Wilson Prates de Oliveira

Arquitetura de software para sistemas de tempo real

São Paulo

2011

Wilson Prates de Oliveira

Arquitetura de software para sistemas de tempo real

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia.

Área de Concentração:
Engenharia de Controle e Automação Mecânica.

Orientador:
Prof. Dr. Newton Maruyama

São Paulo

2011

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 12 de dezembro de 2011.

Assinatura do autor ____________________________________

Assinatura do orientador_________________________________

FICHA CATALOGRÁFICA

Oliveira, Wilson Prates de
Arquitetura de software para sistemas de tempo real / W.P. de Oliveira. -- ed.rev. -- São Paulo, 2011.
110 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Sistemas de tempo-real 2.Arquitetura de software I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

DEDICATÓRIA

A minha família, especialmente a minha querida irmã Elza Prates de Oliveira que foi minha principal incentivadora para deixar o calor da pequena cidade de Januária-MG em busca de um sonho que se realizou e em memória de José Geraldo Prates de Oliveira com quem tive o prazer de conviver durante a minha infância e graduação na USP-São Carlos.

"Que sentido teria a vida se não existisse os obstáculos para serem vencidos" (José Geraldo Prates de Oliveira).

AGRADECIMENTOS

Ficam meus sinceros agradecimentos a todos que de alguma maneira contribuíram para que fosse possível chegar até o presente momento.

As pessoas com quem tive o prazer de conviver durante o desenvolvimento desta pesquisa no laboratório de sistemas embarcados do Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos da Escola Politécnica da USP. Especialmente Roberto e Douglas pelos constantes esclarecimentos das dúvidas sobre controle. Ao Marcos pela colaboração durante a montagem da plataforma para RCP.

Ao meu orientador Prof. Dr. Newton Maruyama pelas críticas que inicialmente me pareciam sem sentido, mas durante a confecção do trabalho foi possível constatar que tinham fundamentos e pelo direcionamento da pesquisa.

Ao Prof. Dr. Diolino José dos Santos Filho por mostrar a necessidade da aplicação de métodos e metodologias no desenvolvimento de trabalhos e aos membros da banca de qualificação Prof. Dr. Paulo Eigi Miyagi e Prof. Dr. Oswaldo Horikawa pelas críticas construtivas.

RESUMO

O desenvolvimento de sistemas de controle em plataformas de tempo real é uma tarefa que envolve Engenharia de Controle e Ciência da Computação. Nas últimas décadas, estas áreas se desenvolveram como áreas independentes. Este trabalho busca diminuir distância entre as áreas propondo a utilização de métodos de Engenharia de *Software* em uma fase de modelagem do *software* de controle.

Uma das propostas apresentadas no trabalho é a utilização de *frameworks* orientados a objetos no processo de *Rapid Control Prototyping* (RCP) para substituir a geração automática de código, eliminar os problemas de integração com código legado e tornar o processo RCP mais interativo. Outra proposta é a utilização de plataformas para RCP composta por uma camada de *hardware* real, uma camada SOTR e uma camada de aplicação formada pelo *framework* para análise e desenvolvimento de sistemas de controle centralizado ou distribuído.

Palavras chave: RCP, *frameworks*, orientação a objetos, tarefas, plataformas de tempo real, *hardware, software, host* e *target*.

ABSTRACT

The development of real-time platform control systems is a task that involves Control Engineering and Computer Science. In the last decade, these areas have developed independent from each other. This paper seeks to decrease the distance between this areas, by proposing the use of Software Engineering methods in a software control modeling phase.

One of the propositions in this paper is the use of object orientated frameworks in the Rapid Control Prototyping (RCP) process to substitute the automatic code generation, thus eliminating the problems with the legacy code and making the RCP process more interactive. Another proposition is the use of RCP directed platforms composed by a real hardware layer, a RTOS layer and an application layer formed by the framework for the analysis and development of the centralized or distributed control systems.

Keywords: RCP, frameworks, tasks, object orientation, real-time platforms, hardware, software, host and target.

LISTA DE FIGURAS

Figura 1.1: Processo de desenvolvimento de um sistema de controle por computador. 2

Figura 1.2: Sistema de controle computadorizado (TAKARABE, 2009). 5

Figura 1.3: Sistema de controle distribuído em rede (TAKARABE, 2009). 6

Figura 1.4: Diagrama de blocos de uma plataforma para RCP. 8

Figura 1.5: Processo de desenvolvimento de sistemas de controle aplicando RCP. 9

Figura 2.1: Principais partes da malha de controle. 23

Figura 2.2: Grandezas que caracterizam o comportamento temporal das tarefas de controle. 26

Figura 2.3: *Jitter* em sucessivas ativações de uma tarefa de controle periódica. 29

Figura 2.4: Execução do algoritmo FP para um conjunto de tarefas. 31

Figura 2.5: Conjuntos de tarefas periódicas escalonadas pelo algoritmo RR. 31

Figura 2.6: Escalonamento de sub-tarefas restritas a prioridades. 36

Figura 2.7: Escalonamento de sub-tarefas usando deslocamento. 36

Figura 3.1: Representação de um SCC com controlador discreto e planta contínua. 46

Figura 3.2: Representação de um SCC com planta e controlador no domínio discreto. 47

Figura 3.3: Arquitetura hierárquica para FCC. 50

Figura 3.4: Diagrama de módulos e suas dependências na arquitetura do FCC. 50

Figura 3.5: Diagrama de classes do módulo Controladores. 51

Figura 3.6: Diagrama de classes do módulo Plantas. 52

Figura 3.7: Diagrama de classes do módulo Sensores/Atuadores. 53

Figura 3.8: Diagrama de classes do módulo aplicação. 54

Figura 3.9: Diagramas de classes do módulo Útil. 55

Figura 3.10: Representação de um SCD com planta contínua e controlador discreto............56
Figura 3.11: Representação de um SCD com planta e controlador no domínio discreto. 57
Figura 3.12: Modelo de arquitetura hierárquica para FCD..58
Figura 3.13: Diagrama de módulos e suas dependências na arquitetura do FCD. 59
Figura 3.14: Diagrama de componentes do módulo Comunicação. 60
Figura 3.15: Diagramas de classes do módulo TCP. 60
Figura 3.16: Diagramas de classes do módulo UDP. 61
Figura 3.17: Diagrama de classes do módulo CAN..61
Figura 3.18: Diagrama de classes do módulo Controladores distribuído. 62
Figura 3.19: Diagrama de classe do módulo aplicação distribuída no Nó Controlador. 62
Figura 3.20: Diagrama de classes do módulo aplicação distribuída no Nó Planta/Sensor/Atuador...63
Figura 3.21: Diagrama de classes do módulo atuadores para controle distribuído. 64
Figura 3.22: Diagrama de classes do módulo sensores para controle distribuído. 65
Figura 4.1: Arquitetura de plataforma para RCP. 66
Figura 4.2: Processo de desenvolvimento do sistema de controle aplicando RCP modificado. 67
Figura 4.3: Diagrama de blocos do sistema em malha fechada com controlador PID digital e planta contínua. 69
Figura 4.4: Diagrama de blocos do sistema em malha fechada com controlador e planta no domínio de Z. 69
Figura 4.5: Diagrama de classes da planta de segunda ordem..72
Figura 4.6: Resposta da planta sem controle a entrada em degrau unitário..........................72
Figura 4.7: Diagrama de classes do controlador PID. 74

Figura 4.8: Resposta da planta ao sinal de saída do controlador PID centralizado para a entrada em degrau unitário. 75
Figura 4.9: Plataforma RCP para sistemas de controle distribuído em rede. 77
Figura 4.10: Resposta da planta ao sinal de saída do controlador PID distribuído para uma entrada em degrau unitário. 78
Figura 4.11: Plataforma RCP para sistema centralizado com plantas reais conectadas às interfaces de entrada e saída. 80
Figura 4.12: Respostas das plantas 1, 2 e 3 a uma entrada em degrau de 4,2 volts. 81
Figura 4.13: Diagrama de classes da planta de primeira ordem. 83
Figura 4.14: Respostas dos modelos das plantas a uma entrada em degrau de 4,2 volts: (a) Planta1; (b) Planta 2; (c) Planta 3. 84
Figura 4.15: Resposta do modelo de Planta 1 e saída do protótipo do controlador C_1. 86
Figura 4.16: Resposta do modelo de Planta 2 e saída do protótipo do controlador C_2. 86
Figura 4.17: Resposta do modelo de Planta 3 e saída do protótipo do controlador C_3. 87
Figura 4.18: Resposta da Planta real 1 e saída do controlador C_1. 88
Figura 4.19: Resposta da Planta real 2 e saída do controlador C_2. 88
Figura 4.20: Resposta da Planta real 3 e saída do controlador C_3. 88
Figura 5.1: Desempenho ITSE das instâncias do conjunto de tarefas Γ_1 para os algoritmos FP e RR: (a) f_{clk} = 1kHz e *quantum* = 1 ms; (b) f_{clk} = 5kHz e *quantum* = 0,2 ms. 103
Figura 5.2: Desempenho ITSE das instâncias do conjunto de tarefas Γ_2 para os algoritmos FP e RR: (a) f_{clk} = 1kHz e *quantum* = 1 ms; (b) f_{clk} = 5kHz e *quantum* = 0,2 ms. 104

Figura 5.3: Desempenho ITSE das instâncias do conjunto de tarefas Γ_3 para os algoritmos FP e RR: (a) f_{clk} = 1kHz e *quantum* = 1 ms; (b) f_{clk} = 5kHz e *quantum* = 0,2 ms. 105

LISTA DE TABELAS

Tabela 4.1: Constantes das plantas reais. 82

Tabela 4.2: Modelos matemáticos das plantas. 83

Tabela 4.3: Constantes dos modelos de plantas. 84

Tabela 4.4: Parâmetros dos controladores PI. 85

Tabela 5.1: Valores do índice ITSE durante a análise do método de compensação do J_h. 92

Tabela 5.2: Valores dos WCET das implementações das tarefas controlador e planta. 96

Tabela 5.3: Valores dos atributos dos conjuntos de tarefas Γ_{Planta} e $\Gamma_{convencional}$. 98

Tabela 5.4: Valores dos atributos para os conjuntos de tarefas $\Gamma_{subtask}$. 98

Tabela 5.5: Desempenho dos controladores para as implementações convencional e *subtask*. 100

LISTA DE ABREVIATURAS E SIGLAS

RCP	*Rapid Control Prototyping*
NCS	*Networked Control Systems*
FP	*Fixed Priority*
RR	*Round Robin*
RM	*Rate Monotonic*
WCET	*Worst Case Execution Time*
CPU	*Central Processing Unit*
EDF	*Earliest Deadline First*
SOTR	Sistema Operacional de Tempo Real
A/D	Analógico-Digital
D/A	Digital-Analógico
RTSD	*Real-Time Structured Analysis and Design*
WCTA	*Worst Case Timing Abstraction*
LAN	*Local Area Network*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
CAN	*Controller Area Network*
FCC	*Framework* para Controle Centralizado
SCC	Sistema de Controle Centralizado
TS	Tempo de Simulação
FCD	*Framework* para Controle Distribuído
SCD	Sistema de Controle Distribuído

API	*Application Programming Interface*
RT-UML	*Real Time Unified Modeling Language*
IP	*Internet Protocol*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
BSD	*Berkeley Software Distribution*
PID	Controlador Proporcional Integral Derivativo
ZOH	*Zero-Order-Hold*
CC	Corrente Contínua
RP	Regime Permanente
IDE	*Integrated Development Environment*
RTP	*Real-Time Process*
FIFO	*First In First Out*
ITSE	*Integral of Time multiplied by the Squared Error*

LISTA DE SÍMBOLOS

Símbolo	Descrição
τ_i	Tarefa periódica
C_i	Tempo de computação da tarefa periódica
Ti	Período de ativação da tarefa periódica
Di	*Deadline* da tarefa periódica
P_i	Prioridade da tarefa periódica
J_i	*Jitter* de liberação da tarefa periódica
L_{io}	Latência de entrada e saída
L_s	Latência de amostragem
U	Utilização do processador
T_r	Tempo de subida
N_r	Número de amostragem por tempo de subida
h	Intervalo de amostragem nominal
K	k-ésima ativação da tarefa
h_k	Intervalo de amostragem entre as ativações k e (k+1)
J_s	*Jitter* de amostragem
J_h	*Jitter* no intervalo de amostragem h_k
J_{io}	*Jitter* de entrada-saída
C_a	Tempo de execução da ativação atual
Γ	Conjunto de tarefas
P_Γ	Prioridades do conjunto de tarefas Γ

P_{US}	Prioridade da sub-tarefa *Update State*
P_{CO}	Prioridades da sub-tarefa *Calculate Output*
τ_{CO}	Sub-tarefa *Calculate Output*
τ_{US}	Sub-tarefa *Upadate State*
C_{CO}	Tempo de execução de pior caso para *Calculate Output*
C_{US}	Tempo de execução de pior caso para *Update State*
D_{CO}	*Deadline* de *Calculate Output*
D_{US}	*Deadline* de *Update State*
Φ	Deslocamento fixo
$\mathbf{K_p}$	Ganho proporcional do controlador PID
$\mathbf{K_i}$	Ganho integral do controlador PID
$\mathbf{K_d}$	Ganho derivativo do controlador PID
$\boldsymbol{\tau}$	Constante de tempo do sistema de primeira ordem[1]
$\mathbf{K}$	Ganho do regime permanente[2]
f_{clk}	Freqüência de *clock* do sistema de tempo real
T_{min}	Período de ativação mínimo teórico tarefas periódicas
$\mathbf{C_i}$	Controlador i[3]
P_i	Planta i
$\tau_{\mathbf{Ci}}$	Tarefa controlador i
τ_{Pi}	Tarefa planta i

[1] Negrito para diferenciar da tarefa τ.
[2] Negrito para diferenciar da k-ésima ativação da tarefa.
[3] Negrito para diferenciar do tempo de computação da tarefa.

Sumário

LISTA DE FIGURAS i

LISTA DE ABREVIATURAS E SIGLAS vi

LISTA DE SÍMBOLOS viii

1 Introdução 1

1.1 Contextualização 4

1.2 Rapid Control Prototyping (RCP) 8

1.3 Contribuições e Objetivos 10

1.4 Apresentação do trabalho 11

2 Sistemas de controle e tempo real 13

2.1 Modelo de tarefa de tempo real 16

2.1.1 Restrições temporais 16

2.1.2 Relações de Precedência e exclusão 18

2.2 Escalonamento em tempo real 18

2.2.1 *Rate Monotonic* (RM) 20

2.2.2 *Earliest Deadline First* (EDF) 22

2.3 Implementação de malhas de controle 23

2.3.1 Restrições temporais na malha de controle 25

2.3.2 Compensação do *jitter* e atraso 27

2.3.3 Escalonamento de tarefas de controle 30

2.4 Análises do WCET 37

2.5 Redes de computadores 40

2.6 Trabalhos relacionados 42

3 Arquiteturas de *Frameworks* para RCP 44

3.1 Modelagem do *Framework* para Controle Centralizado (FCC) 46

3.1.1 Arquitetura do FCC 48

3.1.2 Módulos do FCC 51

3.2 Modelagem do *Framework* para Controle Distribuído (FCD) 56

3.2.1 Arquitetura do FCD 58

3.2.2 Módulos do FCD 59

4 Arquitetura de plataforma para RCP 66

4.1 Análise de um sistema de controle utilizando as arquiteturas de *frameworks* e plataformas para RCP 68

4.1.1 Sistema de Controle Centralizado 70

4.1.2 Sistema de Controle distribuído (protocolo TCP/IP) 76

4.1.3 Comparação entre o SCC e o SCD 79

4.2 Plataforma RCP para sistema centralizado com plantas reais 80

4.3 Projeto do protótipo do sistema de controle 85

4.3.1 Simulação do protótipo com modelos de plantas 86

4.3.2 Simulação do sistema de controle com plantas reais 87

5 Resultados e análises 90

5.1 Método de compensação do *jitter* no intervalo de amostragem 90

5.2 Cálculos do WCET das tarefas controlador e planta 92

5.3 Análises de *subtask scheduling* e dos escalonadores FP e RR 97

5.3.1 Análise da metodologia *subtask scheduling* 99

5.3.2 Análise dos escalonadores FP e RR 101

6 Conclusões 108

6.1 Sugestões para trabalhos futuros 109

Referências Bibliográficas 111

Apêndice A - Código Ada 2005 das classes que compõem as arquiteturas dos *frameworks*. 116

1 Introdução

Sistemas de tempo real é uma área de pesquisa interdisciplinar que inclui aspectos da Engenharia de Controle e Ciência da Computação (ÅRZÉN, K. ET AL., 1999). Esta é uma área de vital importância para todos os engenheiros de controle. Atualmente praticamente todos os controladores são implementados na forma digital em computadores.

O desenvolvimento bem sucedido de sistemas de controle em uma plataforma de tempo real requer *co-design* do sistema computacional e do sistema de controle (CERVIN, 2003). A plataforma computacional deve ser dimensionada de tal maneira que, todos os requisitos do sistema de controle possam ser atendidos. Os controladores devem ser projetados levando em consideração as limitações da plataforma computacional.

O sucesso da aplicação de abordagens como *co-design* visando integrar as áreas de controle e tempo real, depende exclusivamente da capacidade dos profissionais envolvidos no processo de desenvolvimento de um sistema de controle por computador (engenheiros de controle e engenheiros de *software*) ampliarem seus conhecimentos nas áreas de controle e computação. Pelo lado da Engenharia de Controle, é necessário que os engenheiros entendam melhor as plataformas computacionais utilizadas para controle em tempo real nos níveis de arquitetura do *hardware*, *software* de tempo real, bem como, dos métodos de Engenharia de *Software* que possam ser aplicados durante o projeto e desenvolvimento dos sistemas de controle. Por outro lado, a comunidade de computação deve compreender os modelos matemáticos, bem como, as especificações que determinam os requisitos de desempenho dos sistemas de controle.

Um processo tradicional de desenvolvimento de um sistema de controle por computador é ilustrado na Figura 1.1.

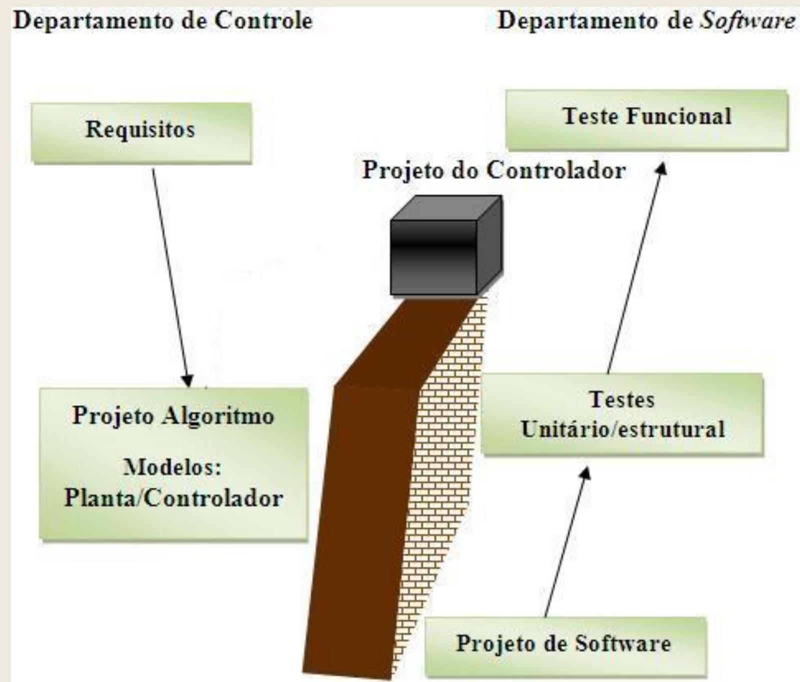


Figura 1.1: Processo de desenvolvimento de um sistema de controle por computador.

No processo ilustrado na figura, a distância entre as áreas de controle e computação é evidenciada pela existência de uma barreira. O processo de desenvolvimento é composto por dois estágios bem distintos. No primeiro estágio, o projeto do sistema de controle é elaborado pelo departamento de controle. O segundo estágio do projeto, especificamente projeto e testes do *software* de controle, é executado pelo departamento de *software*. O departamento de *software* recebe a especificação dos módulos, interfaces, funções que devem ser executadas e das saídas que o sistema deve fornecer. Com base nestas especificações o departamento de *software* implementa o algoritmo de controle e realiza os testes estruturais e funcionais.

Um dos principais problemas deste processo de desenvolvimento está relacionado à análise da plataforma computacional. Por exemplo, como os atrasos e *jitters* na plataforma computacional podem afetar o desempenho do sistema de controle, qual a melhor política para alocar recursos computacionais para as tarefas de controle ou qual a melhor maneira de implementar a malha de controle.

Atualmente, com a utilização da metodologia *Rapid Control Prototyping* (RCP) em conjunto com ferramentas capazes de gerar códigos automáticos, o segundo estágio do processo foi totalmente modificado. O projeto e a implementação do *software* podem ser realizados de maneira automática utilizando geradores de código. Devido a sua simplicidade de utilização e por não exigir do engenheiro de controle nenhum conhecimento sobre processos de desenvolvimento de *software* ou linguagens utilizadas na implementação, estes geradores têm sido largamente utilizados na automatização do processo de implementação (LEE,W. ; SHIN, M. & SUNWOO, M., 2004).

Porém, o código gerado apresenta alguns problemas e nem sempre consegue atender aos requisitos de desempenho determinados para as aplicações de controle. Isto ocorre porque as principais causas da degradação do desempenho em malhas de controle em tempo real estão relacionadas aos atrasos e *jitters,* que podem ser decorrentes da forma como a malha de controle é implementada e da capacidade da plataforma computacional fornecer um ambiente determinístico.

De acordo com Marti e Fuertes (2001), o desempenho do sistema de controle é degradado pelos atrasos e *jitters*. Sendo que, a principal causa dos atrasos e *jitters* em plataformas de tempo real estão relacionadas à capacidade da plataforma escalonar e alocar recursos computacionais para as tarefas de controle. Marti e Fuertes (2001) propõem que as restrições de tempo na malha de controle possam variar, de tal forma que, os parâmetros dos controladores possam ser alterados durante a execução.

Para Cervin (2003), o problema de encontrar uma plataforma computacional de custo mínimo capaz de escalonar um conjunto de controladores com especificação ótima é tratado como um problema de *co-design* entre o sistema de controle e o escalonamento de tempo real. Entres as soluções propostas por Cervin (2003) para reduzir atrasos e *jitters* em malhas de controle estão às alterações no nível de programação das tarefas e no tipo de escalonamento.

Necessariamente, uma solução para os problemas da implementação dos sistemas de controle em plataforma de tempo real dependem da pesquisa, desenvolvimento e aplicação de métodos e metodologias específicas para o tratamento dos atrasos e *jitters*. Tanto a pesquisa, quanto a aplicação de novos métodos e metodologias de tratamento dos problemas relacionados aos atrasos e *jitters* não são mapeados de imediato pelos geradores de código automático, que possuem como requisito principal o aumento da produtividade na confecção de programas computacionais e se baseiam em construções de blocos específicos.

Uma opção à utilização dos geradores de códigos automáticos é aplicar métodos de Engenharia de *Software* no desenvolvimento de *frameworks* orientados a objetos. Estes *frameworks* podem ser utilizados como ferramentas auxiliares durante a pesquisa, desenvolvimento e incorporação de novos métodos e metodologias de tratamento, para tratar os problemas provenientes da implementação da malha de controle em uma plataforma de tempo real citados acima. Além disso, *frameworks* permitem que o código e o projeto do *software* de controle sejam reutilizados. O desenvolvimento dos *frameworks* pode ser realizado em paralelo com a fase de modelagem do *software* de controle. Os artefatos de *software* gerados para o sistema de controle podem ser agrupados nos módulos que constituem a arquitetura dos *frameworks*.

1.1 Contextualização

A teoria de controle e a teoria de sistemas de tempo real se desenvolveram nas últimas décadas como áreas independentes (CERVIN, 2003). Para a comunidade de controle, o sistema de tempo real é visto somente como uma plataforma na qual o controlador pode ser implementado de maneira trivial. Para a comunidade de sistemas de tempo real, o controlador é apenas um código de programa caracterizado por três parâmetros: um período de tempo fixo

T_i, um tempo de computação de pior caso C_i e um instante de finalização D_i (*hard deadline*). Recentemente, foi observado em (CERVIN, 2003) a necessidade de utilizar uma abordagem do tipo *co-design*, o que implica na aproximação entre as áreas de controle e de tempo real no projeto de sistemas de controle.

A base para a implementação de sistemas de controle em computadores digitais, mais precisamente, de tempo discreto para a realização de sistemas de controle tem sido a teoria de sistemas de controle discreto. Obviamente, o surgimento dos microcomputadores a partir da década de 70 permitiu a materialização de tais sistemas de controle.

Tradicionalmente, na implementação de sistemas controlados por um computador digital como o ilustrado na Figura 1.2, a comunidade de controle tem assumido que a plataforma computacional utilizada é capaz de escalonar as tarefas de controle, de tal maneira que, seja possível fornecer uma amostragem determinista e eqüidistante, que é a base da teoria de controle de sistemas de tempo discreto. No entanto, esta garantia de determinismo mesmo em plataformas de computação desenvolvidas para dar suporte a sistemas *hard real-time* não é fácil de ser cumprida.

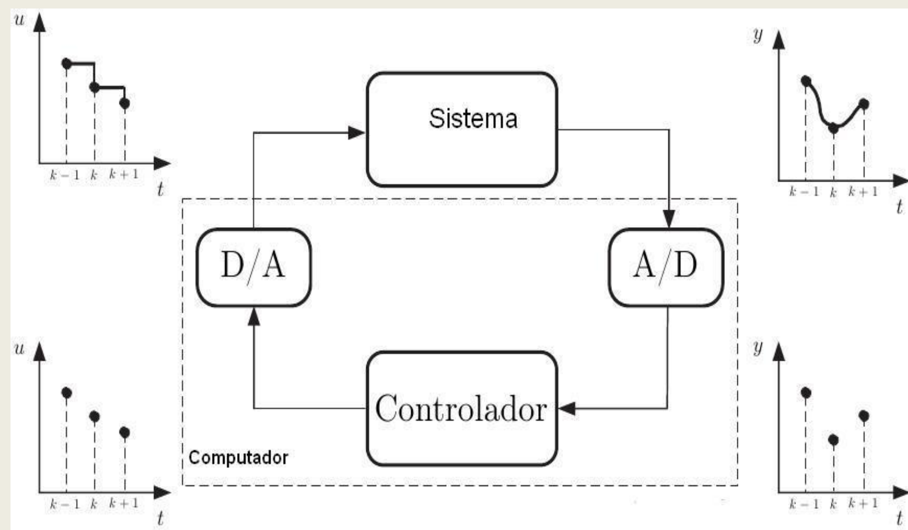


Figura 1.2: Sistema de controle computadorizado (TAKARABE, 2009).

Os principais problemas em sistemas controlados por computador são os atrasos e os *jitters* de amostragem, computação e liberação das tarefas que são introduzidos pelas variações dos atrasos. Muitas metodologias foram propostas por Cervin (2003) e Marti (2002) com o objetivo de reduzir os atrasos, ou torná-los mais determinísticos, ou seja, livre dos *jitters*. Nestas propostas, o atraso é compensado no projeto dos controladores utilizando a teoria de controle discreto (ÅSTRÖM, K. J. & WITTENMARK, B., 1997).

Os atrasos e *jitter* são agravados quando os sistemas de controle são distribuídos em redes ou NCSs (*Networked Control Systems*) como mostrado na Figura 1.3. Em um NCS, os valores dos sinais de controle enviados pelos controladores para os atuadores e os valores das saídas das plantas amostrados pelos sensores e enviados para os controladores, trafegam por uma rede de comunicação através da qual atuadores, sensores e controladores se comunicam. Neste caso, além do atraso para o processamento dos dados no nó controlador e o *jitter* relativo à variação deste atraso, existem também os atrasos nas comunicações entre sensores e controladores, entre controladores e atuadores e os respectivos *jitters* relativos às variações destes atrasos.

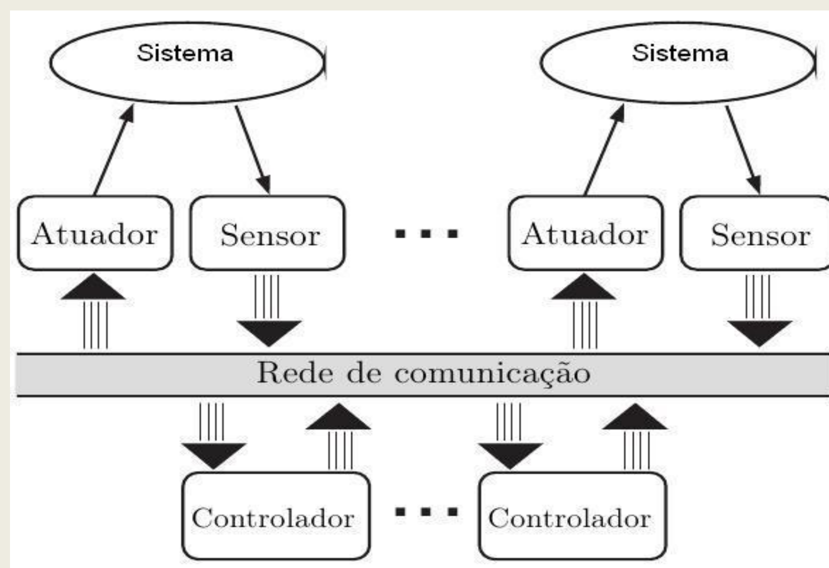


Figura 1.3: Sistema de controle distribuído em rede (TAKARABE, 2009).

Nilsson (1998) propõe três modelos para os atrasos na malha de controle. No primeiro caso, o atraso é considerado constante para todas as transmissões. Em um segundo caso, o atraso é modelado como uma seqüência aleatória e independente e no terceiro caso a distribuição dos atrasos é governada por cadeias de Markov.

Em termos de implementação, as malhas de controle centralizadas ou distribuídas podem ser implementadas através de uma única tarefa ou várias tarefas. Basicamente a malha de controle é implementada como uma tarefa com um laço infinito e uma diretiva capaz de suspender a execução do *loop* por um intervalo de tempo T, freqüentemente denotado por h. O intervalo de tempo h depende da dinâmica da planta ou processo.

Durante uma fase de modelagem do sistema de controle, técnicas de Engenharia de *Software* como *Real Time* UML (DOUGLAS, 2000), têm sido utilizadas tornando possível reutilização do *software* de controle através de mecanismos como herança e composição. Em (PINHO, 2001) é apresentada uma proposta de *framework* para replicação de aplicações de tempo real tolerantes a falhas. Outra proposta de *framework* orientado a objetos para sistemas de controle pode ser visto em (BLUM, A.; CECHTICKY, V. & PASETTI, A., 2010).

Do ponto de vista da análise, a equipe do *Lund Institute of Technology* desenvolveu duas importantes ferramentas de projeto e simulação: *Jitterburg* (CERVIN, A. & LINCOLN, B., 2006) e *TrueTime* (ANDERSSON, M.; CERVIN, A. & HENRIKSSON, D., 2005). Estás ferramentas podem ser utilizadas para a análise do *jitter* e escalonamento do sistema em plataformas de tempo real.

Uma técnica bastante utilizada atualmente na indústria automotiva, como um método rápido de testar controladores diretamente na plataforma computacional é RCP. Em (HÖLTTÄ, V.; PALMROTH, L. & ERICKSON, L., 2004), é mostrado um exemplo da aplicação de RCP no qual, após um estágio de prototipagem o protótipo do sistema de controle é convertido automaticamente em um controlador para tempo real e embarcado no

target. Com a utilização de RCP é possível analisar se a plataforma computacional será capaz de atender aos requisitos de desempenho do sistema de controle durante a fase de projeto.

1.2 Rapid Control Prototyping (RCP)

Rapid Control Prototyping fornece uma solução de prototipagem rápida de novas funções em diferentes tipos de processos industriais e dispositivos controlados por um sistema de controle complexo (HÖLTTÄ, V.; PALMROTH, L. & ERICKSON, L., 2004). A idéia básica contida nesta abordagem é desenvolver e testar novas estratégias de controle em um ambiente de simulação e na seqüência testar em um ambiente real. O diagrama de blocos ilustrado na Figura 1.4 mostra uma plataforma RCP utilizada atualmente na indústria para aplicação da metodologia RCP.

Figura 1.4: Diagrama de blocos de uma plataforma para RCP.

No modelo de plataforma ilustrado na figura o estágio RCP do sistema de controle, que ocorre no *host*, é encerrado quando os resultados da simulação são considerados satisfatórios. Após o estágio de RCP, o protótipo do controlador pode ser transformado em um protótipo de controlador para tempo real. Este protótipo é utilizado para testar o algoritmo de controle em condições de trabalho reais com *target* e planta real.

Os processos de modelagem e simulação do sistema são executados no *host* utilizado ferramentas como SIMULINK, por exemplo. A tradução do protótipo desenvolvido no estágio de simulação no *host* para o *target* é realizado por geradores de código automático

sem que haja necessidade de modelagem e implementação do *software* de controle. Após a tradução, o código do protótipo é compilado, linkado e embarcado no *target* para controlar a planta real.

Processo de desenvolvimento de sistemas de controle aplicando RCP

O processo de desenvolvimento de sistemas de controle aplicando RCP utilizado atualmente na indústria é mostrado na Figura 1.5. O processo mostrado na figura é composto pelo estágio de RCP, fases 1, 2, 3, 4 e pelo estágio de implementação do código para o *target*, fase 5. Este processo apresenta problemas decorrentes da geração automática de código e integração posterior entre o sistema de controle que está sendo desenvolvido e sua execução no *target*.

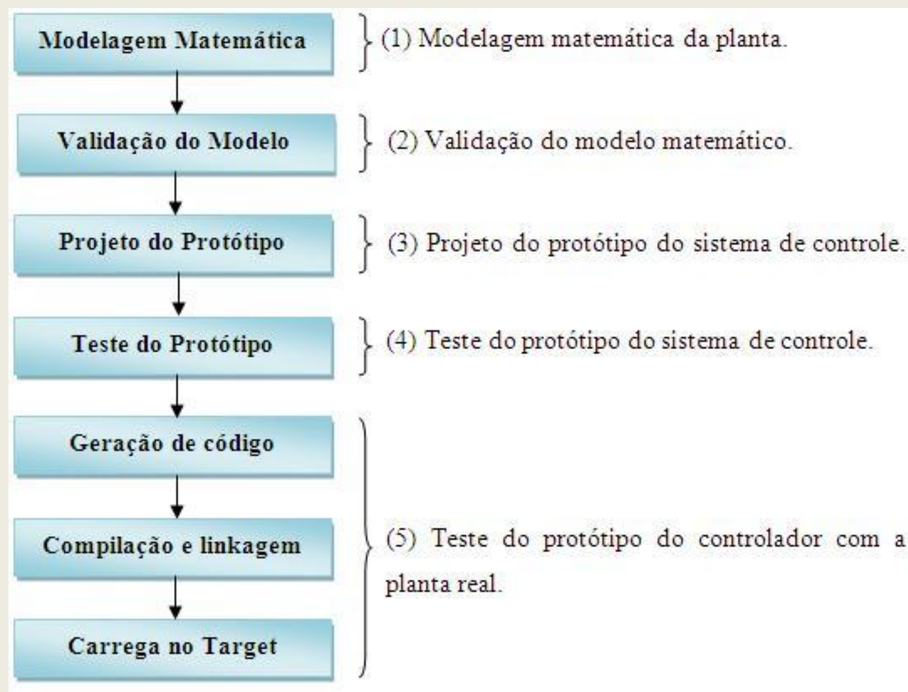


Figura 1.5: Processo de desenvolvimento de sistemas de controle aplicando RCP.

