulas de caminho ou de estado. Sobre o conjunto AP de proposições atômicas e o conjunto C de relógios, as fórmulas de estado em TCTL são formadas de acordo com a gramática

$$\Phi ::= true \quad | \quad a \quad | \quad g \quad | \quad \Phi \wedge \Phi \quad | \quad \neg\Phi \quad | \quad \exists\varphi \quad | \quad \forall\varphi \tag{2.9}$$

onde $a \in AP$, g pertence ao conjunto δ de restrições de relógio definido na gramática da Equação 2.5 e φ é uma fórmula de caminho definida por:

$$\varphi ::= \Phi \, U^J \, \Phi \tag{2.10}$$

onde $J \subseteq \mathbb{R}_{\geq 0}$ é um intervalo no qual os limites são números naturais e é de alguma das formas: [n,m], (n,m], [n,m) ou (n,m), para $n, m \in \mathbb{N}$ e $n \leq m$.

A semântica utilizada pelo UPPAAL estabelece o operador U da Equação 2.9 em que dadas duas fórmulas de caminho φ_1 e φ_2, temos as seguintes propriedades que podem ser verificadas:

- $E\Box\ \varphi_1$, que indica o fato da fórmula ser satisfeita sempre em pelo menos um caminho;
- $A\Box\ \varphi_1$, que indica o fato da fórmula sempre ser satisfeita para todo e qualquer estado de qualquer caminho e é logicamente equivalente a *not* $E\Box$ *not* φ_1;
- $E\Diamond\ \varphi_1$, que indica que existe um caminho no qual φ_1 é satisfeita;
- $A\Diamond\ \varphi_1$, que indica o fato de que em todo caminho a fórmula é eventualmente satisfeita em algum estado e é logicamente equivalente a *not* $E\Box$ *not* φ_1;
- $\varphi_1 \rightarrow \varphi_2$, que indica o fato de que satisfazer φ_1 implica ou leva a satisfazer φ_2.

2.3 Teoria de controle

A teoria de controle é uma abordagem matemática para solucionar problemas no comportamento de sistemas dinâmicos. Dentro dessa teoria existem dois principais métodos, sendo estes o controle em malha aberta e o controle em malha fechada. Este último em especial busca a partir de uma realimentação fazer com que a resposta do sistema siga uma referência desejada, garantindo a estabilidade do mesmo a fim de evitar respostas indesejadas.

2.3.1 Controle em malha fechada

O controle em malha fechada se baseia, como o nome já diz, em uma malha fechada que em sua forma mais simples envolve a planta a ser controlada, juntamente com um controlador, que é outro sistema responsável por controlar o sistema a fim de seguir a referência que se deseja, e, se necessário, um transdutor ou sensor, que converterá a saída para o tipo de variável desejada (mecânica, elétrica, etc), como mostrado na Figura 2.5.

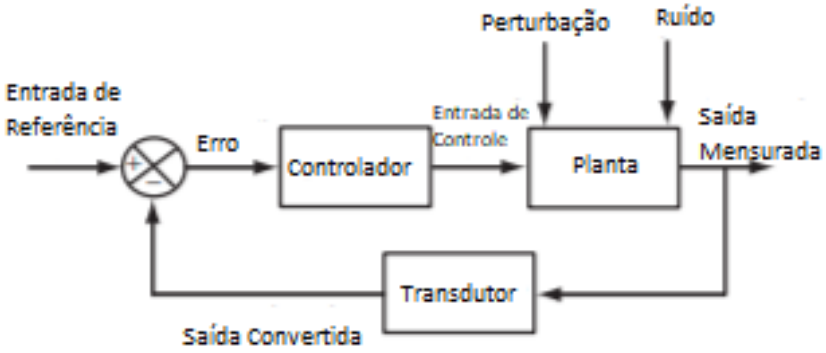

Figura 2.5: Malha fechada básica (traduzido de [3])

Da Figura 2.5 podem ser explicitadas variáveis importantes dentro do sistema de controle, que serão listadas a seguir.

- A referência r(t), que é o valor desejado para a saída do sistema;
- O erro e(t), que é a diferença entre a referência e a saída atual e que será alimentado para o sistema composto por planta e controlador;
- A entrada de controle u(t), que é a saída do controlador para a planta;
- A perturbação d(t), que são entradas adicionais do sistema, as quais o controlador deve responder e que são modeladas no projeto do controlador;
- O ruído n(t), que são entradas não modeladas mas que podem ter seu efeito diminuído;
- A saída y(t), que é a saída da planta em si.

O projeto do controlador se baseia na planta a ser controlada e nos requisitos que se deseja para a resposta, visando um comportamento adequado do sistema para a aplicação. Requisitos usualmente utilizados são o tempo de subida, que é o tempo que a saída leva para atingir seu primeiro pico, o denominado *overshoot*, que é a porcentagem que o valor da saída ultrapassa o valor desejado para a mesma, e o tempo de assentamento, que é o tempo que o sistema leva para atingir uma margem de erro aceitável e permanecer nela. Além disso, existem requisitos que o sistema deve possuir como a estabilidade e erro em estado estacionário baixo ou nulo, sendo este último a diferença entre o valor obtido com o controle e o valor desejado após o regime transiente, visando um sistema funcional.

2.3.2 Teoria de controle e sistemas tempo real

Para que o escalonador de um sistema tempo real se adapte para funcionar baseado na teoria de controle devem-se ter em mente primeiramente dois conceitos de grande importância. O primeiro

é o conceito de variável controlada, que é a grandeza que se deseja controlar e cujo valor definirá o desempenho do sistema, baseado no erro com relação à referência. O segundo é o conceito de variável manipulada, que é a grandeza que será alterada visando atingir um valor de referência da variável controlada. Uma ilustração dessa adaptação é mostrada na Figura 2.6.

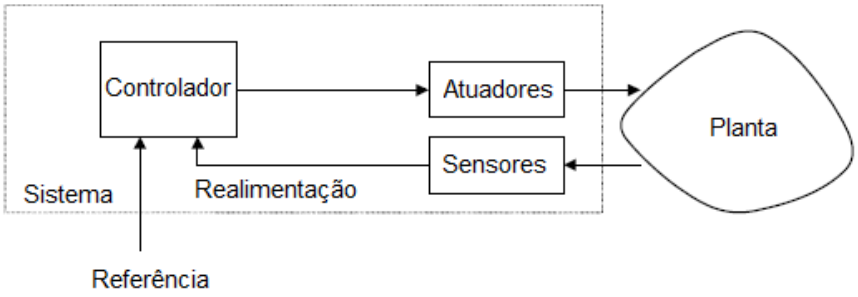

Figura 2.6: Controle realizado no sistema computacional pelo escalonador (traduzido de [4])

Como um sistema tempo real envolve um sistema computacional, as atividades realizadas para o controle podem ser descritas em passos como a seguir. [4]

- O sistema periodicamente monitora e compara o valor da variável controlada com a referência para determinar o erro;
- O controlador, que nesse caso é o próprio escalonador, computa o controle, ou seja, a quantia pela qual deve ser variada a variável manipulada, a partir da função de controle baseado no erro;
- Os atuadores alteram o valor da variável manipulada de maneira a controlar o sistema.

Finalmente, para se ter um sistema tempo real em malha fechada deve-se definir as variáveis manipulada e controlada, o valor de referência desejado, os atuadores, a margem de erro aceitável e a função de controle. Vale ressaltar que tais escolhas dependem dos requisitos de projeto e do objetivo do sistema, havendo diversas configurações possíveis com cada uma delas obtendo resultados diferentes.

2.3.3 Controle PID

O controle PID (Proporcional-Integral-Derivativo) é uma teoria de controle bem consolidada e amplamente utilizada tanto academicamente como industrialmente [24]. Tal teoria explicita um controlador com 3 ganhos: o proporcional (K_p), o integral (K_i) e o derivativo (K_d), como mostrado na Equação 4.4(a), em que se tem a função de transferência do mesmo no domínio da frequência.

$$G(s) = K_p + K_i\frac{1}{s} + K_d s \tag{2.11}$$

Adaptando tal teoria para sistemas computacionais deve-se primeiramente considerar a fórmula no domínio do tempo e após isso discretizá-la, resultando na equação 2.12 (adaptado de [4]).

$$g(k) = K_p u(k) + K_i \sum_{IW} u(k) + K_d \frac{u(k) - u(k - DW)}{DW} \tag{2.12}$$

Onde u(k) é a entrada do controlador e g(k) sua saída em determinado momento k. Além disso tem-se os parâmetros IW e DW (*Integral Window* e *Derivative Window*, respectivamente) que são as janelas de tempo para a integração, que no caso discreto é um somatório de um número finito de termos IW, e para a derivação, que no caso discreto é um termo em que os dois pontos utilizados para o cálculo são separados por DW amostras, ou seja, um termo sendo da amostra mais recente k e o outro da amostra $k - DW$.

2.3.3.1 Método Ziegler-Nichols

O método Ziegler-Nichols é um método amplamente difundido na comunidade de controle [25], utilizado para sintonização dos parâmetros de um controlador PID. Existem dois tipos do método: o primeiro é aplicável para controle em malha aberta e o segundo é aplicável para controle em malha fechada. Como neste trabalho estamos lidando com controle em malha fechada, temos que a ideia por trás do segundo método é simples e pode ser descrita nos seguintes passos [7]:

1. Faz-se K_i e K_d iguais a zero.
2. Aumenta-se K_p para um valor crítico onde o sistema apresenta uma oscilação harmônica como resposta.
3. Verifica-se o valor do ganho crítico K_{cr} e do período crítico P_{cr} e calcula-se os parâmetros do controlador a partir da Tabela 2.1. Vale ressaltar que $K_i = T_i^{-1}$.

K_p	T_i	K_d
$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Tabela 2.1: Tabela do método Ziegler-Nichols para cálculo dos parâmetros do controlador PID (adaptado de [7])

Após a definição inicial dos parâmetros faz-se um ajuste fino empírico dos parâmetros levando em conta o efeito de cada um no sistema, ou seja, como a variação de seu valor afeta a resposta do mesmo. Os efeitos para um aumento do valor de cada um dos ganhos são listados a seguir.

- K_p: Um aumento em seu valor diminui o tempo de subida, aumenta o overshoot, não afeta consideravelmente o tempo de assentamento e diminui o erro estacionário.

- K_i: Um aumento em seu valor diminui o tempo de subida, aumenta o overshoot, aumenta o tempo de assentamento e elimina o erro em estado estacionário.
- K_d: Um aumento em seu valor não afeta consideravelmente o tempo de subida, diminui o overshoot, diminui o tempo de assentamento e não afeta o erro estacionário.

2.4 Body Sensor Network

Body Sensor Networks (BSNs) ou Body Area Networks (BANs) são redes de sensores corporais sem fio que medem de maneira contínua informações fisiológicas como temperatura, batimento cardíaco etc. Avanços tecnológicos das últimas décadas, como tecnologias de comunicação sem fio mais eficientes e produção em massa de sensores miniaturizados, têm proporcionado um cenário favorável para a utilização massiva das BSNs nas áreas esportiva, de jogos interativos, militar, e, principalmente, médica. Informações mais detalhadas sobre o funcionamento, arquitetura da rede e dos sensores utilizados e comparações com redes de sensores sem fio tradicionais podem ser encontradas no Anexo I.

Estudos realizados a respeito desses sistemas mostram as características essenciais que eles devem possuir para que possam ser amplamente utilizados. Em [26] foi realizada uma reunião das informações mostradas nesses trabalhos e percebeu-se que os maiores desafios no projeto das BSNs são o equilíbrio entre processamento e comunicação, a taxa de aquisição de dados e o consumo de energia, maior atenuação de interferências em relação à outras redes de sensores sem fio e a capacidade de armazenamento dos nós.

