

Extensões de Tempo Real para UML

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

**ROTA2030
FUNDEP**

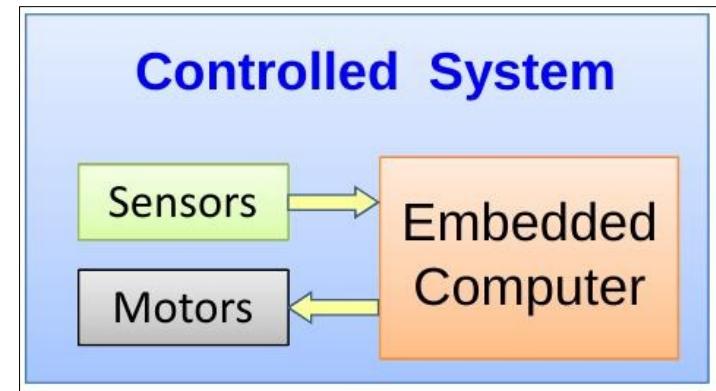
- Dia
- Eclipse com papyrus e suporte a UML-Marte no papyrus
- Ou outro editor de UML com suporte a estereótipos
- Papel

- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- Modelagem de requisitos não-funcionais
- Expressão dos dispositivos de hardware
- Exemplos
- Exercícios

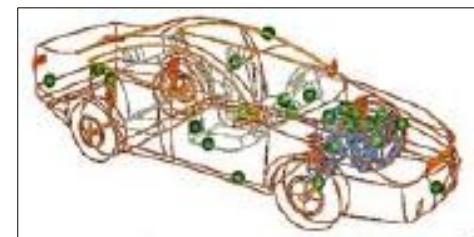
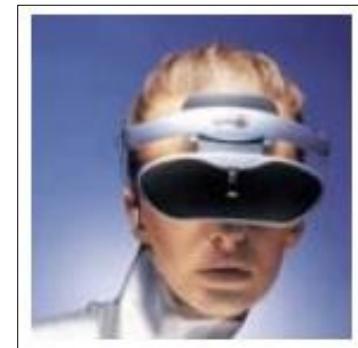
- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- Modelagem de requisitos não-funcionais
- Expressão dos dispositivos de hardware
- Exemplos
- Exercícios

LISHA Introdução a Sistemas de Tempo Real

- Definição formal
 - A corretude do sistema não depende somente dos resultados lógicos da computação, mas também de quando esses resultados são computados
 - Um **resultado correto** em **um tempo errado** é **uma falha**
- Tipicamente estão “embarcados” em um sistema para controlar suas funções
 - Sistemas Embarcados e de Tempo real



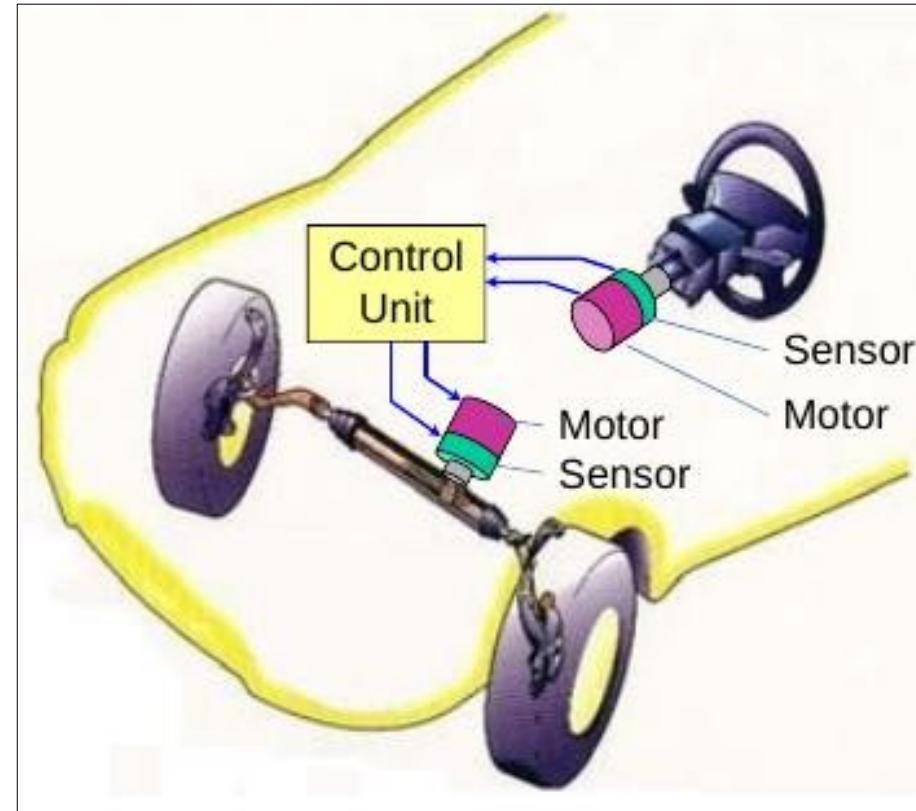
- Aplicações típicas
 - Aviação
 - Automotiva
 - Robótica
 - Automação industrial
 - Telecomunicação
 - Sistemas multimídia
 - Eletrônica de consumo