Em (BOSTIC, D. ET AL., 2002) são enfatizados alguns dos problemas com a geração automática de código tais como: ineficiência no uso dos recursos computacionais, falta de integração com código legado, etc.. Lee (2002), enfatiza a necessidade de estreitar a barreira

técnica existente entre o estágio de RCP que acontece no *host* e a implementação do código para o *target*. Para tanto, Lee (2002) propõe uma nova plataforma RCP com uma camada de abstração do *hardware* do *target* e uma linguagem para integrar o código gerado com o código legado.

1.3 Contribuições e Objetivos

Entre as contribuições deste trabalho está a proposta de duas arquiteturas de *frameworks* orientados a objetos para substituir a geração de código automático na metodologia *Rapid Prototyping Control*. Estes *frameworks* serão integrados a proposta de um modelo de arquitetura para plataforma RCP. Neste modelo de arquitetura pretende-se eliminar a integração posterior entre *software* de controle que está sendo desenvolvido e os testes no *target* tornando o processo de desenvolvimento utilizando RCP mais interativo.

Outras contribuições incluem a proposta de um método para compensação do *jitter* de amostragem baseado na utilização do período de amostragem médio; Proposta de um algoritmo para cálculo do WCET das tarefas de controle em plataformas de tempo real utilizando o método prático e análise estruturada;

O objetivo deste trabalho é propor alterações no processo de desenvolvimento do sistema de controle aplicando RCP e utilizar os *frameworks* RCPs como uma das camadas da plataforma RCP para realizar as seguintes análises:

- Analisar a influência entre os algoritmos de escalonamento de processo, *Fixed Priority* (FP) e o *Round Robin* (RR), no desempenho do sistema de controle utilizando o esquema de atribuição de prioridade *Rate Monotonic* (RM) introduzido por Liu e Layland (1973).

- Analisar o desempenho dos sistemas de controle quando as tarefas de controle são implementadas aplicando a metodologia *subtask scheduling* proposta por Cervin (2003) sem compensação do *jitter* no intervalo de amostragem e com compensação do *jitter* no intervalo amostragem utilizando o método de compensação proposto;
- Possibilitar que seja feita uma comparação de desempenho quando os protótipos dos controladores desenvolvidos utilizando o *framework* FCC estiverem atuando sobre modelos de plantas, com os resultados obtidos quando os protótipos estiverem atuando sobre as plantas reais conectadas as interfaces de entrada e saída (AD/DA) da plataforma RCP.

1.4 Apresentação do trabalho

Capítulo 2: Este capítulo apresenta uma introdução a sistemas de tempo real, modelos de tarefas, sub-tarefas, algoritmos de escalonamento. São apresentados aspectos relevantes da integração entre controle e tempo real, focando principalmente nas restrições temporais que devem ser consideradas durante a implementação das tarefas de controle em plataformas de tempo real. O capítulo é finalizado com uma introdução sobre redes de computadores e a apresentação de alguns trabalhos sobre controle e tempo real.

Capítulo 3: Neste capítulo são apresentadas duas arquiteturas propostas para os *frameworks* orientados a objetos que serão a base para a modelagem e desenvolvimento de sistemas de controle no decorrer do trabalho.

Capítulo 4: Neste capítulo é apresentada uma proposta de arquitetura para plataforma RCP e sua integração com os *frameworks* implementados em Ada 2005 a partir das arquiteturas de

frameworks mostradas no Capítulo 3. O capítulo mostra um exemplo da utilização dos *frameworks* na análise do desempenho de sistemas de controle centralizado e distribuído em uma rede *Ethernet* (protocolo TCP/IP). No exemplo, os protótipos dos controladores desenvolvidos utilizando os *frameworks* são embarcados nas plataformas RCP, centralizada e distribuída, em várias etapas do processo de desenvolvimento dos sistemas de controle tornando o processo de desenvolvimento aplicando RCP mais interativo. O capítulo é finalizado com a apresentação e o desenvolvimento de um sistema de controle para uma plataforma RCP centralizada com plantas reais.

Capítulo 5: Neste capítulo é apresentado um método para compensação do *jitter* no intervalo de amostragem utilizando período médio das últimas n ativações das tarefas de controle e um algoritmo para calcular o WCET utilizando o método prático e análise em nível de estrutura. Além disso, são apresentados os resultados da implementação das tarefas de controle aplicando a metodologia *subtask scheduling* com compensação do *jitter* no intervalo de amostragem e sem compensação do *jitter* no intervalo amostragem e os resultados da análise dos algoritmos de escalonamento de processos (FP e RR) no desempenho dos sistemas de controle.

Capítulo 6: Neste capítulo são apresentadas as conclusões e sugestões para trabalhos futuros relacionados à integração entre sistemas de controle e tempo real.

2 Sistemas de controle e tempo real

Sistemas de tempo real é uma área de pesquisa interdisciplinar que inclui Engenharia de Controle e Ciência da Computação (ÅRZÉN, K. ET AL., 1999). Esta área é de vital importância para todos os engenheiros de controle. Atualmente praticamente todos os controladores são implementadas em computadores digitais e a implementação bem sucedida de um sistema controlado por computador requer uma boa compreensão tanto da teoria de controle quanto da teoria de sistemas de tempo real.

Muitos sistemas de controle em tempo real são sistemas embarcados, onde o computador é um componente de um “grande” sistema de engenharia. Estes sistemas de controle são freqüentemente implementados em microprocessadores usando linguagens de programação de tempo real, tais como Ada, Modula-2, Java ou usando linguagens de programação seqüencial como C ou C++ juntamente com um sistema operacional de tempo real (SOTR). Exemplos de tais sistemas de controle são encontrados em sistemas mecatrônicas, aplicações aeroespaciais, sistemas de veículos e produtos eletrônicos.

Tradicionalmente na implementação de sistemas de controle por computador, a comunidade de controle tem assumido que a plataforma computacional utilizada é capaz de fornecer uma amostragem determinista e eqüidistante, que é a base teoria de controle de sistemas amostrados. No entanto, muitas das plataformas de computação comumente usadas para implementar sistemas de controle não são capazes de dar quaisquer garantias determinísticas e de eqüidistância da amostragem. Um caso especial é quando se utiliza um sistema operacional comercial como o Windows NT, por exemplo. Estes sistemas operacionais são tipicamente concebidos para alcançar um “bom” desempenho, em vez de

garantir um desempenho de pior caso. Eles freqüentemente apresentam um indeterminismo significativo no agendamento tarefas.

A comunidade de computação em tempo real, pó outro lado, assume que todos os algoritmos de controle devem ser modelados como tarefas com as características abaixo:

- Periódicas com um conjunto de períodos fixos (T);
- Prazos rígidos, ou seja, devem ser executadas dentro de um *deadline* (D);
- Tempo de computação do pior caso C (WCET *Worst Case Computation Time*) conhecido;

Esta abordagem tem permitido que a comunidade de controle concentre no seu domínio do problema sem se preocupar com o escalonamento que está sendo feito. Ficando a cargo da comunidade de computação estudar o impacto que os atrasos na programação das tarefas têm sobre a estabilidade e desempenho do sistema de controle.

Após uma inspeção mais realista fica evidente que nenhuma das três hipóteses citadas acima é necessariamente verdadeira. Muitos algoritmos de controle não são periódicos, ou podem alternar entre diferentes períodos de amostragem. Entretanto, controladores robustos suportam variações no período de amostragem e tempo de resposta e em muitos casos, é possível compensar as variações dos intervalos de amostragem recalculando os parâmetros do controlador utilizando uma abordagem de compensação. Existe também a possibilidade de considerar sistemas de controle capazes de escolher como utilizar o tempo de computação, ou seja, quando há muito tempo disponível o controlador pode utilizar o tempo para cálculo do novo sinal de controle ou cálculo do desempenho malha através do ajuste dos parâmetros do controlador em tempo de execução.

Outro problema quando se utiliza a abordagem acima é obter um valor exato para o WCET. Este é um problema muito difícil na área de escalonamento em tempo real (FARINES, J. ;FRAGA, J. S. & OLIVEIRA, R. S., 2000). Temos que muitas das construções das linguagens de programação de caráter geral são fontes de indeterminismo, tornando o sistema não previsível (FARINES, J. ;FRAGA, J. S. & OLIVEIRA, R. S., 2000), como exemplo podemos citar: laços não limitados, ou ainda primitivos da linguagem Ada como *delay* que determina um limite mínimo que uma tarefa fica esperando na fila de Pronto, mas não é capaz de determinar um limite máximo e o *select* que introduz uma escolha aleatória na opção de fluxos alternativos.

Considerando os objetivos específicos dos projetos de sistemas de controle por computador, no qual se pretende desenvolver um controlador ótimo que minimize um determinado índice de desempenho. A otimização destes sistemas pode ser limitada pelos recursos computacionais disponíveis especialmente em aplicações avançadas como no caso de um estimador de estados onde: pretende-se controlar a dinâmica de uma planta rápida usando estimador de estados e algoritmos de controle sofisticados.

O objetivo da utilização de programação em tempo real na implementação do sistema de controle como o citado anteriormente, é alocar os recursos computacionais limitados de tal forma que, a estimação de estado e os algoritmos de controle possam garantir a estabilidade e otimização do desempenho do sistema de controle como um sistema de tempo real. Os recursos de computação podem incluir tempo de CPU, largura de banda de comunicação em sistemas de controle distribuído, etc.

Necessariamente, o sucesso do desenvolvimento de sistemas de controle em plataformas de tempo real depende de metodologias de desenvolvimento de projeto que busquem aproximar as áreas de controle e computação em tempo real.

2.1 Modelo de tarefa de tempo real

O conceito de tarefa é uma das abstrações básicas que faz parte do que chamamos de problema de escalonamento (FARINES, J. ;FRAGA, J. S. & OLIVEIRA, R. S., 2000). Tarefas ou processos formam as unidades de processamento seqüencial que concorrem sobre um ou mais recursos computacionais do sistema. Aplicações de controle são constituídas tipicamente por tarefas de tempo real com restrições severas de prazos. Neste tipo de tarefa é importante satisfazer a correção lógica (*Correctness*) e os prazos ou restrições temporais. As restrições temporais, as relações de precedência e de exclusão usualmente impostas sobre tarefas são determinantes na definição de um modelo de tarefa que é parte integrante de um problema de escalonamento.

2.1.1 Restrições temporais

Todas as tarefas de tempo real tipicamente estão sujeitas a prazos ou *deadlines*. A princípio, uma tarefa deve ser concluída antes de seu *deadline*. As conseqüências de uma tarefa ser concluída após o seu *deadline* definem dois tipos de tarefas de tempo real:

- Tarefas críticas: uma tarefa é dita crítica quando ao ser completada após seu *deadline* pode causar falhas catastróficas no sistema de tempo real e em seu ambiente;
- Tarefas brandas: são tarefas que quando completadas depois de seus *deadlines* no máximo implicam na diminuição de desempenho do sistema;

Quanto a regularidade ou freqüência de suas ativações, as tarefas em sistemas de tempo real são classificadas em dois modelos: periódicas e aperiódicas.

- Tarefas periódicas: quando as ativações do processamento da tarefa ocorrem em intervalos regulares chamado de período;
- Tarefas aperiódicas: quando a ativação do processamento da tarefa responde a eventos internos ou externos definindo uma característica aleatória;

Outras restrições temporais são: os tempos de computação, início, término, chegada e liberação. Dependendo do modelo de tarefa, o tempo de liberação pode ou não coincidir com o tempo de chegada da tarefa. Em geral, é assumido que uma tarefa é liberada imediatamente após sua chegada na fila de Pronto. Porém, a liberação da tarefa pode ser retardada pelo *polling* de um escalonador ativado por (*tick scheduler*) ou pelo bloqueio na recepção de uma mensagem, no caso de tarefas ativadas por mensagens. Esta diferença entre os tempos de chegada e liberação das tarefas conduz ao que é identificado como *jitter* de liberação (J_i).

Considerando as restrições temporais citadas, o comportamento temporal de uma tarefa periódica τ_i pode ser descrito através da quádrupla (J_i, C_i, T_i, D_i) onde:

C_i é o tempo de computação igual ao WCET;

T_i é o período de ativação;

D_i é o *deadline*;

J_i é o *jitter* de liberação da tarefa, que corresponde à pior situação de liberação da mesma;

Neste modelo de tarefas periódicas, J_i e D_i são intervalos medidos a partir do início do período de ativação T_i. O *deadline* absoluto e o tempo de liberação da k-ésima ativação da tarefa periódica τ_i são determinados a partir dos períodos anteriores.

2.1.2 Relações de Precedência e exclusão

A ordem na qual os processamentos são executados em aplicações de tempo real define as relações de precedência entre as tarefas da aplicação. Quando uma tarefa τ_k é precedida por uma tarefa τ_i, denotado por ($\tau_i \rightarrow \tau_k$). Se τ_k pode iniciar sua execução somente após o término da execução de τ_i pode-se expressar relações de dependências de informações ou sinais de sincronização entre as tarefas.

Para representar as relações de precedência em um conjunto de tarefas, habitualmente é utilizado um grafo acíclico orientado onde, os nós correspondem às tarefas do conjunto e os arcos descrevem as relações de precedência existentes entre as tarefas.

Outra relação significativa que é definida pelo compartilhamento de recursos mutuamente exclusivos entre tarefas de tempo real, são as relações de exclusão. Uma tarefa τ_i exclui τ_k quando a execução de uma seção crítica de τ_k que manipula o recurso compartilhado não pode executar porque o recurso já está sendo usado por τ_i. Um dos problemas das relações de exclusão em escalonamentos dirigidos a prioridade é a inversão de prioridades onde tarefas mais prioritárias são bloqueadas por tarefas menos prioritárias (BURNS, A. & WELLINGS, A. J., 2001).

2.2 Escalonamento em tempo real

Escalonamento ou *scheduling* é o procedimento de ordenar tarefas em uma fila de Pronto utilizando uma escala de execução que indica a ordem de ocupação do processador por um conjunto de tarefas na fila. O escalonador ou *scheduler* é o componente do sistema responsável pela gestão dessa ordenação de tarefas a serem executadas no processador. É o escalonador quem implementa a política de escalonamento ao ordenar um conjunto de tarefas

para execução sobre o processador. As políticas de escalonamento definem critérios ou regras para a ordenação das tarefas. Ao adotar estas políticas, os escalonadores produzem escalas que se forem realizáveis, garantem o cumprimento das restrições temporais impostas às tarefas de tempo real. Uma escala é dita ótima se a ordenação do conjunto de tarefas, de acordo com os critérios pré-estabelecidos pela política de escalonamento, é a melhor possível no atendimento das restrições temporais.

Tomando como base o cálculo da escala, algumas classificações são encontradas para a grande variedade de algoritmos de escalonamento de tempo real encontrados na literatura (FARINES, J. ;FRAGA, J. S. & OLIVEIRA, R. S., 2000). Os algoritmos são ditos preemptivos ou não preemptivos quando a qualquer momento, a tarefa executando pode ou não ser interrompida por outra de maior prioridade. Quando o cálculo da escala é feito tomando como base apenas os parâmetros atribuídos às tarefas do conjunto em tempo de projeto, os algoritmos de escalonamento são identificados como estáticos. Os algoritmos dinâmicos são baseados em parâmetros que podem mudar em tempo de execução com a evolução do sistema. Os algoritmos também são identificados de acordo com a produção da escala. Quando a escala é produzida em tempo de projeto os algoritmos são ditos *off-line*. Se a escala é produzida em tempo de execução o algoritmo de escalonamento é dito *on-line*.

Na sua forma geral, o problema de escalonamento envolve um conjunto de processadores, recursos compartilhados e tarefas com suas restrições temporais, de precedência e exclusão (FARINES, J. ;FRAGA, J. S. & OLIVEIRA, R. S., 2000). O escalonamento de tempo real é identificado como um problema intratável (NP - Completo (GARCEY, M. & JOHNSON, D. S., 1979), (BURNS, A. & WELLINGS, A. J., 2001)). Freqüentemente os algoritmos existentes são ótimos apenas para uma determinada classe de problemas. Assim, os algoritmos estão ligados a classes de problemas, sendo que um algoritmo é identificado como ótimo se minimiza ou maximiza algum custo ou métrica

definida sobre a sua classe de problema. Se nenhum custo ou métrica é definido, a única preocupação é então encontrar uma escala realizável. Neste caso, o algoritmo é dito ótimo quando consegue encontrar uma escala realizável para o conjunto de tarefas. Se o algoritmo ótimo falha na determinação de uma escala realizável então todos os algoritmos da mesma classe de problema também falharão.

Entre os algoritmos ótimos para suas classes de problemas temos o algoritmo de prioridades fixas *Rate Monotonic* (RM) e o algoritmo *Earliest Deadline First* (EDF) que apresenta atribuição dinâmica de prioridades. Estes algoritmos foram propostos por Liu e Layland (1973). As análises dos conjuntos de tarefas escalonadas pelo RM e EDF são realizadas utilizando os testes de escalonabilidade disponíveis para os mesmos. Em (CERVIN, 2003), estes algoritmos são aplicados com a metodologia *subtask scheduling* a um conjunto de tarefas de controle para analisar a latência de entrada e saída (L_{io}) e a latência de amostragem (L_s) das tarefas do conjunto.

2.2.1 *Rate Monotonic* (RM)

O escalonamento RM é um esquema de prioridade fixa que produz escalas em tempo de execução através de escalonadores preemptivos, dirigidos a prioridades. O RM é um algoritmo ótimo entre os escalonamentos de prioridade fixa na sua classe de problemas.

A análise de escalonabilidade do RM define um modelo de tarefas bastante simples de acordo com as quatro premissas abaixo:

1. As tarefas são periódicas e independentes;
2. O *deadline* de cada tarefa é igual ao período ($D_i = T_i$);
3. O tempo de computação (C_i) de cada tarefa é conhecido e constante;
4. O tempo de chaveamento entre tarefas é assumido como nulo;

Na prática as premissas 1 e 2 são muito restritivas, contudo essas simplificações facilitam o entendimento sobre o escalonamento de tarefas periódicas. A política de atribuição de prioridades usando o RM é baseada nos valores dos períodos das tarefas do conjunto: quanto mais freqüente a tarefa, maior a sua prioridade no conjunto. Como os períodos das tarefas não mudam, o RM define uma atribuição estática de prioridades.

A análise de escalonabilidade do RM feita em tempo de projeto é baseada no cálculo da utilização (U). O teste condicional da Equação 2.1 define uma condição suficiente, mas não necessária, para que, um conjunto com n tarefas tenham suas restrições temporais atendidas quando escalonadas pelo RM.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \ \ 2^{1/n} - 1 \qquad (2.1)$$

Quando $n \to \infty$ na Equação 2.1 a utilização do processador converge para 0,693. Este baixo valo de utilização **U** do processador implica no descarte de muitos conjuntos de tarefas com utilização **U** maior que o limite, mas que apresentam escalas realizáveis. Quando, o conjunto de tarefas apresenta períodos múltiplos do período da tarefa mais prioritária, esta condição suficiente (Equação 2.1) pode ser relaxada. Neste caso, a utilização alcançada sob o RM se aproxima do máximo teórico igual a 100%, coincidindo com o teste da Equação 2.2 como uma condição necessária e suficiente (KOPETZ, 1992).

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad (2.2)$$

2.2.2 *Earliest Deadline First* (EDF)

O EDF define um escalonamento baseado em prioridades dinâmicas onde a escala é produzida em tempo de execução por um escalonador preemptivo. Este é um algoritmo ótimo na classe dos escalonamentos de prioridade dinâmica. As premissas que determinam o modelo de tarefas no EDF são às mesmas do RM.

A atribuição dinâmica de prioridades no EDF define a ordenação das tarefas segundo os seus *deadline*s. A tarefa mais prioritária é a que tem o *deadline* mais próximo de expirar. A cada chegada de uma nova tarefa, a fila de Pronto é reordenada, considerando a nova distribuição de prioridades. A cada ativação de uma tarefa τ_i um novo valor de *deadline* absoluto é determinado considerando o número de períodos que antecede a atual ativação.

A escalonabilidade do conjunto de tarefas pode ser verificada em tempo de projeto, analisando a utilização do processador. Um conjunto de tarefas periódicas é escalonável com o EDF se e somente se:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.3}$$

Este teste é necessário e suficiente na classe de problemas definidos para o EDF. Se qualquer uma das premissas é relaxada (por exemplo, assumindo $D_i \neq T_i$), a condição contínua a ser necessária, porém não é mais suficiente.

2.3 Implementação de malhas de controle

Uma malha de controle é composta basicamente por três partes principais: a amostragem ou coleta de dados a computação do algoritmo de controle e a transmissão da saída ou atuação, como ilustrado na Figura 2.1. No caso mais simples a amostragem de dados e a transmissão da saída são compostas de chamadas para uma interface de I/O externa, por exemplo, como conversores A/D e D/A ou um barramento. Em uma estrutura mais complexa, os dados de entrada podem ser provenientes de outros elementos computacionais como um sinal que pode ser filtrado e ter a saída envidada para outros elementos computacionais ou outras malhas de controle.

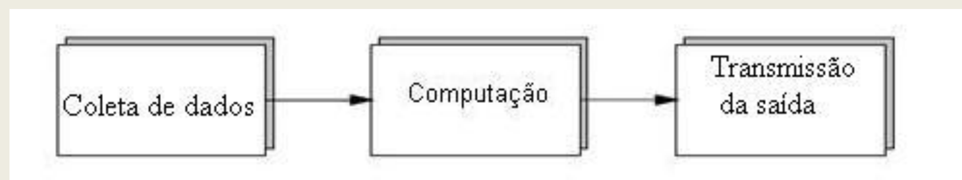


Figura 2.1: Principais partes da malha de controle.

Em geral, o algoritmo de controle é executado periodicamente com um período ou intervalo de amostragem constante e igual a T (também denotado por h). O período é determinado pela dinâmica do processo que se deseja controlar e pelos requisitos de desempenho na malha fechada. Uma regra prática apresentada em (ÅSTRÖM, K. J. & WITTENMARK, B., 1997) estabelece que o período de amostragem h deve estar no intervalo [4 a 10] vezes por tempo de subida T_r do sistema em malha fechada. Seja N_r o número de amostragem por tempo de subida T_r temos que:

$$N_r = \frac{T_r}{h} \approx 4..10 \tag{2.4}$$

Existem duas alternativas para implementar uma malha de controle em um sistema operacional de tempo real utilizando o conceito de tarefas. Uma das alternativas é utilizar uma única tarefa periódica. Neste caso o cálculo do algoritmo de controle é executado como código seqüencial na tarefa. Outra alternativa é utilizar múltiplas tarefas, sendo que, cada parte da malha de controle é implementada como uma tarefa separada. Está abordagem se originou da metodologia de Engenharia de *Software* RTSD (*Real-Time Structured Analysis and Design*) (WARD, 1985). Nesta metodologia cada transformação de dados é mapeada em uma tarefa separada. Aplicando a metodologia para o caso da malha de controle da Figura 2.1 teríamos três tarefas distintas e periódicas. A vantagem desta abordagem é a facilidade de distribuir o processamento em vários processadores e sistemas distribuídos. A desvantagem é que o grande número de tarefas criadas simultaneamente gera uma potencial sobrecarga para sincronização, comunicação e mudanças de contexto.

Na abordagem de tarefa simples as transformações de dados são agrupadas dentro das tarefas de acordo com a seqüência temporal de execução. As principais vantagens desta abordagem são: menor número de tarefas a serem executadas, menor demanda de sincronização e comunicação entre processos, menos trocas de contexto. Está é também a abordagem mais natural para os engenheiros de controle, especialmente na implementação de pequenos sistemas de controle utilizando plataformas com um único processador. A principal desvantagem desta abordagem é a introdução de atrasos desnecessários que deterioram o desempenho dos algoritmos de controle (MARTI, 2002). Estes atrasos são introduzidos devido à execução de todas as transformações de dados antes da transmissão da saída.

Existem situações onde pode ser difícil escolher que abordagem deve ser utilizada. Particularmente nos casos onde partes da malha de controle executam suas tarefas em freqüências diferentes. Uma situação é ilustrada em (ÅRZÉN, K. ET AL., 1999), onde uma filtragem *anti-aliasing* é implementada como filtro passa-baixa analógico em combinação

com um filtro passa-baixa discreto.

2.3.1 Restrições temporais na malha de controle

As características básicas da malha de controle estabelecem as seguintes restrições temporais:

1. o período de amostragem h, entre sucessivas ativações da tarefa de controle τ deve ser constante, ou seja, livre de jitter;
2. envolve o atraso computacional, entre a amostragem dos dados de entrada e a transmissão da saída ou latência de entrada e saída;

Do ponto de vista de controle o atraso computacional tem efeitos similares a um atraso na entrada do processo que se deseja controlar.

Em relação a uma tarefa τ, as seguintes considerações devem ser feitas:

1. após ser liberada a tarefa pode ter seu início de execução atrasado devido a uma tarefa com prioridade mais elevada estar sendo executada;
2. existe também a possibilidade de ocorrer atraso na liberação da saída da tarefa nas seguintes circunstâncias:
   i. a tarefa começar a executar e em seguida é preemptada por uma tarefa de prioridade mais alta;
   ii. a tarefa fica bloqueada aguardando a liberação de um recurso mutuamente exclusivo que está em poder de outra tarefa;

As restrições citadas acima e as considerações em relação ao modelo de tarefas apresentado na seção 2.2.1, definem as grandezas que caracterizam o comportamento temporal da malha de controle durante sucessivas ativações de uma tarefa τ. Estas grandezas

estão ilustradas na Figura 2.2. As setas verticais indicam os instantes de amostragem da tarefa τ (Conversão A/D). Os instantes de tempo que ocorrem as liberações das saídas (Conversão D/A) estão representados por setas verticais com círculos preenchidos nas pontas. Os instantes em que a tarefa está sendo processada são representados por um quadrado preenchido em tons de cinza.

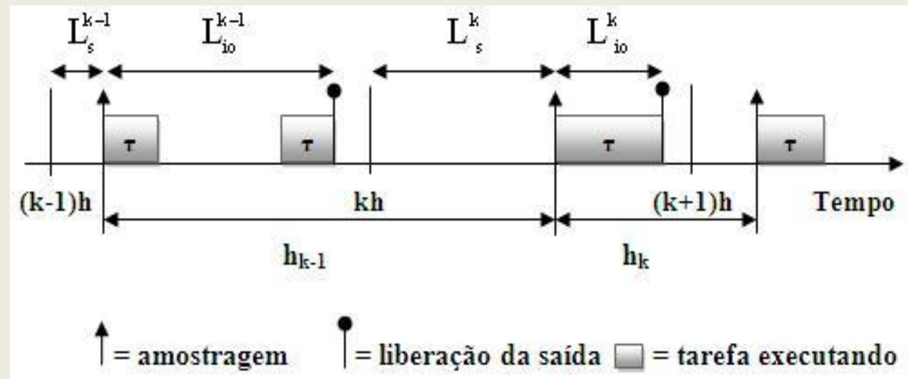


Figura 2.2: Grandezas que caracterizam o comportamento temporal das tarefas de controle.

As seguintes grandezas são mostradas na figura:

- Intervalo de amostragem h_k: intervalo de tempo entre os instantes de amostragem k e (k+1);
- Latência de entrada-saída L_{io}^k : atraso entre a amostragem do sinal de entrada e a saída do sinal de controle para k-ésima ativação da tarefa;
- Latência de amostragem L_s^k : atraso entre a chegada da tarefa na fila de prontos e o início de execução para a k-ésima ativação da tarefa;
- *Jitter*: variações na latência de amostragem são denominadas *jitter* de amostragem (J_s). O *jitter* de amostragem também pode causar *jitter* (J_h) no intervalo de amostragem h_k. Outro tipo de *jitter* é o de entrada-saída (J_{io}) que são causados pelas variações no tempo de execução C_i do algoritmo de controle;

Segundo Årzén et al. (1999), as quatro abordagens a seguir são possíveis para tratar o atraso em malhas de controle considerando as restrições e as grandezas de tempo que caracterizam a malha.

1. A abordagem mais simples é implementar o sistema de tal forma que o atraso seja minimizado e depois ignorá-lo no projeto do controlador;
2. Garantir que o atraso seja constante e incluir o atraso no projeto do controlador. Uma maneira de fazer isso é esperar a transmissão de saída até o início da próxima amostragem. Desta forma, o atraso computacional se aproxima do período de amostragem. Se o controlador é projetado de acordo com a teoria de controle discreto (ÅSTRÖM, K. J. & WITTENMARK, B., 1997) fica relativamente fácil compensar esse atraso;
3. Projetar controladores robustos onde o atraso seja tratado como um parâmetro de incerteza. Esta abordagem é substancialmente mais complexa do que as duas primeiras;
4. Compensar o atraso computacional ou parte dele durante cada ativação do algoritmo de controle em determinadas situações. Esta abordagem foi utilizada em (NILSSON, 1998) para compensar os atrasos computacionais em sistemas de controle distribuído e em (MARTI, P. & FUERTES, J. M., 2001) para compensar os atrasos em controladores do tipo PID;

2.3.2 Compensação do *jitter* e atraso

Através da abordagem de tarefa única é relativamente simples mapear as restrições de tempo da malha de controle em parâmetros da tarefa (ÅRZÉN, K. ET AL., 1999). O período ou intervalo de amostragem h da malha corresponde ao período de ativação T_i da tarefa τ_i. No

entanto, os requisitos de atraso e *jitters* computacionais são difíceis de manusear (ÅRZÉN, K. ET AL., 1999).

Uma abordagem relativamente simples para reduzir o atraso computacional é transmitir a saída do controle no final da tarefa. Isto fará com que o atraso computacional seja sempre menor que o *deadline* da tarefa. O problema neste caso é que o *jitter* pode se tornar considerável, ou seja, a execução entre duas ativações sucessivas da tarefa não serão separadas por um período exatamente igual a T_i. Por exemplo, o limite superior (J_i) para o jitter relativo a duas sucessivas ativações quando $D_i = T_i$ é dada pela Equação 2.5. Quando $C_i << T_i$, o limite pode chegar a 200% e o *jitter* máximo se aproxima de duas vezes o período de amostragem.

$$J_i = 2\left(1 - \frac{C_i}{T_i}\right) \tag{2.5}$$

Considerando a situação ilustrada na Figura 2.3 e o *jitter* de amostragem $J_h = h^{max} - h^{min}$, onde: h^{max} é o maior intervalo entre duas ativações sucessivas da tarefa e h^{min} é o menor intervalo entre duas ativações sucessivas da tarefa. Temos que o tempo de execução atual (C_a) da tarefa varia entre ativações sucessivas. Entretanto, este tempo é sempre menor que o tempo de computação da tarefa (C_i). A execução da tarefa e a transmissão da saída são finalizadas antes do *deadline* D_i que é igual ao período T_i. A variação do tempo de execução da tarefa entre ativações causam o *jitter* de entrada-saída (J_{io}). Pode ser facilmente verificado através da Figura 2.3 que o intervalo de amostragem $h_{(k-1)}$ é diferente do intervalo de amostragem h_k. A amostragem na primeira ativação da tarefa, instante k-1, ocorre imediatamente após a liberação da tarefa (L_s= 0). Na segunda ativação da tarefa, instante k, a amostragem ocorre em

um instante próximo de expirar o *deadline* ($L_s \neq 0$). A variação de L_s causa o *jitter* de amostragem (J_s). O *jitter* de amostragem causa variação no valor do intervalo de amostragem e conseqüentemente o J_h. O valor de J_h na situação ilustrada na figura é igual à h_{k-1} - h_k. O maior intervalo de amostragem (h^{max}) é igual à h_{k-1} e ocorre entre as ativações k-1 e k. O menor intervalo de amostragem (h^{min}) é igual à h_k e ocorre entre as ativações k e k+1.

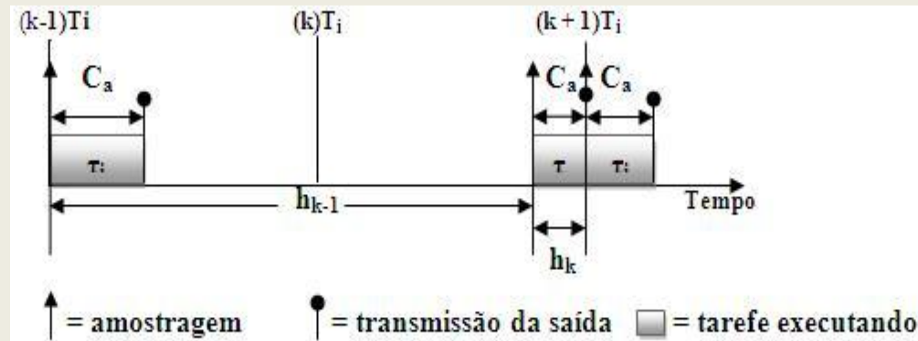


Figura 2.3: *Jitter* em sucessivas ativações de uma tarefa de controle periódica.

Uma possibilidade para reduzir o *jitter* no intervalo de amostragem proposta por Marti e Fuertes (2001), é utilizar uma abordagem de compensação ajustando os parâmetros do controlador em tempo de execução a cada ativação da tarefa de controle. Ao se realizar a compensação em tempo de execução, cada ativação da tarefa de controle pode ocorrer em qualquer instante de tempo dentro de um intervalo h_k.

O intervalo de variação do período de amostragem pode ser determinado através de uma análise *off-line* que inclui análise da estabilidade. Marti e Fuertes (2001) definem a seguinte regra para seleção do intervalo de amostragem h_k: $h^{min} \leq h_k \leq h^{max}$, onde h^{min} tem que ser maior que 5% do período equivalente a freqüência de corte do sistema e h^{max} não pode ser menor que 25% do período equivalente a freqüência de corte do sistema.

Outra possibilidade para tratar o atraso computacional proposta por Åström e Wittenmark (1997) e Cervin (2003) é escrever o código da tarefa de tal forma que os atrasos

sejam minimizados. Isso pode ser realizado separando os cálculos do algoritmo de controle em duas partes: *Calculate Output* e *Update State*.

2.3.3 Escalonamento de tarefas de controle

Nesta seção são apresentados os algoritmos de escalonamento comumente utilizados em SOTR, novos modelos de tarefas para controle em tempo real baseados na divisão da tarefa em sub-tarefas e os métodos para analisar a escalonabilidade dos algoritmos.

Prioridades Fixas (FP)

O algoritmo de prioridade fixa é o tipo de escalonamento mais comum entre os SOTR comerciais. Para utilizar este algoritmo, cada controlador digital é modelado como tarefa periódica τ_i. A cada tarefa periódica τ_i é associada um período fixo T_i, um *dedline* D_i, um tempo de execução de pior caso C_i e uma prioridade fixa P_i.