Devido ao crescente aumento da população idosa e à grande parcela da população com deficiências graves, se faz necessário no ambiente médico o uso de tecnologias que auxiliem os processos clínicos para que estes sejam sustentáveis e eficientes. De acordo com *Nadeem et al.* [26], a Organização Mundial da Saúde (OMS) relatou que existem cerca de 600 milhões de pessoas com 60 anos ou mais, sendo esperado que esse número cresça para 1.2 bilhão de pessoas no ano de 2025, e também que cerca de 1 bilhão de pessoas sofrem de algum tipo de deficiência, sendo que 10% desse número são vítimas de deficiências severas como paralisia e cegueira. Nesse contexto, as BSNs podem vir a ter um papel muito importante na transformação do ambiente médico, auxiliando na obtenção de diagnósticos mais precisos e provendo monitoramento constante.

Capítulo 3

Desenvolvimento

3.1 Introdução

Para o desenvolvimento do projeto foi necessário primeiramente definir as características da BSN a ser modelada em conjunto com o escalonador e o funcionamento da mesma, dado que assim poderá ser verificado o modelo do escalonador e avaliada sua corretude. A BSN foi pensada como um sistema distribuído composto por nós sensores onde se encontram os *sensornodes*, que são as aplicações responsáveis pela aquisição e pré-processamento dos dados. Além disso, existe um nó central onde se encontra a *bodyhub*, que é a aplicação responsável pelo processamento dos dados e assumido como um módulo mais robusto que os nós sensores, possuindo maior quantidade de memória e poder de processamento. Os sensores apenas se comunicam com o nó central a partir de trocas de mensagens, não havendo memória compartilhada, e o processamento é realizado localmente em cada um dos componentes do sistema.

Cada nó tem uma camada de software de aplicação implementada que é controlada pelo *middleware* OpenDaVINCI, sendo este outra camada de software que está entre a aplicação e o sistema operacional escolhido que permite o gerenciamento dos dados e a comunicação entre os módulos do sistema que estão distribuídos e se comunicam via rede sem fio. Uma ilustação deste funcionamento é mostrada na Figura 3.1. Todo o escalonamento é realizado pelo middleware, o qual monitora as taxas de perda e realiza um controle das prioridades de escalonamento a partir delas.

Algumas outras considerações devem ser esclarecidas pois sustentam o modelo criado neste trabalho. A primeira diz respeito ao escalonador implementado, onde este pode estar atuando dentro do nó central ou ainda em um nó separado, porém usualmente atua dentro do nó central em conjunto com a *bodyhub*. A segunda diz respeito à confiabilidade dos dados, sendo esta papel do sensor, ou seja, o sensor irá, a partir de uma política predefinida, verificar se o dado é confiável ou não, sendo enviado no primeiro caso e descartado no segundo. A política utilizada para a verificação de confiabilidade foi uma política de redundância onde se realizam até 5 leituras, enviando o dado caso 3 leituras se encontrem no mesmo nível e descartando-o caso contrário. A terceira e última diz respeito à ativação do sistema, onde todos os nós são ativados periodicamente.

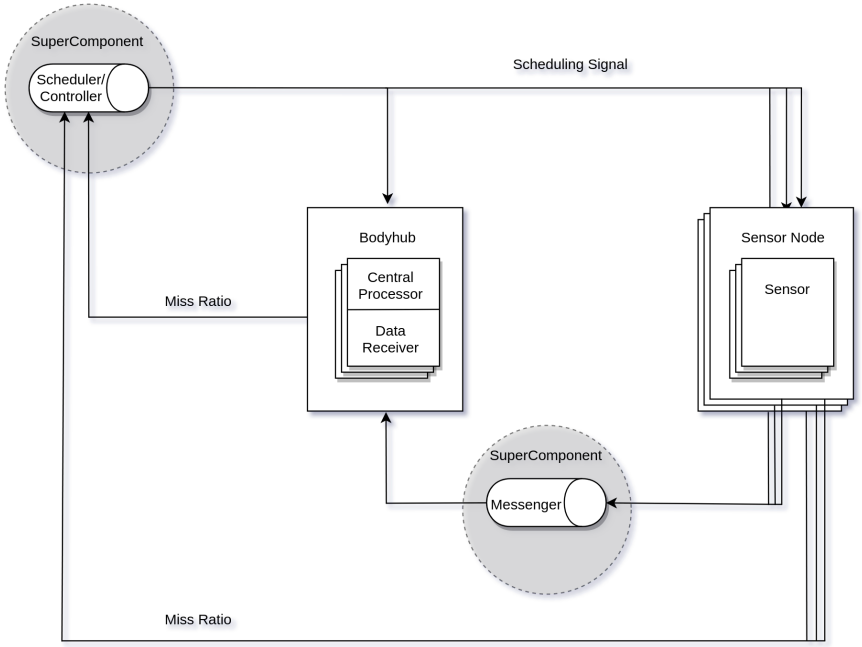

Figura 3.1: Funcionamento básico do sistema

Além do que foi citado anteriormente, vale ressaltar que os valores assumidos para períodos, *deadlines* e tempos computacionais do pior caso, *worst case execution times* (WCETs) em inglês, foi arbitrário para fins de generalização dos resultados.

3.2 Modelagem no UPPAAL

O modelo no UPPAAL é uma modelagem do nó central e o escalonador EDF para o mesmo, sendo que a modelagem do sensor foi feita de maneira simples uma vez que assume-se que este realiza verificações para que seja feito apenas o envio de dados confiáveis. Nesta seção, serão descritos todos os 7 modelos locais, ou `templates`, que foram criados dentro do modelo global, sendo eles: `scheduler`, `Aux_Scheduler`, `centralprocessor`, `datareceiver`, `gen_timer`, `sensor` e `observer`. Neste modelo realizou-se não só a introdução do escalonador EDF ao invés do escalonador FIFO como alterações estruturais com relação ao modelo implementado por Farias et al [15]. Isto foi feito buscando adequá-lo melhor à visão de um sistema distribuído e facilitar a verificação de novos requisitos temporais decorrentes da mudança de escalonador, principalmente na questão de obediência às deadlines. Além disso, uma ilustração em alto nível da estrutura global da modelagem é dada na Figura 3.2, onde são mostrados os modelos locais e os canais de comunicação entre eles.

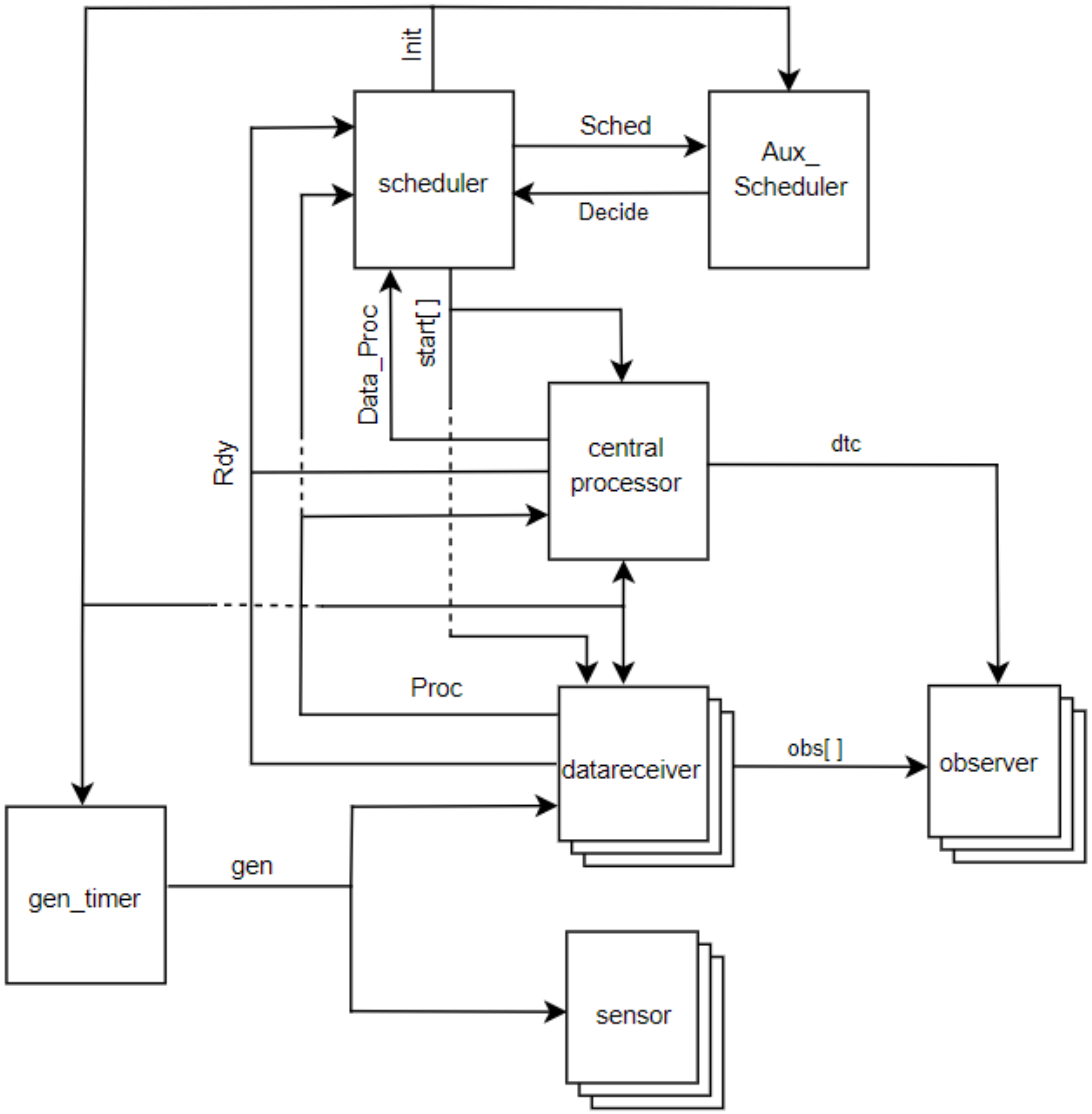

Figura 3.2: Ilustração em alto nível da modelagem no UPPAAL

3.2.1 Variáveis e funções globais

Antes de dar início a descrição dos modelos locais é importante deixar claro quais são as variáveis e funções globais e suas utilidades, uma vez que estas podem ser usadas em qualquer local que se desejar, respeitando a visão de um sistema distribuído onde não há compartilhamento de memória entre sensores e nó central.

3.2.1.1 Variáveis globais

`const int SENSORS` - Variável que instancia o número de sensores a serem considerados pelo UPPAAL, aumentando o número de modelos locais do `datareceiver` que é parametrizado de acordo com esse valor.

`const int BUFFER_SIZE` - Variável que expressa o número de dados na fila do nó central, sendo dado como igual ao número de sensores no modelo para fins de simplificação da verificação e porque assume-se que cada sensor terá apenas uma leitura na fila. Em um sistema real, esse tamanho dependerá da memória do mesmo e poderá compensar eventuais desvios de comportamento que ele venha a ter.

`const int low, medium, high` - Variáveis que definem um valor para os níveis que cada sensor possa vir a atingir. O valor low é dado como 1, o valor medium é dado como 2 e o valor high é dado como 3.

`const int PD` - Tempo de execução escalonador, sendo o tempo que o mesmo leva para poder tomar uma decisão do próximo processo a ser executado. Considerado como 25 ms.

`const int Cb` - Tempo de execução do `centralprocessor`, considerado como 10 ms.

`const int Cs` - Tempo de execução de cada `datareceiver`, considerado como 5 ms.

`const int SP` - Período dos sensores, considerado como 1000 ms (1s).

`const int Low_Deadline` - Deadline de cada `datareceiver`, sendo considerada como 200 ms.

`const int Processor_Deadline` - Deadline do `centralprocessor` em execução normal, considerada como 2000 ms (2s) visando a execução prévia de todos os `datareceivers` para que assim o mesmo possa executar.