LISHA

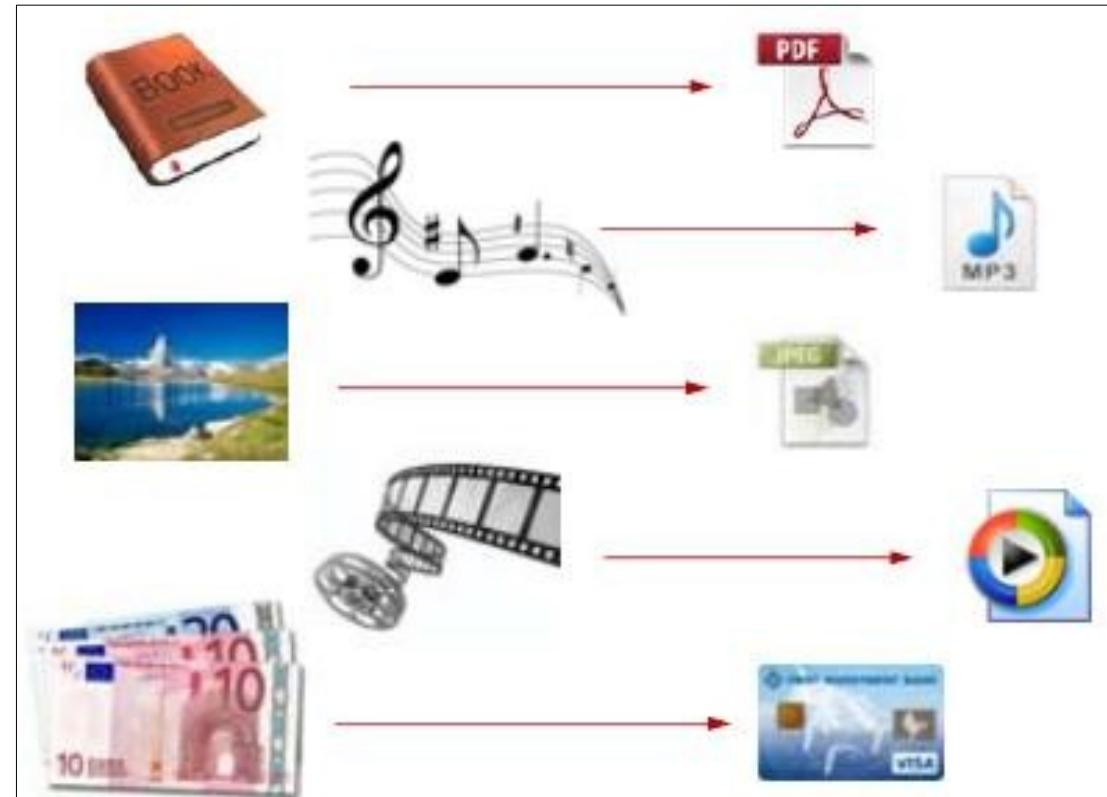
Introdução a Sistemas de Tempo Real

- Steer by Wire



LISHA Introdução a Sistemas de Tempo Real

- Estamos em um processo avançado de “**desmaterialização**” na qual várias funções estão sendo convertidas para software
 - Dinheiro
 - Livros
 - Música
 - Fotos
 - Entradas
 - Documentos
 - Educação
- E também com STR





Introdução a Sistemas de Tempo Real

■ Evolução das linhas de código



~1.7 milhões de linhas de código no caça F22



Estima-se que no futuro os carros terão em torno de 200 milhões de linhas de código



~20 milhões de linhas de código na Mercedes Classe S



Introdução a Sistemas de Tempo Real



- Porque estamos indo em direção ao software?
 - Software é mais flexível do que hardware
 - Pode ser facilmente modificado, adaptado e atualizado
 - Pode ser atualizado remotamente
 - Pode evoluir em algoritmos de controle inteligentes
 - Não tem massa, então “viajar” para qualquer lugar do mundo rapidamente