A Figura 2.4 ilustra execução do algoritmo para um conjunto de tarefas $\Gamma = \{\tau_1, \tau_2, \tau_3\}$ com a seguinte distribuição de prioridades P_Γ: $P_1 < P_2 < P_3$. De acordo com a Figura 2.4, quando várias tarefas estão prontas para serem executadas no mesmo instante de tempo, a tarefa com a maior prioridade ganha acesso ao processador. Se uma tarefa com prioridade maior que a prioridade da tarefa que sendo executada chega à fila de prontos, a tarefa que está sendo executada é preemptada pela tarefa de maior prioridade.

A atribuição de prioridades para o conjunto de tarefas pode ser realizada utilizando o RM (*rate monotonic*). Caso seja assumido que D_i seja igual T_i, o RM permite uma maneira ótima de atribuir prioridades para o conjunto de tarefas. A análise da escalonabilidade pode ser realizada utilizado o teste apresentado para o RM (Equação 2.1).

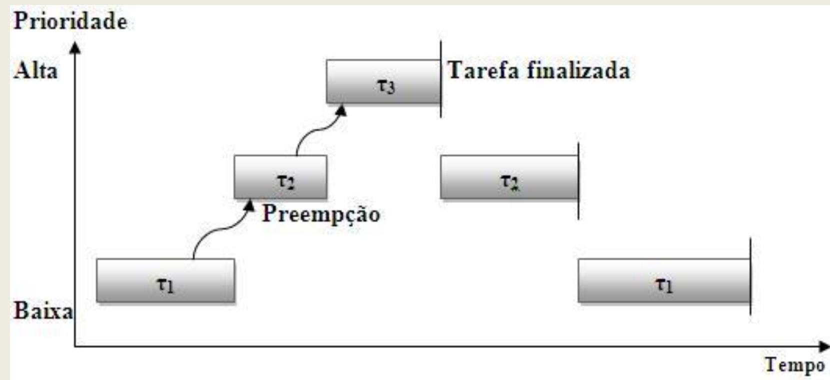


Figura 2.4: Execução do algoritmo FP para um conjunto de tarefas.

Escalonamento *Roudin Robin* (RR)

O *Round Robin* é uma extensão do algoritmo FP para casos em que existe mais de uma tarefa do conjunto de tarefas com a mesma prioridade. O algoritmo mantém CPU compartilhada entre as tarefas usando *time-slicing*. Cada tarefa em um grupo de tarefas com a mesma prioridade executa por um determinado *quantum* ou *time slice* antes de liberar a CPU para a próxima tarefa no grupo. Nenhuma das tarefas pode se apoderar do processador enquanto ele está bloqueado. A Figura 2.5 ilustra a execução do algoritmo para dois conjuntos de tarefas: $\Gamma_{1} = \{ \tau_1, \tau_2, \tau_3 \}$ e $\Gamma_{2} = \{ \tau_4 \}$ onde, a prioridade atribuida as tarefas do conjunto Γ_1 ($P_{\Gamma 1}$) é menor que a prioridade atribuida as tarefas do conjunto Γ_2 ($P_{\Gamma 2}$).

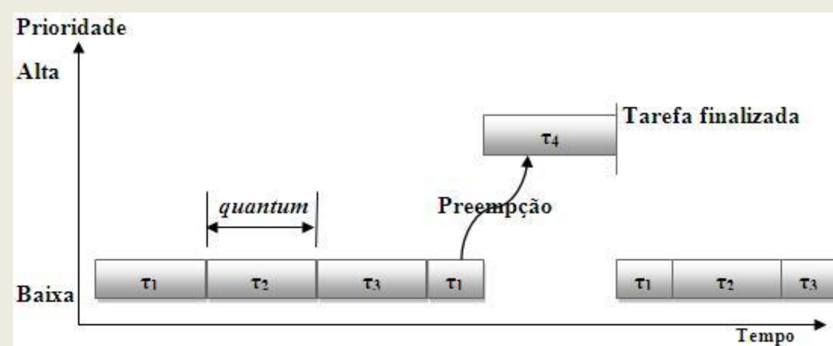


Figura 2.5: Conjuntos de tarefas periódicas escalonadas pelo algoritmo RR.

Em implementações tradicionais do RR presentes na maioria dos Sistemas Operacionais permite-se que uma tarefa seja executda até expirar seu *quantum*, independente de uma tarefa de prioridade mais alta ter chegado a uma das filas de Pronto. No exemplo mostrado na Figura 2.5, ilustra-se, entretanto o caso onde assim que o escalonador tem conhecimento da chegada de uma tarefa com maior prioridade na fila de Pronto, a tarefa de prioridade mais baixa é preemptada.

Neste trabalho, o RR será utilizado em conjunto com esquema de atribuição de prioridades RM. A análise de escalonabilidade será realizada através do teste proposto para o RM (Equação 2.1). Em (BRITO, R. & NAVET, N., 2003) é proposto um algoritmo para análise de escalonabilidade para o RR e um método de análise *off-line* para reduzir o custo do algoritmo proposto.

Escalonamento de Sub-tarefas (*Subtask scheduling*)

A possibilidade de decompor uma tarefa de controle em sub-tarefas foi identificado em (GERBER, R. & HONG, S., 1993). O objetivo era transformar um conjunto de tarefas não escalonáveis em um conjunto escalonável. Cada tarefa foi dividida em três sub-tarefas sendo, uma parte para *Calculate Output* (mantendo o período original) e duas partes para *Update State* (dobrando o período original e permitindo *deadline*s maiores que o período). A partir da proposta de Gerber e Hong (1993), vários autores apresentaram propostas de divisões das tarefas com diferentes objetivos.

Foi salientado em (BURNS, A.; TINDELL, K. & WELLINGS, A. J. , 1994), que o método utilizado por Gerber e Hong (1993) era desnecessário, complicado e não ideal com base numa análise mais exata do tempo de resposta da tarefa R_i. Além disso, era possível incluir o caso quando $D_i > T_i$ no teste do tempo de resposta.

Cervin (2003) propõe alterações no nível de programação das tarefas e no tipo de escalonamento através da divisão da tarefa de controle em sub-tarefas, como uma maneira de tratar o atraso computacional e reduzir as latências de amostragem e entrada e saída. A proposta consiste em separar os cálculos do algoritmo de controle em *Calculate Output* e *Update State* da seguinte forma:

Calculate Output**:** contém apenas as partes do algoritmo de controle que usam os dados amostrados no instante atual de ativação da tarefa;

Update State**:** atualiza os estados do controlador e executa os pré-cálculos que são necessários para minimizar o tempo de realização de *Calculate Output* na próxima ativação da tarefa.

A divisão da tarefa de controle em *Calculate Output* e *Update State* introduz as seguintes restrições temporais para as diferentes partes da tarefa:

1. A amostragem deve ser realizada no mesmo instante em cada período, para eliminar o *jitter* de amostragem.
2. *Calculate Output* deve começar e terminar o mais rapidamente possível, de modo que o atraso computacional seja minimizado.
3. A transmissão de saída deve ser realizada imediatamente após o *Calculate Output*, ou em um instante fixo após a amostragem de dados, dependendo de como o controlador foi projetado.
4. *Update State* deve terminar antes do início do próximo período, ou pelo menos antes da liberação de *Calculate Output*.

No nível de programação das tarefas os algoritmos mostrados nas Listagens 2.1. e 2.2, foram propostos para reduzir o *jitter* quando as tarefas são escalonadas pelo FP (*fixed priority*) ou pelo EDF (earliest deadline first). Em ambos os casos, os controladores podem ser implementados como uma, duas, três ou quatro tarefas distintas.

As variáveis P_{US} e P_{CO} mostrados na Listagem 2.1 são as prioridades para *Update State* e *Calculate Output*, h é período de ativação da tarefa. A prioridade atribuída à *Calculate Output*, que é a parte crítica, deve ser maior que a prioridade atribuída à *Update State* cujo único requisito e terminar antes do final do período. Dependendo do tipo de escalonamento que será utilizado para o conjunto de tarefas gerado, a análise da escalonabilidade pode ser realizada aplicando um dos testes propostos para os algoritmos RM (Equação 2.1) ou EDF (Equação 2.3).

Listagem 2.1: Implementação de *subtask scheduling* utilizando o escalonador FP.

```
t := CurrentTime;
SetPriority(P_CO);
Loop
    ReadInput;
    CalculateOutput;
    WriteOutput;
    SetPriority(P_US);
    UpdateState;
    t := t + h;
    SetPriority(P_CO);
    Delay Until (t);
End Loop;
```

Listagem 2.2: Implementação de *subtask scheduling* utilizando o escalonador EDF.

```
t := CurrentTime;
SetAbsDeadline(t + D_CO);
Loop
    ReadInput;
    CalculateOutput;
    WriteOutput;
```

```
    SetAbsDeadline(t + h);
    Delay Until(t + D_CO);
    UpdateState;
    t := t + h;
    SetAbsDeadline(t + D_CO);
    Delay Until(t);
End Loop;
```

Modelos de tarefas e atribuição de *deadlines*

Os modelos de tarefas escalonamento restrito a prioridades e escalonamento deslocado, foram propostos por Cervin (2003) na metodologia *subtask scheduling*. Em teoria, todos os modelos de tarefas propostos são aplicáveis aos escalonadores RM e EDF. Nestes modelos cada tarefa de controle consiste de duas sub-tarefas: τ_{CO} (*Calculate Output*) e τ_{US} (*Update State*). Os tempos de execução de pior caso das sub-tarefas são conhecidos e iguais a C_{CO} e C_{US} respectivamente. Para simplificar os tempos entre as conversões A/D e D/A foram negligenciados.

Escalonamento restrito a prioridades

Neste modelo é assumido que as tarefas são executadas em seqüência usando restrições de prioridades como ilustrado na Figura 2.6. No modelo as partes das sub-tarefas são liberadas no início do período de ativação. Para garantir que a ordem de execução esteja correta, a prioridade de τ_{CO} deve ser maior que τ_{US}.

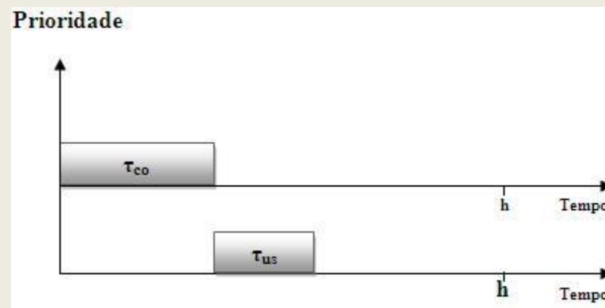


Figura 2.6: Escalonamento de sub-tarefas restritas a prioridades.

A atribuição de *deadline*s para as sub-tarefas seguem as seguintes restrições:

Tarefa τ_{CO} com *deadline* D_{CO}: $C_{CO} \leq D_{CO} \leq h - C_{US}$;

Tarefa τ_{US} com *deadline* D_{US}: $D_{US} = h$;

Escalonamento deslocado

Neste modelo é assumido que τ_{US} é liberada com deslocamento fixo Φ_{US} em relação à liberação de τ_{CO}, como ilustrado na Figura 2.7. As prioridades e o deslocamento devem ser selecionados de forma a garantir que τ_{CO} seja finalizada antes da execução τ_{US}.

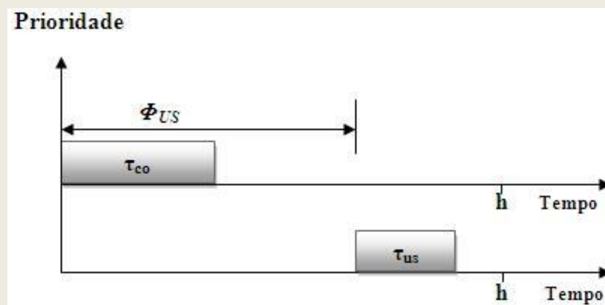


Figura 2.7: Escalonamento de sub-tarefas usando deslocamento.

A atribuição de *deadline*s para as sub-tarefas seguem as seguintes restrições:

Tarefa τ_{CO} com *deadline* D_{CO}: $C_{CO} \leq D_{CO} \leq h - C_{US}$ e $D_{CO} \leq \Phi_{US} \leq h - C_{US}$;

Tarefa τ_{US} com *deadline* D_{US}: $D_{US} = h - \Phi_{US}$;

2.4 Análises do WCET

Analisar o *Worst Case Computation Time* (WCET) das tarefas é uma atividade importante quando se constrói sistemas de tempo real (LUNDQVIST, T. & SANDIN, P., 2008). Essencialmente todos os algoritmos de escalonamento para tempo real são baseados em estimativas do WCET das tarefas a serem escalonadas.

Uma predição de WCET deve fornecer uma previsão que seja o limite superior do WCET real, ou seja, as estimativas do WCET devem ser próximas, mas não inferior ao seu valor real (ÅRZÉN, K. ET AL., 1999). Delimitar o tempo de execução de pior caso é um problema duplo, onde podem ser realizados dois tipos de análises:

1. análises no nível de estrutura para determinar o tempo de execução do caminho mais longo possível do programa, ou parte dele, com base no tempo de execução das partes programa;
2. análises no nível de *hardware* para determinar o WCET de um pedaço de código em uma plataforma específica em ciclos ou nanosegundos.

Análises no nível de estrutura

Shaw (1989) apresenta um esquema para raciocinar com e sobre o tempo especificando as propriedades de tempo em programas concorrentes. Os objetivos são predizer o comportamento temporal de programas escritos em linguagens de alto nível e provar que os mesmos atendem as restrições de tempo através da análise direta de declarações nos programas. Duas idéias foram desenvolvidas por Shaw (1989):

1. consiste em derivar os limites superiores e inferiores dos tempos de execução das instruções baseando se nos limites das declarações primitivas e elementos da linguagem e do sistema subjacente;

2. estende a lógica de Hoare para incluir os efeitos das atualizações em tempo real depois de cada execução da instrução;

Na abordagem proposta por Shaw (1989), uma previsão de WCET é associado a cada “bloco atômico”. Um bloco atômico é essencialmente qualquer pedaço de código fonte seqüencial. Uma vez que, as previsões de tempo sejam produzidos para os blocos atômicos em um programa, as previsões de tempo podem ser calculados para construções compostas utilizando os seus constituintes. Por exemplo, o tempo de execução da instrução de atribuição

S: c = a * b;

deve ser computado da seguinte maneira:

T(S) = T(b) + T(c) + T(*) + T(a) + T(=);

onde, T(X) denota o tempo de execução de pior caso do nó X em uma árvore de sintaxe abstrata. O valor de T(S) é a soma dos tempos de execução dos cinco blocos atômicos que compõem a instrução.

O esquema proposto por Shaw (1989) foi a base para o desenvolvimento de uma ferramenta experimental para determinar tempo de execução de programas escritos em um subconjunto da linguagem C mostrada em (PARK, C. Y & SHAW, A. C., 1990). Entretanto, no trabalho o esquema de tempo foi usado apenas como uma “metodologia de raciocínio para tempo deterministico” e não como uma implementação real.

Um importante subproblema em análise no nível de estrutura é determinar os limites para as iterações do *loop* e a profundidade das recursões. Neste caso ou o programador fornece anotações explícitas sobre *loops* e chamadas recursivas, ou a análise de alguma forma calcula automaticamente estes limites (ÅRZÉN, K. ET AL., 1999).

De acordo com Lundiqvist e Stenstrom (1998), é possível utilizar execução simbólica como uma maneira de determinar automaticamente alguns limites para *loops* e profundidade

das recursões. Execução simbólica corresponde a executar o programa com as variáveis habituais que permitem presumir um valor desconhecido. Em um programa com entrada que assuma um valor desconhecido, qualquer cálculo que envolve um valor desconhecido resulta em um valor desconhecido. Entretanto, nas instruções de desvio de fluxo de controle que dependem de um valor desconhecido, todos os caminhos serão executados.

A principal desvantagem da execução simbólica é que ela pode nunca terminar, ainda que o programa analisado tem uma WCET limitado.

Surprendetemente, pouca pesquisa tem sido publicada na área de predição de tempo deterministico (PARK, C. Y & SHAW, A. C., 1990). Geralmente, na prática tem sido utilizado um método padrão que consiste em "rodar" o programa para uma entrada de dados representativa e medir o tempo de execução do mesmo. Embora esta abordagem seja bastante útil, ela apresenta algumas falhas como de depuração, pois os dados de teste podem não cobrir todo o dominio de interesse e as medidas podem não ser realisticas caso não seja criado um ambiente de produção real. Outra técnica muito utilizada é simular o sistema *target* usando algumas especificações ou modelos de linguagens. O problema neste caso é que os resultados podem ser um tanto imprecisos, pois a simulação necessariamente assume um modelo aproximado do sistema real.

Análises no nível de *hardware*

Processadores modernos utilizam uma variedade de técnicas para acelerar o tempo de execução (PETTERS, S. M. & FARBER, G., 1999). O foco principal é diminuir o gargalo existente entre a velocidade do processador e a memória principal. Inicialmente foi adicionado uma mémoria cache externa e atualmente existe cahe o *on-chip* e em vários níveis. Outras alterações visando melhorar o desempenho dos processadores são: *pipelines* adicionais, múltiplas unidades de execução execução especulativa, etc. Embora estas técnicas

melhorem o desempenho do caso médio, as mesmas tornam consideravelmente mais difícil prever o desempenho de pior caso. Supondo que o pior caso ocorra para cada instrução do processador, pode ocorrer uma superestimativa do WCET por um fator de 10-100 vezes (ÅRZÉN, K. ET AL., 1999). A programação baseada neste cenário pessimista reduz efetivamente o desempenho dos processadores modernos.

Técnicas de análise no nível de *hardware* operam sobre a representação do programa em código objeto. Uma abordagem é a análise iterativa do fluxo de dados em baixo nível empregada por White et al. (1997), abordando os efeitos da *cache* de dados, *cache* de instrução e *pipeline*. Segundo Lundiqvist e Stenstrom (1998), a utilização da execução simbólica para a análise no nível de estrutura, em alguns casos, podem ser usados para prever a performance do *hardware*.

O esquema de abordagem de tempo original proposto por Shaw (1989), não lida com aspectos da análise do WCET no nível de *hardware*. Uma extensão do esquema de tempo proposta em (LI ET AL., 1995) é projetado para tratar os aspectos de *hardware* considerando que as estimativas dos WCETs dos blocos atómicos são representados por uma abstração do tempo de pior caso WCTA (*Worst Case Timing Abstraction*).

2.5 Redes de computadores

Uma rede de comunicação é formada por um conjunto de módulos processadores autônomos (SOARES, L. F. ; LEMOS, G. & COLCHER, S., 1995). Estes módulos são capazes de trocar informações e compartilhar recursos. As redes de comunicação foram introduzidas nos sistemas de controle digital em meados da década de 1970 movidos pelo crescimento da indústria automobilística. Entre os principais motivos para a introdução de redes de comunicação em sistemas de controle podemos citar: redução dos custos com cabeamento,

modularização do sistema e flexibilidade na configuração do sistema. A partir de então, vários tipos de redes de comunicação têm sido desenvolvidas.

Ethernet

Ethernet é uma das mais utilizadas tecnologias de rede locais (LANs) que surgiram em institutos de pesquisa e universidades. Uma rede Ethernet é capaz de transmitir dados com velocidades entre 10 Mbit/s a 100 Mbit /s. Este tipo de rede não é destinado a comunicação em tempo real. No entanto, devido o grande número de redes Ethernets instaladas, elas tem-se tornado atraente para uso em sistemas de controle em tempo real. Nestas redes não existe um controlador central do barramento, em vez disso as redes Ethernet utilizam um método chamado CSMA/CD para que cada estação ou nó possa acessar o barramento. Isto significa que antes de enviar um pacote para a rede, cada nó escuta o canal. Quando o canal de transmissão estiver ocioso o nó inicia a transmissão e permanece escutando o canal. Se uma colisão é detectada a tradição e abortada e o nó espera por um intervalo de tempo aleatório para tentar retransmitir. Está espera aleatória faz com que as redes Ethernets não sejam muito apropriadas para aplicações em sistemas de controle, devido às características dos intervalos de amostragem dos controladores serem determinísticos.

Um número quase ilimitado de estações pode ser conectado a uma rede Ethernet. Este número de estações é limitado pelo endereço de seis bytes. Os três primeiros bytes do endereço são usados para fornecer a identificação, e os últimos três bytes são definidos pelo vendedor de modo que cada interface Ethernet tem um endereço exclusivo. Um quadro ou pacote Ethernet possui entre 64 e 1.500 bytes de comprimento.

CAN (Controller Area Network)

CAN foi desenvolvido pela empresa alemã Bosch para a automação industrial e foi um dos primeiro *Fieldbus*. Atualmente é usada em carros de diversos fabricantes. CAN é definido pelas normas ISO 11898 e 11519-1. A velocidade de transferência do barramento pode ser configurada através dos registradores do controlador de CAN. Está velocidade pode chegar 1 Mbit/s se o comprimento do barramento não for superior à 50 metros. Para barramentos superiores a 50 metros a velocidade pode ser de até 500 Kbit/s. Quando a qualidade do cabeamento é baixa, como em carros populares, a velocidade máxima de transferência pode ser ainda menor. Não existe limite no número de nós em uma rede CAN, sendo que qualquer nó da rede pode iniciar uma transmissão a qualquer momento se o barramento está em ocioso. Se vários nós estão tentando transmitir é iniciado uma arbitração e o nó que estiver tentando enviar a mensagem com maior prioridade terá o direito de transmissão.

2.6 Trabalhos relacionados

A integração entre controle e tempo real tem despertado o interesse da comunidade científica nos últimos anos. Neste contexto diversas metodologias têm sido propostas visando resolver os problemas de atrasos e *jitters* tanto nos sistemas de controle centralizado, quanto nos sistemas de controle distribuído. O objetivo principal das metodologias propostas é tornar o ambiente computacional o mais determinístico possível, de tal maneira que, o desempenho dos sistemas de controle não seja afetado pelos atrasos e *jitters*.

O estado da arte no campo do controle e escalonamento é apresentado em (ÅRZÉN, K. ET AL., 1999). Cervin (2003) propõe que o algoritmo de controle seja dividido em duas partes com o objetivo de minimizar o atraso computacional e a latência de entrada e saída.

Outra proposta de Cervin (2003) é a utilização do conceito de *co-design* entre o sistema de controle e algoritmo de escalonamento para aproximar as áreas de controle e computação.

Princípios de controle e escalonamento são combinados para projetar controladores com restrições temporais mais flexíveis em (MARTI, 2002). Marti e Fuertes (2001) propõem um método para compensar o *jitter* de amostragem e os atrasos entre a amostragem e atuação.

Modelos para atrasos em sistemas de controle distribuído em redes foram propostos em (NILSSON, 1998).

3 Arquiteturas de *Frameworks* para RCP

No contexto da Engenharia de *Software*, diferentes abordagens têm sido utilizadas para melhorar a qualidade dos artefatos de *software*, bem como diminuir o tempo e o esforço necessários para produzi-los (SILVA, 2000). Entre estas abordagens podemos citar: utilização de *frameworks* e a utilização de componentes orientados a objetos.

Frameworks são estruturas de classes inter-relacionadas que correspondem a uma implementação incompleta para um conjunto de aplicação de um determinado domínio (JOHNSON, 1992). Esta estrutura de classes deve ser adaptada para a produção de aplicações específicas. A grande vantagem da utilização de *frameworks* é o reuso de código e de projeto, que pode diminuir o tempo e o esforço exigidos na produção de *software*. Em contrapartida, *frameworks* são complexos para se desenvolver e difíceis de utilizar. A abordagem de *frameworks* pode se valer de padrões de projeto para a obtenção de estruturas de classes bem organizadas e mais aptas a modificações e extensões.

Um componente é uma unidade de composição com interfaces contratualmente especificadas e dependências de contexto explícitas (SZYPERSKI, 1996). Os componentes podem ser desenvolvidos utilizando o paradigma de orientação a objetos como estruturas de classes inter-relacionadas, semelhantes aos *frameworks*, com visibilidade externa limitada (KRAJNC, 1997). Porém, no caso geral, componentes não dependem de tecnologia de implementação. Assim, qualquer artefato de *software* pode ser considerado um componente, desde que possua uma interface definida (SZYPERSKI, 1996).

No desenvolvimento orientado a componentes pretende-se organizar a estrutura de um *software* como uma interligação de artefatos de *software* independentes isto é, os componentes (SILVA, 2000). O reuso de componentes previamente desenvolvidos, como no

caso dos *frameworks*, permite a redução de tempo e esforço para a obtenção de um *software*. Por outro lado, devido a deficiências das formas correntes de descrição de componentes, é complexo avaliar a adequação de um componente existente a uma situação específica. Também é complexo adaptar componentes quando estes não são adequados às necessidades específicas.

Um fator relevante em termos de aumento de produtividade do desenvolvimento de *software* é a granularidade do artefato de *software* reutilizado. Para Meyer (1988), por exemplo, o uso de bibliotecas de funções por um programador que usa linguagem C se classifica como reutilização de rotinas. A reutilização no nível de módulo corresponde a um nível de granularidade superior à reutilização de rotinas. Segundo Meyer (1988), a reutilização de classes em orientação a objetos corresponde à reutilização de módulo. Quando uma classe é reutilizada, um conjunto de rotinas que são os métodos da classe também é reutilizado, bem como a estrutura de dados que são os atributos da classe.

Meyer (1988) enfatiza a necessidade da disponibilidade de artefatos genéricos e afirma que: as classes de linguagens orientadas a objetos podem ser vistas tanto como módulos de projeto, quanto como módulos de implementação.

A reutilização de classes pode ser realizada através dos mecanismos de composição e herança. Na composição as classes disponíveis em bibliotecas são utilizadas para originar os objetos da implementação. Na herança as classes disponíveis em bibliotecas são aproveitadas no desenvolvimento de novas classes. A reutilização promovida por *frameworks* situa num patamar de granularidade superior à reutilização de classes, por reusar classes interligadas, ao invés de classes isoladas. Esta reutilização de código e projeto em um nível de granularidade superior às outras abordagens de reutilização citadas, confere aos *frameworks* o potencial de contribuir mais significativamente com o aumento de produtividade no desenvolvimento de *software*.

O desenvolvimento de *software* orientado a componentes é outra abordagem que promove o reuso de artefatos de alta granularidade, com a capacidade de promover reuso de projeto e de código. Nesta abordagem uma aplicação é constituída a partir de um conjunto de módulos (componentes) interligados. Na medida em que parte das responsabilidades de uma aplicação seja delegada a um componente reutilizado, estar-se-á reduzindo o esforço necessário para produzir a aplicação caracterizando o aumento de produtividade semelhante ao que ocorre com os *frameworks*.

Observa-se em comum nas abordagens de desenvolvimento baseadas em *frameworks* e em componentes a possibilidade de reutilização de artefatos de *software* de alta granularidade e a caracterização de reuso de projeto e de código (SILVA, 2000).

3.1 Modelagem do *Framework* para Controle Centralizado (FCC)

Uma representação para um sistema controlado por computador é ilustrada na Figura 3.1. Nesta representação as principais partes da malha de controle (amostragem, atuação e computação do algoritmo de controle) estão fechadas em um único nó de processamento. Caracterizando um Sistema de Controle Centralizado (SCC).

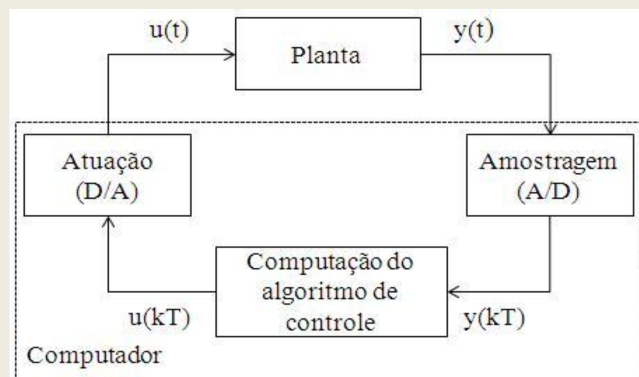


Figura 3.1: Representação de um SCC com controlador discreto e planta contínua.

De acordo com a Figura 3.1, a saída da planta ou processo é um sinal contínuo y(t). Esta saída é amostrada por sensores e convertida para um sinal digital pelo conversor analógico digital (A/D). O conversor A/D pode ser incluído no computador ou considerado como uma unidade separada. A conversão ocorre periodicamente nos instantes kT. O controlador recebe como entrada o sinal amostrado da planta e convertido para a forma digital y(kT) e gera uma seqüência u(kT) que é convertida para a forma analógica por um conversor digital analógico (D/A), gerando o sinal de atuação u(t). Entre os intervalos de amostragem o sistema fica em malha aberta. O sinal gerado pelo controlador é mantido constante por um segurador, em geral, um segurador de ordem zero, durante o intervalo entre a amostragem e atuação.

Os elementos do sistema de controle mostrado na Figura 3.1, possuem dados no domínio contínuo e discreto. O controlador, o sinal de entrada y(kT) e o sinal de saída u(kT) do controlador estão no domínio discreto. A planta, o sinal u(t) e o sinal de saída y(t) da planta estão no domínio continuo.

A análise e o desenvolvimento de um controlador para o sistema mostrado na Figura 3.1, pode ser realizada tanto no domínio discreto quanto no domínio contínuo. Uma maneira de simplificar a análise do sistema é converter os dados do mesmo para um único domínio (discreto ou continuo). A Figura 3.2 mostra a representação do sistema em um computador digital após a conversão para o domínio discreto.

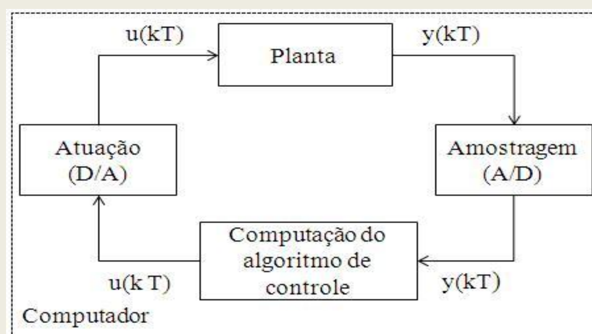


Figura 3.2: Representação de um SCC com planta e controlador no domínio discreto.

Utilizando a representação do sistema mostrado na Figura 3.2 com o controlador, a planta, o sensor, o atuador e os sinais de entrada do controlador e da planta no domínio discreto é possível modelar e implementar em uma plataforma de tempo real um *framework* para análises e testes de sistemas de controle centralizado.

3.1.1 Arquitetura do FCC

A arquitetura de um sistema define sua organização, em termos de componentes e suas possíveis interconexões e interações, bem como suas propriedades fundamentais (JONG, 1997).

No Capítulo 1, foram apresentados diferentes processos de desenvolvimento de sistemas de controle. O primeiro processo de desenvolvimento (Figura 1.1), o departamento de *software* recebe a especificação do sistema de controle em uma “caixa preta”, implementa e testa o sistema de controle em uma plataforma de tempo real. A implementação do sistema de controle é realizada assumindo tarefas periódicas. As tarefas de controle executam periodicamente o código referente a equação de diferenças característica do controlador, acessam as rotinas da API (*Application Programming Interface*) do sistema e os *drivers* de dispositivos. Após a implementação do sistema de controle, são realizados os testes unitário, estrutural e funcional. Caso os resultados dos testes não sejam satisfatórios, é preciso realizar uma nova análise, alterar o algoritmo de controle, mudar algum dos requisitos do sistema ou redimensionar a plataforma computacional. Neste caso, o projeto retorna ao departamento de controle que refaz a projeto do controlador.

Um outro processo de desenvolvimento, ilustrado na Figura 1.5, é uma solução que utiliza a metodologia RCP. No processo utilizando RCP, todo o projeto pode ser realizado pelo departamento de controle e a codificação é gerada de forma automática. Foi enfatizado

que o processo RCP apresenta os seguintes problemas: dificuldade de integração do código gerado com código legado, dificuldade em transformar o protótipo do controlador gerado no estágio de RCP em um executável para o *target*, ineficiência em aproveitar recursos computacionais, dificuldade de integrar durante o processo de desenvolvimento novos métodos ou metodologias para tratar os *jitters* e atrasos que são as principais causas da degradação do desempenho da malha de controle em plataformas de tempo real.

Os problemas apresentados em ambos os processos, são resultados da distância existentes entres as áreas de controle e computação, envolvidas no desenvolvimento do sistema de controle em plataformas de tempo real. Como forma de aproximar as duas áreas, este trabalho propõe a utilizar *frameworks* orientados a objetos nos estágios da metodologia RCP para substituir a geração automática de código.

A principal motivação para utilizar *frameworks* no processo RCP, é apresentar uma maneira estruturada e organizada de substituir os gerados de código automático possibilitando o reaproveitamento dos projetos e artefatos de *software* gerados durante o processo RCP.

Os *frameworks* podem ser embarcados no *target* real durante a prototipagem facilitando a integração com códigos legados, melhorando aproveitamento dos recursos computacionais e excluindo a necessidade de conversão do protótipo gerado no estágio de prototipagem em um executável para o *target*.

A proposta de arquitetura de *framework* RCP para sistemas de controle centralizado é mostrada na Figura 3.3. A arquitetura está organizada em quatro camadas hierárquicas sendo que, cada camada possui um ou mais módulos. O diagrama dos módulos e as dependências são mostrados na Figura 3.4. Cada módulo da arquitetura é composto por pacotes e ou outros módulos. As camadas **SOTR** e ***Hardware***, mostradas no modelo são parte da arquitetura proposta para plataforma RCP descrita no Capítulo 4.

Figura 3.3: Arquitetura hierárquica para FCC.

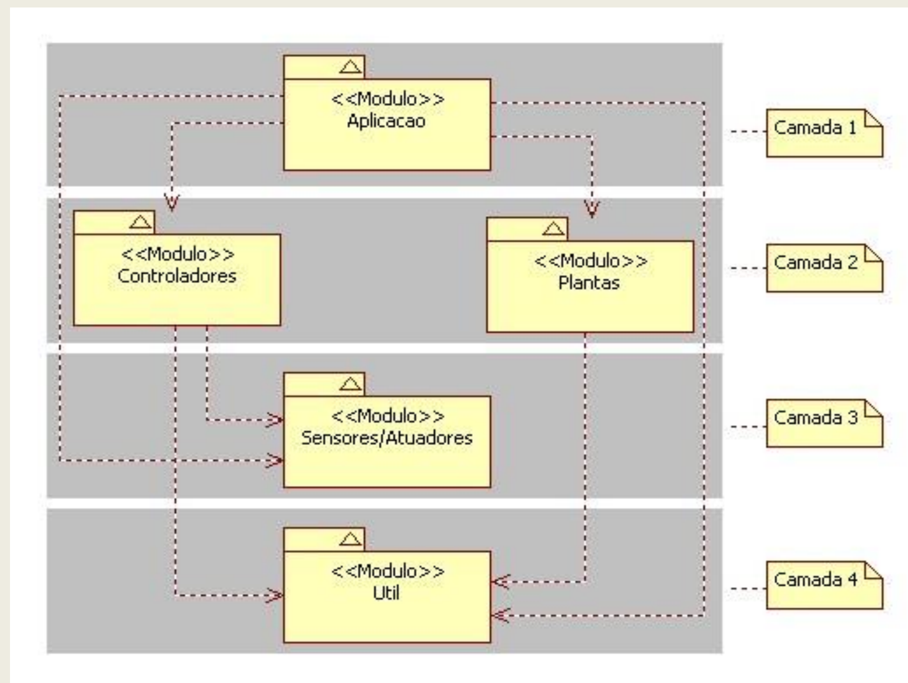


Figura 3.4: Diagrama de módulos e suas dependências na arquitetura do FCC.