`const int Critical_Processor_Deadline` - Deadline do `centralprocessor` quando um dado crítico é detectado, considerado como 50 ms.

`typedef int[0,0] bh_t` - Definição de tipo utilizada para parametrização do número de `centralprocessors`, assumindo apenas o valor 0.

`typedef int[1,SENSORS]` sensor_t - Definição de tipo utilizada para parametrização dos sensores, definindo a identificação dos mesmos. Pode assumir valores de 1 a SENSORS, variando de acordo com o número de sensores definidos.

`typedef int[0,SENSORS] module_t` - Definição de tipo para identificação de todos os módulos pertencentes ao nó central.

`typedef struct Data` - Definicão do tipo de dados que será enviado pelos `datareceivers` para o `centralprocessor` possuindo como valores o estado do sensor no último dado processado (int[0,high] status) e a identificação do mesmo (int[0,SENSORS] id).

`int[low,high] m_status[sensor_t]` - Vetor de estados dos sensores, sendo a posição referente a cada sensor enviada ao seu respectivo `datareceiver`.

`Data sn_status[sensor_t]` - Vetor de dados dos `datareceivers`, onde cada posição corresponde a um em específico.

`Data m_buffer[BUFFER_SIZE]` - Vetor de dados que representa o *buffer* do `centralprocessor`.

`chan start[module_t], Data_Proc, Rdy, obs[sensor_t]` - Canais diversos para comunicação entre módulos.

`broadcast chan gen, Init, Proc, Sched, Decide, dtc` - Canais de *broadcast* diversos para comunicação entre módulos.

`bool Ready[module_t]` - Vetor que expressa se os módulos que fazem parte do nó central, sendo eles os `datareceivers` e o `centralprocesor`, estão aptos a processar.

`bool critical` - Variável que indica se o sistema realizou uma leitura de nível crítico.

`bool crit[sensor_t]` - Vetor de variáveis que expressam se o dado encontrado pelo datareceiver correspondente está ou não em nível alto.

`module_t mid, m` - Variáveis para identificar qual módulo está sendo considerado em alguma operação do escalonador. A variável mid indica qual módulo indicou que está apto a processar e a variável m indica o módulo escolhido pelo escalonador para processamento.

`clock m_clks[module_t]` - Relógios globais de cada um dos processos do nó central.

`int[0,BUFFER_SIZE]` len - Variável que expressa o tamanho do buffer que está sendo utilizado em cada momento.

`int Deadlines[module_t]` - Vetor das *deadlines* de cada processo pertencente ao nó central.

3.2.1.2 Funções globais

`bool isEmpty()` - Função que indica se o *buffer* do `centralprocessor` está vazio.

`bool isFull()` - Função que indica se o *buffer* do `centralprocessor` destá cheio.

`Data dequeue()` - Função que retira um dado do *buffer* do `centralprocessor`.

`void insert(Data message)` - Função que insere um dado no buffer `centralprocessor`.

3.2.2 scheduler

Módulo responsável pelo escalonamento do nó central, tomando decisões de qual processo deve ser executado dados os estado do sistema naquele momento. Seu modelo local é mostrado na Figura 3.3

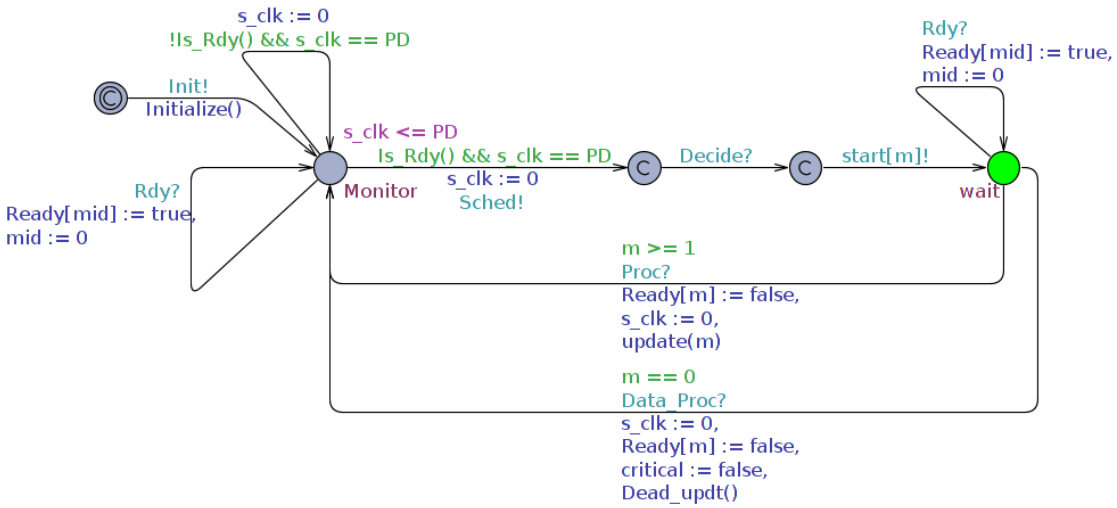

Figura 3.3: Modelo local do scheduler

Em seu estado inicial, este módulo é reponsável por enviar o sinal `Init` para inicialização do sistema. Uma vez incializado e enquanto não há nenhum processo executando, ele permanece no estado `Monitor` monitorando as atividades do sistema e periodicamente verificando se há um ou mais processos prontos para serem executados. Caso haja pelo menos um processo apto a executar altera-se o estado e envia-se um sinal para o `Aux_Scheduler`, uma vez que este é o mecanismo de decisão do processo a ser executado, e espera o final da execução do mesmo dada a partir de um sinal `Decide`. Com a decisão tomada, é enviado um sinal `start` para o processo correspondente para que este inicie sua execução, permanecendo no `wait` monitorando o sistema e à espera de um sinal `Proc` advindo dos `datareceivers`, ou de um sinal `Data_Proc` advindo do `centralprocessor`.

As variáveis e funções locais utilizadas pelo módulo são listadas a seguir.

`clock s_clk` - Relógio interno para contagem do tempo de execução.

`void Initialize()` - Função de inicialização do sistema, onde são alterados os valores das variáveis `Deadlines[id]`, das variáveis `Ready[id]` e dos parâmetros `id` das variáveis `sn_status[id]`.

`Dead_updt()` - Função que altera o valor da deadline do `centralprocessor`, caso esta tenha sido alterada dada a detecção de um estado crítico.

`update()` - Função que altera o valor da deadline do `centralprocessor` para o valor de estado crítico, caso esta esteja em seu valor normal.

`Is_Rdy()` - Função que verifica se algum processo está apto a executar.

3.2.3 Aux_Scheduler

Módulo responsável por tomar as decisões de escalonamento baseado na política de escalonamento EDF, não modelando nenhum componente do sistema real. Sua necessidade se dá pelo fato

de não serem permitidas comparações entre variáveis do tipo `clock` no UPPAAL, por conta de sua representação simbólica, mas por serem permitidas comparações nas condições de guarda. Seu modelo local é mostrado na Figura 3.4.

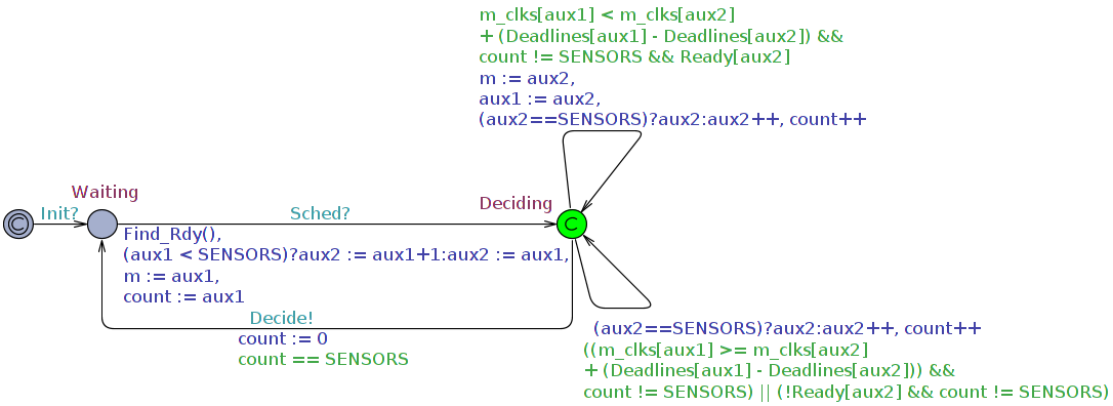

Figura 3.4: Modelo local do Aux_Scheduler

Ao inicializar o sistema, este módulo se encontra em seu estado inicial à espera de um sinal `Init` advindo do `scheduler`. Ao receber o sinal, ele transita para o estado `Waiting` onde ficará a espera do sinal `Sched`, também proveniente do `scheduler`, que dará início ao processo de decisão no estado `Deciding`. Na transição do estado é verificado qual o primeiro processo que está apto a processar e atribuído seu identificador ao valor de uma variável de decisão auxiliar para da início ao processo que comparará o valor das deadlines absolutas utilizando os valores das variáveis `m_clks[id]` e `Deadlines[id]`, enviando um sinal `Decide` para o `scheduler` ao final.

As variáveis e funções locais utilizadas pelo módulo são listadas a seguir.

`int[0,SENSORS] aux1` - Variável auxiliar do processo de decisão que armazena o menor valor de identificador dentre os dois processos sendo comparados.

`int[1,SENSORS+1] aux2` - Variável auxiliar do processo de decisão que armazena o maior valor de identificador dentre os dois processos sendo comparados.

`int[0,SENSORS] count` - Contador do número de sensores analisados.

`void Find_Rdy()` - Função que encontra o primeiro processo apto a executar.

3.2.4 centralprocessor

Módulo responsável pelo processamento central das informações advindas dos `datareceivers`, analizando e agrupando dados já classificados e tomando alguma ação que seja necessária, como um envio de sinal para algum dispositivo externo. Seu modelo local é mostrado na Figura 3.5

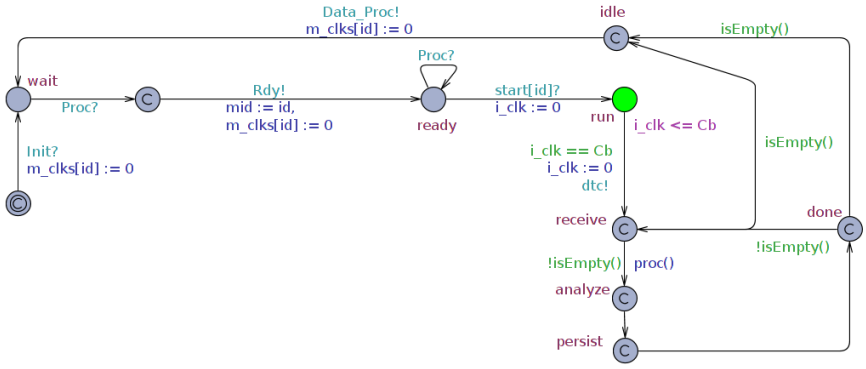

Figura 3.5: Modelo local do centralprocessor

Ao iniciar o sistema, este módulo se encontra na posição inicial à espera do sinal `Init`. Após o recebimento do sinal, ele permanece na posição `wait` até o recebimento de outro sinal, denominado `Proc`, que virá do primeiro datareceiver processado, se tornando assim apto a processar e enviando um sinal `Rdy` para o escalonador. Após se tornar apto a processar, ele permanece na posição `ready` até o envio de um sinal `start[0]` vindo do processador, sendo válido ressaltar aqui que o número de identificação deste módulo é 0. Com o sinal do escalonador, o módulo passa para o estado `run`, que modela o tempo para o processamento do mesmo, passando após um tempo igual a `Cb` para os estados que representam suas ações propriamente ditas e enviando ao mesmo tempo um sinal `dtc` para o observador. Sua execução termina quando não existem mais dados a serem processados, mediante o envio de um sinal `Data_Proc` para o escalonador.