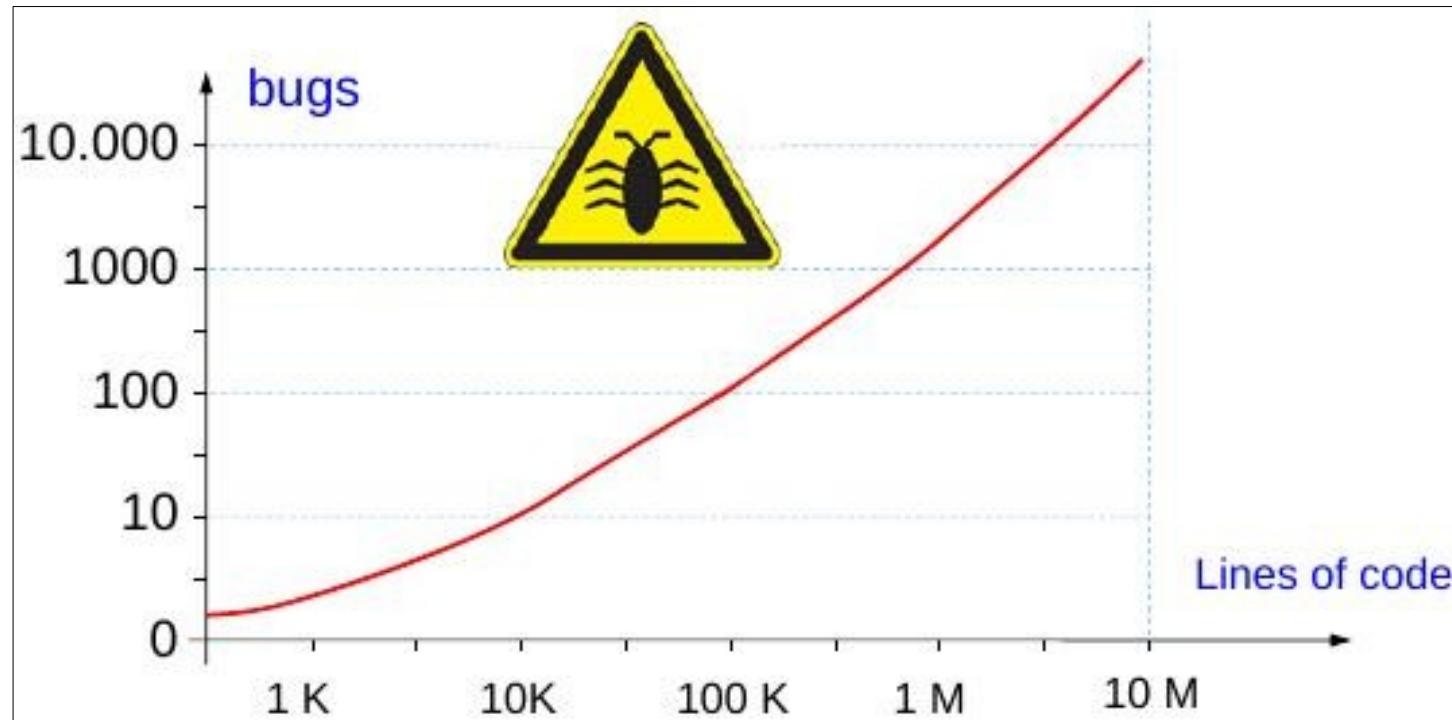


Introdução a Sistemas de Tempo Real

- Quais as desvantagens?
 - O preço pago é **maior complexidade**
- Problemas relacionados
 - Projetar é mais difícil
 - Menos previsibilidade
 - Menos confiabilidade
- **Métodos modernos** para
 - Projeto de software modular e reutilizável envolvendo software e hardware
 - Análise para garantir previsibilidade e segurança
 - Métodos eficientes de teste de software (e também certificação)

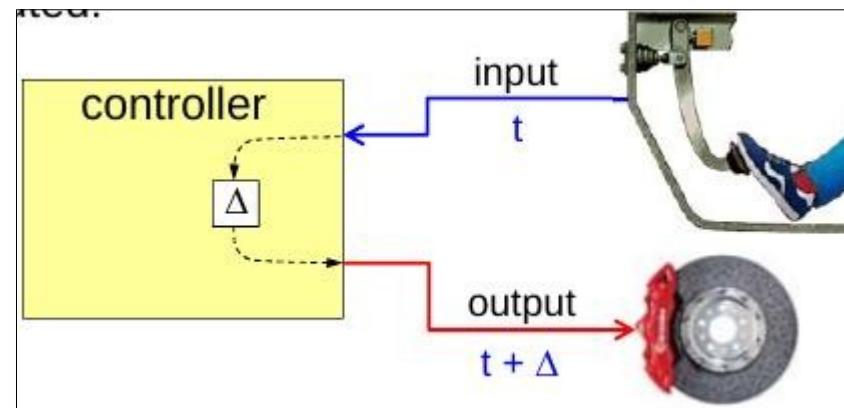
LISHA Introdução a Sistemas de Tempo Real

- Bugs de software aumenta com a complexidade



LISHA Introdução a Sistemas de Tempo Real

- A **confiabilidade** não depende somente da corretude de uma única instrução, mas também **quando** ela é executada



- Uma ação correta executada muito tarde pode não ter utilidade ou ser **perigosa**



Introdução a Sistemas de Tempo Real



Um sistema computacional que deve garantir tempos de resposta dentro dos **limites e previsíveis** são chamados de **sistemas de tempo real**

- As funções estão relacionadas com restrições temporais (**deadlines**)
- Previsibilidade dos tempos de resposta deve ser garantida
 - Para toda atividade crítica
 - Para todas as combinações de eventos