Os componentes do SCC: Controladores, Plantas, Atuadores, Sensores e os sinais u(kT) e y(kT) estão encapsulados nos pacotes que fazem partes dos módulos que formam as camadas da arquitetura do *framework*. Durante o processo de desenvolvimento de novos componentes no *framework*, a hierarquia entre as camadas da arquitetura é mantida ao se utilizar apenas componentes das camadas inferiores e ou componentes da mesma camada.

3.1.2 Módulos do FCC

Esta seção descreve os componentes presentes em cada um dos módulos da arquitetura do *framework*. Os componentes e suas relações estão representados por diagramas de *Real Time* UML (DOUGLAS, 2000).

Controladores

Este módulo é composto pelo modelo de controlador mostrado no diagrama de classes da Figura 3.5.

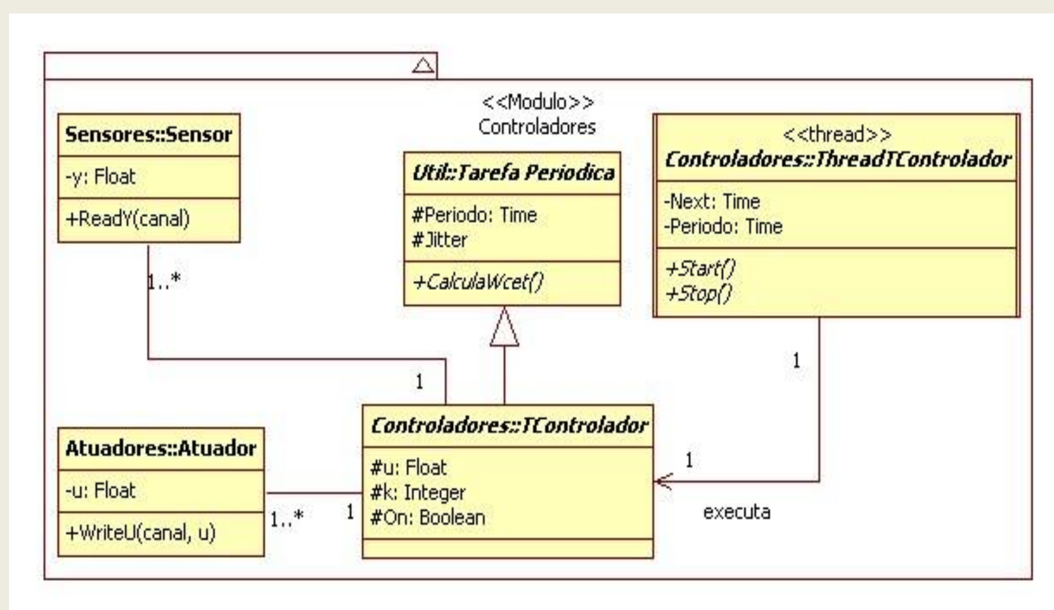


Figura 3.5: Diagrama de classes do módulo Controladores.

A classe *TControlador*[4] é uma classe abstrata que herda os métodos e atributos da interface *Tarefa Periodica*, mas não implementa nenhum método. A *thread ThreadTcontrolador* é uma abstração de tarefa do sistema operacional responsável pela execução do código do controlador durante os períodos de ativação.

[4]Nomes de classes abstratas em itálico (ferramenta de modelagem Star-UML).

Plantas

Este módulo é composto pelo modelo de planta mostrado no diagrama da Figura 3.6.

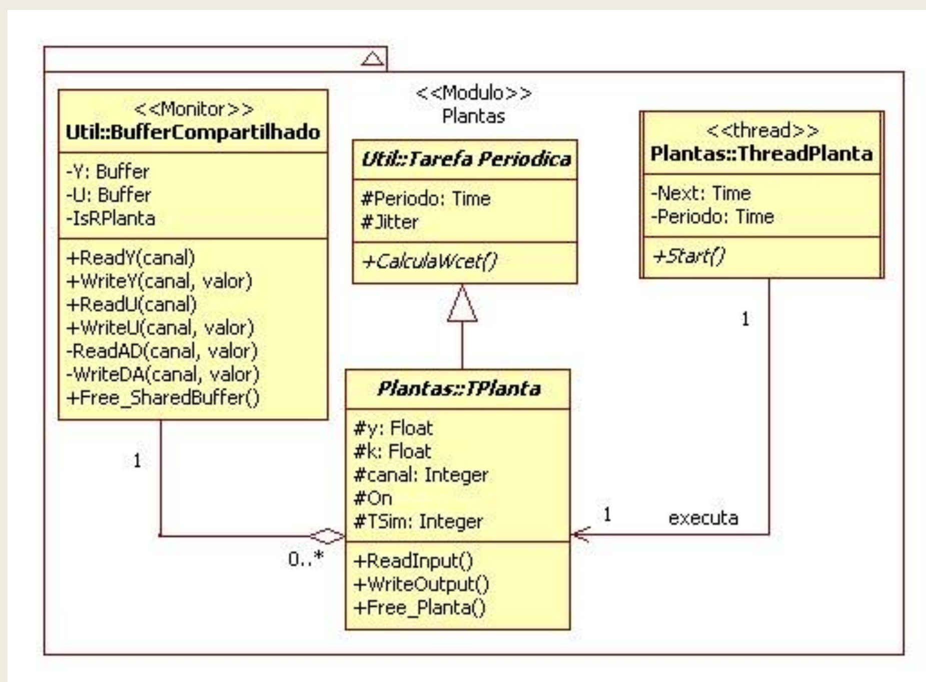


Figura 3.6: Diagrama de classes do módulo Plantas.

A classe abstrata *TPlanta* herda os métodos e atributos da interface *Tarefa Periodica* e agrega um objeto do tipo monitor (Buffer Compartilhado). No monitor estão os buffers de dados compartilhados entre as tarefas de controle e as plantas, os métodos para leitura e escrita dos dados contidos nos buffers e os métodos para leitura dos conversores A/D e escrita nos conversores D/A. A *thread ThreadPlanta* é uma abstração de tarefa do sistema operacional responsável pela execução do código da planta durante os períodos de ativação.

Sensores/Atuadores

Este módulo é formado pelo modelo de sensor e pelo modelo de atuador mostrados no diagrama de classes da Figura 3.7.

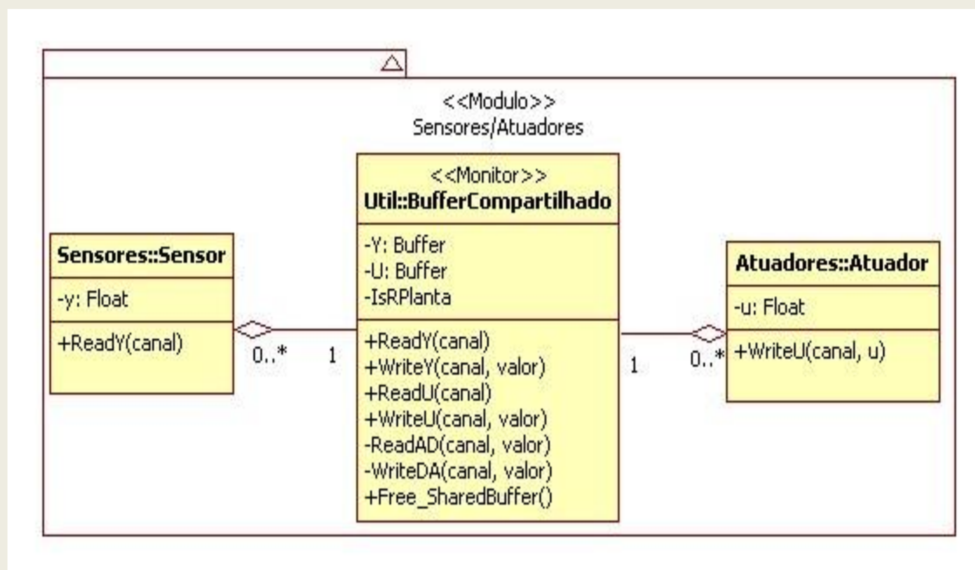


Figura 3.7: Diagrama de classes do módulo Sensores/Atuadores.

As classes Sensor e Atuador fazem parte do modelo de controlador mostrado no diagrama de classes da Figura 3.5. Estas classes agregam um objeto do tipo monitor (Buffer Compartilhado) que é utilizado como canal de comunicação pelas instâncias das tarefas do modelo de planta e pelas plantas reais com os controladores nas aplicações.

Aplicação

Este módulo possui um modelo de aplicação para desenvolvimento de um SCC mostrado no diagrama da Figura 3.8. No modelo de aplicação mostrado no diagrama os objetos tp, tc, s, a, sb, ThP, ThC são instanciados na tarefa principal (*thread Main*). A instanciação dos objetos é realizada através da chamada dos respectivos construtores das classes e *threads*. A chamada do construtor das classes e *threads* retorna uma referência ou ponteiro para a instância do objeto. As referências destes objetos são visíveis para a tarefa principal da aplicação permitindo esta tarefa tenha controle sobre as demais *threads* criadas durante a execução da aplicação.

O controle da execução das tarefas na camada de aplicação foi adotado para que, a tarefa principal funcione como um observador, ou seja: durante uma simulação do sistema de controle a tarefa principal verifica se a *ThreadPlanta* instanciada ainda está executando. Caso a resposta seja negativa, ou seja, se o tempo de simulação da planta (TSim) expirou, a tarefa principal finaliza a execução da tarefa de controle referente à planta e imprime os resultados amostrados na saída da planta. Caso existam várias plantas e controladores, a tarefa principal aguarda o término de todas as plantas para finalizar os controladores e imprimir as saídas.

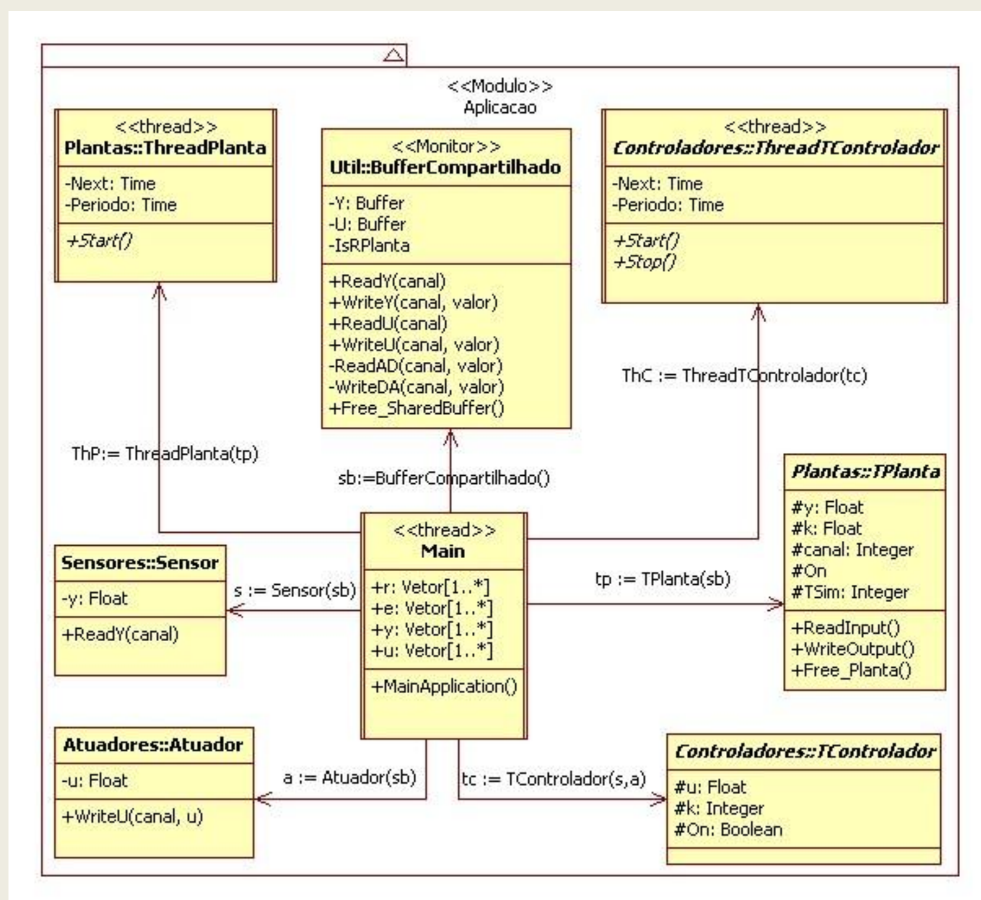


Figura 3.8: Diagrama de classes do módulo aplicação.

Os outros elementos que aparecem no diagrama de classes mostrado na figura são os vetores r, e, y e u que são utilizados para armazenar os valores do sinal de referência, erro, saída da planta e o sinal de controle calculado.

Em relação ao modelo de aplicação mostrado no diagrama da Figura 3.8, são realizadas as seguintes considerações:

1. As tarefas “controladores e as “plantas” são independentes, ou seja, as ativações de tarefa de controle nos instantes kT não dependem das ativações da planta nos mesmos instantes. Desta maneira, a malha de controle permanece aberta entre os instantes de amostragem;
2. As tarefas “plantas” e “controladores” podem ser preemptadas pelas tarefas obrigatórias para o funcionamento do SOTR;
3. Os pares de tarefas “plantas” e “controladores” possuem a mesma prioridade. Assim, não haverá preempção entre as “plantas” e “controladores” durante a simulação;

Útil

Este módulo é utilizado para organizar componentes que serão utilizados por outros módulos ou que não tenha uma relação bem definida com os módulos existentes.

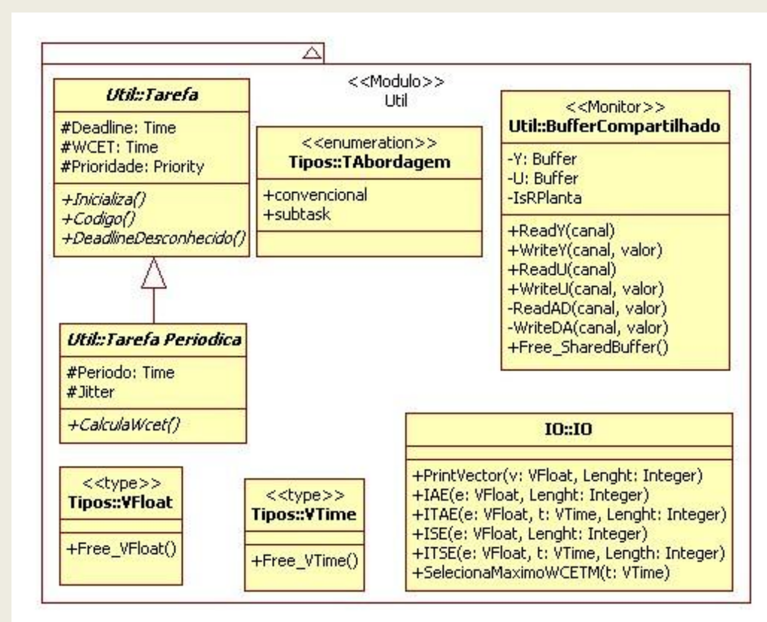


Figura 3.9: Diagramas de classes do módulo Útil.

3.2 Modelagem do *Framework* para Controle Distribuído (FCD)

Uma representação de sistema de controle distribuído (SCD) é mostrada na Figura 3.10. Nesta representação as partes principais da malha de controle (amostragem, computação do algoritmo de controle e atuação) estão fechadas através de uma rede de comunicação.

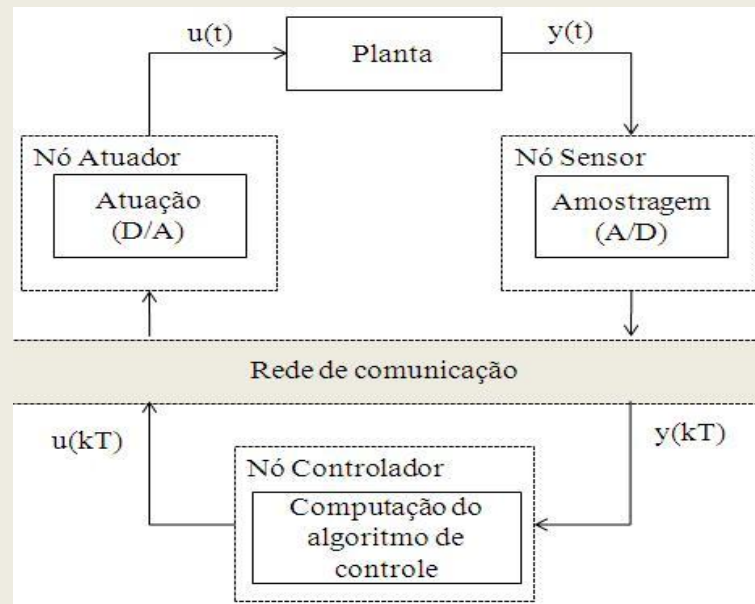


Figura 3.10: Representação de um SCD com planta contínua e controlador discreto.

De acordo com a figura, a saída contínua y(t) da planta é amostrada pelo sensor, e convertida para uma sinal digital y(kT) por um conversor A/D. O sinal convertido é enviado pelo Nó Sensor para o Nó Controlador. O cálculo do algoritmo de controle é realizado utilizando no Nó Controlador, onde a entrada y(kT) é recebida através de uma interface de rede. Após o cálculo do esforço de controle, a saída do controlador u (kT) é enviada para o Nó Atuador através da rede. O sinal de controle u (kT) recebido no Nó Atuador é convertido para um sinal digital por um conversor D/A gerando o sinal de atuação continuo u(t).

O processo descrito acima se repete a cada intervalo de amostragem kT. Sendo que, entre os intervalos de amostragem, o sistema permanece em malha aberta e o sinal de controle

é mantido constante durante este intervalo, em geral por um segurador de ordem zero implementado no conversor A/D presente no Nó Atuador.

Para desenvolver o modelo de *framework* para controle distribuído, foi realizado o mesmo procedimento utilizado para o FCC, ou seja, todos os elementos do sistema de controle mostrado na Figura 3.10 foram transformados para o domínio discreto. O resultado desta transformação é ilustrado na Figura 3.11.

O modelo de sistemas de controle distribuído (SCD) em rede mostrado na Figura 3.11, é baseado na estrutura de controle direto proposta em (TIPSUWAN, Y. & MON-YUEN, C., 2003). A estrutura de controle direto é composta por um controlador e um sistema remoto contendo a planta real, sensores e atuadores. O controlador e a planta estão fisicamente em locais diferentes e são ligados através de uma rede de comunicação. Durante o funcionamento do sistema, o sinal de controle é encapsulado em um *frame* ou pacote e enviado para a planta através da rede. A saída da planta é retornada para o controlador encapsulando as medidas do sensor dentro de um *frame* ou pacote. Na prática, múltiplos controladores podem ser implementados em um mesmo Nó Controlador para controlar múltiplas plantas.

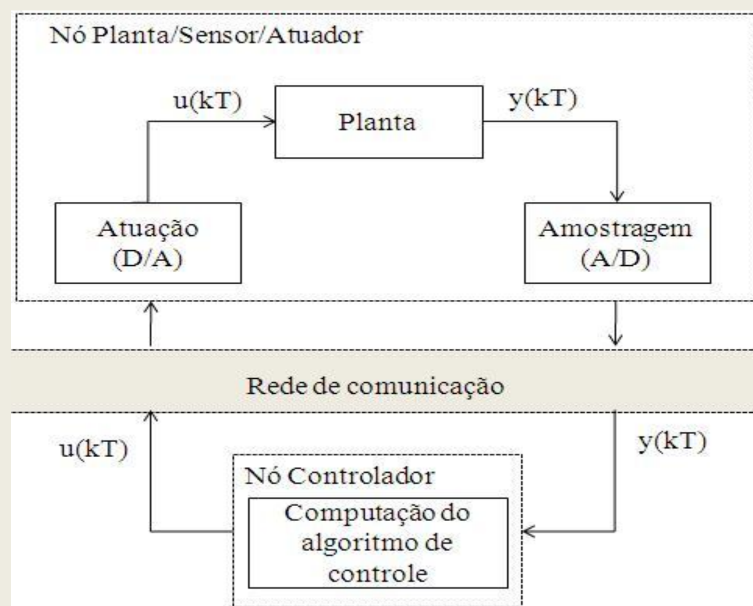


Figura 3.11: Representação de um SCD com planta e controlador no domínio discreto.

3.2.1 Arquitetura do FCD

A modelagem da arquitetura do *framework* RCP para sistemas de controle distribuído (FCD) foi realizada identificando os componentes do sistema. Após a identificação, os componentes foram organizados nos módulos que formam as camadas hierárquicas da arquitetura do FCD.

Identificação dos elementos do sistema

Como o sistema está distribuído em dois tipos de nós, foram identificados os componentes de cada nó.

Componentes do Nó Controlador: Controladores e os canais de comunicação entre os Controladores e os Sensores e entre os Controladores e os Atuadores.

Componentes do Nó Planta/Sensor/Atuador: Plantas, Sensores, Atuadores e os canais de comunicação entre os Sensores e os Controladores e entre Atuadores e o Controladores.

A proposta de arquitetura de *framework* RCP para sistemas de controle distribuído em redes é ilustrada na Figura 3.12. Os módulos e suas dependências são mostrados no diagrama da Figura 3.13. A estrutura hierárquica da arquitetura é a mesma proposta para a arquitetura do FCC, com exceção do módulo de comunicação adicionado à camada 4. O processo de desenvolvimento de componentes segue o mesmo critério descrito para a arquitetura do FCC.

Figura 3.12: Modelo de arquitetura hierárquica para FCD.

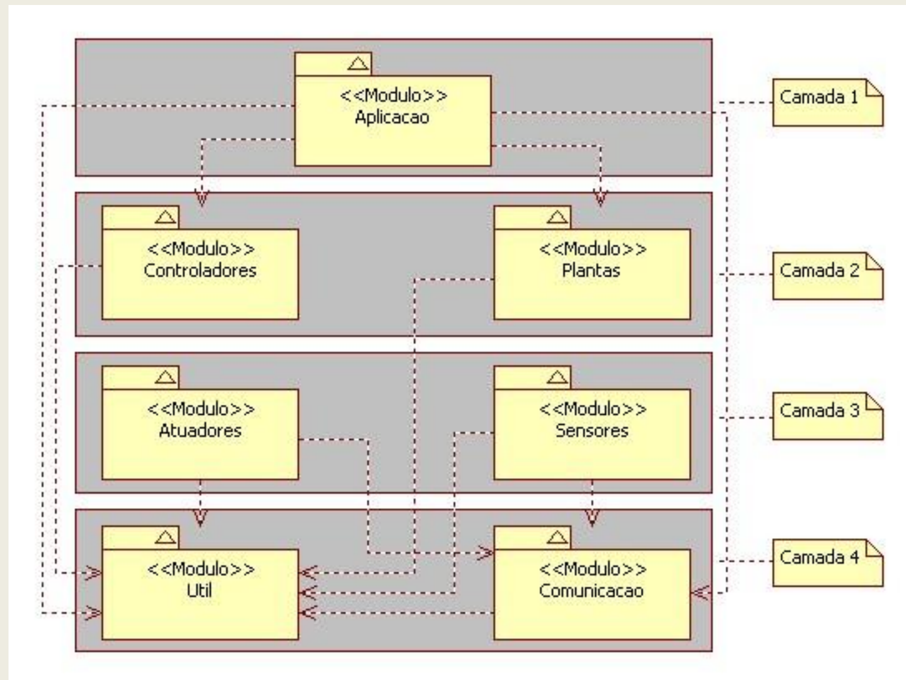


Figura 3.13: Diagrama de módulos e suas dependências na arquitetura do FCD.

3.2.2 Módulos do FCD

Nesta seção serão descritos os módulos da arquitetura do FCD, os componentes dos módulos e suas relações estão descritos através dos diagramas de *Real Time* UML (DOUGLAS, 2000).

Comunicação

Este módulo trata as funções de comunicação através da qual, controladores, sensores, e atuadores enviam e recebem mensagens através da rede de comunicação. As relações de dependências entre os componentes que compõem o módulo são mostradas na Figura 3.14.

O módulo fornece suporte para comunicação utilizando os seguintes protocolos: TCP/IP, UDP/IP e CAN.

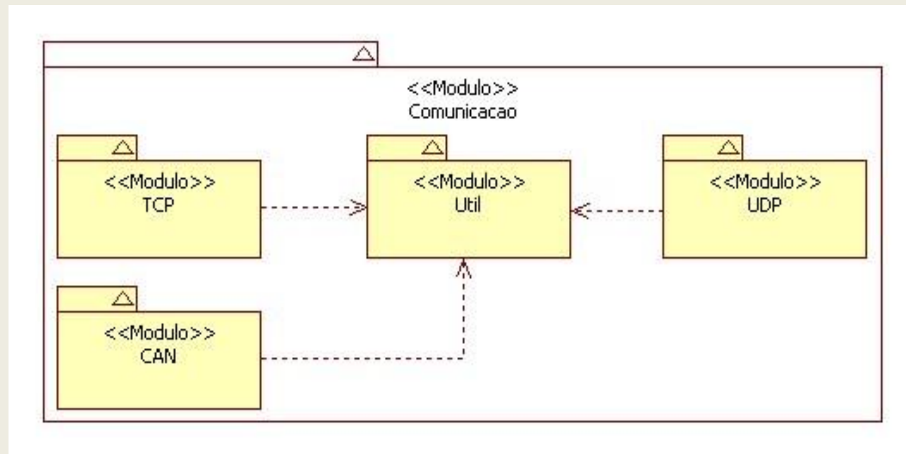


Figura 3.14: Diagrama de componentes do módulo Comunicação.

TCP

O módulo TCP é formado pelas classes SocketTcp e ServidorTcp mostradas no diagrama da Figura 3.15. A classe SocketTcp implementa a classe abstrata Socket que é uma abstração da API de comunicação BSD (*Berkeley Software Distribution*). A API BSD é utilizada para comunicação entre processos distribuídos em redes através dos protocolos TCP/IP ou UDP/IP (SOARES, L. F. ; LEMOS, G. & COLCHER, S., 1995). A classe ServidorTcp é composta por um ou mais SocketTcp.

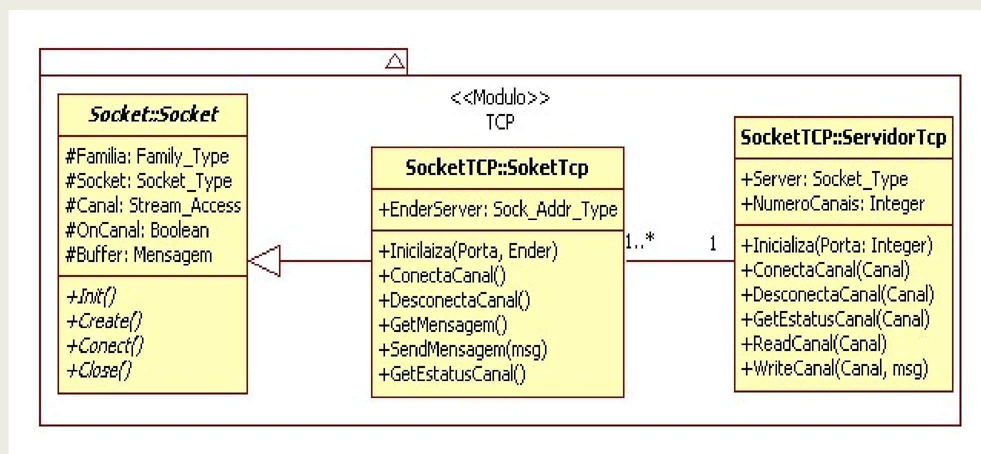


Figura 3.15: Diagramas de classes do módulo TCP.

UDP

O diagrama de classe do módulo UDP mostrado na Figura 3.16 é semelhante ao diagrama do módulo TCP. A diferença entre os dois módulos é a substituição do protocolo TCP pelo protocolo UDP. Embora o protocolo UDP forneça um serviço de comunicação dirigido a pacotes, sem conexão, a classe SocketUdp possui os métodos conecta e desconecta para sincronizar as comunicações entre os sensores e atuadores das plantas com os controladores.

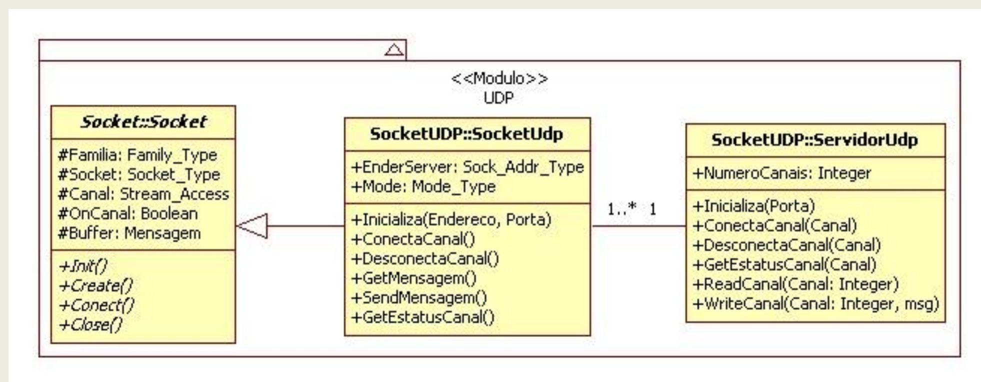


Figura 3.16: Diagramas de classes do módulo UDP.

CAN

Este módulo possui a classe Can mostrado no diagrama da Figura 3.17. Está classe é a base para comunicação entre os sensores e atuadores da planta e o controlador utilizando o protocolo CAN (BOTERENBROOD, 2000).

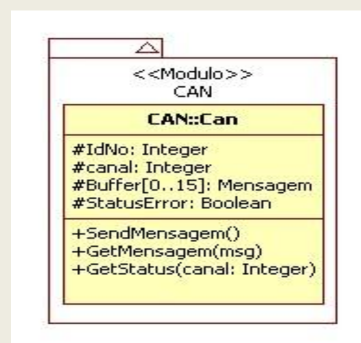


Figura 3.17: Diagrama de classes do módulo CAN.

Controladores

O módulo Controladores é formado pelo modelo de controlador mostrado no diagrama de classes Figura 3.18. A classe abstrata *Controlador* herda os métodos e atributos da classe abstrata *Tarefa Periódica* e possui uma ou mais instâncias da classe abstrata *Socket* para comunicação com os sensores e atuadores da planta. O modelo de aplicação que será distribuída nos nós controladores é mostrado no diagrama de classes da Figura 3.19.

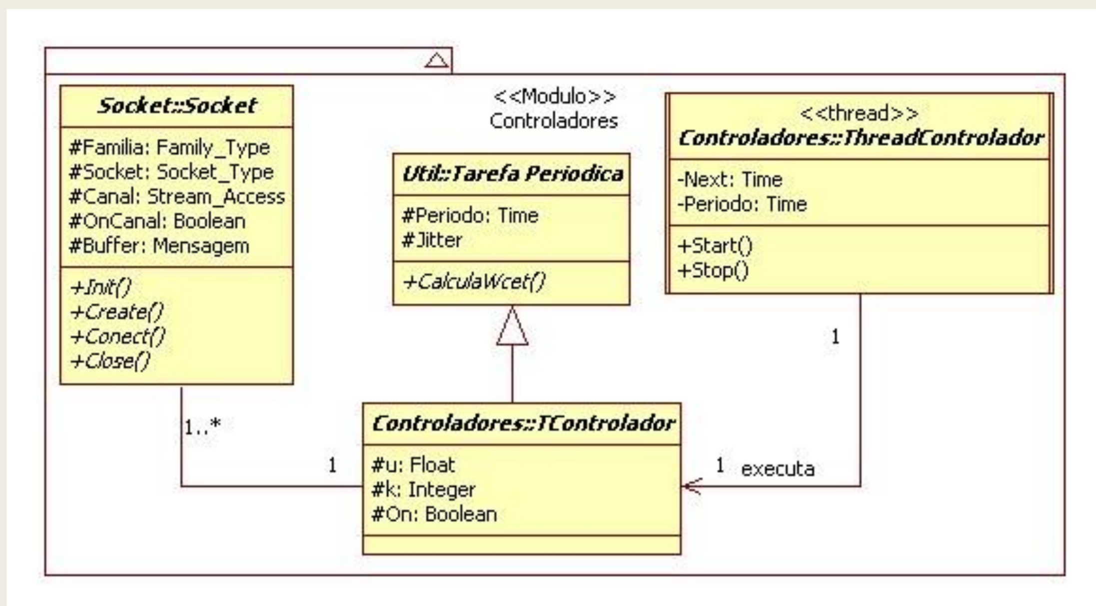


Figura 3.18: Diagrama de classes do módulo Controladores distribuído.

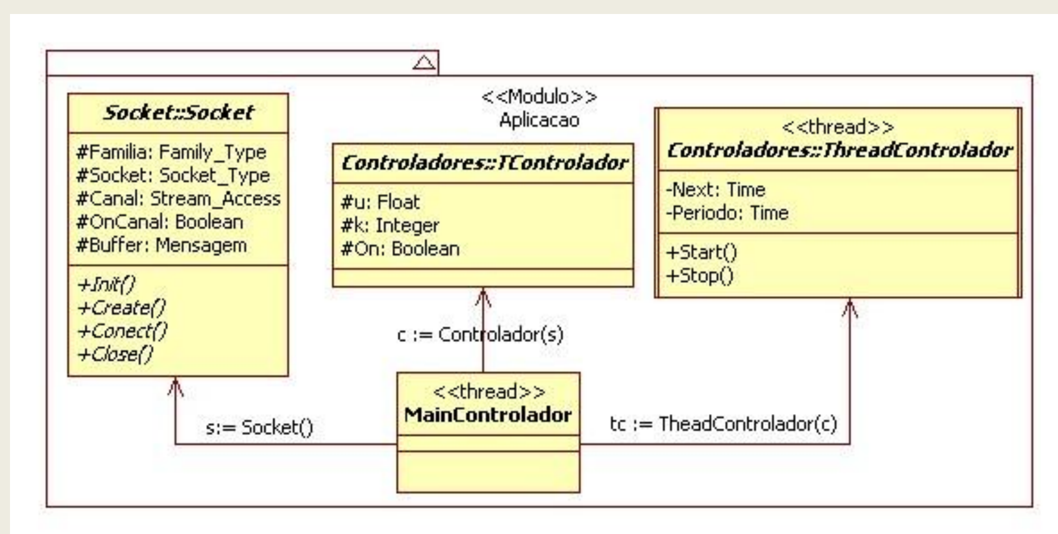


Figura 3.19: Diagrama de classe do módulo aplicação distribuída no Nó Controlador.