As variáveis e funções locais utilizadas pelo módulo são listadas a seguir.

`clock i_clk` - Relógio interno para contagem do tempo de execução.

`Data hstatus` - Variável que representa o dado sendo processado.

`Data received_data` - Variável que representa o dado retirado do *buffer*.

`void proc()` - Função que retira o dado do *buffer* e atribui os valores para as variáveis received_data e hstatus.

3.2.5 datareceiver

Módulo responsável por representar os receptores de dados, sendo parametrizado de acordo com o número desejado de sensores a partir de um parâmetro `const sensor_t id` que faz com que o UPPAAL crie quantas instâncias forem necessárias do módulo, de acordo com a variável `SENSORS`, e atribua para cada instância uma identificação única dentro da faixa de valores possíveis. Suas funções são a de classificação dos dados recebidos e sinalização para o escalonador de que há um dado crítico, fazendo com que o mesmo altere a *deadline* do `centralprocessor`. Seu modelo local é mostrado na Figura 3.6

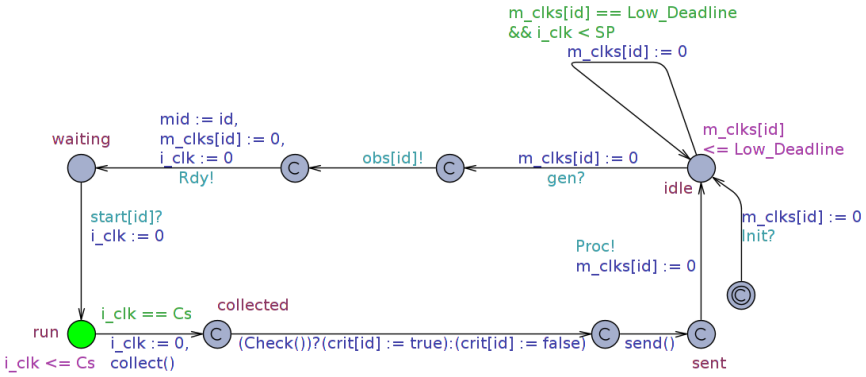

Figura 3.6: Modelo local do datareceiver

Ao iniciar o sistema, este módulo se encontra na posição inicial à espera do sinal `Init`. Após o recebimento do sinal, ele permanece na posição idle até o recebimento do sinal `gen` advindo do `gen_timer` para assim que o receber sinalizar ao escalonador que está apto a processar através de um sinal `Rdy` e ir para a posição waiting, onde esperará o sinal `start[id]` advindo do mesmo, sendo válido ressaltar que `id` é um identificador único, do tipo `sensor_t`, de cada módulo `datareceiver` e devido a isto varia entre 1 e `SENSORS`. Ao receber o sinal do escalonador, o módulo passa para o estado `run` onde ficará por um tempo `Cs`, simulando o tempo de processamento, e depois passará para seu processamento em si, alterando a variável `crit[id]` a depender se o estado do dado lido é alto ou não e enviando um sinal `Proc` para o `centralprocessor` e escalonador.

As variáveis e funções locais utilizadas pelo módulo são listadas a seguir.

`clock i_clk` - Relógio interno para contagem do tempo de execução.

`void collect()` - Função que coleta o dado recebido do sensor e atribui ao vetor `sn_status[id]`.

`bool Check()` - Função que verifica se o dado do sensor está em estado alto ou não.

`void send()` - Função que insere o dado `sn_status[id]` no *buffer* do `centralprocessor`.

3.2.6 gen_timer e sensor

Módulos responsáveis pela modelagem do funcionamento do sensor. O módulo `gen_timer` aciona as instâncias do módulo `sensor`, parametrizado de acordo com a variável `SENSORS` a partir de um parâmetro `const sensor_t id`, a cada período de tempo igual a `SP`, fazendo com que este mude seu estado e altere o valor da variável `m_status[id]`, modelando o envio do dado para os respectivos `datareceivers`. Seus modelos locais são mostrados nas Figuras 3.7 e 3.8.

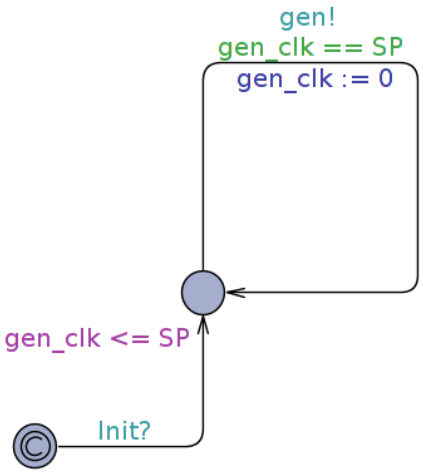

Figura 3.7: Modelo local do gen_timer

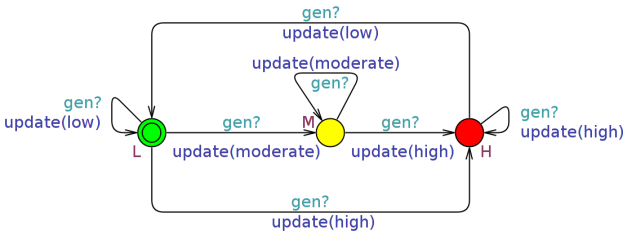

Figura 3.8: Modelo local do sensor

O módulo `gen_timer` começa no estado inicial à espera do sinal de inicialização `Init`. Após o recebimento do sinal seu funcionamento é apenas esperar um tempo igual a `SP`, como dito anteriormente, para o envio de um sinal `gen` para o `sensor` e para os `datareceivers`. O módulo `sensor`, inicialmente no estado L que indica estado baixo, ao receber o sinal `gen` pode transitar para o estado M, que indica estado moderado, para o estado H, que indica estado alto do mesmo, ou ainda permanecer no estado L, possuindo comportamento semelhante quando se encontra nos outros estados. Vale ressaltar que as transições de estado no modelo do sensor podem ocorrer com a mesma probabilidade em todos os casos, havendo um comportamento randômico dos dados utilizados para exercitar os outros modelos.

As variáveis e funções locais utilizadas pelos módulos são listadas a seguir.

`clock gen_clk` - Relógio interno para contagem do tempo de execução do `gen_timer`.

`void update(int[low,high] new_status)` - Função do módulo `sensor` que insere na variável `m_status[id]` o valor passado como parâmetro da função, sendo este igual a low, medium ou high.

3.2.7 observer

Módulo utilzado apenas para fins de verificação, não modelando nenhum componente do sistem real em si. Além disso, é parametrizado de acordo com a variável `SENSORS` a partir de um parâmetro `const sensor_t id`. Seu modelo local é mostrado na Figura 3.9

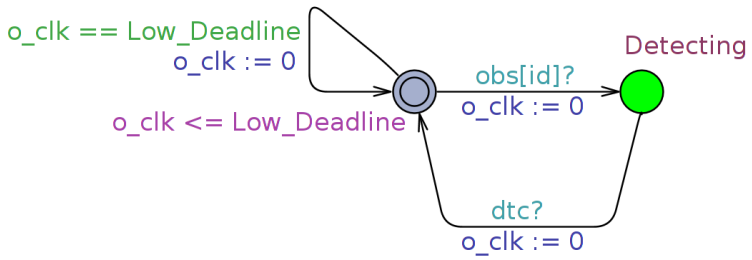

Figura 3.9: Modelo local do observer

O observer permanece em seu estado inicial até receber um sinal `obs[id]` advindo do sensor corresponde à cada instância, começando a partir daí a contar o tempo, a partir de um relógio local, até o recebimento de um sinal `dtc` advindo do `centralprocessor`, que indica que o dado do sensor já foi processado. Busca-se a partir disso verificar se existe algum caminho que, a partir do recebimento do dado do sensor, o sistema ultrapassa um determinado tempo predefinido para a detecção do nível do mesmo.

As variáveis e funções locais utilizadas pelo módulo são listadas a seguir.

`clock o_clk` - Relógio interno para contagem do tempo.

3.3 Implementação do escalonador EDF no *middleware* OpenDaVinci

3.3.1 Visão geral

A implementação do escalonador EDF se deu como uma adição ao código do *middleware* OpenDaVinci, escrito na linguagem C++, que possuía originalmente apenas um escalonador não determinístico deixando todo o escalonamento determinístico dos módulos de *software* por parte do sistema operacional. O escalonador adicional foi implementado para ter funcionamento completo apenas em sistemas Linux/POSIX, onde se tem um kernel apropriado para aplicações em tempo real. Além disso, o escalonador apenas é capaz de escalonar tarefas com comportamento periódico, assim como os processos utilizados na BSN.

O OpenDaVinci possui uma arquitetura orientada a serviços *publish/subscribe* para a comunicação entre módulos. Ele possui um módulo central denominado SuperComponent, que é responsável pela troca de mensagens entre todos os módulos a ele conectados por meio de conexões TCP ou UDP. Essa estrutura básica é mostrada na Figura 3.10, onde N módulos estão conectados ao SuperComponent.

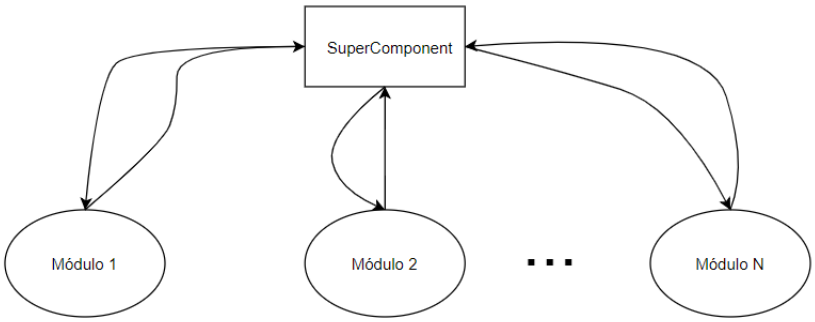

Figura 3.10: Estrutura básica de comunicação do OpenDaVinci

3.3.2 Inicialização dos módulos

3.3.2.1 SuperComponent

Como para qualquer comunicação existir deve-se haver a inicialização de pelo menos um módulo SuperComponent, com um número de identificação específico, certos parâmetros foram adicionados na hora da inicialização do mesmo, os quais deverão ser declarados no terminal. Tais parâmetros são listados a seguir.

- scheduling: Parâmetro que irá definir se o escalonamento utilizado pelo módulo SuperComponent será EDF ou se será o escalonamento não determinístico original;
- scheduling_log: Parâmetro que define se os dados do escalonamento serão ou não registrados em um arquivo de extensão .txt.

Além disso, uma inicialização padrão do SuperComponent para que o escalonamento utilizado seja EDF é mostrada abaixo e uma descrição dos parâmetros ainda não descritos é dada a seguir.

```
sudo odsupercomponent –cid=111 –configuration=/configuration –managed=simulation_rt –freq=1000
    –verbose=0 –scheduling=edf –scheduling_log=yes –realtime=10
```

- cid: Parâmetro que define o número de identificação;
- configuration: Parâmetro que especifica o caminho de um arquivo de configuração para inicialização de determinadas variáveis do SuperComponent;
- managed: Parâmetro que define como serão tratadas as mensagens recebidas, sendo que o escalonador EDF apenas funciona caso o valor do mesmo seja *simulation* ou *simulation_rt*;
- freq: Parâmetro que define a frequência de funcionamento;
- verbose: Parâmetro que define se certas informações serão visualmente mostradas ao usuário por meio do terminal;

- realtime: Parâmetro que define se o escalonamento do sistema operacional será próprio para aplicações em tempo real, sendo seu valor a prioridade inicial do SuperComponent dentro deste.

3.3.2.2 Outros módulos

Qualquer outro módulo que deseje se conectar ao SuperComponent também deve ser inicializado e, para o correto funcionamento do sistema, certos parâmetros devem ser propriamente especificados. Os parâmetros adicionados na inicialização dos módulos são mostrados a seguir.