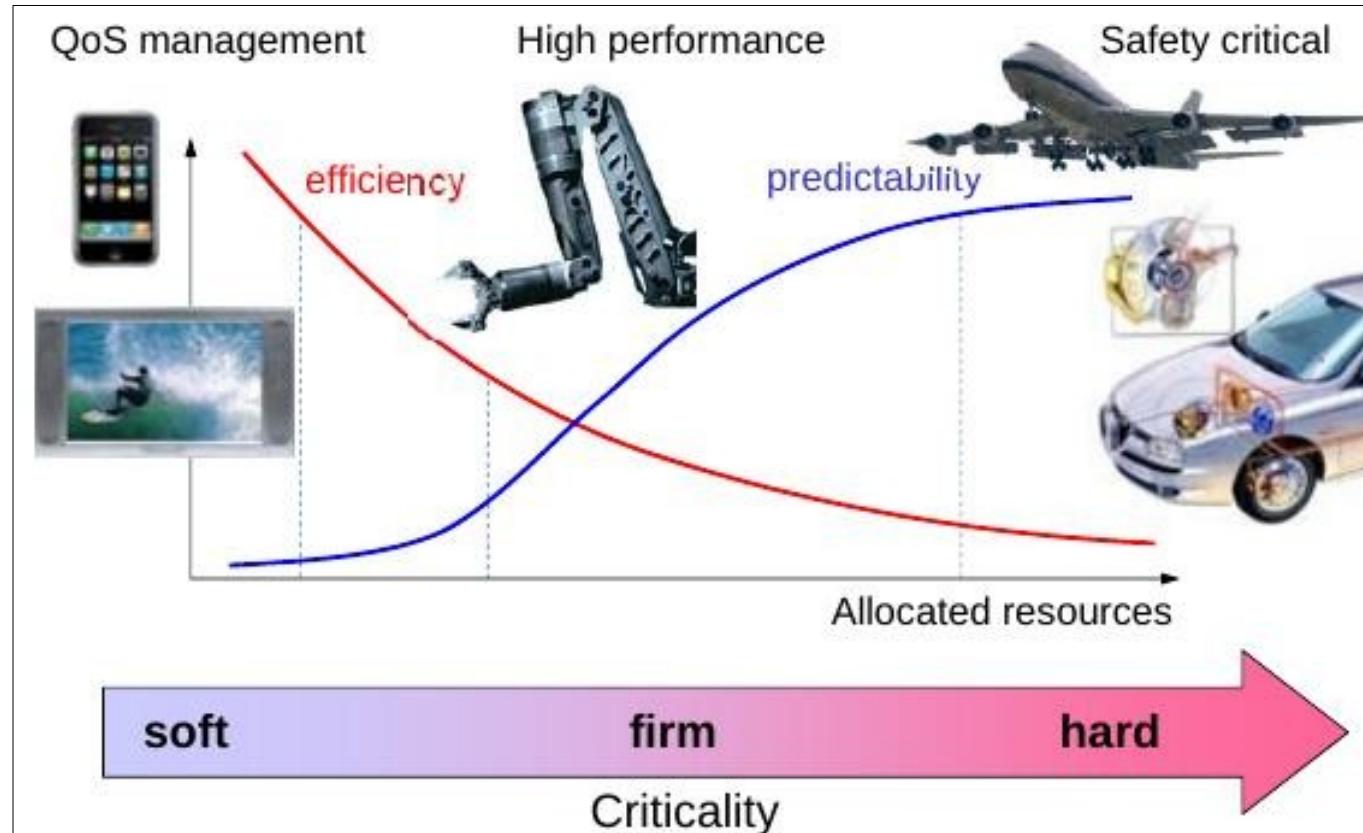


Introdução a Sistemas de Tempo Real

- **Soft Real-Time (SRT)**: a perda de um deadline é indesejável, mas não levará a consequências catastróficas
 - Se relaciona ao conceito de qualidade de serviço (QoS)
 - Tipicamente interessa o tempo de resposta médio
 - Ex: sistemas de reserva, multimídia, telefones
- **Hard Real-Time (HRT)**: a perda de deadlines não é uma opção
 - Interesse sempre no pior tempo de resposta
 - Ex: aviões, usinas nucleares, sistemas militares, automóveis