Plantas

O modelo de plantas mostrado neste módulo é o mesmo modelo mostrado no diagrama de classes Figura 3.6. O modelo de aplicação que será distribuída no nó Planta/Sensor/Atuador é mostrado no diagrama de distribuição da Figura 3.20.

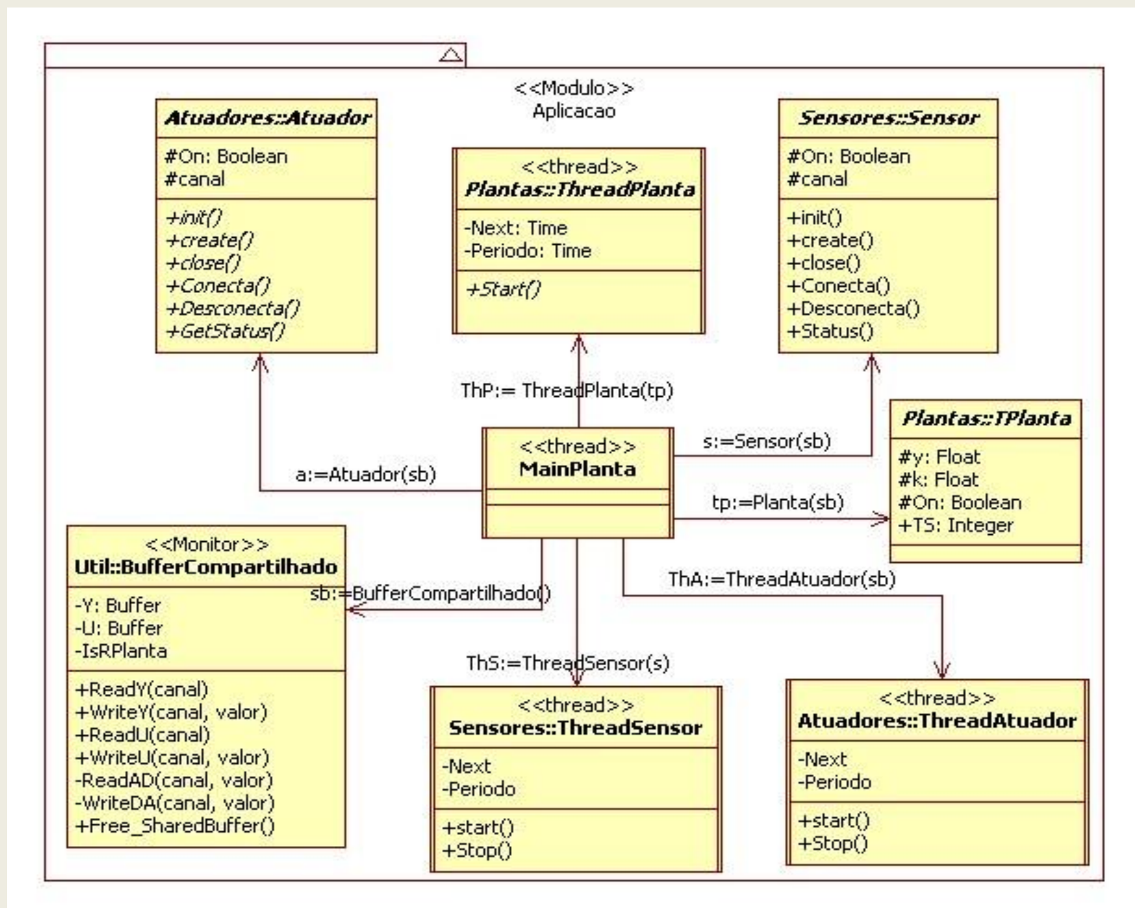


Figura 3.20: Diagrama de classes do módulo aplicação distribuída no Nó Planta/Sensor/Atuador.

Atuadores

Este módulo é formado pelo modelo de Atuador mostrado no diagrama de classes da Figura 3.21. A classe abstrata *Atuador* herda os métodos e atributos da classe abstrata *Tarefa*. Os outros componentes da classe atuador são a classe abstrata *Socket*, a classe Can e o tipo

monitor (Buffer Compartilhado). As classes *Socket* e Can são utilizadas para comunicação com Nó Controlador. O Buffer Compartilhado é utilizado para comunicação com as plantas.

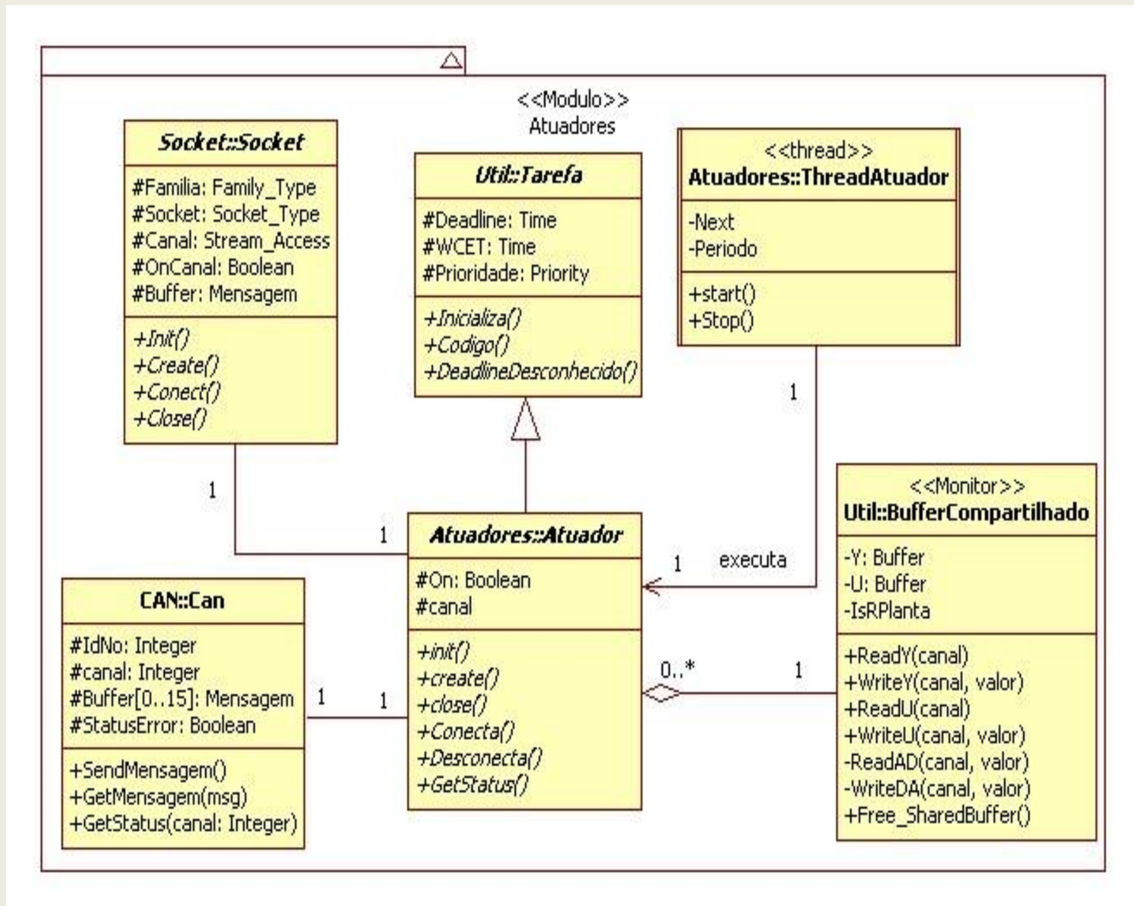


Figura 3.21: Diagrama de classes do módulo atuadores para controle distribuído.

Sensores

O módulo é formado pelo modelo de sensor mostrado no diagrama de classes da Figura 3.22. Comparando o modelo de sensor com o modelo de atuador mostrado no diagrama de classes da Figura 3.21 é possível verificar que a interface e as relações de dependências são praticamente as mesmas. A principal diferença entre os classes “sensor” e “atuador” é o tipo de tarefa. O sensor é uma tarefa periódica e o atuador uma tarefa aperiódica.

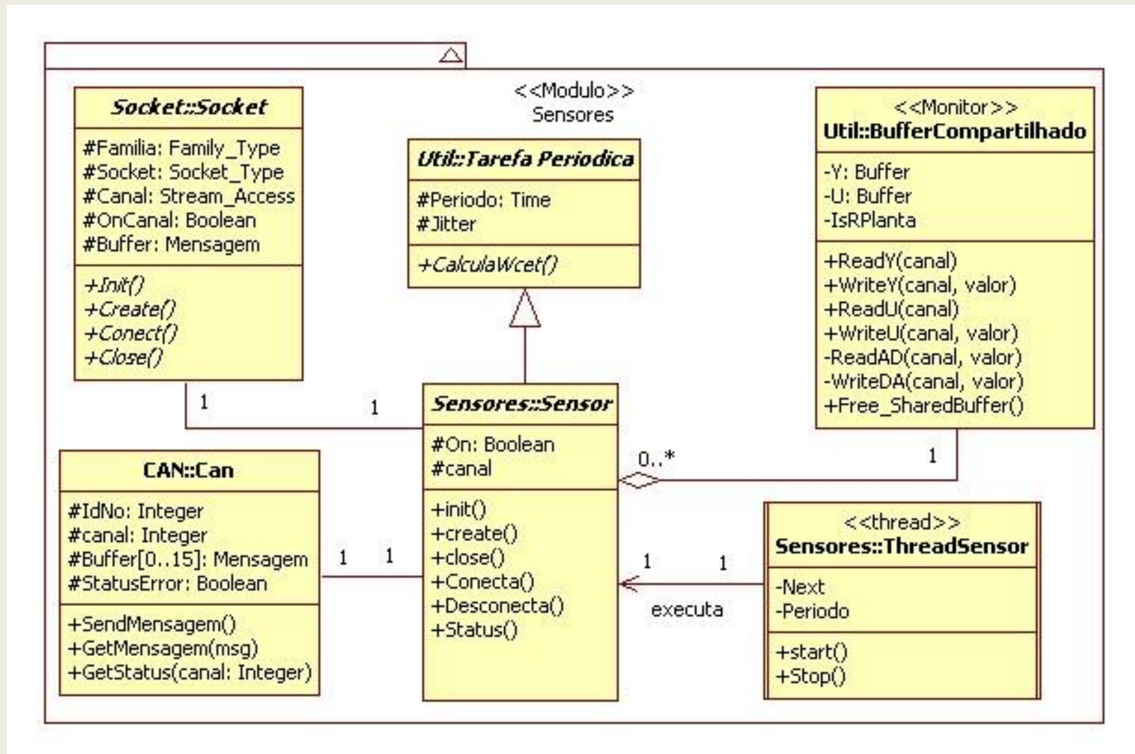


Figura 3.22: Diagrama de classes do módulo sensores para controle distribuído.

Útil

Este módulo é o mesmo mostrado no diagrama de classes da Figura 3.9.

4 Arquitetura de plataforma para RCP

Segundo Lee, Shin e Sunwoo (2004), em um processo de desenvolvimento de um sistema de controle aplicando RCP (Figura 1.5) e utilizando uma plataforma com arquitetura semelhante as plataformas utilizadas atualmente na indústria (Figura 1.4), no estágio de RCP, fases 1, 2, 3 e 4 do processo de desenvolvimento do sistema de controle, são realizados a validação do modelo da planta e o projeto do algoritmo de controle. A implementação do código para o *target* é realizada depois pelos engenheiros de *software*. Para Lee et. al. (2004), esta abordagem aumenta a distância entre as áreas de controle e computação tornando mais difícil a cooperação entre as áreas. Caso o código do *target* seja gerado de forma automática, a cooperação entre as áreas praticamente não existe.

As diferenças entre a prototipagem e os diferentes níveis de abstração do processo de desenvolvimento e produção do *software* de controle, tornam difícil uma perfeita transição do protótipo desenvolvido no estágio RCP para o *target*. No Capítulo 3, foram propostas duas arquiteturas de *frameworks* orientados a objetos para serem utilizados no estágio de RCP. Estes *frameworks* serão utilizados como uma das camadas da arquitetura de plataforma proposta para RCP, mostrada na Figura 4.1, com o objetivo de automatizar o processo de transição do protótipo do controlador para o *target* e eliminar os problemas ocasionados pelo uso dos geradores automáticos de código.

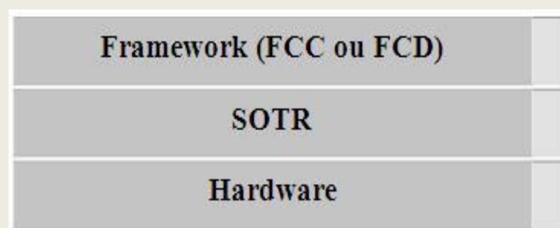


Figura 4.1: Arquitetura de plataforma para RCP.

A arquitetura de plataforma para RCP mostrada na Figura 4.1 possui três camadas. A camada superior da arquitetura é composta por um dos *frameworks* desenvolvidos no Capítulo 3. O tipo de arquitetura de *framework*, FCC ou FCD, presente nesta camada define se a plataforma para RCP é centralizada ou distribuída. Caso seja utilizado o FCC na primeira camada, a plataforma é do tipo centralizada. Se o FCD for utilizado na primeira camada, a plataforma é do tipo distribuída.

A segunda camada da plataforma, SOTR, é o Sistema Operacional de Tempo Real que será embarcado no *target*. A terceira camada representa o *hardware* do *target*. Neste trabalho a segunda camada é o SOTR VxWorks da Wind River. A terceira camada é composta por PCs/104 e alguns módulos (placas conversoras A/D, D/A e de rede CAN).

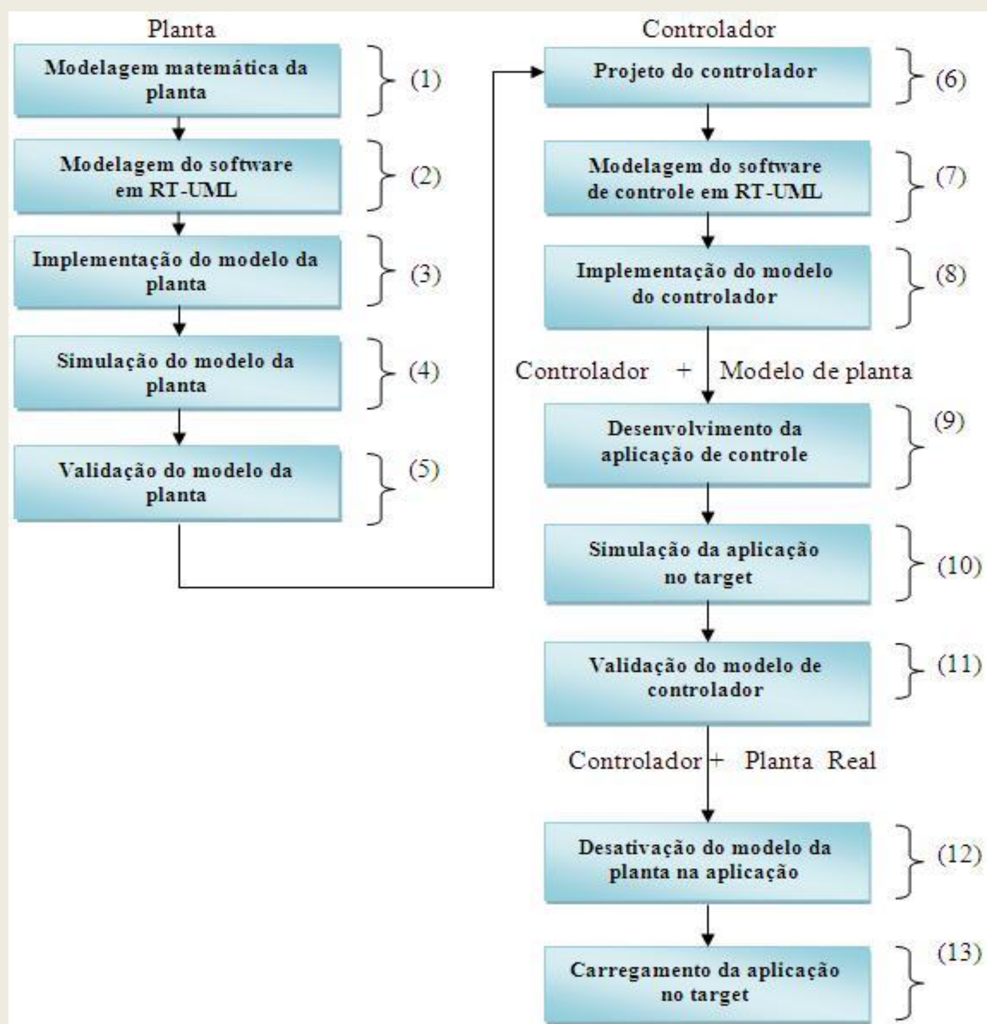


Figura 4.2: Processo de desenvolvimento do sistema de controle aplicando RCP modificado.

O processo de desenvolvimento de um sistema de controle aplicando RCP e utilizando a arquitetura de plataforma e os *frameworks* propostos é ilustrado na Figura 4.2. Os nomes contidos dos retângulos indicam que atividade está sendo realizada no estágio atual. Os nomes sobre os retângulos indicam os principais artefatos de *software* que estão sendo produzidos ou utilizados no atual estágio de desenvolvimento. O processo mostrado na figura está dividido em dois estágios: modelagem e validação do modelo da planta, modelagem e validação do protótipo do controlador.

O estágio de modelagem e validação da planta é composto pelas atividades de 1 a 5. O estágio de modelagem e validação do protótipo do controlador é formado pelas atividades de 6 a 11. As atividades 12 e 13 não são um estágio, pois a “desativação do modelo de planta na aplicação” corresponde a comentar o trecho de código referente à criação da tarefa “planta” e substituir o valor do atributo IsRPlanta para verdadeiro. “Carregamento da aplicação no *target*” é uma atividade que já ocorre durante a simulação da aplicação com os modelos de planta e controlador.

4.1 Análise de um sistema de controle utilizando as arquiteturas de *frameworks* e plataformas para RCP

Nesta seção as arquiteturas de *frameworks* e plataformas propostas para RCP serão utilizados para analisar o comportamento dos sistemas de controle centralizado e distribuído descrito no exemplo 4.1.

Exemplo 4.1

Considere o sistema com controlador digital PID mostrado nos diagramas de blocos das Figuras 4.3 e 4.4:

1) Inicialmente, desenvolve-se um sistema de controle centralizado para analisar a resposta do sistema à uma entrada em degrau unitário.
2) A seguir, desenvolve-se um sistema de controle distribuído em rede *Ethernet* (protocolo TCP/IP) para analisar a resposta do sistema à uma entrada em degrau unitário.
3) Por fim, compara-se o comportamento dos sistemas centralizado e distribuído considerando o sobre-sinal máximo e o tempo de acomodação para um critério de 2%.

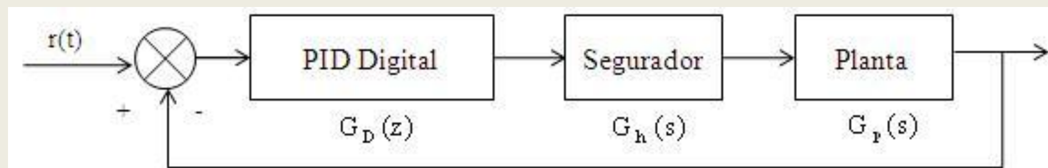


Figura 4.3: Diagrama de blocos do sistema em malha fechada com controlador PID digital e planta contínua.

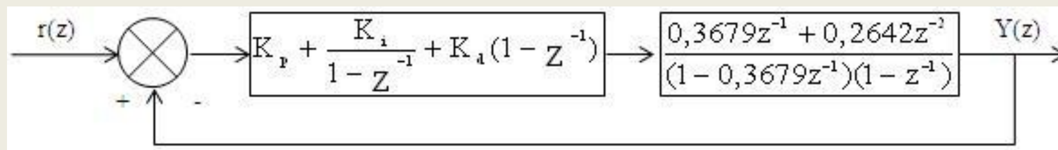


Figura 4.4: Diagrama de blocos do sistema em malha fechada com controlador e planta no domínio de Z.

A função de transferência da planta contínua é:

$$G_p(s) = \frac{1}{s(s+1)} \tag{2.6}$$

A função de transferência do segurador de ordem zero (ZOH) é:

$$G_h(s) = \frac{1-e^{-s}}{s} \tag{2.7}$$

A função de transferência do controlador PID é:

$$G_D(s) = K_p + \frac{K_i}{s} + K_d s \tag{2.8}$$

4.1.1 Sistema de Controle Centralizado

Planta

(1) Modelagem matemática da planta

A análise do modelo de planta mostrado na Equação 4.1, no domínio contínuo, é realizada convertendo o modelo para o domínio discreto. A Equação 4.4 mostra uma maneira de converter o modelo do sistema contínuo para o domínio discreto através do cálculo da transformada **Z** entre a convolução do segurador de ordem zero e planta. A equação de diferenças utilizada durante a simulação do sistema em computador é obtida a partir da transformada **Z** inversa.

$$G_p(z) = Z\left[\frac{1-e^{-s}}{s}\,\frac{1}{s(s+1)}\right] \tag{2.9}$$

A Equação 4.5 mostra a função de transferência pulsada do sistema de dados amostrados equivalente à função de transferência do sistema contínuo para T igual 1 obtido a partir da Equação 4.4.

$$G_p(z) = \frac{Y(z)}{U(z)} = \frac{0,3679z^{-1} + 0,2642z^{-2}}{(1 - 0,3679z^{-1})(1 - z^{-1})} \tag{2.10}$$

Aplicando a transformada **Z** inversa na Equação 4.5 temos a seguinte equação de diferenças para a planta:

$$y(k) - 1,3679y(k-1) + 0,3679y(k-2) = 0,3679u(k-1) + 0,2642u(k-2) \tag{2.11}$$

Rearranjando os termos da Equação 4.6 e substituindo k por (k+1) temos:

$$y(k+1) = 0,3679u(k) + 0,2642u(k-1) + 1,3679y(k) - 0,3679y(k-1) \tag{2.12}$$

onde, **y(0) = 0** e **y(1) = 0,3679**.

(2) Modelagem do software em RT-UML

Para o modelo de software planta no *framework* foram criados a classe Planta2 derivada da classe abstrata *Planta* e a *thread* ThreadPlanta. A ThreadPlanta é uma abstração da tarefa do sistema operacional que executará a equação de diferenças da planta (Equação 4.7) durante os intervalos periódicos.

O modelo de *software* em RT-UML para a planta é mostrado na Figura 4.4. Por conversão, foi utilizado um número na parte final do nome da classe (Planta2) para indicar a ordem do sistema representado pelo modelo. No exemplo a parte final do nome é igual a 2, indicando que este é um modelo de *software* para sistemas de segunda ordem.

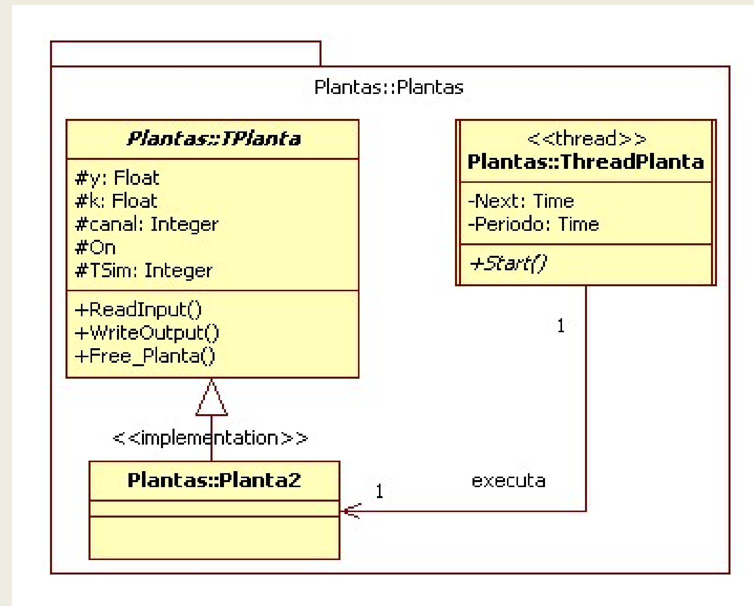


Figura 4.5: Diagrama de classes da planta de segunda ordem.

(3) Implementação do modelo da planta

A implementação do modelo para a planta de segunda ordem é mostrada no Apêndice A.3.

(4) Simulação do modelo da planta

O gráfico da Figura 4.5 mostra o resultado da simulação do modelo da planta por um intervalo de tempo de 40 segundos.

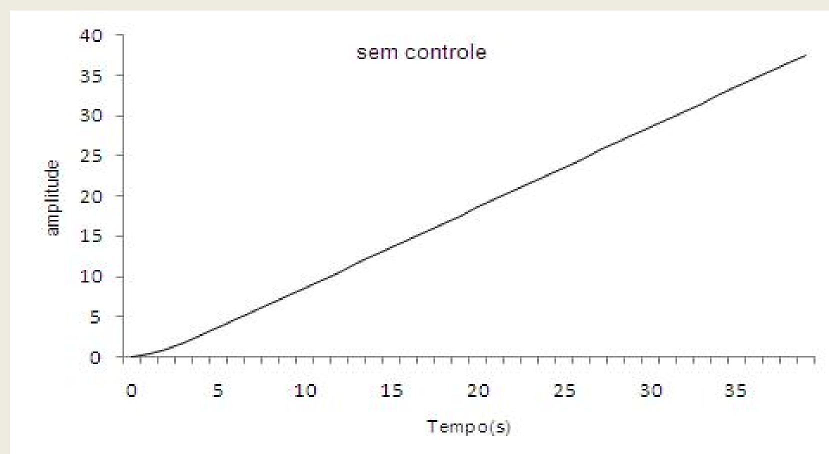


Figura 4.6: Resposta da planta sem controle a entrada em degrau unitário.

(5) Validação do modelo da planta

Neste exemplo o modelo de planta foi fornecido. Portanto, não existe a necessidade de validação.

Controlador

(6) Projeto do controlador

Foi assumido como objetivo para o projeto do sistema de controle que a saída da planta em regime permanente seja igual a 1 para um critério de erro igual a 2%. Para atingir este objetivo é utilizado o controlador PID mostrado no diagrama de blocos da Figura 4.4 com os seguintes valores de ganho: ganho proporcional $\mathbf{K_p = 1}$, ganho integral $\mathbf{K_i = 0{,}2}$ e ganho derivativo $\mathbf{K_d = 0{,}2}$. A Equação 4.8 mostra a função de transferência pulsada do controlador PID na forma de posição.

$$G_D(z) = k_p + K_i T \frac{1}{1-z^{-1}} + \frac{K_d}{T}(1-z^{-1}) \tag{2.13}$$

Aplicando a aproximação de diferenças para trás para a derivada e a integração retangular avançada para a integral obtém-se a equação de diferenças mostrada na Equação 4.9 para o controlador PID.

$$u(k) = K_p\left[e(k) + \frac{T}{T_i}\sum_{i=0}^{k} e(i) + \frac{T_d}{T}(e(k) - e(k-1))\right] \tag{2.14}$$

(7) Modelagem do *software* de controle em RT-UML

O modelo do *software* de controle em RT-UML para o controlador PID é mostrado no diagrama da Figura 4.7. A classe PID implementa os métodos da classe abstrata *TControlador*. Os valores das constantes do controlador PID são os atributos da classe ParametrosPID. O tipo do controlador (P, PI, PD, PID) e a abordagem (convencional ou *subtask scheduling*) são as enumerações TControladoresPID e TAbordagem. O tipo *thread* ThreadPID é a abstração de tarefa do sistema operacional que executa a equação de diferenças do controlador (Equação 4.9) em intervalos periódicos.

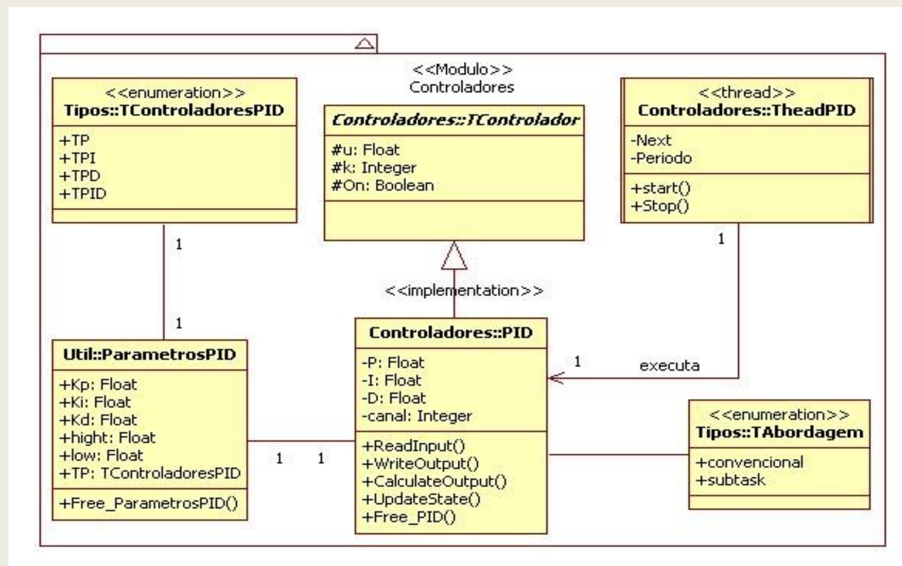


Figura 4.7: Diagrama de classes do controlador PID.

(8) Implementação do modelo do controlador

A implementação do modelo do controlador PID é mostrada no Apêndice A.5.

Controlador + Modelo de Planta

(9) Desenvolvimento da aplicação de controle

O código da aplicação desenvolvida é mostrado no Apêndice A.7.

(10) Simulação da aplicação no *target*

O gráfico da Figura 4.8 mostra o resultado da simulação do sistema durante um período de 40 segundos. O instante de pico do sistema é aproximadamente 4 segundos e o tempo de acomodação considerando um critério de 2% é aproximadamente 17 segundos.

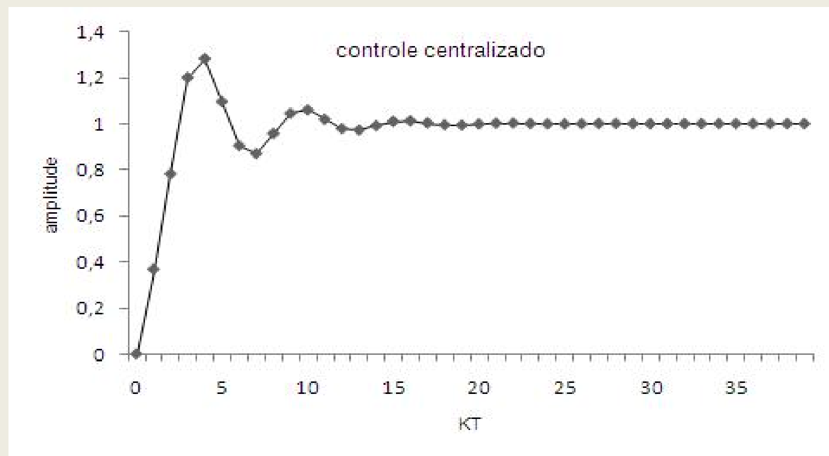


Figura 4.8: Resposta da planta ao sinal de saída do controlador PID centralizado para a entrada em degrau unitário.

(11) Validação do modelo de controlador

Os resultados mostrados no gráfico da Figura 4.8 estão de acordo com os objetivos estabelecidos para o sistema de controle. Portanto, o protótipo do controlador pode ser utilizado para controlar uma planta real com as mesmas especificações.

Controlador + Planta Real

Os passos listados abaixo são executados quando existe uma planta real conectada a plataforma RCP.

(12) Desativação modelo da planta na aplicação;

(13) Carregamento da aplicação no *target*;

4.1.2 Sistema de Controle distribuído (protocolo TCP/IP)

Para a análise do sistema de controle distribuído em rede Ethernet (protocolo TCP/IP), o modelo matemático e de *software* da planta são os mesmo descritos na análise do sistema de controle centralizado.

Controlador

(7) Modelagem do *software* de controle em RT-UML

O modelo de *software* em RT-UML para o controlador PID distribuído é o mesmo modelo do controlador PID centralizado mostrado na Figura 4.7. A diferença entre o controlador PID centralizado e o distribuído está na classe abstrata *TControlador*. No sistema distribuído a classe abstrata *TControlador* possui uma referência para um ou mais *Sockets*.

(8) Implementação do modelo do controlador

A implementação do modelo para controlador PID distribuído é mostrada no Apêndice A.6.

Controlador + modelo de planta

(9) Desenvolvimento da aplicação de controle

O código das aplicações desenvolvidas e distribuídas nos Nós Controlador e Planta/ Sensor/ Atuador são mostrados nos Apêndices A.8 e A.9.

(10) Simulação da aplicação no *target*

Para simular a aplicação de controle distribuído em rede Ethernet (protocolo TCP/IP) foi montada a plataforma RCP distribuída mostrada na Figura 4.9. A plataforma é composta pelo *host* e dois *targets* (*target 1* e *target 2*).

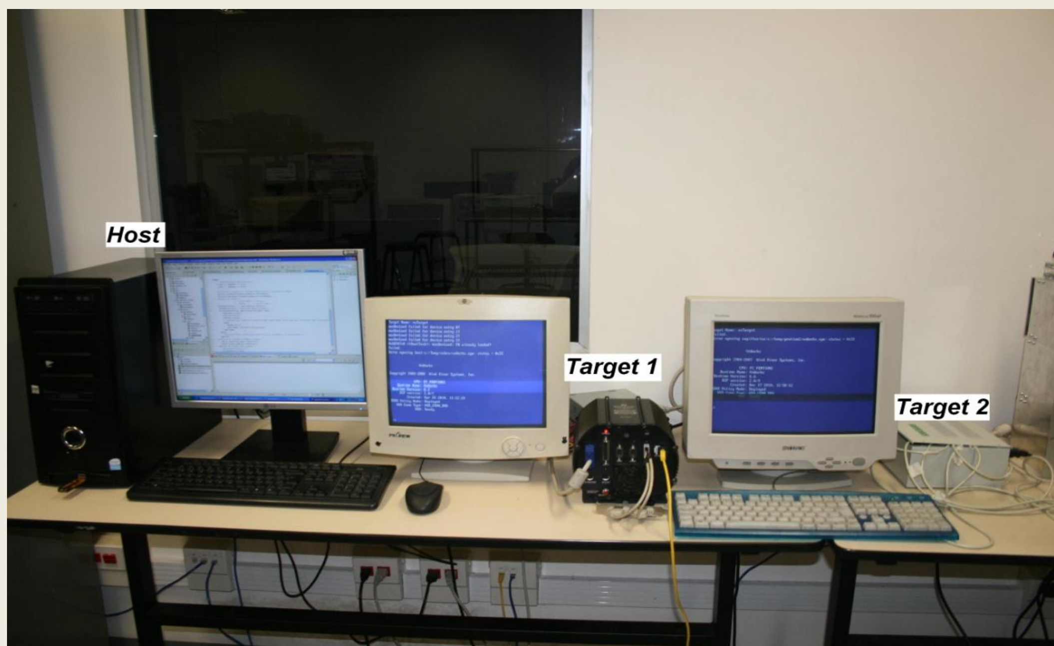


Figura 4.9: Plataforma RCP para sistemas de controle distribuído em rede.