- deadline: Parâmetro que especifica a deadline relativa do módulo em microssegundos;
- wce: Parâmetro que define o tempo de execução de pior caso do módulo.
- scheduling_priority: Parâmetro que define a prioridade inicial do módulo no escalonamento realizado pelo SuperComponent;
- control_type: Parâmetro que define o tipo de controle a ser realizado com o módulo podendo ser via controle simples caso assuma o valor 1, controle PID caso assuma o valor 2, e sem controle caso assuma qualquer outro valo ou não seja declarado.

Além disso, uma iniciaização padrão de um módulo de nome examplemodule para que este esteja devidamente configurado para um escalonamento EDF é mostrada abaixo e uma descrição dos parâmetros ainda não descritos é dada a seguir.

./examplemodule –freq=10 deadline=1000000 –wce=100 –configuration=~/configuration –cid=111 –id=0 –scheduling_priority=0 –control_type=0

- freq: Define a frequência de funcionamento;
- configuration: Parâmetro que especifica o caminho de um arquivo de configuração para inicialização de determinadas variáveis do módulo;
- cid: Parâmetro que define o número de identificação do SuperComponent ao qual o módulo deve ser conectado;
- id: Parâmetro que define o número de identificação do módulo;

3.3.3 Funcionamento

Dada a dificuldade de sempre garantir as *deadlines* em um sistema computacional, devida sua imprevisibilidade, buscou-se, baseado na ideia mostrada em [4], implementar no escalonador a função de um controlador. Para esse controlador a variável controlada é a perda de deadlines de cada módulo em dado instante ($M_{ri}(k)$) e a variável manipulada é a prioridade dos módulos

relacionados ao OpenDaVINCI e conectados ao SuperComponent. Tal ideia foi implementada de duas maneiras: por meio de um controlador simples e por meio de um controlador PID, os quais serão detalhados nas seções seguintes.

O SuperComponent calcula a taxa de perda de deadlines para um dado módulo i ($M_{ri}(k)$) da seguinte maneira:

$$M_{ri}(k) = \frac{N^o\, de\, instancias\, que\, perderam\, a\, deadline}{N^o\, total\, de\, instancias} \tag{3.1}$$

Essa taxa é calculada periodicamente e é a variável controlada do sistema de controle, tendo este como referência $M_r(k) = 0$. A variável manipulada é a prioridade do módulo frente aos outros módulos no escalonamento realizado pelo SuperComponent, a qual se sobrepõe à deadline do módulo, ou seja, primeiramente analisa-se o valor desta e caso hajam módulos com mesma prioridade analisa-se o valor da deadline como critério de seleção. Um algoritmo simples que representa esta ideia é mostrado a seguir.

Algorithm 1: Algoritmo para exemplificar o funcionamento do escalonamento EDF utilizando módulos com prioridades

```
1  available_modules = []
2  for module in modules do
3      if module.is_available() then
4          available_modules.append(module)
5  while available_modules.length > 0 do
6      max_priority = max(module.priority for module in available_modules)
7      possible_modules = []
8      for module in available_modules do
9          if module.priority == max_priority then
10             possible_modules.append(module)
11     while possible_modules.length > 0 do
12         min_deadline = min(module.deadline for module in possible_modules)
13         for module in possible_modules do
14             if module.deadline == min_deadline then
15                 if Actual_time ≤ module.deadline then
16                     execute(module)
17                 break
18         available_modules.erase(module)
19         possible_modules.erase(module)
```

Basicamente, avalia-se quais módulos estão aptos a executar e, dentre estes, avalia-se quais módulos possuem a maior prioridade. Dos módulos com maior prioridade, verifica-se aquele com a

menor *deadline* e, caso o tempo atual seja menor que esta, o executa e o apaga da lista de módulos aptos a executar e da lista dos módulos de maior prioridade. Caso a deadline seja menor que o tempo atual, simplesmente apaga-se o módulo da lista de módulos aptos a executar e da lista dos módulos de maior prioridade e repete-se o processo de escolha pela *deadline* até que esta última esteja vazia. Após verificar todos o módulos da maior prioridade, repete-se o processo de criação da lista dos módulos de maior prioridade, porém desta vez sem os módulos que foram anteriormente escolhidos pela *deadline* e assim em diante, até que a lista dos módulos aptos a executar esteja vazia.

3.3.4 Controlador Simples

Neste controlador a prioridade (p) é calculada fazendo-se uma associação direta entre esta e a taxa de perda $M_r(k)$, com as duas limitadas ao mesmo intervalo [0,100] porém com a prioridade apenas assumindo valores inteiros. O controle realizado é mostrado a seguir, onde num_modules é o número de módulos conectados ao SuperComponent e p_i e M_{ri} são a prioridade e a taxa de perda para cada módulo, respectivamente.

Algorithm 2: Funcionamento do controlador simples

```
for i from (0,num_modules) do
    p_i = round(M_ri(k))
```

O arredondamento realizado no valor da taxa de perda é realizado a partir do truncamento do valor, ou seja, retira-se a parte fracionária da mesma e compõe-se assim a prioridade.

3.3.5 Controlador PID

O controlador PID é utilizado como explicitado na Equação 2.12, onde a saída é a variação de prioridade desejada (Δp) e a entrada é perda de deadlines em dado momento ($M_r(k)$), como mostrado em 3.2.

$$\Delta p = K_p M_r(k) + K_i \sum_{IW} M_r(k) + K_d \frac{M_r(k) - M_r(k - DW)}{DW} \tag{3.2}$$

Como a referência desejada é uma taxa de perda nula e temos realimentação negativa no sistema, a Equação 3.2 se torna a Equação 3.3, similar ao mostrado por Stankovic et al [4].

$$\Delta p = -K_p M_r(k) - K_i \sum_{IW} M_r(k) + K_d \frac{M_r(k - DW) - M_r(k)}{DW} \tag{3.3}$$

Com o valor de Δp em mãos, apenas soma-se este ao valor atual da prioridade p resultando em um novo valor p', como na Equação 3.4. Vale ressaltar que neste controlador a prioridade também se encontra no intervalo [0,100].

$$p' = p + \Delta p \tag{3.4}$$

3.3.5.1 Determinação dos parâmetros do controlador PID

Para a determinação dos parâmetros do controlador PID utilizou-se uma heurística simples, que será descrita nesta seção. Primeiramente definiu-se um número de passos no qual seria desejado que, em casos extremos, o sistema respondesse completamente às variações na taxa de perda, sendo tal caso o seguinte:

1) A alteração súbita da taxa de perda do valor 0 para o valor 100.

2) A alteração súbita da taxa de perda do valor 100 para o valor 0.

Escolheu-se o valor 4, ou seja, a cada 4 ciclos de controle o controlador deveria ser capaz de levar a prioridade do valor 100 para o valor 0 ou vice-versa, sendo este o valor dado para os parâmetros DW e IW das equações 3.2 e 3.3. Começou-se a análise pelo caso 2, o qual será equacionado a seguir utilizando a equação 3.3. Como serão 4 passsos, a variação da prioridade em cada passo será representada por Δp_i com i = 1,2,3 e 4.

$$\Delta p_1 = -K_i(100 + 100 + 100 + 0) + K_d\frac{100 - 0}{4} = -300K_i + 25K_d \tag{3.5}$$

$$\Delta p_2 = -K_i(100 + 100 + 0 + 0) + K_d\frac{100 - 0}{4} = -200K_i + 25K_d \tag{3.6}$$

$$\Delta p_3 = -K_i(100 + 0 + 0 + 0) + K_d\frac{100 - 0}{4} = -100K_i + 25K_d \tag{3.7}$$

$$\Delta p_4 = K_d\frac{100 - 0}{4} = 25K_d \tag{3.8}$$

Com esses 4 valores em mãos, somou-se todos eles de modo que a soma total fosse igual a -100, que seria a variação total de prioridade desejada. Desta maneira,

$$\Delta p = \Delta p_1 + \Delta p_2 + \Delta p_3 + \Delta p_4 = -100 \tag{3.9}$$

o que implica em,

$$\Delta p = -600K_i + 100K_d = -100 \tag{3.10}$$

A partir disso, utilizou-se a tabela do método Ziegler-Nichols mostrada na seção 2.3.3.1 de onde pode-se concluir com simples manipulações a relação $K_i = \frac{0,25}{K_d}$. Inserindo tal relação na fórmula obtida em 3.10, temos

$$\frac{-150}{K_d} + 100K_d = -100 \tag{3.11}$$

daí,

$$100K_d^2 + 100K_d - 150 = 0 \tag{3.12}$$

Resolvendo a Equação 3.12 obteve-se dois possíveis valores para K_d sendo estes aproximadamente 0.3 e -1.3. Como a equação do controlador 3.3 foi obtida por conta da realimentação negativa do sistema, é desejável que os parâmetros K_p, K_d e K_i sejam negativos. Isso se dá com o objetivo de se ter no final o comportamento da Equação 3.2, onde há um valor negativo do termo derivativo caso haja uma diminuição da taxa de perda e um valor positivo do mesmo caso haja um aumento desta. Com isso a prioridade e a taxa de perda tendem a se comportar de maneira diretamente proporcional, o que é o comportamento esperado, dada uma escolha correta dos valores dos parâmetros do controlador. Dito isso, escolheu-se o valor -1.3 como estimativa inicial de K_d. Com esse valor de K_d temos, consequentemente, $K_i = -0.18$.

O processo para o caso 1) é muito similar ao do caso 2), sendo este mostrado a seguir. É importante ressaltar neste caso que dado o fato de que valores obtidos com esta análise serão utilizados apenas como ponto de partida assumiu-se a taxa de perda constante durante todo o processo, seja ele de subida ou descida. Isso se dá com o objetivo de simplificar os cálculos, obtendo assim parâmetros maiores (ou menores) do que os que seriam obtidos caso se considerasse uma taxa de perda variável (que não se mantém exatamente em 100% o tempo todo) como se tem no caso real.

$$\Delta p_1 = -100K_p - K_i(0 + 0 + 0 + 100) = -100K_i - 100K_p \tag{3.13}$$

$$\Delta p_2 = -100K_p - K_i(0 + 0 + 100 + 100) = -100K_i - 200K_p \tag{3.14}$$

$$\Delta p_3 = -100K_p - K_i(0 + 100 + 100 + 100) = -100K_i - 300K_p \tag{3.15}$$

$$\Delta p_4 = -100K_p - K_i(100 + 100 + 100 + 100) = -100K_i - 400K_p \tag{3.16}$$

Com isso, deseja-se que Δp seja igual a 100, pois a prioridade deve aumentar com o aumento da taxa de perda, logo

$$-400K_p - 1000K_i = 100 \tag{3.17}$$

Daí, fazendo $K_i = -0.18$ temos

$$K_p = \frac{80}{400} \implies K_p = 0.2 \tag{3.18}$$

Com os parâmetros determinados passou-se para a parte empírica, onde utilizou-se a BSN com 5 sensores (sensornodes) conectados a uma plataforma central (bodyhub) em que todos esses módulos possuíam uma frequência igual a 25 Hz, visando dar uma grande carga para o sistema fazendo com que este fosse se adaptando de acordo com as perdas monitoradas. Nesta parte foram variados os ganhos de acordo com a teoria mostrada na seção 2.3.3.1 e percebeu-se a necessidade da diminuição do ganho integral, pois este gerava um termo de valor alto ao ter acúmulo do erro e dificultava a resposta do sistema no sentido de diminuir a prioridade.

Com esta diminuição, percebeu-se a necessidade de alterar o sinal do ganho proporcional, o qual estava tendo comportamento diferente do esperado para o sistema, além do fato de que para valores de K_i abaixo de 0.1, tal termo se torna negativo devido à equação 3.17. Após a mudança de sinal de K_p percebeu-se uma certa lentidão na resposta do sistema, ou seja, a prioridade subia em pequenas quantidades, optando-se assim por aumentar o módulo do mesmo.