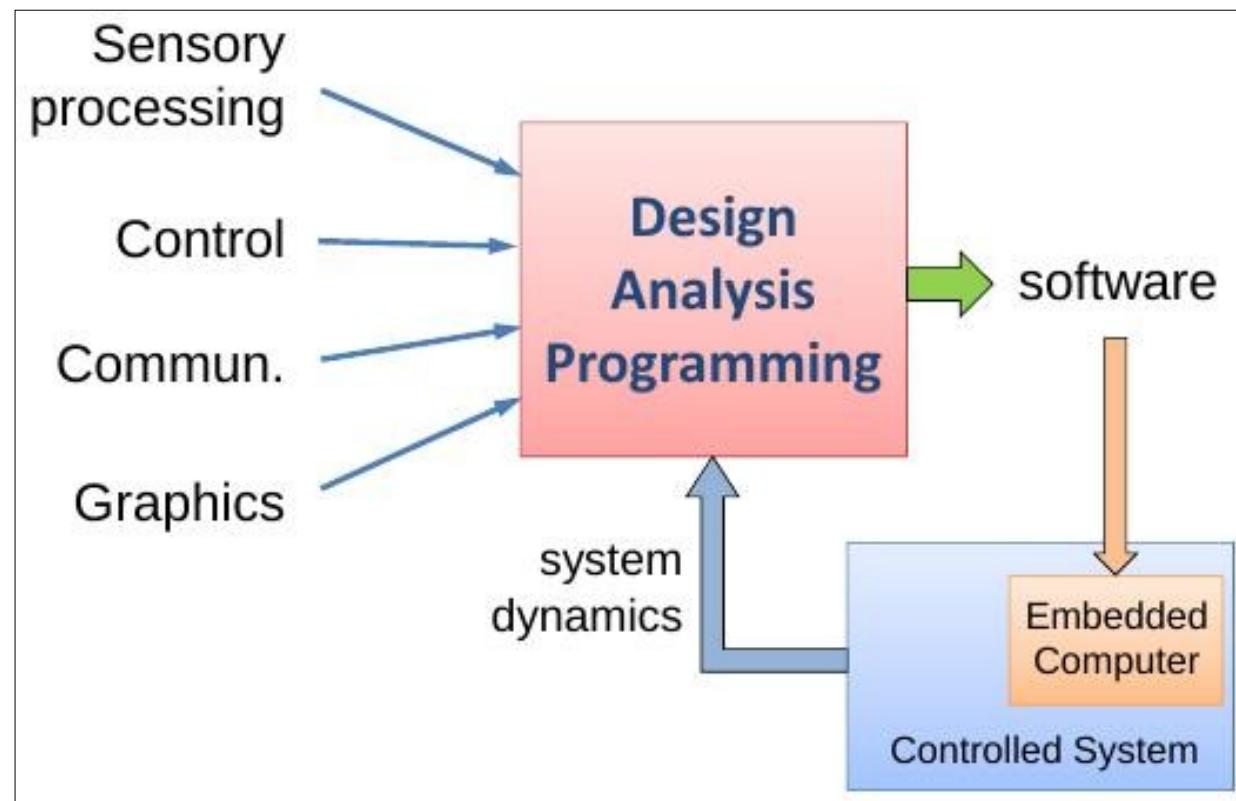
LISHA Introdução a Sistemas de Tempo Real

■ Eficiência x previsibilidade



LISHA Introdução a Sistemas de Tempo Real

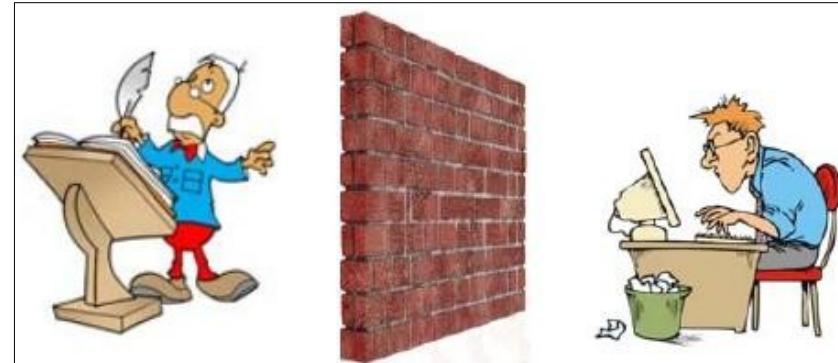
- Nosso foco: software previsível



LISHA Introdução a Sistemas de Tempo Real

■ Controle e implementação

- Geralmente o controle e sua implementação são feitos por pessoas diferentes que não possuem uma boa comunicação



- A teoria de controle assume que o computador tem recursos e poder computacional infinitos
- Em alguns casos, a computação é modelada com um atraso (delay) fixo no modelo



Introdução a Sistemas de Tempo Real

■ Na realidade, um processador:

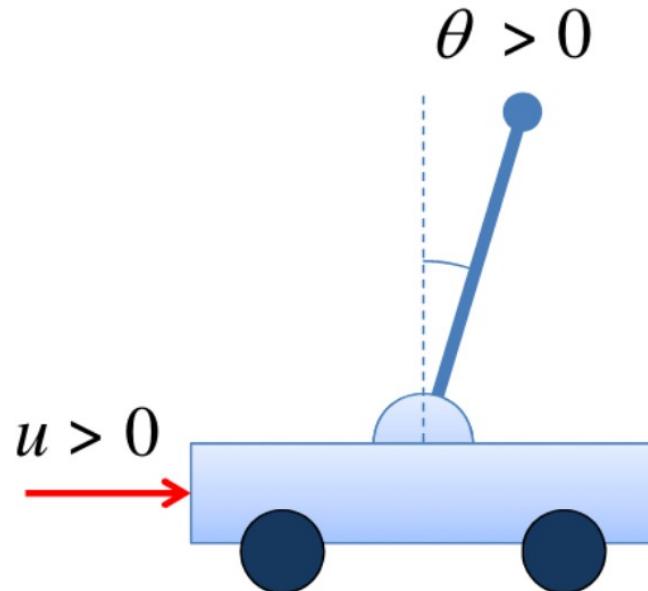
- Tem recursos **limitados**
- Poder computacional **finito** (**tempos de execução não nulos**)
- Executa diversas atividades **concorrentes**
- Introduz atrasos **variáveis** (frequentemente **imprevisíveis**)