No *host*, CPU Pentium D com *clock* de 3.4 GHZ e sistema operacional Windows XP, foi instalada a plataforma de desenvolvimento para sistemas embarcados (*Wind River General Purpose Platform, VxWorks Edition 3.6*). Os principais componentes desta plataforma de desenvolvimento são: o Workbench (IDE baseada em Eclipse) e o SOTR VxWorks.

O VxWorks é um *microkernel* multitarefa que possui como principais requisitos de projeto o determinismo e a resposta rápida a interrupções (características dos sistemas *hard real-time*). Atualmente o sistema fornece suporte para r*eal-time process* (RTP) que inclui a execução de aplicações em modo usuário e outras características comuns aos Sistemas Operacionais tais como: clara delineação entre *kernel* e aplicações, gerenciamento de arquivos e memória, escalonamento preemptivo, comunicação entre processos, suporte a praticamente todas as CPUs modernas, etc. Estas características fazem com que o VxWorks seja amplamente utilizado em aplicações de sistemas embarcados na indústria aeroespacial, indústria automotiva, sistemas de defesa, sistemas de controle, computação móvel, equipamentos de rede, controle de processos industriais, etc.

A IDE Workbench é um ambiente integrado de desenvolvimento com interface gráfica e fornece todas as facilidades para a criação, edição, compilação, *debug* e gerenciamento de projetos como: *boot-loaders*, imagens do *kernel* do SOTR, aplicações baseadas em *Real-Time Process* (RTP) e em módulos de *kernel*, etc. Além de um conjunto de ferramentas que permitem a comunicação entre o *host* e os processos que são executados no *target*.

O *target 1* (Nó Planta/Sensor/Atuador) é um PC/104-Plus cheetah, CPU Intel Pentium M com *clock* de 1.8 GHZ. O *target 2* (Nó Controlador) é um PC/104, CPU Cyrix Geode Gx1 com *clock* de 300 MHZ. Para os *targets* 1 e 2, poderiam ter sido utilizados qualquer tipo de PC. Porém, a opção pelo modelo PC/104, em vez de um PC convencional para embarcar os sistemas, se deve ao fato deste ter sido desenvolvido especificamente para aplicações embarcadas onde a confiabilidade, o consumo de energia e o espaço para armazenamento são críticos. Além de ser um padrão muito bem aceito para placas do tipo mezanino.

O gráfico da Figura 4.10 mostra o resultado da simulação do sistema durante um período de quarenta segundos. O instante de pico do sistema é aproximadamente 4 segundos e o tempo de acomodação considerando um critério de 2% é aproximadamente 27 segundos.

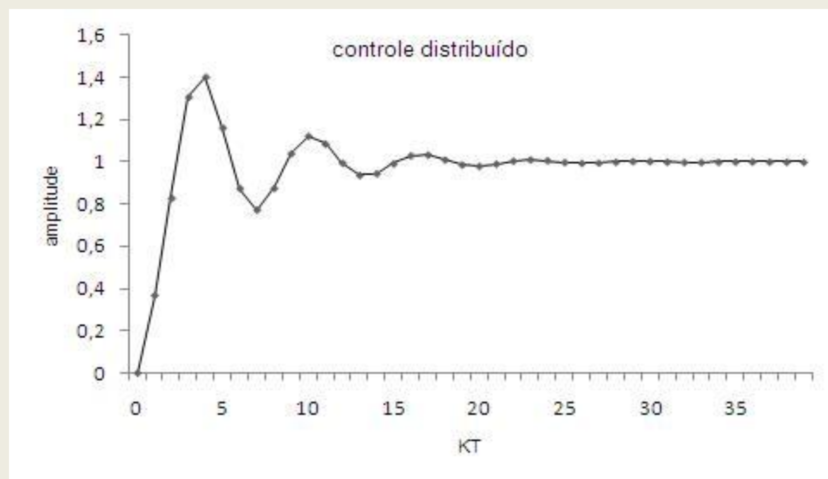


Figura 4.10: Resposta da planta ao sinal de saída do controlador PID distribuído para uma entrada em degrau unitário.

(11) Validação do modelo de controlador

Os resultados apresentados no gráfico da Figura 4.10 estão de acordo com os objetivos estabelecidos para o sistema de controle. Portanto, o modelo de controlador pode ser utilizado para controlar a planta real com as mesmas especificações.

Controlador + Planta Real

Os passos listados abaixo são executados somente quando existem plantas reais conectadas a plataforma RCP.

(12) Desativação modelo da planta na aplicação;

(13) Carregamento da aplicação no *target*;

4.1.3 Comparação entre o SCC e o SCD

Os resultados apresentados mostram que o instante de pico é aproximadamente o mesmo para os dois sistemas de controle. Porém, o valor de pico no sistema de controle distribuído foi aproximadamente 1,5 e no sistema de controle centralizado foi de aproximadamente 1,3. Em relação ao tempo de assentamento, o sistema de controle centralizado apresenta uma resposta melhor com um tempo de assentamento de aproximadamente 17 segundos. Enquanto o sistema de controle distribuído apresenta um tempo de assentamento de aproximadamente 27 segundos. Embora, não tenham sido realizados os cálculos dos atrasos no sistema, é possível inferir através do valor de pico e do tempo de assentamento que o sistema de controle distribuído deve apresenta desempenho inferior ao sistema de controle centralizado. A degradação no desempenho do sistema distribuído é causada pelos atrasos existentes no envio dos dados do sensor para o controlador e do controlador para o atuador através da rede de comunicação.

4.2 Plataforma RCP para sistema centralizado com plantas reais

Nesta seção é apresentada uma plataforma RCP para sistema centralizado, construída para análise e testes de sistemas de controle com plantas reais e para mostrar a facilidade de transição entre o protótipo do controlador utilizado para controlar os modelos de planta para os controladores utilizados para controlar as plantas reais. A Figura 4.11 ilustra a plataforma com 3 motores de CC e algumas inércias acopladas aos eixos dos motores (Planta1, Planta 2 e Planta 3).

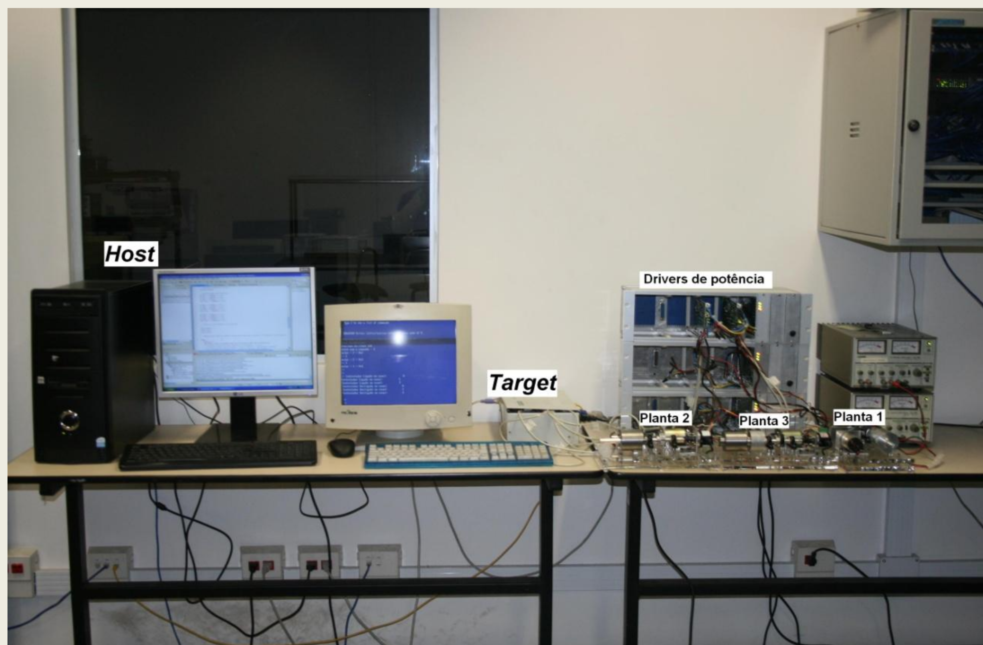


Figura 4.11: Plataforma RCP para sistema centralizado com plantas reais conectadas às interfaces de entrada e saída.

Na plataforma mostrada na figura, as saídas da plataforma, canais do conversor D/A, e as entradas da plataforma, canais do conversor A/D, estão ligadas aos *drivers* de potência dos motores. O acionamento dos motores é realizado aplicando uma tensão de saída nos canais do conversor D/A. A leitura dos sensores dos motores (*encoders*) é realizada através da leitura

dos canais do conversor A/D. Os valores das tensões de saída são proporcionais à rotação dos motores.

Modelo matemático das plantas (Motores CC)

Os modelos matemáticos das plantas foram obtidos através dos ensaios realizados a uma taxa de amostragem de 100 Hz. Durante os ensaios os motores foram acionados individualmente por um intervalo de tempo igual a 4 segundos. Para o acionamento dos motores foi aplicada uma entrada em degrau de 4,2 volts nos canais do conversor D/A (saída da plataforma) conectados aos *drivers* de potência dos motores.

Os gráficos da Figura 4.12 mostram os valores das tensões de saída dos *drivers* de potência ligados aos canais do conversor A/D da plataforma. Os valores da saída podem ser convertidos para velocidade angular, desde que, seja conhecida a relação entre as tensões de saída nos *drivers* de potência e a velocidade angular dos motores.

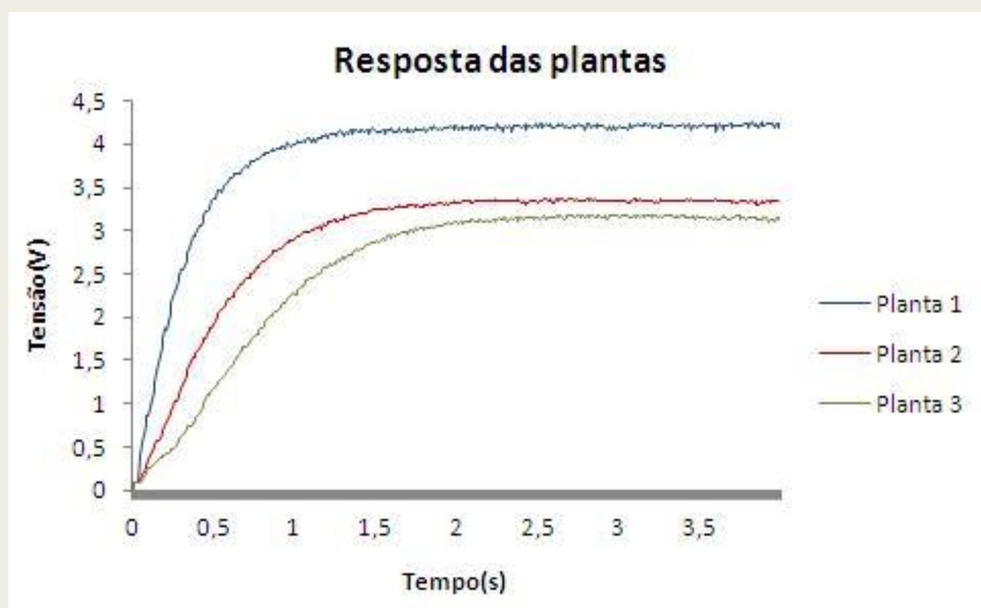


Figura 4.12: Respostas das plantas 1, 2 e 3 a uma entrada em degrau de 4,2 volts.

As funções de transferência $G_p(s)$ das plantas no domínio contínuo podem ser escritas na forma da Equação 4.10.

$$\frac{K}{\tau s + 1} \tag{2.15}$$

Onde, **τ** é a constante de tempo do sistema de primeira ordem e **K** é o ganho do regime permanente. A Tabela 4.1 mostra os valores do ganho, da constante tempo e da saída permanente obtidos durante os ensaios.

Tabela 4.1: Constantes das plantas reais.

Plantas	K	τ[s]	Saída[v]
G_{P1}	1,005	0,33	4,221
G_{P2}	0,794	0,56	3,333
G_{P3}	0,747	0,83	3,137

A função de transferência pulsada $G_p(z)$ do sistema discreto equivalente ao sistema da Equação 4.10 pode ser escrita na forma da Equação 4.11.

$$\frac{1 - e^{-\frac{T}{\tau}}}{z - e^{-\frac{T}{\tau}}} \tag{2.16}$$

onde, T é o período de amostragem e **τ** é a constante de tempo do sistema contínuo.

A Tabela 4.2 mostra os períodos de amostragens T em segundos, as funções de transferência do sistema contínuo $G_P(s)$, as funções de transferência pulsada $G_P(z)$ dos sistemas discretos equivalente aos sistemas contínuo e as equações de diferenças Y(k). As equações de $G_P(z)$ foram obtidas fazendo $N_r = 10$ e $T_r = \tau$ na Equação 2.4 e em seguida substituindo o resultado na Equação 4.11.

Tabela 4.2: Modelos matemáticos das plantas.

Plantas	T[ms]	$G_P(s)$	$G_P(z)$	Y(k+1)
Gp_1	33	$\frac{1,005}{0,33S+1}$	$\frac{0,09564}{Z-0,9048}$	$0,09564u(k)+0,9048y(k)$
Gp_2	56	$\frac{0,794}{0,56S+1}$	$\frac{0,07556}{Z-0,9048}$	$0,07556u(k)+0,9048y(k)$
Gp_3	83	$\frac{0,747}{0,83S+1}$	$\frac{0,07109}{Z-0,9048}$	$0,07109u(k)+0,9048y(k)$

Modelo de *software* para simulação da planta

O modelo de *software* para simulação das plantas é mostrado na Figura 4.13. As constates CTEu1 e CTEy1 são os valores constantes que aparecem na equação de y(k+1) mostrados na Tabela 4.2.

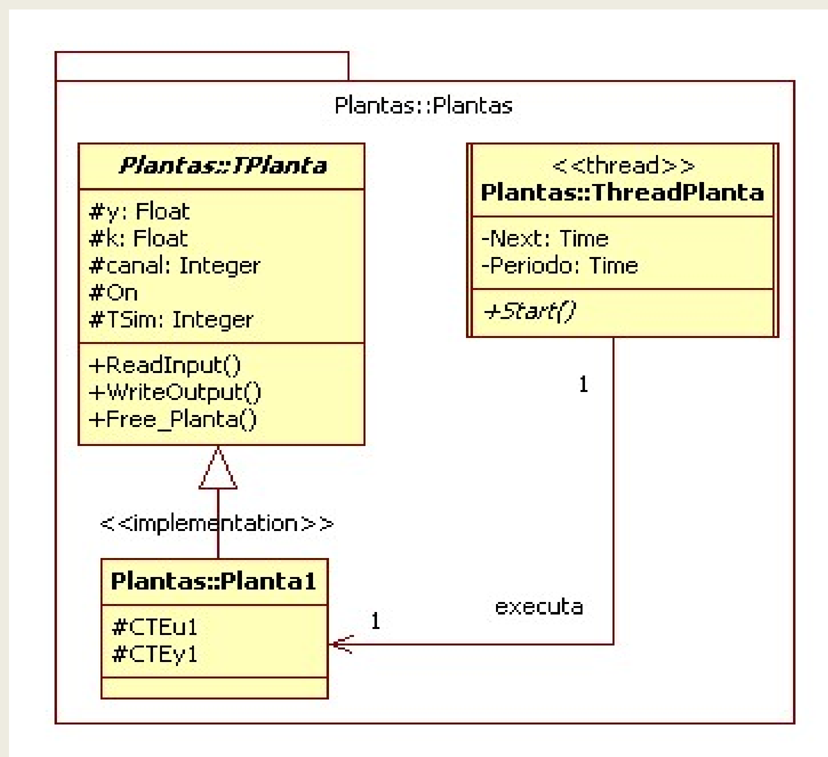
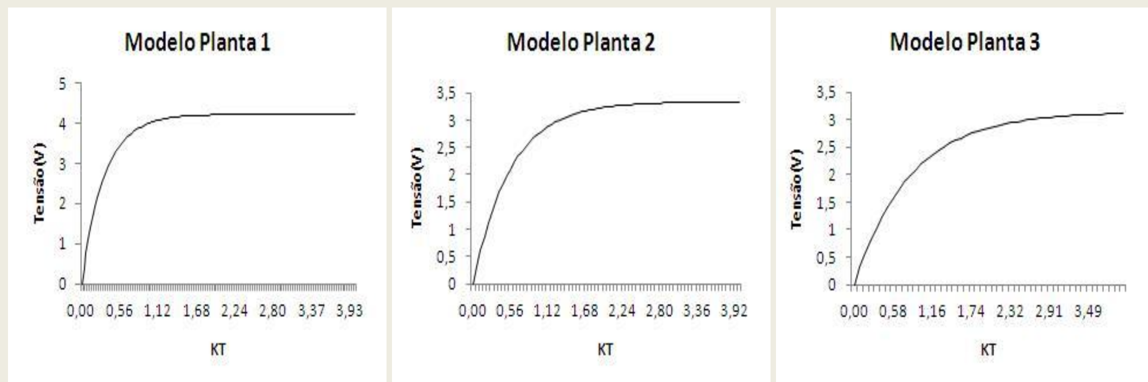


Figura 4.13: Diagrama de classes da planta de primeira ordem.

Validação dos modelos das plantas

A validação dos modelos das plantas foi realizada analisando as respostas dos modelos para uma entrada em degrau de 4,2 volts durante um intervalo de tempo de 4 segundos. Os gráficos das Figuras 4.14a, 4.14b e 4.14c mostram as respostas dos três modelos de plantas durante 4 segundos de simulação.



(a) (b) (c)

Figura 4.14: Respostas dos modelos das plantas a uma entrada em degrau de 4,2 volts: (a) Planta1; (b) Planta 2; (c) Planta 3.

Os resultados dos ensaios apresentados nos gráficos da Figura 4.12, mostram que as três plantas apresentam pequenas oscilações em regime permanente. Porém, os valores das amplitudes destas oscilações são relativamente pequenos, quando comparados com o valor em regime permanente. Portanto, podem ser desconsiderados nos modelos das plantas.

A Tabela 4.3 mostra os valores constantes do ganho **K**, da constante de tempo **τ** em segundos e da saída em volts para os três modelos de plantas.

Tabela 4.3: Constantes dos modelos de plantas.

Modelos	K	τ[s]	saída[v]
Gp_1	1,0046	0,33	4,219
Gp_2	0,793	0,56	3,331
Gp_3	0,745	0,83	3,137

Comparando os valores das Tabelas 4.3 e 4.1 e observando os gráficos das Figuras 4.12 e 4.14, é possível concluir que os modelos matemáticos das plantas G_{P1}, G_{P2} e G_{P3} são equivalentes as plantas reais conectadas à plataforma. Estes modelos de plantas são utilizados no projeto dos controladores para as plantas reais, durante a análise da metodologia *subtask scheduling* e análise dos algoritmos de escalonamento FP e RR.

4.3 Projeto do protótipo do sistema de controle

Especificação do sistema de controle

Durante o projeto do protótipo do controlador para as plantas G_{P1}, G_{P2} e G_{P3}, foi assumido como um único objetivo: o valor das saídas das plantas, em regime permanente, deve ser igual ao valor do sinal de referência de 2,5 volts para um critério de erro de 2%. Para alcançar este objetivo foi projetado o controlador tipo PI descrito a seguir.

Protótipo do controlador

A função de transferência $G_D(z)$ e a equação de diferenças u(k) do controlador PI foram derivadas do controlador PID da seção 4.1 desativando o termo derivativo. Os parâmetros ($\mathbf{K_p}$ e $\mathbf{K_i}$) dos controladores são mostrados na Tabela 4.4. Os parâmetros dos controladores foram obtidos aplicando o método de *Ziegler Nichols* de malha fechada inicialmente e realizando um ajuste manual na seqüência. Os índices dos controladores indicam que planta é controlada pelo respectivo controlador, ou seja, o controlador $\mathbf{C_1}$ controla a planta G_{P1}.

Tabela 4.4: Parâmetros dos controladores PI.

Controladores	K_p	K_i
C_1	1,5	6,0
C_2	2,0	4,0
C_3	2,5	2,7

Modelo de *software* de controle

O modelo do *software* de controle é o mesmo mostrado no diagrama da Figura 4.7.

4.3.1 Simulação do protótipo com modelos de plantas

Para simular o protótipo do sistema de controle com modelo de plantas, foi implementada a aplicação mostrada no Apêndice A10. Os gráficos das Figuras 4.15, 4.16 e 4.17 mostram os valores do sinal de controle u(kT), da saída da planta y(kT) e do erro e(kT) obtidos durante a simulação dos modelos de plantas por um intervalo de tempo igual a 5 segundos.

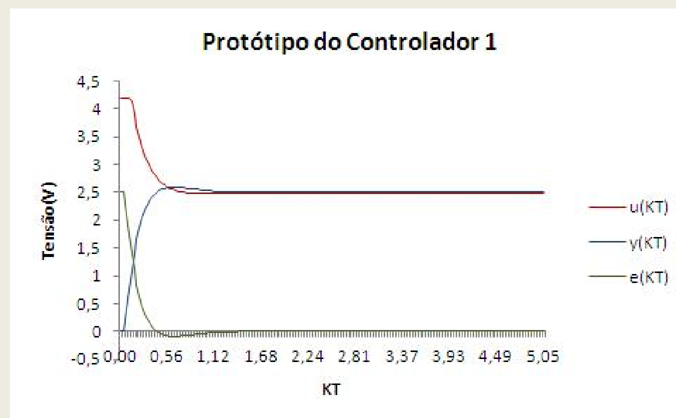


Figura 4.15: Resposta do modelo de Planta 1 e saída do protótipo do controlador $\mathbf{C_1}$.

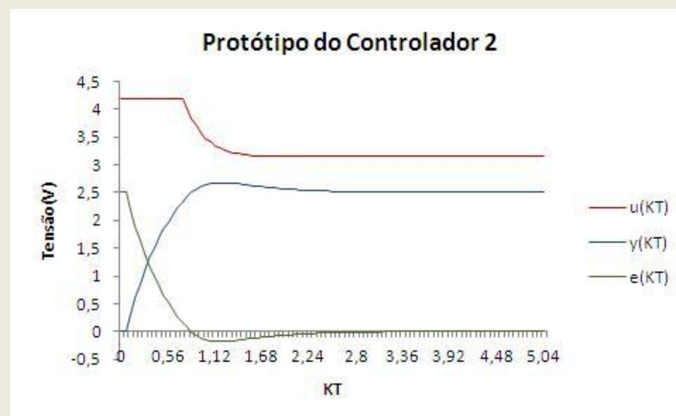


Figura 4.16: Resposta do modelo de Planta 2 e saída do protótipo do controlador $\mathbf{C_2}$.

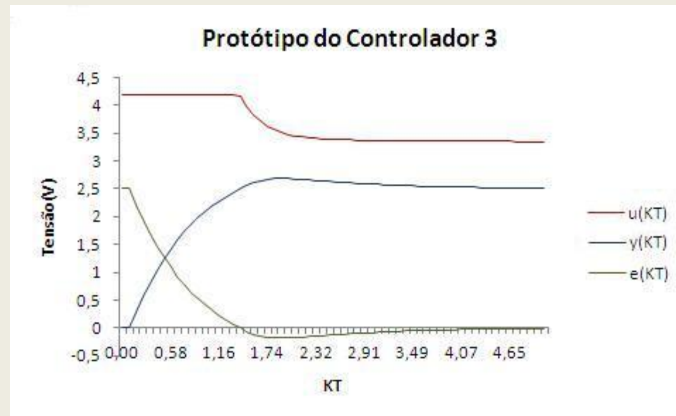


Figura 4.17: Resposta do modelo de Planta 3 e saída do protótipo do controlador $\mathbf{C_3}$.

De acordo com os resultados da simulação dos sistemas de controle com os modelos de plantas mostrados nos gráficos das Figuras 4.15, 4.16 e 4.17 e os protótipos dos controladores ($\mathbf{C_1}$, $\mathbf{C_2}$, $\mathbf{C_3}$), as saídas dos sistemas atendem aos requisitos especificados. Portanto, os protótipos dos controladores podem ser utilizados para controlar as plantas reais.

4.3.2 Simulação do sistema de controle com plantas reais

Para simular os sistemas de controle com as plantas reais as seguintes alterações foram realizadas na aplicação mostrada no Apêndice A.10:

1. Desativação das tarefas referentes aos modelos de plantas;
2. Atribuição do valor verdadeiro ao atributo IsRPlanta *(`Pid.SetIsRealPlanta (This => C1, newIsRPlanta => True)`)*;

Os gráficos da Figuras 4.18, 4.19 e 4.20 mostram as respostas das plantas y(KT), o sinal de controle u(KT) e o erro e(KT) durante a simulação dos sistemas de controle com as plantas reais por um intervalo de tempo igual a 5 segundos.

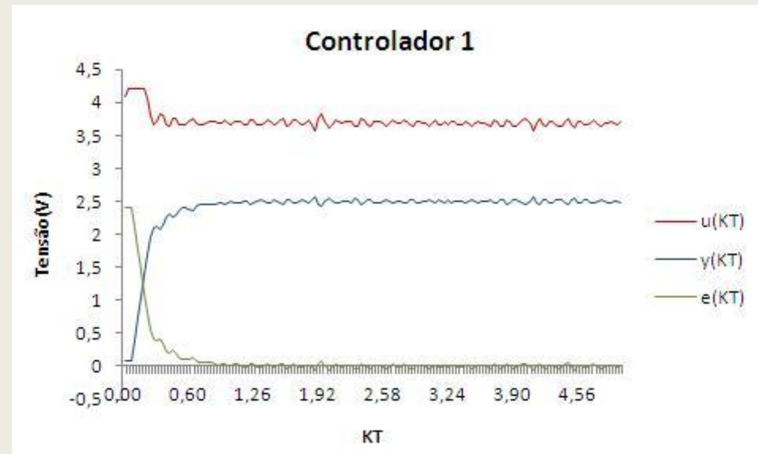


Figura 4.18: Resposta da Planta real 1 e saída do controlador $\mathbf{C_1}$.

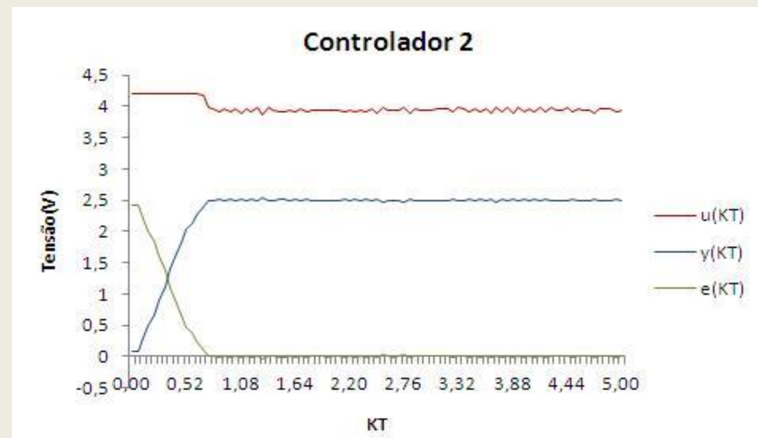


Figura 4.19: Resposta da Planta real 2 e saída do controlador $\mathbf{C_2}$.

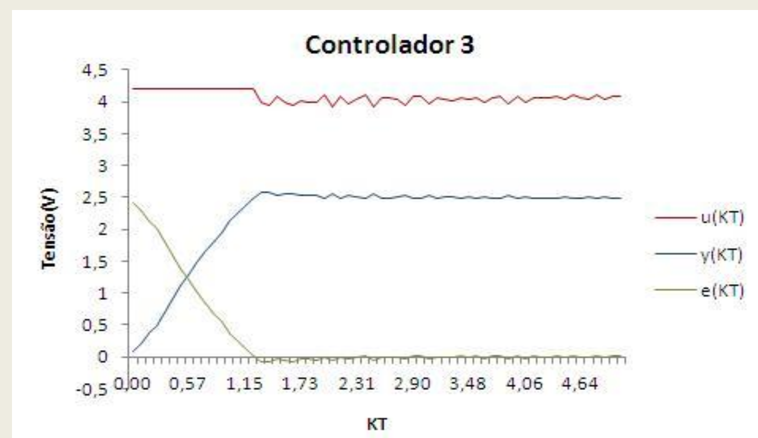


Figura 4.20: Resposta da Planta real 3 e saída do controlador $\mathbf{C_3}$.

De acordo com os resultados mostrados nos gráficos das Figuras 4.18, 4.19 e 4.20, as respostas das plantas em regime permanente apresentam oscilações em torno do valor do sinal de referência (2,5 volts). As oscilações foram causadas pelo mau alinhamento das bases dos motores na plataforma. Desconsiderando as oscilações os requisitos do sistema de controle foram alcançados para o três controladores.

Conclusão

Neste exemplo, foi possível verificar que a arquitetura de *framework* RCP proposto para desenvolvimento de sistema de controle centralizado será eficiente no projeto do *software* de controle. Fica evidente através do exemplo a facilidade de transição entre o protótipo do controlador utilizado para controlar os modelos de plantas durante para um controlador utilizado para controlar as plantas reais conectadas a plataforma, bem como, a geração do código para o *target*. No caso do exemplo apresentado, a transição foi realizada comentando a parte do código referente as plantas e alterando o valor do atributo IsRPlanta para *True*. O exemplo também possibilitou testar a capacidade da plataforma computacional durante a simulação com os modelos de plantas, pois a mesma foi realizada diretamente no *target*.

5 Resultados e análises

5.1 Método de compensação do *jitter* no intervalo de amostragem

Em (MARTI, P. & FUERTES, J. M., 2001) é proposto um método para compensar o *jitter* no intervalo ou período de amostragem. O método é aplicado para compensar o *jitter* em controladores do tipo PID. O cálculo do período de amostragem do controlador é realizado *on-line* durante a execução da tarefa de controle.

A Listagem 5.1 mostra a implementação do método proposto para compensar o *jitter* no período de amostragem baseado no método proposto por Marti e Fuertes (2001). O método de compensação consiste em atribuir ao período de amostragem da tarefa de controle o valor médio dos últimos n períodos de amostragem que são armazenados durante as sucessivas ativações da tarefa. O método deve ser aplicado na implementação de *subtask scheduling*. O valor do período de amostragem é calculado em *Update State* e utilizado na próxima ativação da tarefa.

Listagem 5.1: Implementação em Ada 2005 do método de compensação do $\mathbf{J_h}$.

```
procedure CompensaJitterAmostragem(This: PPid) is
    pragma inline(CompensaJitterAmostragem);
Begin
    -- se número de ativações da tarefa for maior que um
    if This.k > 1 then
        -- retorna o valor da média dos períodos entre o inicio e o fim da fila
        This.Periodo:= Fifo.Media(This.CFifo);
        -- calcula os novos parâmetros do controlador
        This.T := Float(To_Duration(This.Periodo));
        This.Par.Kd := This.Par.Kdc / This.T;
```

```
            This.Par.Ki := This.Par.Kic * This.T;
        end if;
    end CompensaJitterAmostragem;
```

O algoritmo utiliza uma estrutura de fila circular do tipo FIFO (*First In First Out*) para armazenar os últimos n períodos de ativação da tarefa. Os períodos são armazenados no início do intervalo de ativação da tarefa. A média dos períodos é calculada a partir da segunda ativação da tarefa. Uma vantagem do algoritmo em relação ao proposto por Marti e Fuertes (2001) ocorre na liberação da saída do controlador que não será atrasada pelos cálculos dos novos parâmetros do controlador.

Aplicação do método

A aplicação do método de compensação do *jitter* no intervalo de amostragem foi realizada implementado o mesmo como um método privado da classe PID do FCC mostrado na Listagem 5.1. Os três casos a seguir foram considerandos para analisar o método:

1. caso ideal: ausência de *jitter* no intervalo de amostragem;
2. sem compensação: presença de *jitter* no intervalo de amostragem e sem compensação;
3. com compensação: presença de *jitter* no intervalo de amostragem e com compensação;

Todos os casos foram simulados utilizando os conjuntos de tarefas $\Gamma_{subtask}$ e Γ_{planta}. Os valores dos atributos para os conjuntos de tarefas são mostrados nas Tabelas 5.3 e 5.4. O *jitter* de amostragem, presente nos casos 2 e 3, foi modelado como uma variável aleatória uniformemente distribuída no intervalo [-6 .. 6] ms e incorporado ao tipo ThreadPid na aquitetua

do FCC como um atributo. O valor do *jitter* gerado foi adicionado ao período de amostragem das tarefas durante as sucessivas ativações da mesma.

Os desempenho dos controladores (**C_1, C_2 e C_3**) são mostrados na Tabela 5.1 para o índice de desempenho ITSE durante um período de simulação igual a 5 segundos. Os valores mostrados na tabela comprovam a deterioração do desempenho dos sistemas de controle na presença do *jitter* no intervalo de amostragem.

Tabela 5.1: Valores do índice ITSE durante a análise do método de compensação do $\mathbf{J_h}$.

Subtask	$ITSE_{C1}$	$ITSE_{C2}$	$ITSE_{C3}$
Caso Ideal	1,864398	4,900903	8,540793
Sem compensação	2,419763	6,055632	10,876456
Com compensação	2,013313	5,299021	8,785764

Análise do método de compensação

Os resultados obtidos para os três controladores, quando implementados utilizando o método de compensação mostraram que o desempenho do sistema melhorou e se aproximou do caso ideal. Porém, estes resultados são relativos a um *jitter* no período de amostragem com função de distribuição uniforme. A análise do método para *jitters* com outros tipos de distribuição, bem como, a proposta de novos métodos de compensação em tempo de execução visando relaxar as retrições de tempo das tarefas de controle ficam como trabalho futuro.

5.2 Cálculos do WCET das tarefas controlador e planta

Na seção 2.6, foi enfatizada a importância da predição do WCET das tarefas no projeto de sistemas de tempo real para a análise da escalonabilidade do conjunto de tarefas. Segundo Shaw (1989), uma maneira prática de calcular o WCET de uma tarefa utilizando a análise no

nível de estrutura é aplicar o método padrão. O método padrão consiste em executar o código da tarefa para um determinado conjunto de entradas.

Na Listagem 5.2 é mostrada a implementação de um algoritmo proposto para medir os tempos de execução de tarefas de controle em SOTR utilizando análise estruturada e aplicando o método padrão.