Por fim, com o aumento do ganho proporcional o sistema passou a apresentar lentidão na resposta no sentido de diminuição da prioridade, o que fez com que se aumentasse o módulo do ganho derivativo. Ao verificar que se havia alcançado o ponto em que ao aumentar o valor do ganho derivativo degradava a resposta do sistema porém a mesma ainda apresentava lentidão, manteve-se este ganho constante e diminuiu-se o valor do parâmetro DW, visando uma resposta mais rápida com relação às variações da taxa de perda. Os parâmetros obtidos ao final do processo são mostrados a seguir, os quais fizeram com que o sistema apresentasse a melhor resposta dentre as observadas.

- $\rightarrow$ K_p = -0.8
- $\rightarrow$ K_i = -0.08
- $\rightarrow$ K_d = -4
- $\rightarrow$ DW = 2
- $\rightarrow$ IW = 4

Com os parâmetros determinados, a equação do controlador 3.3 se torna a equação 3.19

$$\Delta p = 0.8M_r(k) + 0.08\sum_{j=0}^{3} M_r(k-j) + 2(M_r(k) - M_r(k-2)) \tag{3.19}$$

Capítulo 4

Resultados

4.1 Modelagem formal

Na etapa atual do projeto foi realizada a implementação de um modelo da BSN visando a verificação formal de diversas propriedades com relação às características tempo real da mesma. Para tanto foram criadas consultas, no formato da lógica TCTL apresentada na Seção 2.2, buscando mostrar as principais características que o sistema deve possuir. As descrições das propriedades verificadas para um modelo com 2 sensores e um nó central são dadas na Tabela 4.1. Nela é atribuído uma identificação (ID) para cada uma delas na ordem em que aparecem.

Resultado	ID	Descrição	Especificação em TCTL
✓	**P1**	O sistema é livre de *deadlock*.	$A\Box\, not\, deadlock$
✓	**P2**	Garantia de que nenhuma deadline dos *datareceivers* será ultrapassada em nenhum momento da execução do sistema, dados os tempos escolhidos.	$A\Box\, not\, (m\_clks[1] > Low\_Deadline \,\|\|\, m\_clks[2] > Low\_Deadline)$
✓	**P3**	Garantia de que o tempo entre o recebimento de um dado pelo *datareceiver* do sensor 2 e processá-lo no *centralprocessor* nunca será maior que 250 ms	$A\Box\, not\, (observer(2).Detecting\, and\, oberserver(2).o\_clk > 20)$
✓	**P3**	Garantia de que o tempo entre o recebimento de um dado pelo *datareceiver* do sensor 1 e processá-lo no *centralprocessor* nunca será maior que 250 ms	$A\Box\, not\, (observer(1).Detecting\, and\, oberserver(1).o\_clk > 20)$
✓	**P5**	Garantia de que o *centralprocessor* nunca ultrapassará a deadline crítica quando em. em estado crítico.	$A\Box\, not\, (critical\, \&\&\, m\_clks[0] > Critical\_Processor\_Deadline)$

Tabela 4.1: Propriedades TCTL do sistema com escalonador e suas descrições

Os resultados obtidos são mostrados na Figura 4.1, onde se tem um círculo verde para uma propriedade verificada e um círculo vermelho para uma propriedade não verificada.

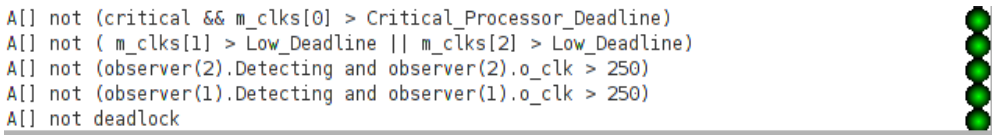

Figura 4.1: Propriedades verificadas e seus resultados

A primeira propriedade diz respeito ao fato de o nó central nunca ultrapassar sua deadline crítica, dada pela constante *Critical_Processor_Deadline*. A segunda propriedade diz respeito ao fato de nenhuma das deadlines dos *datareceivers*, todas sendo iguais a constante *Low_Deadline*, ser ultrapassada em momento algum durante a execução do sistema. Já a terceira e a quarta propriedades dizem respeito ao fato de que o tempo entre receber um dado no datareceiver e processá-lo nunca será maior do que 250 ms, tempo estipulado como máximo tempo de resposta em uma situação de criticidade, sendo notável o fato de que isso se faz válido também para os casos em que o sistema não se encontra em estado crítico. Finalmente, a última propriedade diz respeito ao fato de não haverem *deadlocks* no sistema. Isso quer dizer que não existem estados nos quais nenhuma transição pode ser tomada sem a violação de invariantes ou pela simples impossibilidade de se transitar. Desta maneira, pode-se garantir que o sistema sempre estará alterando seu estado quando necessário.

Os resultados obtidos até o presente momento são positivos e mostram que o modelo implementado possui as características desejadas para o sistema, tendo assim o comportamento que se espera. A verificação formal, por meio de sua fundamentação matemática, garante que tais resultados são obtidos em 100% dos casos, ou seja, não há riscos de haverem estados desconhecidos nos quais as propriedades verficadas não valem, o que garante robustez ao sistema projetado.

4.2 Implementação do escalonador

4.2.1 Métricas

Como a implementação do escalonador se deu via *software* é necessário explicitar quais as configurações da máquina utilizada para a realização dos experimentos com este realizados. Isso se faz necessário uma vez que a variável temporal em sistemas computadorizados não possui uma definição trivial por depender de características do *hardware*, o qual possui limitações intrínsecas. Tais configurações são mostradas na Tabela 4.2.

Recurso	Especificação	Observação
Sistema Operacional	16.04.5 LTS	
Kernel	Linux 4.15.0-45-generic (x86_64)	SMP PREEMPT RT
Compilador	GNU C Compiler 5.3.1	
Processador	4x Intel(R) Core(TM) i7-8550U CPU @1.8GHz	
Memória RAM	16 GB	

Tabela 4.2: Configurações utilizadas para teste

4.2.2 Comparação entre o escalonador EDF e o escalonador original do OpenDaVINCI

O objetivo deste teste foi mostrar que a frequência determinada para o módulo com o escalonador EDF é obedecida, o que não ocorre para o escalonamento original do OpenDaVINCI, anteriormente utilizado por Diniz [16] uma vez que todos os módulos herdam a frequência do SuperComponent. Além disso, devido ao fato do escalonador original não obedecer a frequência determinada também não há nenhum mecanismo que faça ser possível determinar uma deadline para o módulo, dado que esta acaba sendo igual ao período do SuperComponent.

Para tanto, definiu-se o SuperComponent com frequência igual a 2 Hz e utilizando dois nós sensores da BSN, denominados sensornode-0 e sensornode-1, em que estes consumiram aproxidamente 200 ms a cada execução, para melhoria da visualização gráfica. As especificações dos módulos, frequência de operação, deadline e pior tempo computacional (*worst case execution time*) são listadas a seguir.

- sensornode-0

  frequência: 0.5 Hz

  deadline: 1800 ms

  worst case execution time (WCET): 200 ms

- sensornode-1

  frequência: 1 Hz

  deadline: 900 ms

  worst case execution time (WCET): 200 ms

Os gráficos da execução dessa configuração para o escalonador original do OpenDaVINCI e para o escalonador EDF são mostrados na Figura 4.2. A partir dos gráficos obtidos pode-se perceber que, embora o módulo tenha sido definido com frequência igual a 1 Hz o escalonador original do OpenDaVINCI o executou a uma frequência de 2 Hz, assim como a do SuperComponent. Já o escalonador EDF executou o módulo na frequência determinada, obedecendo ainda a deadline o pior tempo computacional também definidos, independetemente da frequência definida no SuperComponent, cuja única restrição é que esta última seja suficientemente maior do que a primeira dado que o SuperComponent é responsável pelo escalonamento do módulo.

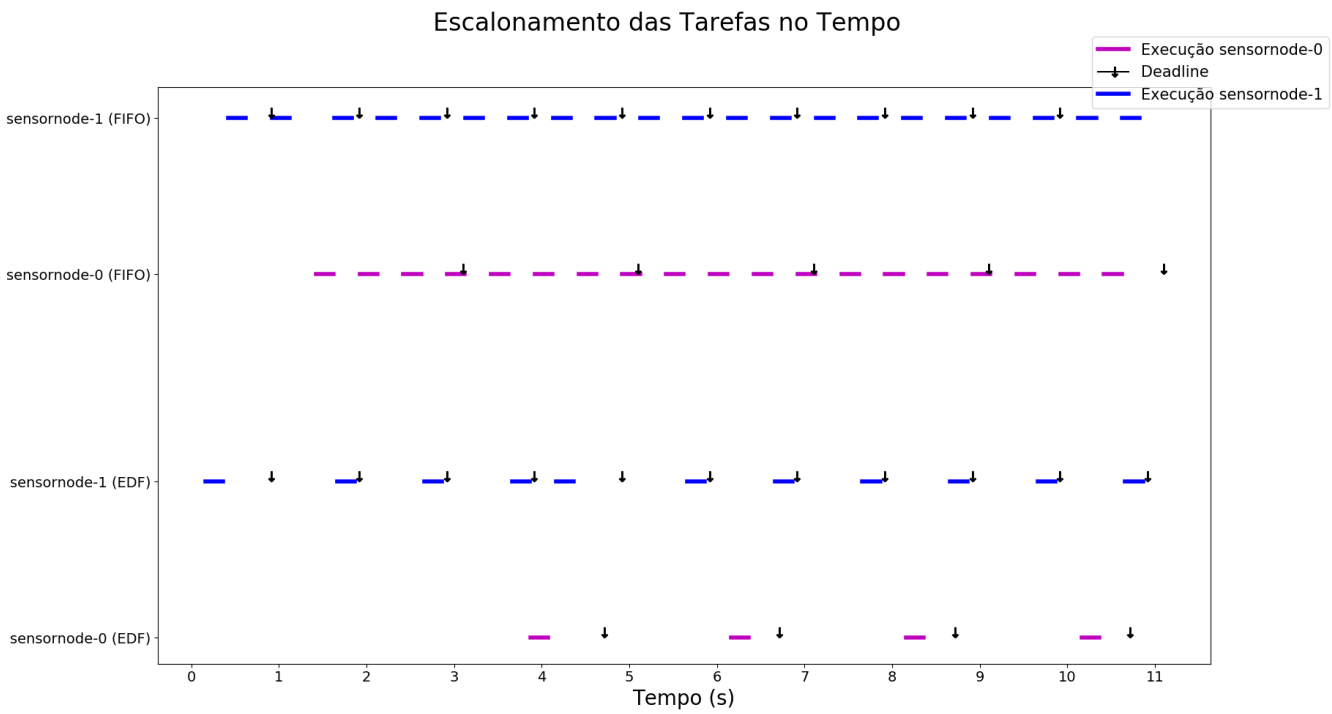

Figura 4.2: Execução do escalonamento para os dois escalonadores

4.2.3 Comparação entre desempenho do sistema controlado e o sistema não controlado utilizando a BSN

Nesta etapa, utilizou-se a BSN para avaliar o comportamento do sistema utilizando o controlador simples e o controlador PID em comparação com o sistema não controlado. Foram utilizados 5 sensores, denominados sensornodes, e 1 plataforma central de processamento, denominada bodyhub, responsável por processar os dados advindos dos sensores. Todos os módulos conectados ao SuperComponent possuíam as seguintes características.