Modelar esses fatores e levá-los em conta na fase de projeto permite uma significante melhora de desempenho e confiabilidade

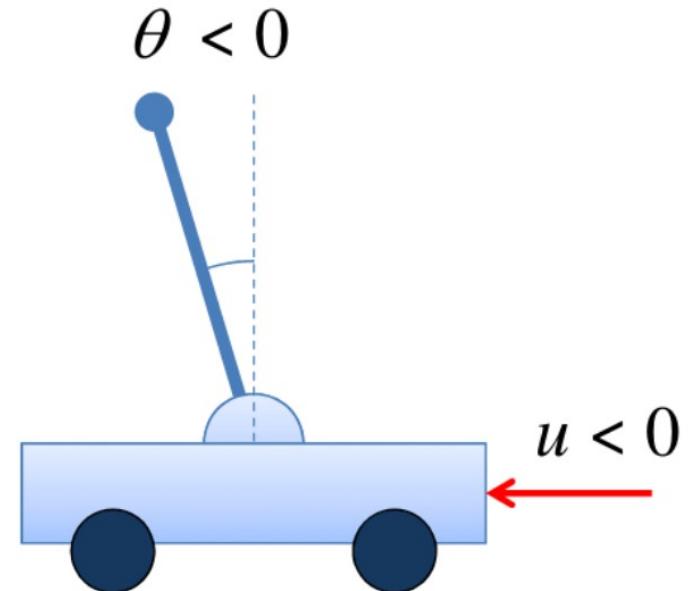
LISHA Introdução a Sistemas de Tempo Real

■ Exemplo de controle

Um ângulo θ positivo requer uma ação de controle positiva u .



$$\theta > 0 \rightarrow u > 0$$

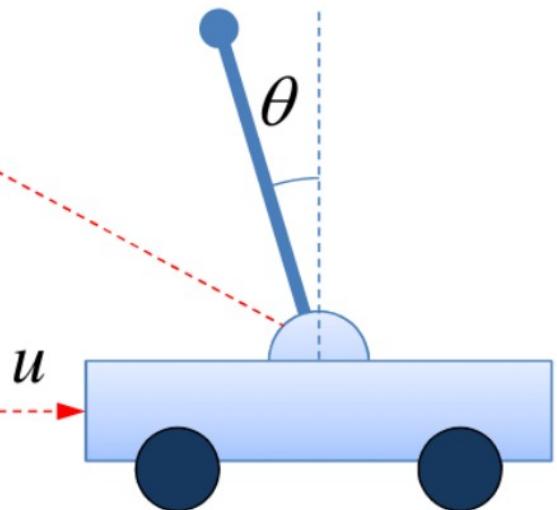


$$\theta < 0 \rightarrow u < 0$$

LISHA Introdução a Sistemas de Tempo Real

```
task control(float theta0, float k)
{
    float error;
    float u;
    float theta;

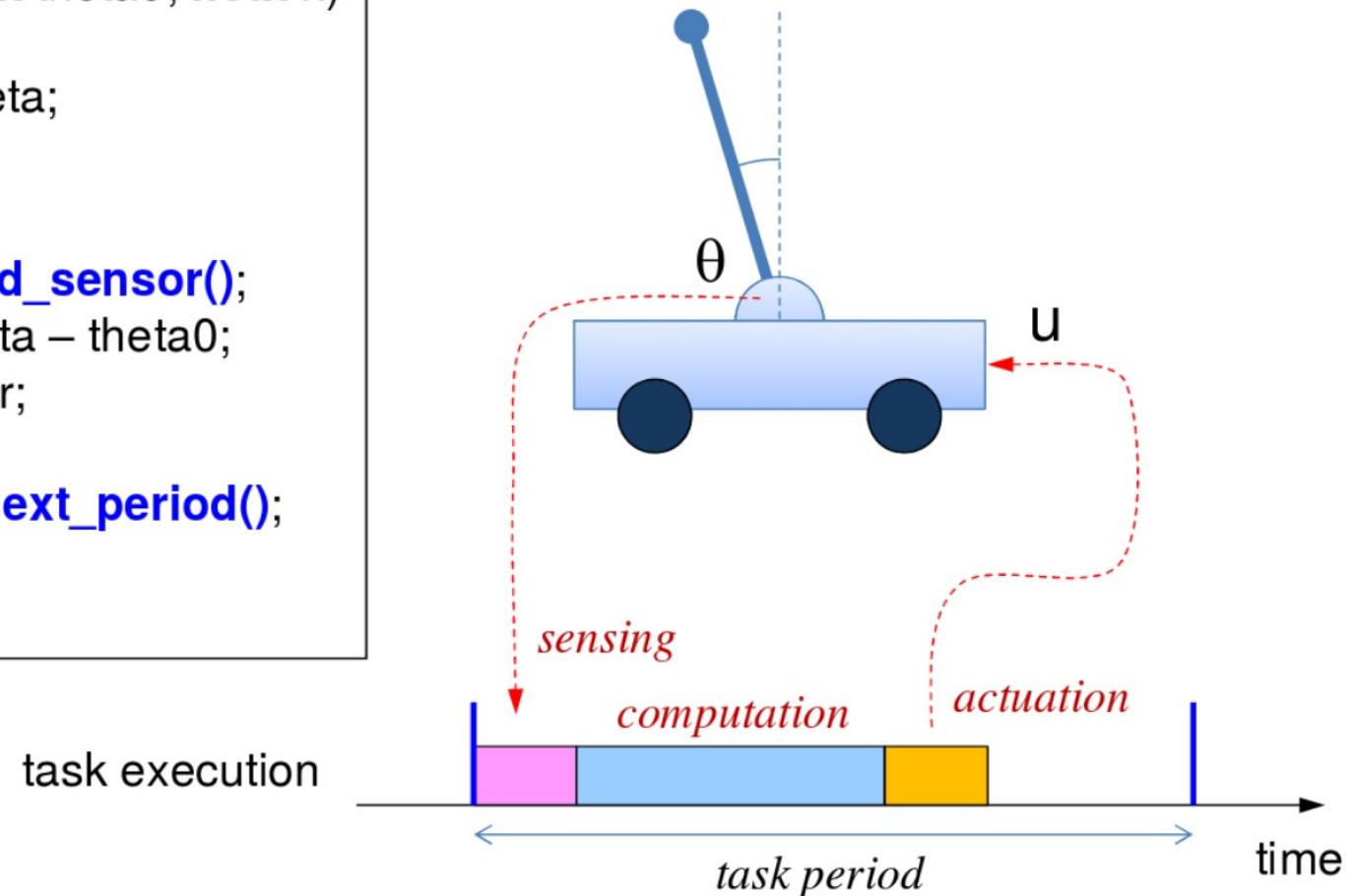
    while (1) {
        theta = read_sensor();           sensing
        error = theta - theta0;          computation
        u = k * error;
        output(u);                     actuation
        wait_for_next_period();         synchronization
    }
}
```