Listagem 5.2: Implementação Ada 2005 do método para cálculo do WCET.

```
function CalculaWCET(This: PPid; erro,ref: PVector; t, ty: PTime) return float is
    NumExecucoes : Integer := erro'Length-1;
     next,clkatual : Time :=Clock ;
begin
    for j in 1..t'Length - 1 loop
     -- captura o instante de tempo atual
     t(j) := Clock;
     -- executa o código de inicialização da tarefa
     inicializa(This);
     -- executa o código da tarefa
     for i in 1..NumExecucoes loop
       ty(This.k) := Clock;
       case This.implementacao is
         when convencional =>
             -- codigo da implementação convencional
             next := next + This.Periodo;
             erro(This.k) := ref(This.k) - ReadInPut(This);
             This.u := Codigo(This,erro);
             -- escreve a saída
             WriteOutput(This);
             This.k := This.k + 1;
         when subtask_update_state =>
             -- seta prioridade para Update state
             Set_Priority(This.Pus);
```

```
            -- executa o a parte do código para update state
            UpdateState(This,erro,ty);
         when subtask_calculate_output =>
            -- executa a parte do código para Calculate Output
            next := next + This.Periodo;
            -- seta a prioridade para Calculate Output
            Set_Priority(This.Pco);
            erro(This.k) := ref(This.k) - ReadInPut(This);
            -- Calcula a saida
            This.u := CalculateOutput(This,erro);
            -- Escreve a saida
            WriteOutput(This);
        end case;
      end loop;
   end loop;
   t(t'Length) := Clock;
   -- seleciona o máximo WCET medido
   return IO.SelecionaMaximoWCET(t,NumExecucoes)/Float(NumExecucoes);
end CalculaWCET;
```

O algoritmo proposto executa o código das tarefas em suas sucessivas ativações por um número de vezes igual à *NumExecucoes*. Os tempos de execução (t_j) das *NumExecucoes* do código das tarefas são armazenados no vetor de tempo (t). Ao final do armazenamento das amostras de tempo, a função *IO.selecionaMaximoWCET* calcula valor do intervalo máximo D_{max} entre as amostras de tempo armazenadas, onde:

$$D_{max} = maximo(t_i - t_{i-1}), \text{ para } i := 2..t'Length.$$

O valor retornado ao final da execução de *CalculaWCET* é o WCET (D_{max}/*NumExecucoes*).

O algoritmo foi utilizado para medir o WCET da tarefa “planta” e das implementações convencional e *subtask* das tarefas “controlador” nas quatro situações a seguir:

1. *subtask* com compensação do *jitter* no intervalo de amostragem;
2. *subtask* sem compensação do *jitter* no intervalo de amostragem;
3. *subtask* com alteração de prioridades em *Calculate Output* e *Update State*;
4. *subtask* sem alteração das prioridades em *Calculate Output* e *Update State*;

A implementação mostrada na Listagem 5.2 é o método da classe PID do FCC. O cálculo do WCET da tarefa “planta” foi realizado através da sobrecarga do método na classe Planta1.

A precisão das medidas do WCET utilizando o algoritmo depende do número de interações executadas no segundo ***“loop for”*** e do tamanho do *tick* configurado para o sistema. Um *tick* é a menor unidade de tempo que pode ser medida pelo sistema operacional, seu tamanho ótimo depende da freqüência do *clock* (f_{clk}) do sistema. O valor da f_{clk} determina o número de interrupções de *clock* que ocorrerão em um segundo. Valores pequenos para a f_{clk} pode causar erros de medida no cálculo do WCET quando o método prático é utilizado. Um exemplo de erro da medida causado pela seleção de um valor baixo para a f_{clk} é mostrado na primeira linha da Tabela 5.2. No exemplo, foi utilizada uma f_{clk} igual a 100 HZ. O valor do WCET para o código convencional executado pela tarefa de controle foi calculado igual a 1 *tick* (10 ms), mesmo tendo um valor real menor que 1 *tick*.

Outro fator importante relacionado ao tamanho do *tick* do sistema é determinar o período mínimo teórico (T_{min}) para o qual uma tarefa de controle possa ter seus requisitos de tempo atendidos. Quando controladores são implementados em SOTR com escalonadores dirigidos por interrupção do *clock*, a cada intervalo de um *tick*, ocorre uma interrupção do *clock* e o escalonador verifica as filas de processos do sistema para determinar qual tarefa deve ser executada pelo processador. Encontrar o valor T_{min} pode ser realizado como no

exemplo seguinte: seja a freqüência máxima de *clock* (f_{clkmax}) suportado pelo sistema igual 5 kHz, neste caso o valor T_{min} de uma tarefa de será igual a 200 microssegundos.

Na Tabela 5.2 são mostrados os valores do WCET das tarefas "controlador" e "planta" para o número de amostras de tempo (*t'Length*) igual a 101 e *NumExecucoes* igual a 1000. A execução da tarefa 1000 vezes e a amostragem de tempo no início e no fim das 1000 execuções melhora a precisão das medidas e reduz o valor do erro cometido. Porém, o WCET calculado é o WCET médio das 1000 execuções da tarefa com um erro igual à *tick*/2000. O valor do WCET para uma ativação pode ser realizado utilizando *NumExecucoes* igual a 1. Neste caso, a precisão máxima da medida será igual a um *tick* e o WCET medido será qualquer valor no intervalo [WCET + *tick*/2..WCET - *tick*/2] onde, *tick*/2 é o erro da medida. Assim, qualquer valor de WCET menor que 1 *tick* será calculado igual a 1 *tick*.

Tabela 5.2: Valores dos WCET das implementações das tarefas controlador e planta.

Código	WCET[ms]	Compensa *Jitter*	Altera Prioridade	fclk [HZ]
Convencional	10			100
Convencional	0,0082			5000
Calculate Ouput	0,0142		Sim	5000
Calculate Ouput	0,0068		Não	5000
Update State	0,0492	Sim	Sim	5000
Update State	0,037	Sim	Não	5000
Update State	0,0074	Não	Sim	5000
Update State	0,0024	Não	Não	5000
Planta	0,007			5000

De acordo com os valores mostrados na Tabela 5.2, o tempo médio de execução do pior caso para a execução de *Update State* (49,2 μ*s*), corresponde a sua execução com compensação do *jitter* e alteração de prioridade durante a execução. A alteração de prioridade, *Set_Priority(This.Pus)*, é necessária quando se implementa *subtask scheduling* utilizando uma

única tarefa para executar as partes *Update State* e *Calculate Output*. Este também é o pior caso para a execução de *Calculate Output* (14,2 μ*s*). Para a execução do código convencional o tempo médio de pior caso calculado foi igual a 8,2 μ*s* e para a planta igual a 7,2 μ*s*.

5.3 Análises de *subtask scheduling* e dos escalonadores FP e RR

Para analisar a metodologia *subtask scheduling* com e sem compensação do *jitter* no intervalo de amostragem e os algoritmos de escalonamento de processo, foram utilizadas os modelos de plantas da plataforma RCP mostrados na Tabelas 4.2 e os controladores PI projetados para as mesmas (Tabela 4.4). A partir dos dados das plantas e controladores foram gerados os três conjuntos de tarefas a seguir:

$\Gamma_{convencional} = \{\tau_{C1}, \tau_{C2}, \tau_{C3}\}$, conjunto de tarefas controlador implementadas de maneira convencional, sem aplicação de *subtask scheduling* e sem compensação do *jitter* no intervalo de amostragem;

$\Gamma_{subtask} = \{\tau_{CO1}, \tau_{CO2}, \tau_{CO3}, \tau_{US1}, \tau_{US2}, \tau_{US3}\}$, conjunto de tarefas controlador implementadas aplicando a metodologia *subtask scheduling* utilizando tarefa única para executar as partes *Calculate Output* e *Update State* com compensação do *jitter* no intervalo de amostragem e sem compensação do *jitter* no intervalo de amostragem.

$\Gamma_{planta} = \{\tau_{P1}, \tau_{P2}, \tau_{P3}\}$, conjunto de tarefas implementadas para executar as equações de diferenças das plantas;

Na Tabela 5.3 são mostrados os valores dos atributos período de ativação (T), tempo de computação (C), prioridade (P) e *deadline* (D) para os conjuntos de tarefas Γ_{planta} e $\Gamma_{convencional}$. Os valores das prioridades (P) e dos *deadlines* (D) para as tarefas dos conjuntos

foram geradas aplicando o algoritmo *Rate Monotonic*. Os tempos de computação (C) foram obtidos a partir da Tabela 5.2.

Tabela 5.3: Valores dos atributos dos conjuntos de tarefas Γ_{Planta} **e** $\Gamma_{convencional}$.

Tarefas	T[ms]	C[ms]	P	D[ms]
τ_{C1}	33	0,0082	10	33
τ_{C2}	56	0,0082	9	56
τ_{C3}	83	0,0082	8	83
τ_{P1}	33	0,007	10	33
τ_{P2}	56	0,007	9	56
τ_{P3}	83	0,007	8	83

Os valores dos atributos para o conjunto de tarefas $\Gamma_{subtask}$ são mostrados na Tabela 5.4. O tempo de computação (C_{jh}) corresponde a execução da tarefa com compensação do *jitter* no intervalo de amostragem e alteração de prioridades em *Calculate Output* e *Update State*. O tempo de computação (C) corresponde à execução da tarefa sem compensação do *jitter* no intervalo de amostragem e com alteração das prioridades em *Calculate Output* e *Update State*.

Tabela 5.4: Valores dos atributos para os conjuntos de tarefas $\Gamma_{subtask}$.

Tarefas	T[ms]	C[ms]	C_{jh}[ms]	P	D[ms]
τ_{CO1}	33	0,0142		10	32
τ_{CO2}	56	0,0142		9	55
τ_{CO3}	83	0,0142		8	82
τ_{US1}	33	0,0074	0,0492	7	33
τ_{US2}	56	0,0074	0,0492	6	56
τ_{US3}	83	0,0074	0,0492	5	83

Os valores das prioridades (P) e dos *deadlines* (D) das tarefas do conjunto mostrados na Tabela 5.4, foram atribuídos de acordo com o modelo de restrição de prioridades considerando que, as prioridades atribuídas às partes *Calculate Output* (τ_{CO}) em todas as tarefas do conjunto, devem ser maiores que as prioridades atribuídas às partes *Update State* (τ_{US}).

5.3.1 Análise da metodologia *subtask scheduling*

Implementações

As quatro diferentes implementações de tarefas, para os controladores $\mathbf{C_1}$, $\mathbf{C}_2$ e $\mathbf{C_3}$, mostradas abaixo foram utilizadas para analisar a metodologia *subtask scheduling*:

1. Convencional, DT: implementação convencional discretizando o termo integral utilizando aproximação por diferenças para trás;
2. Convencional, DF: implementação convencional discretizando o termo integral utilizando aproximação por diferenças para frente;
3. *Subtask*, SCJ_h: implementação de *subtask scheduling* sem compensação do *jitter*;[5]
4. *Subtask*, CCJ_h: implementação de *subtask scheduling* com compensação do *jitter*;[6]

O código da tarefa controlador quando implementada de maneira convencional ou com *subtask* é mostrado no Apêndice A.11. Na implementação convencional (*This.Implementacao = convencional*), a saída do controlador é liberada após todas as instruções do algoritmo serem executadas aumentando a latência de entrada e saída. Na

[5] Neste caso o termo integral deve ser discretizado utilizando aproximação por diferenças para frente.
[6] Neste caso o termo integral deve ser discretizado utilizando aproximação por diferenças para frente.

implementação utilizando *subtask* (*This.Implementacao = subtask*), a saída do controlador é liberada o mais rápido possível ao final da execução de *Calculate Output*.

As tarefas controladores e plantas implementadas para simular as quatro situações foram executados na plataforma RCP centralizada. O tempo de simulação das tarefas plantas foi configurado para 5 segundos. Para a f_{clk} foi utilizado um valor de 5kHz.

Na Tabela 5.5 são mostrados os valores medidos para o índice desempenhos ITSE dos controladores $\mathbf{C_1}$, $\mathbf{C_2}$ e $\mathbf{C_3}$**.**

Tabela 5.5: Desempenho dos controladores para as implementações convencional e *subtask*.

Implementação	$ITSE_{C1}$	$ITSE_{C2}$	$ITSE_{C3}$
Convencional, DF	1,662151	4,861353	8,481887
Convencional, DT	1,864398	4,900903	8,540793
Subtask, SCJ_h	1,864398	4,900903	8,540793
Subtask, CCJ_h	1,864398	4,900903	8,540793

Resultados

De acordo com os resultados apresentados na Tabela 5.5, o melhor desempenho para os três controladores foi obtido utilizando a implementação convencional e aproximando o termo integral por diferenças para frente. O desempenho para a implementação convencional aproximando o termo integral por diferenças para traz e as duas implementações utilizando *subtask*, com compensação e sem compensação do *jitter* apresentaram o mesmo desempenho.

Análise dos resultados

A aplicação de *subtask scheduling* na implementação dos três controladores não melhorou o desempenho do sistema de controle, mas atrasou a saída do termo integral em um período de

ativação degradando o desempenho, uma vez que, a saída do termo integral calculado só foi utilizada na ativação seguinte.

Quanto à implementação de *subtask* em conjunto com o método de compensação do *jitter* proposto, também não melhorou o desempenho dos controladores. Neste caso, a média dos períodos de ativação dos controladores foi igual ao período de ativação atribuído inicialmente aos controladores. Embora, o resultado não seja útil para analisar o método de compensação de *jitter*, na presença do *jitter* gerado pelo SOTR, o mesmo mostrou que o SOTR VxWorks é capaz de fornecer um ambiente determinístico dependente do valor da f_{clk} ser múltiplo dos períodos de ativação de todas as tarefas de controle.

5.3.2 Análise dos escalonadores FP e RR

Definição dos conjuntos de tarefas

Para analisar os algoritmos de escalonamento de processos foram definidos três conjuntos de tarefas (Γ_1, Γ_2, Γ_3). Todos os conjuntos são compostos por instâncias (i) das tarefas do conjunto $\Gamma_{convencional}$. Uma tarefa τ_{Cij} de um conjuntos é identificada pelos índices (**ij**) onde, **i** identifica o i-ésimo controlador representado pelo modelo de tarefa e **j** identifica a j-ésima instância da tarefa. Por exemplo, a tarefa τ_{C13} é a terceira instância do controlador **C₁**.

$\Gamma_1 = \{\tau_{Cij}\}$, para i:= 1..3 e j = 1;

$\Gamma_2 = \{\tau_{Cij}\}$, para i:= 1..3 e j := 1..3;

$\Gamma_3 = \{\tau_{Cij}\}$, para i:= 1..3 e j = 1..6;

Configuração do sistema

As quatro configurações abaixo foram utilizadas, para o SOTR, durante a simulação dos conjuntos de tarefas:

1. Algoritmo de escalonomento FP e f_{clk} igual a 1 kHz;
2. Algoritmo de escalonamento FP e f_{clk} igual a 5 kHz;
3. Algoritmo de escalonomento RR e tamanho do *quantum* igual a 1 ms;
4. Algoritmo de escalonamento RR e tamanho do *quantum* igual a 0,2 ms;

Os valores da f_{clk} iguais a 1kHz e 5 kHz, no caso do algoritmo FP, foram utilizadas para evitar a degradação de desempenho do sistema pelo *jitter* de amostragem. Porém, qualquer f_{clk} cujo tamanho do *tick* seja múltiplo dos períodos de ativação das tarefas controlador poderia ter sido utilizada. Os quatum de 1ms e 0,2 ms, no caso do algoritmo RR, são os equivalentes a 1 *tick* quando a f_{clk} é igual a 1kHz e 5 kHz.

Casos analisados

Os três casos a seguir foram simulados nos ambientes descritos acima utilizando os respectivos conjuntos tarefas. O tempo de simulação para as tarefas planta foi igual a 5 segundos.

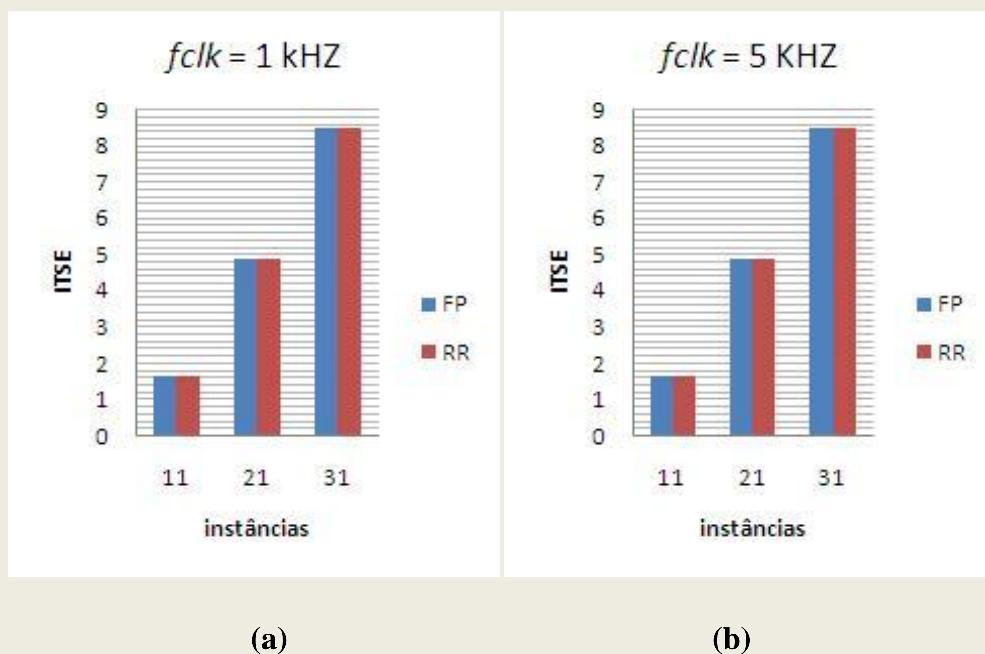
- Caso 1: o sistema executa o conjunto de tarefas Γ_1, corresponde ao caso ideal onde, o desempenho dos sistemas de controle é o máximo para o acionamento simultâneo das três plantas conectadas à plataforma RCP;
- Caso 2: o sistema executa o conjunto de tarefas Γ_2, neste caso a carga do sistema foi triplicada em relação ao Caso 1;

- Caso 3: o sistema executa o conjunto de tarefas Γ_3, neste caso a carga do sistema foi duplicada em relação ao Caso 2;

Resultados

Os resultados para os três casos são mostrados nos gráficos das Figuras 5.1, 5.2 e 5.3. Os gráficos mostram os valores dos índices ITSE para as instâncias dos controladores escalonadas pelo algoritmo FP e RR no eixo y. Os índices das instâncias das tarefas de controle são mostradas no eixo x. O tamanho do *quantum* utilizado durante a simulação com o RR é igual um período da f_{clk} identificada no título do gráfico.

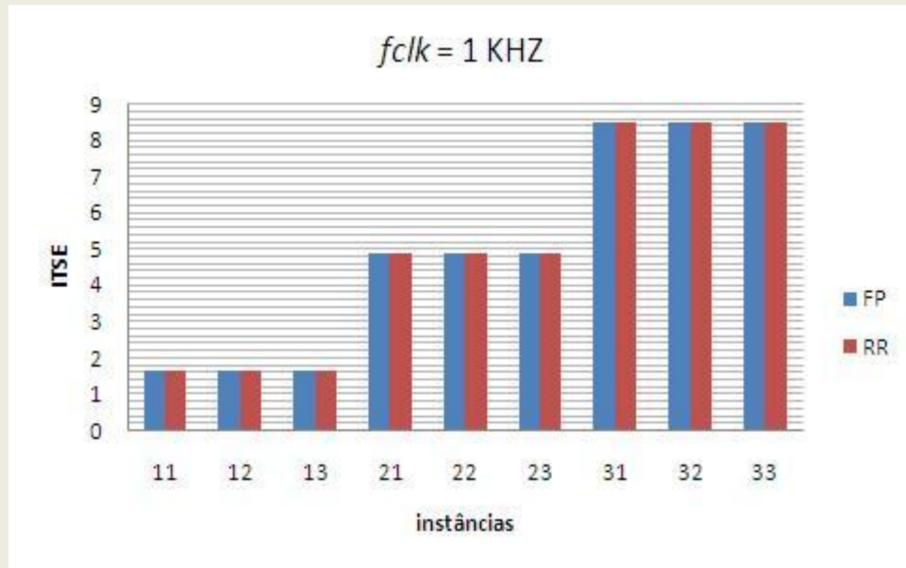
Caso 1:



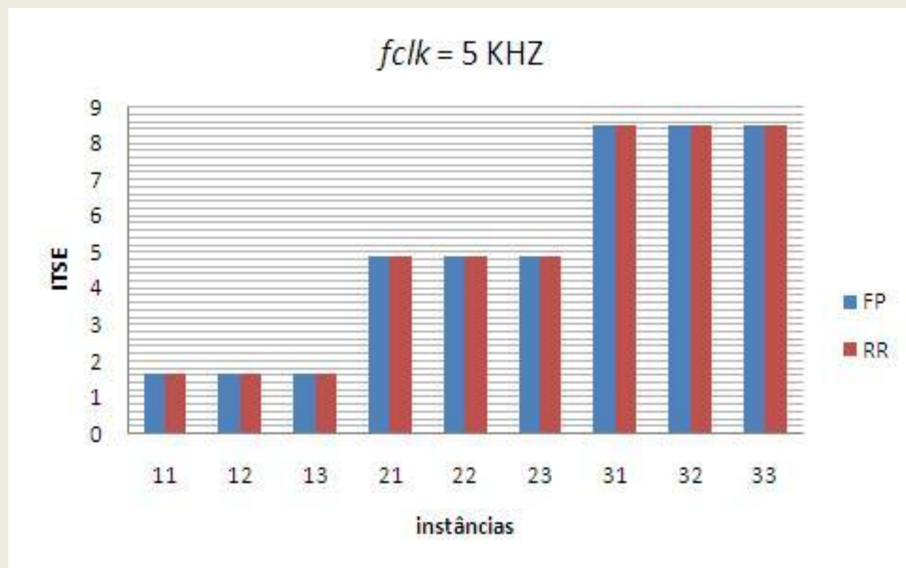
(a) **(b)**

Figura 5.1: Desempenho ITSE das instâncias do conjunto de tarefas Γ_1 para os algoritmos FP e RR: (a) f_{clk} = 1kHz e *quantum* = 1 ms; (b) f_{clk} = 5kHz e *quantum* = 0,2 ms.

Caso 2:



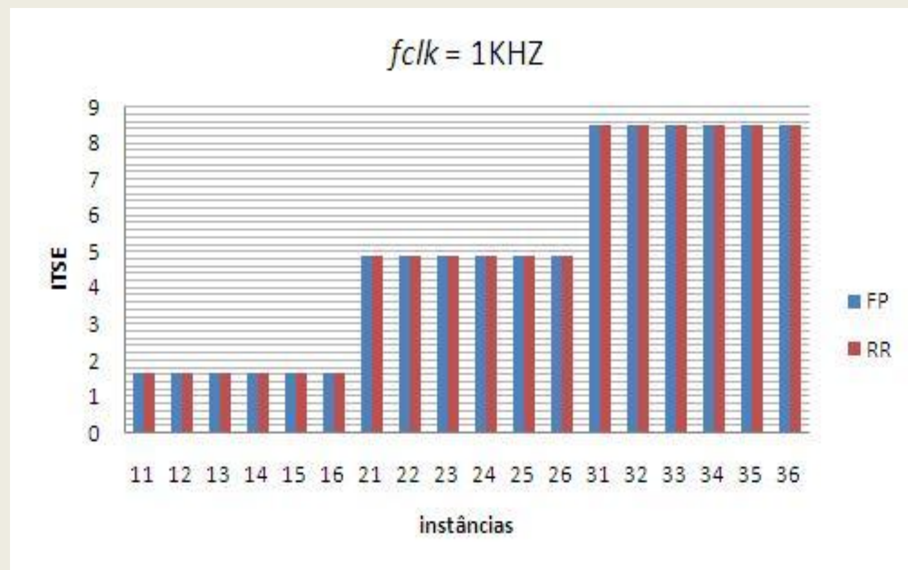
(a)



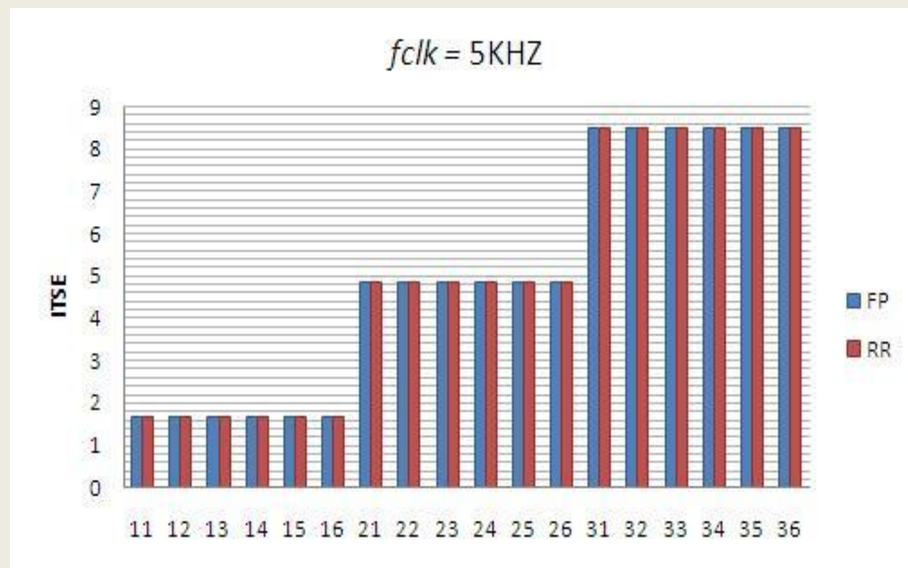
(b)

Figura 5.2: Desempenho ITSE das instâncias do conjunto de tarefas Γ_2 para os algoritmos FP e RR: (a) f_{clk} = 1kHz e *quantum* = 1 ms; (b) f_{clk} = 5kHz e *quantum* = 0,2 ms.

Caso 3:



(a)



(b)

Figura 5.3: Desempenho ITSE das instâncias do conjunto de tarefas Γ_3 para os algoritmos FP e RR: (a) f_{clk} = 1kHz e *quantum* = 1 ms; (b) f_{clk} = 5kHz e *quantum* = 0,2 ms.

Análise dos resultados

Os resultados da simulação para os três casos aplicando as configurações descritas para o SOTR mostram que o desempenho dos algoritmos FP e RR são muito próximos. Praticamente imperceptivéis através dos gráficos, para os respectivos conjuntos de tarefas. A alteração da f_{clk} para o FP ou do tamanho do *quatum* para o RR não alterou o desempenho das instâncias em nenhum dos casos analisados.

A análise dos efeitos do aumento da carga computacional foi realizada comparando o desempenho de cada uma das instâncias das tarefas de controle com o desempenho das instâncias do Caso 1 para ambos os escalonadores (FP e RR), nas repectivas f_{clk} e tamanho do *quatum*. Nestes casos, também não houve mudanças consideravéis no desempenho individual das instâncias das tarefas de controle em relação as instâncias do Caso 1.

Os resultados obtidos permitem concluir que os algoritmos de escalonamento FP e RR presentes no VxWorks possuem desempenho semelhantes. Porém, as seguintes considerações devem ser feitas sobre os resultados aqui apresentados:

- Os tempos de computação (C) das tarefas controlador e plantas são muito menores que o menor tamanho do *quantum* (0,2 ms), configurado para o RR. Assim, o comportamento dos algoritmos de escalonamento RR e FP são praticamente iguais quando analisamos as trocas de contexto. A probabilidade de ocorrer uma troca de contexto antes do fim da execução da tarefa é muito pequena para os dois algoritmos. Em relação a implementação, o algoritmo RR presente no VxWorks é semelhante ao FP, pois não permite que uma tarefa execute até expirar o seu *quantum* quando uma tarefa mais prioritária chega a fila de Prontos.

- O aumento de carga não foi muito significativo, considerando o limite de utilização dos sistema quando aplicado o teste de escalonabilidade para o RM (Equação 2.1). Ainda que este seja um teste nescessário, é possivél conhecer através da análise *off-line* a carga máxima teórica aplicando o teste.
- A arquitetura de *framework* para controle centralizado proposta e implementada para desenvolver as aplicações, ainda não se encontra em um estágio que a torne útil para analisar a carga máxima suportada pela plataforma computacional. A realização deste tipo de análise depende da modelagem e implementação de coleções de objetos (*containers*) e métodos que permitam realizar operações como aumento de carga, através da inserção de novos elementos nos *containers* e analisar o desempenho de todas as instâncias presentes no *container* em tempo de execução. O aumento da carga é nescessário para conhecer o valor da carga máxima suportada pelo sistema e conseqüentemente determinar o desempenho máximo de cada algoritmo de escalonamento.

6 Conclusões

O projeto de sistemas de controle em plataformas de tempo real é uma tarefa que envolve as áreas de controle e tempo real. Nas últimas décadas, estas áreas se desenvolveram como áreas independentes. O desenvolvimento em paralelo contribuiu para aumentar a distância entre as áreas. Neste trabalho buscou-se reduzir esta distância propondo a utilização de métodos de Engenharia de *Software* em uma fase de modelagem do *software* de controle.

Uma das propostas apresentadas no trabalho foi a utilização de *frameworks* orientados a objetos durante o processo de desenvolvimento do *software* de controle aplicando *Rapid Control Prototyping* para substituir a geração automática de código, eliminar os problemas de integração com código legado e tornar o processo mais interativo. Entre as vantagens da utilização de *frameworks* na fase de modelagem do *software* de controle no processo RCP temos: reaproveitamento do código e dos projetos, eliminação dos problemas causados pela conversão automática do código gerado no *host* para o *target*, possibilidade de desenvolver e analisar novos métodos para o tratamento dos atrasos e *jitters* na malha de controle, possibilidade de analisar diferentes formas de implementar a malha de controle, etc..

Como desvantagens podem ser citadas: necessidade de conhecer o domínio da aplicação para modelar o *framework*, dificuldade de aprender a utilizar *frameworks*, alto custo de desenvolvimento do *framework*, etc.

As arquiteturas de *frameworks* propostas foram implementados e embarcados em uma plataforma para RCP composta pelo *hardware* do *target* (PC-104) e o SOTR (VxWorks).

A utilização da plataforma possibilitou realizar as seguintes análises durante a prototipagem do sistema de controle: avaliação se o *hardware* do *target* e o SOTR selecionados são capazes de atender aos requisitos do projeto dos controladores, estudo de

diferentes políticas de escalonamentos, comparação do desempenho dos protótipos desenvolvidos quando os mesmo atuam sobre modelos de plantas com o desempenho quando os protótipos dos controladores atuam sobre as plantas reais conectadas as interfaces de entrada e saída da plataforma.

Porém, é importante ressaltar que apesar das vantagens apresentadas ao se utilizar a plataforma proposta, o custo do projeto é elevado devido o custo da aquisição do *hardware* e do SOTR, das dificuldades em encontrar profissionais capacitados para embarcar o SOTR no *hardware* da plataforma, alterar e desenvolver novos módulos de *kernel* e ou *drivers* de dispositivos em linguagem C ou *assembly*, modelar e desenvolver aplicações de controle em linguagens de alto nível (Ada 2005, por exemplo). Assim, o sucesso de qualquer proposta de aproximação entre as áreas de controle e tempo real depende da capacidade dos profissionais envolvidos no projeto dos sistemas de controle em plataformas de tempo real ampliar seus horizontes de conhecimentos nas duas áreas.

6.1 Sugestões para trabalhos futuros

A metodologia de projeto para desenvolvimento de sistemas de controle em plataformas de tempo real apresentada neste trabalho indica que o sucesso no desenvolvimento depende da aplicação de métodos de Engenharia de *Software* para modelar o sistema de controle e da utilização de uma plataforma formada pelo *hardware* do *target* ou semelhante em conjunto com um SOTR.

Com base nesta filosofia de desenvolvimento, este trabalho pode ser expandido através da alteração da camada de aplicação da plataforma para RCP revendo as arquiteturas de *frameworks* propostas e propondo novas arquiteturas de *frameworks*. Em relação à revisão das arquiteturas de *frameworks*, podem ser propostos métodos adaptativos para compensar o *jitter*

e o atraso em malhas de controle centralizadas e distribuídas em rede baseados nos modelos de distribuição dos *jitters* e atrasos obtidos durante a execução das tarefas de controle; propostas e implementação de novos modelos de controladores tipo avanço atraso, adaptativos, etc.; modelagem e implementação de *containers* de tarefas e métodos que interajam sobre o conjunto de tarefas para solucionar problemas como o *co-design* entre o sistema controle e o escalonamento;

Para a camada SOTR embarcada na plataforma, podem ser propostos e implementados novos algoritmos de escalonamentos diferentes do FP e RR.

Outras possibilidades de novos trabalhos são a utilização de Algoritmos Genéticos (AGs) para desenvolver controladores adaptativos e a análise da taxa de amostragem ótima para os diferentes modelos de plantas.

Referências Bibliográficas

ALT, M. ET AL. Cache Behavior prediction by abstract interpretation. Science of Computer Programming, v. 35, p. 52--66, 1996.

ANDERSSON, M.; CERVIN, A. & HENRIKSSON, D. True Time 1.3 Reference Manual. Lund Institute of Technology. Lund, p. 107. 2005.

ÅRZÉN, K. ET AL. Integrated Control and Sheduling. Lund Institute of Technology. Lund, p. 51. 1999. (0280-5316).

ÅSTRÖM, K. J. & WITTENMARK, B. Computer-Controlled Systems: Theory and design. 3. ed. [S.l.]: Prentice-Hall, 1997.

BARNES, J. Programming in Ada 2005. 1rd edition. ed. [S.l.]: Addison-Wesley, 2006.

BLUM, A.; CECHTICKY, V. & PASETTI, A. A Java-Based Framework for Real-Time Control Systems, Zurich, 1 January 2010.

BOSTIC, D. ET AL. Automatic code generation requeriments for production automotive powertrain applications. IEEE International Symposium on Computer Aided Control System Design, Kohala Coast, Hawwii , USA, p. 200-206, 6 Agosto 2002. ISSN 0-7803-5500-8.

BOTERENBROOD, H. CANopen high-level protocol for CAN-bus NIKHEF. Amsterdam. 2000.

BRITO, R. & NAVET, N. Low Power Round-Robin Scheduling. 12ème Conférence Internationale sur les Systèmes Temps Réel. Nancy, France: [s.n.]. 2003. p. 20.

BURNS, A. & WELLINGS, A. J. Real-Time systems and Programming Language. 3rd edition. ed. [S.l.]: Addison Wesley, 2001. 611 p.

BURNS, A. & WELLINGS, A. J. Concurrent and Real-Time Programming in Ada. 1st edition. ed. [S.l.]: Addison-Wesley, 2007.

BURNS, A. AND WELLINGS, A. J. Real-Time systems and Programming Language. 3rd edition. ed. [S.l.]: Addison-Wesley, 2001.

BURNS, A.; TINDELL, K. & WELLINGS, A. J. . Fixed priority scheduling with deadlines prior to completion. InProceedings of the 6th Euromicro Workshop on Real-Time Systems, 1994. 138–142.

BURNS, A.; TINDELL, K. & WELLINGS, A. J. An extendible approach for analyzing fixed priority hard real-time tasks. Real-Time Systems, 1994. 133–151.

CERVIN, A. & LINCOLN, B. Jitterburg 1.21 reference manual. Lund Institute of technology. Lund, Sweden. 2006.

CERVIN, A. Integrated control and real-time sheduling, Phd. Thesis. Lund Institute of Techonology. [S.l.], p. 172. 2003. (0280-5316).

DOUGLAS, B. P. Real-Time UML: Developing Efficient Objects for Embedded Systems. [S.l.]: Addison Wesley, 2000.