- frequência: 25 Hz
- deadline: 40 ms
- *worst case execution time* (WCET): 1 ms

Vale ressaltar que as frequências utilizadas foram escolhidas para que houvesse garantia de sobrecarga do sistema, dado que para frequências mais baixas (menores ou iguais a 5Hz e em alguns casos até 10Hz) detectou-se que as taxas de perda são nulas ou pequenas o suficiente para serem consideradas irrelevantes. O SuperComponent rodou à uma frequência de 100 Hz, na tentativa de suportar a carga advinda do sistema. Cada sistema foi executado 6 vezes e todos os resultados serão mostrados a seguir.

Os dados são ilustrados graficamente na Figura 4.3. Além disso, a taxa média de perda, o

desvio-padrão médio, o valor mínimo e o valor máximo para cada tipo de controle são mostrados na Tabela 4.3.

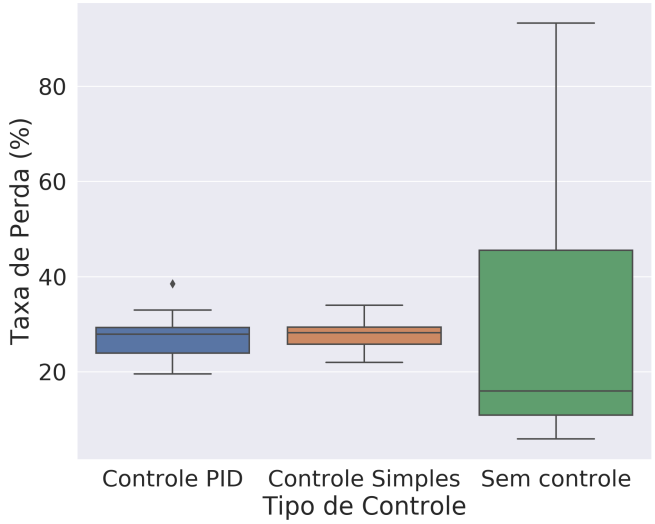

Figura 4.3: Dados obtidos para cada um dos tipos de controle

Tipo de controle	Taxa de perda média	Valor mínimo da taxa de perda média	Valor máximo da taxa de perda média
Sem controle	28.908 ± 23.18	5.893	93.242
Controle simples	27.932 ± 2.51	21.928	32.207
Controle PID	27.057 ± 3.924	19.514	38.516

Tabela 4.3: Dados obtidos para o sistema utilizando cada um dos tipos de controle

Dos resultados mostrados anteriormente podem ser retiradas algumas conclusões a respeito dos resultados que o controle do sistema foi capaz de trazer. Primeiramente, tem-se que o sistema não controlado tem a capacidade de fazer com que alguns módulos possuam taxas de perda muito baixas mesmo em condições de alta carga porém com a contrapartida de haverem módulos com altíssimas taxas de perda, como mostrado na Tabela 4.3 e na Figura 4.3.

O controle da prioridade dá ao sistema uma melhor garantia de justiça entre os módulos, conceito denominado em inglês como fairness, como pode-se perceber pelos baixos valores de desvio-padrão exibidos tanto com o uso do controle simples como com o uso do controle PID, em contrapartida aos altos valores de desvio padrão observados no sistema não controlado. Com relação aos dois controles, tem-se que o controle PID consegue atingir menores valores mínimos de taxa de perda do que o controle simples porém também apresenta alguns valores máximos maiores do que o segundo, o que é comprovado dado seu maior desvio padrão (uma média de quase 1.5% acima).

4.2.4 Comparação de desempenho utilizando tipos de controle mistos

Nesta seção executou-se o sistema com as mesmas configurações da seção passada, havendo 5 sensornodes e 1 bodyhub. A diferença foi a utilização do controle PID ou controle simples em apenas dois módulos: sensornode-0 e sensornode-1. Foram realizadas 10 execuções com cada tipo de controle, sendo os dados ilustrados graficamente na Figura 4.4, onde S-x representa o *sensornode* de número x e B-0 representa a *bodyhub*. Além disso, a taxa média de perda e o desvio-padrão médio para cada um são apresentados na Tabela 4.4.

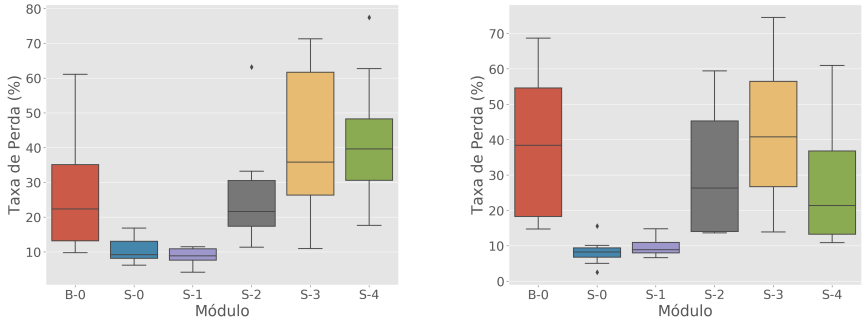

Figura 4.4: (a) Dados obtidos para o sistema com os módulos S-0 e S-1 utilizando controle PID. Todos os outros móodulos não utilizaram nenhum tipo de controle, possuindo prioridade estática igual a zero. (b) Dados obtidos para o sistema com os módulos S-0 e S-1 utilizando controle simples. Todos os outros móodulos não utilizaram nenhum tipo de controle, possuindo prioridade estática igual a zero.

Módulo	Taxa de perda média controle PID	Taxa de perda média controle simples
bodyhub-0	27.112 ± 17.845	38.421 ± 20.258
sensornode-0	10.465 ± 3.699	8.242 ± 3.282
sensornode-1	8.801 ± 2.251	9.945 ± 2.673
sensornode-2	26.133 ± 14.093	30.281 ± 16.819
sensornode-3	41.710 ± 20.504	41.485 ± 18.995
sensornode-4	42.231 ± 16.782	27.355 ± 16.533

Tabela 4.4: Médias da taxa de perda dos módulos do sistema para as duas configurações, com apenas os módulos sensornode-0 e sensornode-1 utilizando algum tipo de controle.

Dos resultados mostrados anteriormente pode-se perceber que os módulos que utilizaram algum tipo de controle obtiveram taxas de perda consideravelmente menores, o que mostra o efeito da variação da prioridade dos mesmos. Além disso, o controlador simples obteve um resultado melhor devido à uma média da taxa de perda bem menor em um dos módulo, quando se trata em porcentagem. Por fim, vale a pena ressaltar que os módulos não controlados foram mantidos na

prioridade mais baixa, sendo esta 0, e todos os módulos começaram com a mesma prioridade.

4.2.5 Demonstração do correto funcionamento da BSN com o escalonador EDF

Este teste teve como objetivo mostrar o funcionamento da BSN, ou seja, que ela está funcionando como deveria. A variação da frequência existente serve para exemplificar casos com frequências distintas e casos com frequências idênticas, como era o caso do funcionamento do OpenDaVINCI anteriormente, dado que os módulos herdavam a frequência de execução do SuperComponent não havendo a possibilidade de ter dois ou mais módulos de frequências diferentes conectados a um mesmo SuperComponent.

Foram realizados dois testes com a BSN utilizando em ambos 3 sensores conectados a uma plataforma central (denominada bodyhub). No primeiro teste a bodyhub foi executada com frequência igual a 20 Hz, o termômetro com frequência igual a 10 Hz, o sensor de frequência cardíaca com frequência igual a 5 Hz e o oxímetro com frequência igual a 1 Hz. No segundo teste todos os módulos foram executados com a mesma frequência, sendo esta igual a 10 Hz. Para ambos os testes gerou-se um gráfico com a evolução do estado do paciente, entre criticidade baixa, moderada e alta, destacando em vermelho os momentos em que uma emergência foi detectada e desenhando uma linha tracejada vertical preta nos momentos em que houve mudança de estado. Tais gráficos são mostrados nas figuras 4.5, para o primeiro teste, e 4.6, para o segundo.

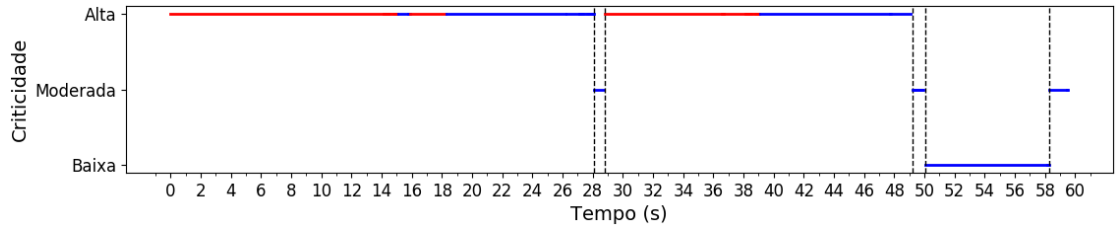

Figura 4.5: Estado do paciente ao longo do tempo para o teste com frequências mistas. No caso foram utilizadas as frequências iguais a 20 Hz, 10 Hz e 5 Hz.

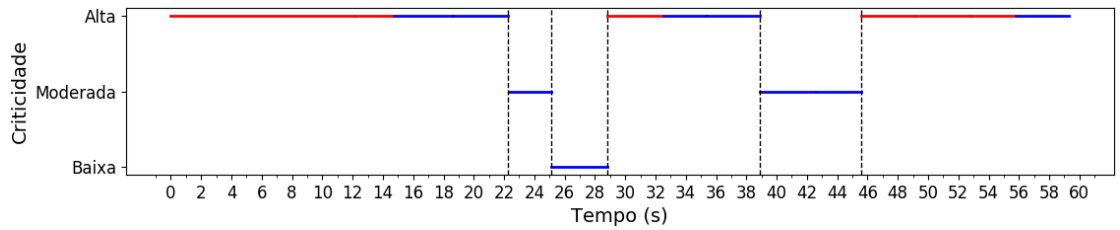

Figura 4.6: Estado do paciente ao longo do tempo para o teste com frequências idênticas. No caso foram utilizadas todas as frequências iguais a 10 Hz.

Como pode-se perceber o estado do paciente é de fato alterado de acordo com a métrica implementada na bodyhub. Isso quer dizer que se o último sensor lido estiver em valor alto, uma emergência será detectada, e caso o último sensor lido estiver em valor moderado ou baixo, a

mesma não o será. Por conta deste comportamento só se vê traços vermelhos quando o estado do paciente é de alta criticidade, uma vez que só se atinge este estado caso um dos sensores possua valor alto. Ademais, não se vê apenas traços vermelhos no estado de alta criticidade uma vez que a detecção da emergência é realizada baseando-se apenas no último sensor lido e não no estado geral do paciente.

Capítulo 5

Conclusões e Trabalhos Futuros

A verificação formal é um mecanismo que traz consigo diversos benefícios para um projeto eficiente dado que, além de haver uma diminuição do tempo dedicado para testes do sistema, há uma garantia matematicamente fundamentada de funcionamento do sistema em todos os casos, dada por uma busca exaustiva. O mesmo não ocorre quando se opta pela realização de testes do sistema pois torna-se muito dispendiosa a verificação de todos os possíveis casos, além de ser quase impossível a definição prévia dos mesmos.

O presente trabalho foi bem sucedido na criação e verificação formal de um modelo para a *Body Sensor Network* com escalonador que segue a política de escalonamento *Earliest Deadline First*, visando garantir as propriedades tempo real do sistema de maneira eficiente. Além disso, a implementação do escalonador em software também se mostrou bem sucedida. As mudanças aqui realizadas com relação ao que foi feito por Farias et al [15] foram:

- Mudanças estruturais no modelo, procurando alinhar o mesmo à uma visão de sistema distribuído.
- A introdução de um escalonador com algoritmo dinâmico e online para garantias de tempo real.
- A adição de uma política de controle ao escalonador, visando garantir um escalonamento justo.