LISHA Introdução a Sistemas de Tempo Real

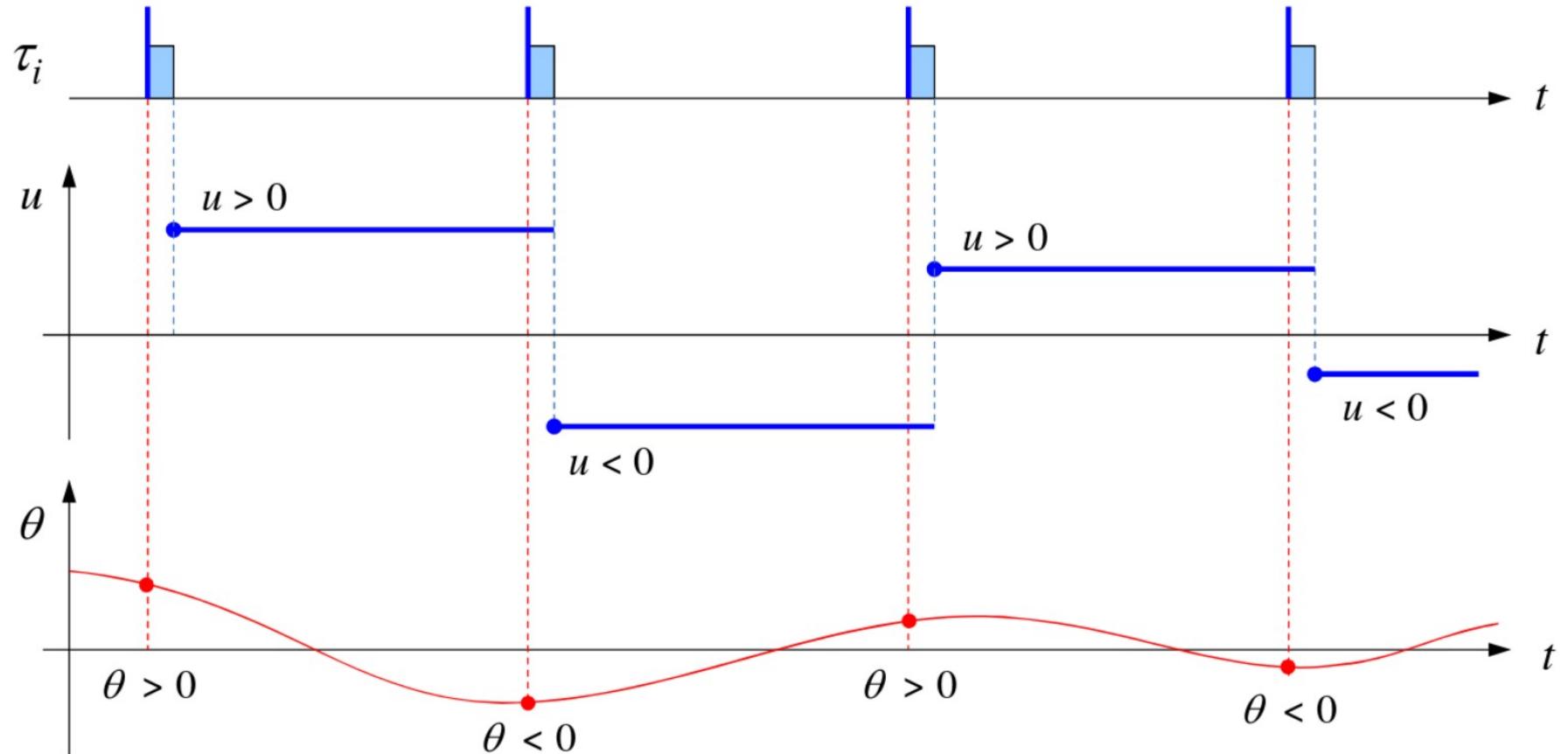
```
task control(float theta0, float k)
{
    float error, u, theta;

    while (1) {
        theta = read_sensor();
        error = theta - theta0;
        u = k * error;
        output(u);
        wait_for_next_period();
    }
}
```