DOUGLASS, B. P. Real-time design patterns robust scalable architecture for real-time systems. [S.l.]: Addison Wesley, 2002.

FARINES, J. ;FRAGA, J. S. & OLIVEIRA, R. S. Sistemas de tempo real. Florianópolis: [s.n.], 2000.

GARCEY, M. & JOHNSON, D. S. Computer and Intractability: a Guide to the Theory of the NP-Completeness. New York: W.H. Freeman and Company, 1979.

GERBER, R. & HONG, S. Semantic-based compiler transformations for enhanced schedulability. In Proceedings of the 14th IEEE Real-Time Systems Symposium, 1993. 232–242.

HÖLTTÄ, V.; PALMROTH, L. & ERICKSON, L. Rapid control Prototyping tutorial with application examples,Sim-Serv, www.sim-serv.com, 2004. 6.

JOHNSON, R. E. Documenting frameworks using patterns, New York, v. 27, 1992. ISSN 10.

JONG, E. Software Architecture for Large Control Systems: A case Study Description. Source. Lecture Notes In Computer Science, 1282 , 1997. 150-156.

KOPETZ, K. Scheduling. on Proceedings of Advanced Course on Distributed Systems, Estoril, Portugal, 1992.

KRAJNC, M. What is component-oriented programming? Zurich: Oberon Microsystems, 1997.

LEE,W. ; SHIN, M. & SUNWOO, M. Target-idetical rapid control prototyping platform for model-based engine control. Journal of Process Mechanical Engineering, Seoul, v. 218, p. 755-765, 26 mar. 2004.

LI ET AL. Efficient microarchitecture modeling and path analysis for real-time systems. In Proceedings of the IEEE Real-Time Systems Symposium, Pisa, 5 december 1995. 298 - 307.

LIU, C. & LAYLAND, J. W. Sheduling algorithms for multi-programming in a hard real-time environment. Journal of ACM, New York, USA, 1 January 1973. 46-61.

LUNDIQVIST, T. & STENSTROM, P. Integrating path and timing analysis using instruction-level simulation techniques. In Proceedings of ACM SIGPLAN Workshop on Languaged, Compilers and Tools for Embedded Systems, 1998. 1-15.

LUNDQVIST, T. & SANDIN, P. Towards a practical WCET analysis approach based on testing. 20th Euromicro conference on real-time systems (ECRTS'08 WIP), june 2008.

MARTI, P. & FUERTES, J. M. Jitter Compensation for Real-Time Control Systems. in 22nd IEEE Real-Time Systems Symposium. London: [s.n.]. 2001. p. 39 - 48.

MARTI, P. ; VILLA, R. & FUERTES,J. M. On Real-Time control tasks schedulability. European Control Conference. Porto: [s.n.]. 2001. p. 6.

MARTI, P. Analysis and Design of Real-Time control Systems with Varying Control Timing Constraints, Phd Thesis. Univesitat Politècnica de Catalunya. Barcelona, p. 169. 2002.

MARTI, P. ET AL. Jitter Compensation for Real-Time Control Systems. in 22nd IEEE Real-Time Systems Symposium. London: [s.n.]. 3-6 December 2001. p. 39 - 48.

MEYER, B. Object-oriented software construction. Englewood Cliffs: Prentice Hall, 1988.

NILSSON, J. Real Time Control Systems with Delay, Phd Thesis. Lund Institute of Techonology. Lund, p. 128. 1998. (0280-5316).

OGATA, K. Discrete-Time Control Systems. 3rd edition. ed. [S.l.]: Prentice-Hall, , 1997.

PARK, C. Y & SHAW, A. C. Experiments with a Program Timing Tool Based on Source-Level Timing Schema. In Proc. 11 th IEEE Real-Time Systems Symposium, December 1990. 72-81.

PETTERS, S. M. & FARBER, G. Making Worst Case Execution Time Analysis for Hard Real-Time Tasks. Sixth International Conference on Real-Time Computing Systems and Applications, 1999. RTCSA '99., Hong Kong , China, 1999. 442 - 449.

PINHO, L. M. R. S. A Framework for Transparent Replication of Real-Time Applications, Phd Thesis. Faculdade de Engenharia da Universidade do Porto. Porto, p. 174. 2001.

POHJOLA, M. Pid Controller Design in Networked Controle Systems, Master's Thesis. Helsinki University of Technology. Espoo, Finland, p. 83. 2006.

SHAW, A. C. Reasoning about time in higher-level language software. IEEE Transactions on Software Engineering, July 1989. 875-889.

SILVA, R. P. Suporte ao desenvolvimento e uso de frameworks e componentes, Tese de doutorado. UFRGS. Porto Alegre, p. 262. 2000.

SOARES, L. F. ; LEMOS, G. & COLCHER, S. Redes de computadores: Das Lans, Mans e Wans às redes ATM. Rio de Janeiro: Editora Campus, 1995.

SZYPERSKI, C. E. A. Sumary of the First International Workshop on Component-Oriented Programming. International Workshop on Component-Oriented Programming. Linz: [s.n.]. 1996.

TAKARABE, E. W. Sistemas de controle distribuído em redes de comunicação, Dissertação de Mestrado. USP - Universidade de São Paulo. São Paulo, p. 101. 2009.

TIPSUWAN, Y. & MON-YUEN, C. Control methodologies in networked control systems. Control Engineering Practice, 11, 16 February 2003. 1099-1111.

TÖRNGREN, M. H. Fundamentals of implementing real-time control applications in distributed computer systems. Real-Time Systems, 1998.

VxWorks Kernel Programmers guide 6.6, Wind River Systems. Alameda, CA 94501-4100, USA, p. 834. 2007.

WARD, P. A. M. S. J. Fundamentals of implementing Real-Time Systems, v. IV, Yourdon Press, 1985.

WHITE, R. T. ET AL. Timing Analysis for data caches and set-associative caches. In Proceedings of the IEEE Real-Time Technology and Applications Symposium, 1997. 48-57.

Apêndice A - Código Ada 2005 das classes que compõem as arquiteturas dos *frameworks*

Este apêndice mostra partes das implementações das classes que compõem as arquiteturas dos *frameworks* FCC e FCD em Ada 2005. A principal motivação para se utilizar a linguagem Ada 2005 na implementação dos *frameworks* para RCP se deve ao fato de Ada 2005 ser a única linguagem de programação padronizada (ISO/IEC 8652:1995/Amd 1:2007) para tempo real, orientação a objetos e concorrência. Ada foi projetada para ser utilizada no desenvolvimento de grandes aplicações de longa vida e aplicações críticas onde segurança e eficiência são essenciais. Particularmente aplicações de tempo real e sistemas embarcados**.**

As vantagens da utilização de Ada 2005 no desenvolvimento de sistemas de controle em detrimento a linguagens como C++ e Java, por exemplo, é que Ada 2005 possui a segurança e portabilidade de Java e a eficiência e flexibilidade de C++. Além da vantagem de ser um padrão internacional com semânticas bem claras e definidas.

Listagem A.1: Implementação da classe abstrata TPlanta derivada de TarefaPeriodica no FCC.

```
package FCC.Plantas.Planta is

   -- definição do tipo abstrato TPlanta
   type TPlanta is abstract new TarefaPeriodica with
      record
         y : Float;      -- saida atual da planta
         k : Integer;    -- k-esima ativação
         On: Boolean := False;
      end record;

   type PPlanta is access all TPlanta'Class;
```

```
task type ThreadPlanta(This: PPlanta; TS: Integer;Name: PString) is
      entry Start;
      pragma Priority(This.Prioridade);
   end ThreadPlanta;

   -- Codigo que será executado na inicialização da planta
   procedure Inicializa(This: PPlanta; y: PVFloat) is abstract;
-- Codigo que será executado periodicamente pela threadMyPlanta
   procedure Codigo(This: PPlanta; y , u: PVFloat) is abstract;
   -- Le a saida do controlador
   function ReadInput(This: PPlanta) return Float is abstract;
   -- Escreve a saida da planta
   procedure WriteOutput(This: PPlanta) is abstract;
   -- calcula tempo de execução de pior caso (WCET)
function   CalculaWCET(This:   PPlanta;   y,u:   pVFloat;   t:   PTime)
return float is abstract;
   -- libera a memoria alocada pelo objeto
   procedure Free_Planta(This: PPlanta)is abstract;

end FCC.Plantas.Planta;
```

Listagem A.2: Implementação da classe Planta1 para sistemas de primeira derivada da classe abstrata TPlanta no FCC.

```
package FCC.Plantas.Planta1 is
   pragma Elaborate_Body;
   type Planta1 is new FCC.Plantas.Planta.TPlanta with private;

   -- ponteiro de acesso para o tipo Planta1
   type PPlanta1 is access Planta1;

   task  type  ThreadPlanta(This:  PPlanta1;  y,u:  pVFloat;  tm:  PTime)
is
      entry Start;
      pragma Priority(this.Prioridade);
   end ThreadPlanta;

   -- rotina de inicialização do tipo planta1
   procedure Inicializa(This: PPlanta1; y, u: PVFloat);
   -- procedimento para calcular o WCET da planta
function CalculaWCET(This: PPlanta1; y,u: pVFloat; t: PTime) return
float;

   -- procedimentos para setar os parâmetros da planta
   procedure SetBuffer(This: PPlanta1; newBuffer: PSharedBuffer);
   procedure SetKg(This: PPlanta1; newKg: PFloat);
```

```
   procedure SetKt(This: PPlanta1; newKt: PFloat);
   procedure SetCanal(This: PPlanta1; newCanal: Unsigned);
   procedure SetTS(This: PPlanta1; newTS: Integer);
   procedure Free(This: in out PPlanta1);

private

   type Planta1 is new Plantas.Planta.TPlanta with
      record
         B : PSharedBuffer;
         kg: Float; -- constante de ganho da planta digital
kt: Float; -- constante de conversão de continuo para discreto = exp( -1/Nr)
      -- onde: Nr=Tr/T, numero de amostragem por periodo [6 .. 10]
         -- Tr constante de tempo contínuo
         Canal: unsigned := 0;-- canal nos conversores AD/DA
         TS : Time_Span:= Time_Span_Zero; -- tempo de simulação
      end record;
   -- Codigo que será executado periodicamente pela threadMyPlanta
   procedure Codigo(This: PPlanta1; y, u: PVFloat);
   -- Le o valor de entrada da planta
   function ReadInPut(This: PPlanta1) return Float;
   -- escreve a saida da planta
   procedure WriteOutput(This: PPlanta1);
end FCC.Plantas.Planta1;
```

Listagem A.3: Implementação da classe Planta2 para sistemas de segunda ordem derivada da classe abstrata TPlanta no FCC.

```
package FCC.Plantas.Planta2 is
   pragma Elaborate_Body;

   type Planta2 is new FCC.Planta.TPlanta with
      record
         B : PSharedBuffer;
      end record;

   type PPlanta2 is access Planta2;

     task type ThreadPlanta(This: PPlanta2; TS: Integer; Buffer: PSharedBuffer; y , u: pVector) is
      entry Start;
      pragma Priority(this.Prioridade);
   end ThreadPlanta2;

   -- Codigo que será executado na inicialização da planta
   procedure Inicializa(This: PPlanta2; y , u: PVector);
```

```
   -- Codigo que será executado periodicamente pela threadPlanta
   procedure Codigo(This: PPlanta2; y , u: PVector);
   -- Le o valor de entrada da planta, saída do controlador
   function ReadInPut(This: PPlanta2) return Float;
   -- escreve a saída da planta
   procedure WriteOutput(This: PPlanta2);

end FCC.Plantas.Planta2;
```

Listagem A.4: Implementação da classe abstrata TControlador derivada da classe TarefaPeriodica no FCC.

```
package FCC.Controladores.Controlador is

   type TControlador is abstract new TarefaPeriodica with
      record
         u : Float;  -- armazena a saída do controlador
         k : Integer; -- Estado do controlador
         On: Boolean := False; -- Liga e desliga o controlador

      end record;

   type PControlador is access TControlador'Class;

   task type ThreadControlador(This: PControlador) is
      entry Start;

      pragma Priority(This.Prioridade);

   end ThreadControlador;

   -- Codigo de inicializacao da tarefa controlador
   procedure Inicializa(This: PControlador) is abstract;
   -- Codigo que sera executado pela threadControlador
   function Codigo(This: PControlador) return Float is abstract;
   -- Le a a entrada do controlador
   function ReadInPut(This: PControlador) return Float is abstract;
-- calcula a parte do algoritimo de controle que faz uso da
informação de amostragem atual
function CalculateOutput(This: PControlador) return Float is
abstract;
   -- Escreve a saída do controlador
   procedure WriteOutput(This: PControlador) is abstract;
```

```
-- Contém a alteração dos estados do controlador e os pre-calculos
que são nescessários para minizar o tempo de execução
   procedure UpdateState(This: PControlador) is abstract;
   -- calcula tempo de execução de pior caso do controlador
function  CalculaWCET(This:  PControlador;  erro,ref:  PVFloat;  t:
PTime) return float is abstract;

end FCC.Controladores.Controlador;
```

Listagem A.5: Implementação da classe PID derivada da classe abstrata TControlador no FCC.

```
package FCC.Controladores.PID is

   pragma Elaborate_Body;

   -- Definição do Tipo Controlador PID
   type PID is new FCC.Controladores.Controlador.TControlador
with private ;
   -- ponteiro de acesso para o tipo controlador PID
   type PPid is access PID;

   task type ThreadPid(This: PPid; e,ref: PVFloat; t: PTime) is
      entry Start;
      pragma Priority(This.Prioridade);

   end ThreadPid;

   -- rotina de inicializacao da classe Pid
   procedure Inicializa(This: PPid);
   -- rotina para calculo do tempo de execucao de pior caso
   function CalculaWCET(This: PPid; erro,ref: PVFloat; t,ty: PTime)
return float;
   -- procedimentos para setar os parametros da planta
   procedure SetBuffer(This: PPid; newBuffer: PSharedBuffer);
   procedure SetParametros(This: PPid ; newPar: PParametrosPid);
   procedure SetCompensaJitter(This: PPid; newCjitter: Boolean);
   procedure SetCanal(This: PPid; newCanal: Unsigned);
   --procedure SetIsRealPlanta(This: PPid; newIsRPlanta: Boolean);
   procedure SetSubTarefa(This: PPid; newSTask: PSubTarefa);
   procedure SetCFifo(This: PPid; newFifo: Fifo_Ptr);
   procedure    SetImplementacao(This:    PPid;    newImplementacao:
TAbordagem);
```

```
   procedure Free(This: in out PPid);

private
   type TPID is new Controladores.Controlador.TControlador with
      record
         P   : Float;                 -- termo proporcional
         I   : Float;                 -- termo Integral
         D   : Float;                 -- termo derivativo
         Par: PParametrosPID;        -- parametros do controlador
         B    :  PSharedBuffer;               --  Buffer  compartilhado
controlador/planta
         CJitter   : Boolean := False;-- True :compensa jitter de
amostragem
         Canal    : unsigned := 0;   -- canal nos conversores AD/DA
  CFifo     : Fifo_Ptr := null ;  -- fila para compensação do jitter
de amostragem
         Implementacao : TAbordagem := convencional;
      end record;

--    Codigo    básico    para    qualquer    um    dos    tipos    de
controladores(P,PI,PD,PID)
   function Codigo(This: PPid; e: PVFloat) return Float;
   -- Le o valor de entrada do controlador (y(KT))
   function ReadInPut(This: PPid) return Float;
   -- Escreve a saida do controlador (u(KT))
   procedure WriteOutput(This: PPid);
--  calcula  a  parte  do  algoritimo  de  controle  que  faz  uso  da
informação de amostragem atual
   function CalculateOutput(This: PPid; e: PVFloat) return Float;
-- Contém a alteração dos estados do controlador e os pre-calculos
que são nescessários para minizar o tempo de execução
   procedure UpdateState(This: PPid; e: PVFloat);
   -- procedimento para compessar o jitter de amostragem
  procedure CompensaJitterAmostragem(This: PPid);

end FCC.Controladores.PID;
```

Listagem A.6: Implementação da classe PID derivada da classe abstrata TControlador no FCD.

```
package FCD.PID is

   pragma Elaborate_Body;

   type PID is new FCD.Controladores.Controlador.TControlador with
```

```
      record
         P   : Float;              -- termo proporcional
         I   : Float;              -- termo Integral
         D   : Float;              -- termo derivativo
         Par: PParametrosPid;      -- parâmetros do controlador
         Server:  PServidor;          --  Servidor  com  sockets  para
comunicação
      end record;

   type PPid is access PID;

   task  type  ThreadPid(This:  PPid;  Parametros:  PParametrosPid;
Server: PServidor;
                        NumCanais,  Base:  Integer;  e,  Referencia:
PVector) is
      entry Start;

      pragma Priority(This.Prioridade);

   end ThreadPid;

   -- Código de inicialização da tarefa Pid
   procedure Inicializa(This: PPid);
   -- código executado pela threadPid.
   function Codigo(This: PPid; e: PVector) return Float;


end FCD.PID;
```

Listagem A.7: Implementação da aplicação para analisar o sistema de controle centralizado com planta de segunda ordem utilizando a plataforma RCP centralizada.

```
   procedure Main is
--------------------------------------------------------------------
-- sinais de referência(r)/erro(e)/saidas planta(y)e controlador(u)
   r: PVector := new Vector(0..99);
   e: PVector := new Vector(0..99);
   y: PVector := new Vector(0..99);
   u: PVector := new Vector(0..99);

s: PSensor := new Sensor;
pa:PParametrosPid := new ParametrosPid(new Float'(1.0), new
Float'(0.2),new Float'(0.2)); -- Kp,Ki,Kd
-- Parametros: T,D,C,Prioridade,pa,s,a
   c: PPid := new Pid.Pid(1000,1000,10,10,pa,s,a);
-- Parametros :T,D,C,Prioridade,Tempo de simulação,s,a
   p: PPlanta2 := new Planta2.Planta2(1000,1000,10,10,40000,s,a);
```

```
   tc:ThreadPid(c, ,r,e,u);
   tp:ThreadPlanta(p,y);
-------------------------------------------------------------------
begin
-- inicialização das variaveis
      y.all := (others => 0.0);
      u.all := (others => 0.0);
      e.all := (others => 0.0);
      r.all := (others => 1.0);
      tp.start;
      tc.start;
-- Aguarda o término do tempo de simulação da planta
      loop
         delay 40.0; -- Aguarda 40 segundos
         exit when (tp'Terminated); -- verifica se a planta términou
      end loop;
      Abort_Task(tc'Identity); -- finaliza a tarefa controlador
-- imprime as saida da planta
      PrintVector(y.all);
-------------------------------------------------------------------
end Main;
```

Listagem A.8: Implementação da aplicação distribuída no Nó Controlador para análise do sistema de controle distribuído com planta de segunda ordem utilizando a plataforma RCP distribuída.

```
procedure MainControlador is
-------------------------------------------------------------------
-- Armazena o endereço destino dos datagramas para o protocolo UDP.
   Endereco: String:= "";
   Server : Servidor.PServidor := new Servidor.Servidor(2);
   -- sinal de referência
   r: PVector := new Vector(0..99);
   p : PParametrosPid := new ParametrosPid(new Float'(1.0),
   new Float'(0.2),new Float'(0.2));
   Pid: ControladorPid.PPid := new
ControladorPid.Pid(1000,1000,10,10,
   p,Server);
   -- diferença entre a saida da planta e a referência
   e: PVector := new Vector(0..99);
   -- Pid:Modelo tarefa; Num de canais = 2; Canal base = 1
   thPid: ThreadPid(Pid,2,1,e,r);
   -- endereço base para os sockets udp
   PortaBase: Integer := 5000;
-------------------------------------------------------------------
begin
```

```
-- inicializa o sinal de referência com um
   r.all := (others => 1.0);
-- Inicializa o servidor
   Inicializa(Server,Endereco,PortaBase);
-- Conecta os canais : 1: Controlador => Atuador 2: Sensor  =>
Controlador
   if ConectaCanais(Server) then
-- Ativa os controladores
      thPid.start;
   else
      Put_Line("Erro conectando ou habilitando canais");
   end if;


--------------------------------------------------------------------
end MainControlador;
```

Listagem A.9: Implementação da aplicação distribuída no Nó Planta/Sensor/Atuador para análise do sistema de controle distribuído com planta de segunda ordem utilizando a plataforma RCP distribuída.

```
procedure MainPlanta is
--------------------------------------------------------------------

      Buffer: PSharedBuffer := new SharedBuffer;
      y: PVector := new Vector(0..99);
      u: PVector := new Vector(0..99);
      Planta      : PPlanta2 := new Planta2.Planta2(1000,1000,10,10);
      thPlanta    : ThreadPlanta(Planta,40000,Buffer,y,u);
      Sensor      : PSensorTcp := new
SensorTcp.SensorTcp(500,500,10,10);
      thSensor    : ThreadSensor (PSensor(Sensor),Buffer);
      Atuador     : PAtuadorTcp := new
AtuadorTcp.AtuadorTcp(500,10,9);
      thAtuador   : ThreadAtuador(PAtuador(Atuador),Buffer);
      Endereco        : String := "143.107.99.137";
      OnCanaisSensor : Boolean := False;
OnCanaisAtuador: Boolean := False;
      -- Porta do servidor/controlador
      PortaBase       : Integer := 5000;
--------------------------------------------------------------------
   begin
   -- inicialização das variaveis
      y.all := (others => 0.0);
      u.all := (others => 0.0);
      InicializaSensor(Sensor,Endereco,PortaBase);
      InicializaAtuador(Atuador,Endereco,PortaBase);
```

```
      OnCanaisSensor := ConectaSensor(Sensor);
      OnCanaisAtuador:= ConectaAtuador(Atuador);

if OnCanaisSensor and OnCanaisAtuador then
         thAtuador.Start;
         thPlanta.Start;
         thSensor.Start;
   -- Aguarda que todas as plantas sejam desativadas
         loop
            delay 40.0;
            exit when (thPlanta'Terminated);
         end loop;
      else
         Put_Line("Erro Conectando Sensores e/ou Atuadores
controlador");
      end if;
-- finaliza as tarefas atuador e sensor
      Abort_Task(thAtuador'Identity);
      Abort_Task(thSensor'Identity);
-- imprime as saidas das plantas
      PrintVector(y.all);
--------------------------------------------------------------------
end MainPlanta;
```

Listagem A.10: Implementação da aplicação utilizada para simular os sistemas de controle com modelos de planta, plantas reais e comparar as abordagens de implementação utilizando a plataforma RCP centralizada.[i]

```
procedure ComparaAbordagens is
--------------------------------------------------------------------
-- fila circular para compensação do jitter de amostragem nos 3
controladores
   FC1: Fifo_Ptr := new Fifo_Type(50);
   FC2: Fifo_Ptr := new Fifo_Type(30);
   FC3: Fifo_Ptr := new Fifo_Type(20);
-- buffer de dados compartilhados entre os controladores e as
plantas
   Buffer: PSharedBuffer := new SharedBuffer(new Float'(0.0));
-- paramentros dos controladores
   PC1: PParametrosPid := new ParametrosPid(new Float'(1.5), new
Float'(6.0), new Float'(0.0),
                                           new Float'(-4.2),new
Float'(4.2),T_PI);
   PC2: PParametrosPid := new ParametrosPid(new Float'(2.0), new
Float'(4.0), new Float'(0.0),
                                           new Float'(-4.2),new
Float'(4.2),T_PI);
```

```
   PC3: PParametrosPid := new ParametrosPid(new Float'(2.5), new
Float'(2.7), new Float'(0.0),
                                             new Float'(-4.2),new
Float'(4.2),T_PI);
   -- Modelo de tarefas controlador e planta
   -- (ms, ms,ns  , - )
   -- (T , D, WCET, Pr)
   C1: PPid := new Pid.Pid(33,33,8200,10);
   C2: PPid := new Pid.Pid(56,56,8200,9);
   C3: PPid := new Pid.Pid(83,83,8200,8);
   -- Plantas
   G1: PPlanta1 := new Planta1.Planta1(33,33,7000,10);
   G2: PPlanta1 := new Planta1.Planta1(56,56,7000,9);
   G3: PPlanta1 := new Planta1.Planta1(83,83,7000,8);

   -- Modelo de subtarefas
   -- (ms, ms,  ns    , ns   ,  - , -  ,  ns   )
   -- (DCO,DUS, WCETCO,WCETUS, PCO, PUS, Offset)
   STC1: PSubTarefa:= new SubTarefa(32,33,14200,49200,10,7,0);
   STC2: PSubTarefa:= new SubTarefa(55,56,14200,49200,9,6,0);
   STC3: PSubTarefa:= new SubTarefa(82,83,14200,49200,8,5,0);

   -- vetores de dados
   e1: PVector := new Vector(0..155);
   y1: PVector := new Vector(1..155);
   u1: PVector := new Vector(1..155);
   tc1: PTime   := new VTime(1..155);
   tg1: PTime   := new VTime(1..155);
   ref1: PVector := new Vector(1..155);

   e2: PVector := new Vector(0..91);
   y2: PVector := new Vector(1..91);
   u2: PVector := new Vector(1..91);
   tc2: PTime   := new VTime(1..91);
   tg2: PTime   := new VTime(1..91);
   ref2: PVector := new Vector(1..91);

   e3: PVector := new Vector(0..61);
   y3: PVector := new Vector(1..61);
   u3: PVector := new Vector(1..61);
   tc3: PTime   := new VTime(1..61);
   tg3: PTime   := new VTime(1..61);
   ref3: PVector := new Vector(1..61);

   -- Threads controladores e plantas
   ThC1: Pid.ThreadPid(C1,e1,ref1,tc1);
   ThC2: Pid.ThreadPid(C2,e2,ref2,tc2);
```

```
   ThC3: Pid.ThreadPid(C3,e3,ref3,tc3);
   -- plantas
   ThG1: Planta1.ThreadPlanta(G1,y1,u1,tg1);
   ThG2: Planta1.ThreadPlanta(G2,y2,u2,tg2);
   ThG3: Planta1.ThreadPlanta(G3,y3,u3,tg3);

   indDesempenho: Float;
--------------------------------------------------------------------
begin
   -- inicialização das variaveis
   Set_Priority(155);

   y1.all := (others => 0.0);
   u1.all := (others => 0.0);
   e1.all := (0.0,2.5, others => 0.0);
   ref1.all := (others => 2.5);

   y2.all := (others => 0.0);
   u2.all := (others => 0.0);
   e2.all := (0.0,2.5, others => 0.0);
   ref2.all := (others => 2.5);

   y3.all := (others => 0.0);
   u3.all := (others => 0.0);
   e3.all := (0.0,2.5, others => 0.0);
   ref3.all := (others => 2.5);

   -- aplicação
   Put_Line("convencional, DF");

   -- seta os valores dos atributos dos controladores e das plantas
   ------------------
   -- Controladores
   ------------------
   Pid.SetParametros(This => C1,newPar => PC1);
   Pid.SetBuffer(This => C1, newBuffer => Buffer);
   Pid.SetCanal(This => C1, newCanal => 0);
   Pid.SetIsRealPlanta(This => C1, newIsRPlanta => False);

   Pid.SetImplementacao(This => C1, newImplementacao =>
convencional);
   Pid.SetCompensaJitter(This => C1, newCJitter => False);
   Pid.SetCFifo(This => C1, newFifo => FC1);
   Pid.SetSubTarefa(This => C1, newSTask => STC1);


   Pid.SetParametros(This => C2,newPar => PC2);
```

```
   Pid.SetBuffer(This => C2, newBuffer => Buffer);
   Pid.SetCanal(This => C2, newCanal => 1);
   Pid.SetIsRealPlanta(This => C2, newIsRPlanta => False);

   Pid.SetImplementacao(This => C2, newImplementacao =>
convencional);
   Pid.SetCompensaJitter(This => C2, newCJitter => False);
   Pid.SetCFifo(This => C2, newFifo => FC2);
   Pid.SetSubTarefa(This => C2, newSTask => STC2);

   Pid.SetParametros(This => C3,newPar => PC3);
   Pid.SetBuffer(This => C3, newBuffer => Buffer);
   Pid.SetCanal(This => C3, newCanal => 2);
   Pid.SetIsRealPlanta(This => C3, newIsRPlanta => False);

   Pid.SetImplementacao(This => C3, newImplementacao =>
convencional);
   Pid.SetCompensaJitter(This => C3, newCJitter => False);
   Pid.SetCFifo(This => C3, newFifo => FC3);
   Pid.SetSubTarefa(This => C3, newSTask => STC3);

   ------------------
   -- Plantas
   ------------------
   Planta1.SetBuffer(This => G1, newBuffer => Buffer);
   Planta1.SetKg(This => G1, newKg => new Float'(0.09564));
   Planta1.SetKt(This => G1, newKt => new Float'(0.9048));
   Planta1.SetTS(This => G1,newTS => 5000);
   Planta1.SetCanal(This => G1, newCanal => 0);

   Planta1.SetBuffer(This => G2, newBuffer => Buffer);
   Planta1.SetKg(This => G2, newKg => new Float'(0.07556));
   Planta1.SetKt(This => G2, newKt => new Float'(0.9048));
   Planta1.SetTS(This => G2,newTS => 5000);
   Planta1.SetCanal(This => G2, newCanal => 1);

   Planta1.SetBuffer(This => G3, newBuffer => Buffer);
   Planta1.SetKg(This => G3, newKg => new Float'(0.07109));
   Planta1.SetKt(This => G3, newKt => new Float'(0.9048));
   Planta1.SetTS(This => G3,newTS => 5000);
   Planta1.SetCanal(This => G3, newCanal => 2);

   ThC3.start;
   ThG3.start;
   ThC2.start;
   ThG2.start;
   ThC1.start;
```

```
   ThG1.start;

   -- Aguarda a finalização de todas as threads plantas para
imprimir as saidas
   loop
      delay 5.0;
      exit when (ThG1'Terminated and ThG2'Terminated and
ThG3'Terminated);
      -- Selecionar um delay de tal maneira que, a tarefa principal
      -- não interfira nos resultados da simulacao.
      -- A tarefa principal deve preemptar o minimo possivel o(s)
      -- controlador(es) e a(s)  planta(s)
   end loop;

   -- finaliza as tarefas de controle
   Abort_Task(ThC1'Identity);
   Abort_Task(ThC2'Identity);
   Abort_Task(ThC2'Identity);

   -- imprime as saidas das plantas
   Put_line("-------------------------------------------------------
-----");
   Put_Line(" Planta 1 ");
   Put_Line("Y(t)");
   PrintVector(y1,tg1,G1.k);
   Put_Line("u(t)");
   PrintVector(u1,tc1,G1.k);
   Put_Line("e(t)");
   PrintVector(e1,tc1,G1.k);
   -- calcula indices de desempenho
   indDesempenho:= ITAE(e1,tc1,G1.k);
   indDesempenho:= IAE(e1,G1.k);
   indDesempenho:= ITSE(e1,tc1,G1.k);
   indDesempenho:= ISE(e1,G1.k);
   Put_line("-------------------------------------------------------
-----");
   Put_Line(" Planta 2");
   Put_Line("Y(t)");
   PrintVector(y2,tg2,G2.k);
   Put_Line("u(t)");
   PrintVector(u2,tc2,G2.k);
   Put_Line("e(t)");
   PrintVector(e2,tc2,G2.k);
   -- calcula indices de desempenho
   indDesempenho:= ITAE(e2,tc2,G2.k);
   indDesempenho:= IAE(e2,G2.k);
   indDesempenho:= ITSE(e2,tc2,G2.k);
```

```
   indDesempenho:= ISE(e2,G2.k);
   Put_line("-------------------------------------------------------
-----");
   Put_Line(" Planta 3 ");
   Put_Line("Y(t)");
   PrintVector(y3,tg3,G3.k);
   Put_Line("u(t)");
   PrintVector(u3,tc3,G3.k);
   Put_Line("e(t)");
   PrintVector(e3,tc3,G3.k);
   -- calcula indices de desempenho
   indDesempenho:= ITAE(e3,tc3,G3.k);
   indDesempenho:= IAE(e3,G3.k);
   indDesempenho:= ITSE(e3,tc3,G3.k);
   indDesempenho:= ISE(e3,G3.k);
   Put_line("-------------------------------------------------------
-----");

   -- libera memoria alocada

   Free_Vector(e1);
   Free_Vector(y1);
   Free_Vector(u1);
   Free_Vector(ref1);
   Free_VTime(tc1);
   Free_VTime(tg1);

   Free_Vector(e2);
   Free_Vector(y2);
   Free_Vector(u2);
   Free_Vector(ref2);
   Free_VTime(tc2);
   Free_VTime(tg2);

   Free_Vector(e3);
   Free_Vector(y3);
   Free_Vector(u3);
   Free_Vector(ref3);
   Free_VTime(tc3);
   Free_VTime(tg3);

   Pid.Free(C1);
   Pid.Free(C2);
   Pid.Free(C3);
   Planta1.Free(G1);
   Planta1.Free(G2);
   Planta1.Free(G3);
```

```
---------------------------------------------------------------------
end ComparaAbordagens;
```

Listagem A.11: Implementação do corpo da tarefa ThreadPid no FCC.

```
-- Task pid
   task body ThreadPid is
      next : Time := Clock; -- proximo periodo de ativacao da tarefa
      periodo : Time_Span := This.periodo; -- periodo da tarefa de
controle
      --Jh: Util.Random.Jitter_Amostragem; -- Jitter de amostragem
   begin

      loop
         -- variavel aleatoria jitter de amostragem
         --This.Jitter                                              :=
Time_Span(Milliseconds(Util.Random.JitterUniforme(Jh)));
         next := next + periodo + This.Jitter;
         select
            accept Start do
               inicializa(This);
               -- inicializa o gerador de numeros randomicos
               --Util.Random.Init(J);
            end Start;

            else if This.On and This.k <= t'Length then
               -- Armazena o instante de ativacao atual
               t(This.k) := Clock;
               -- valor de entrada do controlador
               e(This.k) := ref(This.k) - ReadInPut(This);
               -- implementação convencional
               if This.Implementacao = convencional then
                  This.u := Codigo(This,e);
                  -- Escreve a saida
                  WriteOutput(This);
                  -- Altera o estado
                  This.k := This.k + 1;
                  -- subtask scheduling
               else
                  -- seta a prioridade para Calculate Output
                  Set_Priority(This.STask.PCO);
                  -- Calcula a saida
                  This.u := CalculateOutput(This,e);
                  --  Escreve a saida
```

```
                  WriteOutput(This);
                  -- seta a prioridade para Update State
                  Set_Priority(This.STask.PUS);
                  -- insere o periodo de ativação atual na fila do
controlador

Util.Fifo.Push(This.CFifo,Element_Type(t(This.k)));
                  UpdateState(This,e);

               end if;
            end if;

         end select;
         exit when (This.K = t'Length);
         delay until(next);
      end loop;

      exception when E : others => Ada.Text_IO.Put_Line
      (Exception_Name (E) & ": " & Exception_Message (E));
   end ThreadPid;
```

---

[i] Todas as aplicações para validação dos modelos de plantas, controle das plantas reais e análises utilizando a plataforma RCP centralizada com as plantas de primeira ordem, foram desenvolvidas com base nesta aplicação removendo, inserindo ou alterando partes do código.