Os resultados obtidos até o momento se mostraram positivos, como mostrado no Capítulo 4, abrindo espaço para novas abordagens com relação às características de auto-adaptatividade. As ameaças à validade são a utilização de valores aleatoriamente escolhidos para as frequências de operação e estimativas pouco precisas de tempos de execução no modelo formal. Em trabalhos futuros, vislumbramos os seguintes itens para dar continuidade ao que foi realizado neste trabalho:

- A implementação do escalonador seguindo políticas de controle diferentes da atualmente implementada.

- A otimização dos parâmetros do controlador PID, a partir da utilização de técnicas não heurísticas ou de mais testes empíricos.
- A aproximação dos valores utilizados para parâmetros de dados concretos advindos de possíveis bases de dados existentes.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] BUTTAZZO, G. C. *Hard real-time computing systems: predictable scheduling algorithms and applications*. [S.l.]: Springer Science & Business Media, 2011.

[2] COTTET, F. et al. *Scheduling in real-time systems*. [S.l.: s.n.], 2002.

[3] ABDELZAHER, T. et al. Introduction to control theory and its application to computing systems. In: *Performance Modeling and Engineering*. [S.l.]: Springer, 2008. p. 185–215.

[4] STANKOVIC, J. A. et al. The case for feedback control real-time scheduling. In: IEEE. *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*. [S.l.], 1999. p. 11–20.

[5] CHEN, M. et al. Body area networks: A survey. *Mobile networks and applications*, Springer, v. 16, n. 2, p. 171–193, 2011.

[6] LATRÉ, B. et al. A survey on wireless body area networks. *Wireless Networks*, Springer, v. 17, n. 1, p. 1–18, 2011.

[7] COPELAND, B. R. The design of pid controllers using ziegler nichols tuning. *Zieglar-Nicholos Method*, 2008.

[8] KOPETZ, H. *Real-time systems: design principles for distributed embedded applications*. [S.l.]: Springer Science & Business Media, 2011.

[9] KOPETZ, H.; OCHSENREITER, W. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36, n. 8, p. 933–940, Aug 1987. ISSN 0018-9340.

[10] PESSOA, L. et al. Building reliable and maintainable dynamic software product lines: An investigation in the body sensor network domain. *Information and Software Technology*, Elsevier, v. 86, p. 54–70, 2017.

[11] RODRIGUES, G. N. et al. Modeling and verification for probabilistic properties in software product lines. In: *HASE*. [S.l.]: IEEE Computer Society, 2015. p. 173–180.

[12] AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, v. 1, n. 1, p. 11–33, Jan 2004. ISSN 1545-5971.

[13] WEYNS, D. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering*, Springer, 2017.

[14] 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. ISSN 0924-4247. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0924424710002657>.

[15] FARIAS, A. J. R. et al. A learning approach to support the assurance for real-time self-adaptive systems. 2018.

[16] CALDAS, R. D. *Prototipação e Verificação Formal de Sistema Autônomo com Propriedades Tempo-Real: Um estudo de caso no Body Sensor Network*. 59 p. Monografia (Graduação) — Universidade de Brasília, Brasília, 2017.

[17] NISSANKE, N. *Realtime systems*. [S.l.]: Prentice-Hall, Inc., 1997.

[18] BEHRMANN, G.; DAVID, A.; LARSEN, K. G. *A tutorial on uppaal 4.0. vol.* 2006.

[19] BAIER, C.; KATOEN, J.-P. *Principles of model checking*. [S.l.]: MIT press, 2008.

[20] ALUR, R.; DILL, D. The theory of timed automata. In: SPRINGER. *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*. [S.l.], 1991. p. 45–73.

[21] BOUYER, P.; LAROUSSINIE, F. Model checking timed automata. *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, Wiley Online Library, p. 111–140, 2010.

[22] ALUR, R. *Techniques for automatic verification of real-time systems*. Tese (Doutorado) — stanford university, 1991.

[23] CLARKE, E. M.; EMERSON, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. In: SPRINGER. *Workshop on Logic of Programs*. [S.l.], 1981. p. 52–71.

[24] COMINOS, P.; MUNRO, N. Pid controllers: recent tuning methods and design to specification. *IEE Proceedings-Control Theory and Applications*, IET, v. 149, n. 1, p. 46–53, 2002.

[25] ÅSTRÖM, K. J.; HÄGGLUND, T. Revisiting the ziegler–nichols step response method for pid control. *Journal of process control*, Elsevier, v. 14, n. 6, p. 635–650, 2004.

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

ANEXOS

I. ARQUITETURA E COMUNICAÇÃO DA BSN

I.1 BSN versus WSN tradicional

A BSN é um tipo de rede de sensores sem fio, ou Wireless Sensor Network (WSN) em inglês, existente atualmente, porém algumas características tornam a BSN substancialmente diferente da WSN tradicional. *Chen et al.* aponta tais diferenças em [5], sendo elas listadas a seguir:

- BSNs não empregam nós redundantes para lidar com os mais diversos tipos de falhas, algo que é comum em WSNs. Logo, as BSNs não possuem uma densidade de nós grande, dada a facilidade do acesso para troca.
- A maioria das WSNs são empregadas para monitoramento baseado em eventos. Já as BSNs são empregadas para registrar dados fisiológicos, o que ocorre de maneira mais periódica. Isso faz com que a taxa de aquisição de dados das BSNs seja mais estável em relação às WSNs.
- A latência nas WSNs costuma ser maior em troca da vida da bateria, dada a dificuldade de reposição desta última. Nas BSNs ela não necessita ser tão alta, uma vez que a reposição é feita muito mais facilmente.
- Os nós de uma BSN compartilham de um padrão de mobilidade, uma vez que seus usuários podem se mover. Em contrapartida, os nós de uma WSN são considerados estáticos.

Essas e outras diferenças fazem com que técnicas e algoritmos utilizados em WSNs tradicionais não se adequem tão bem no contexto das BSNs, o que mostra a necessidade de um estudo específico na área das redes de sensores corporais, visando o desenvolvimento de metodologias de projeto para tais sistemas.

I.2 Hardware

O hardware de uma BSN é composto basicamente por seus nós sensores e, a depender do caso, um nó central de processamento. Esses nós devem ter características específicas, sendo as principais o tamanho reduzido, a capacidade de se comunicar através de uma rede sem fio, o consumo de energia muito baixo e a implantação superficial, visto que não devem ser invasivos. *Chen et al.* mostra em [5] a arquitetura básica de um nó sensor com os módulos típicos que esse possui, sendo esta mostrada na Figura I.1. Os módulos são o módulo RF, composto de emissores e receptores sem fio responsáveis pela comunicação, o módulo sensor, composto por sensores do mesmo tipo, filtros e conversores A/D responsáveis pela aquisição e pré-processamento de dados, o módulo de memória, composto de memórias embutidas responsáveis pelo armazenamento de dados e o módulo

microprocessador, composto de um circuito microprocessado responsável pelo processamento dos dados adquiridos, realizando o comando de envio e armazenamento na memória após o término do processamento.

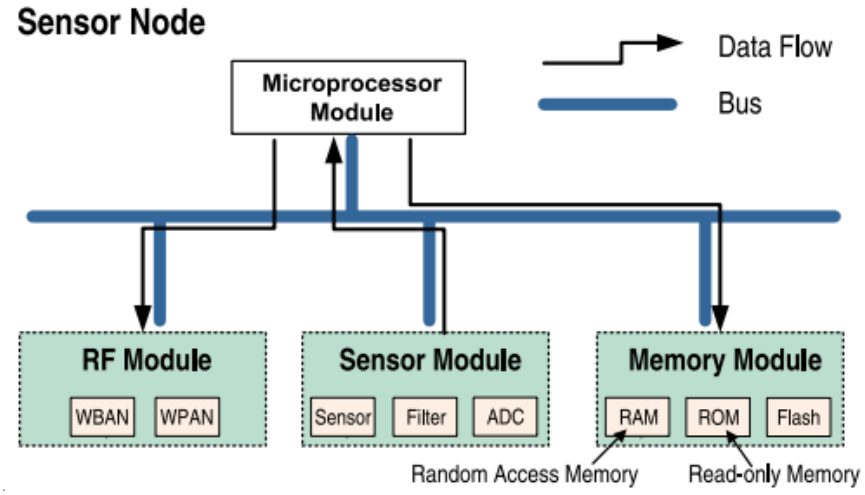

Figura I.1: Arquitetura básica de um nó sensor [5]

No mesmo trabalho os autores citam os tipos mais comuns de sensores utilizados em BSNs e suas respectivas taxas de aquisição de dados, o que é mostrado na Tabela I.1.

Sensor	Topologia	Taxa de amostragem
Acelerômetro/giroscópio	Estrela	Alta
Glicose sanguínea	Estrela	Alta
Pressão sanguínea	Estrela	Baixa
Sensor de gás $C0_2$	Estrela	Muito baixa
Eletrocardiógrafo (ECG)	Estrela	Alta
Eletroencefalógrafo (EEG)	Estrela	Alta
Eletromiógrafo (EMG)	Estrela	Muito alta
Oxímetro de pulso	Estrela	Baixa
Umidade	Estrela	Muito baixa
Temperatura	Estrela	Muito baixa
Sensor de imagem/vídeo	P2P	Muito alta

Tabela I.1: Tipos comuns de sensores utilizados em BSNs e suas taxas de aquisição [5]

I.3 Comunicação

A arquitetura de comunicação da BSN é dividida nas comunicação denominadas intra-corporais, que são comunicaçãoes entre os nós que compõem a BSN e também a comunicação dos nós com o aparelho responsável pelo armazenamento de dados, que pode ser um celular, computador ou dispositivo semelhante, e as comunicações extra-corporais, que são comunicações via redes com servidores médicos distantes, computadores de uma central de emergência ou de um médico responsável ou dispositivos relacionados que estejam fora do alcance limitado dos sensores. Em [5] a nomenclatura utilizada é intra-BAN para a camada de comunicação intra-corporal e inter-BAN para a camada de comunicação extra-corporal. Um exemplo do arranjo dessa arquitetura de comunicações é mostrado na Figura I.2

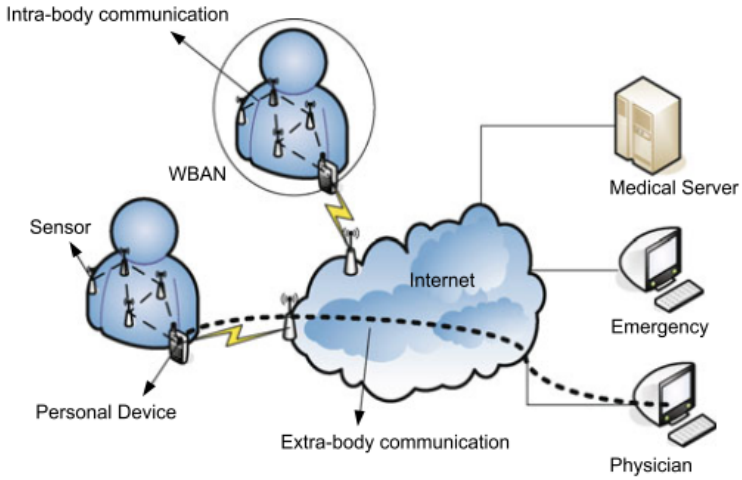

Figura I.2: Arquitetura de comunicação da BSN [6]

A intra-BAN geralmente utiliza o protocolo IEE 802.15.4 (ZigBee) ou o protocolo Bluetooth, que são protocolos de alcance limitado porém de muito baixo consumo, e a inter-BAN geralmente utiliza o padrão IEE 802.11 (WiFi), que é um protocolo de maior alcance e consequentemente de maior consumo de energia. A escolha do protocolo de maior consumo para a inter-BAN é feita uma vez que a comunicação nesta se dá por um dispositivo cujas restrições de consumo não são limitadas como as dos nós sensores, além da necessidade de comunicação da BSN com o meio externo que se dá a partir da comunicação com pontos de acesso (roteadores) que podem estar e provavelmente estarão a uma distância razoável do paciente.