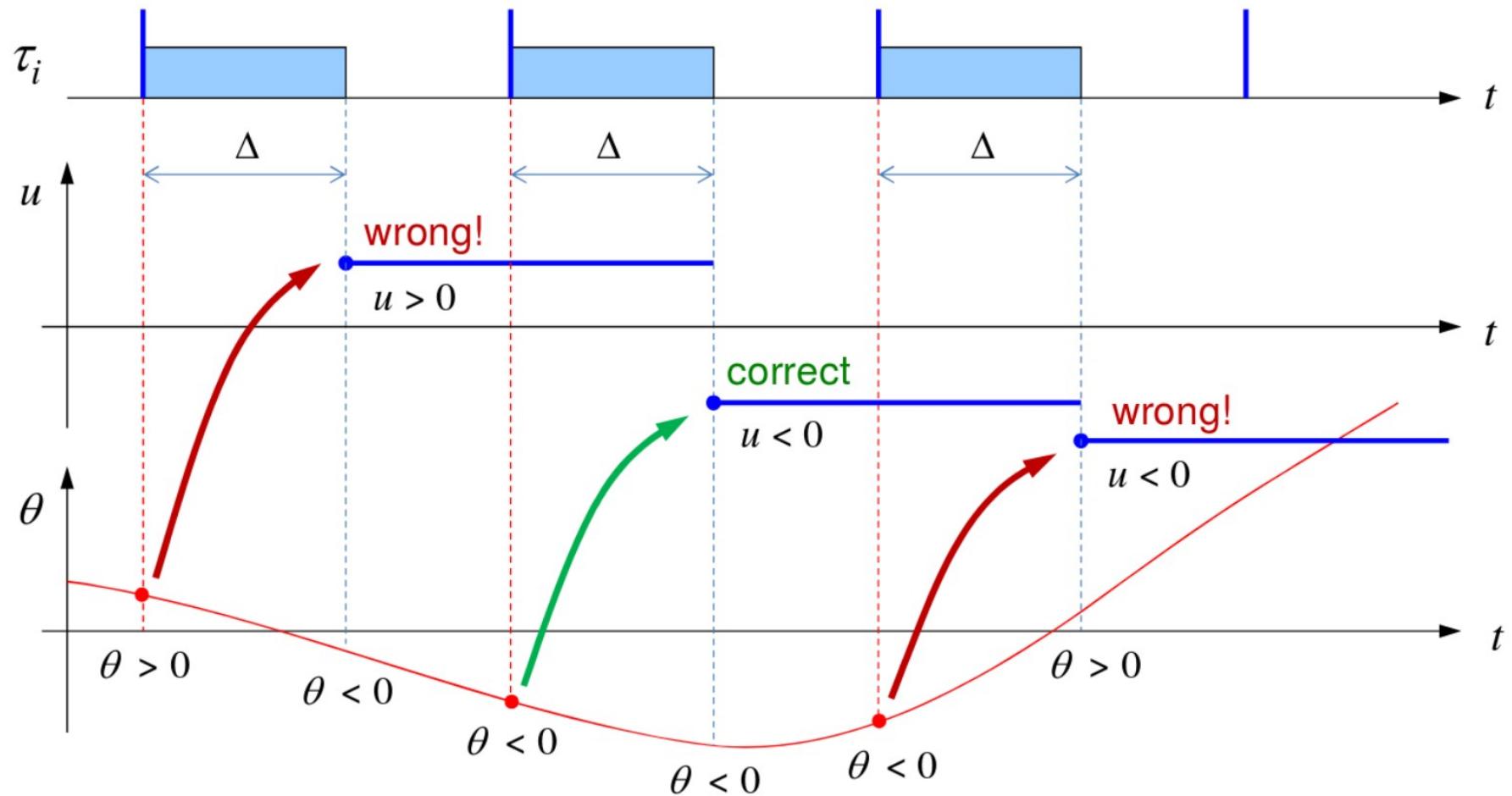
LISHA Introdução a Sistemas de Tempo Real

Visão tradicional de controle: atraso e jitter negligeáveis



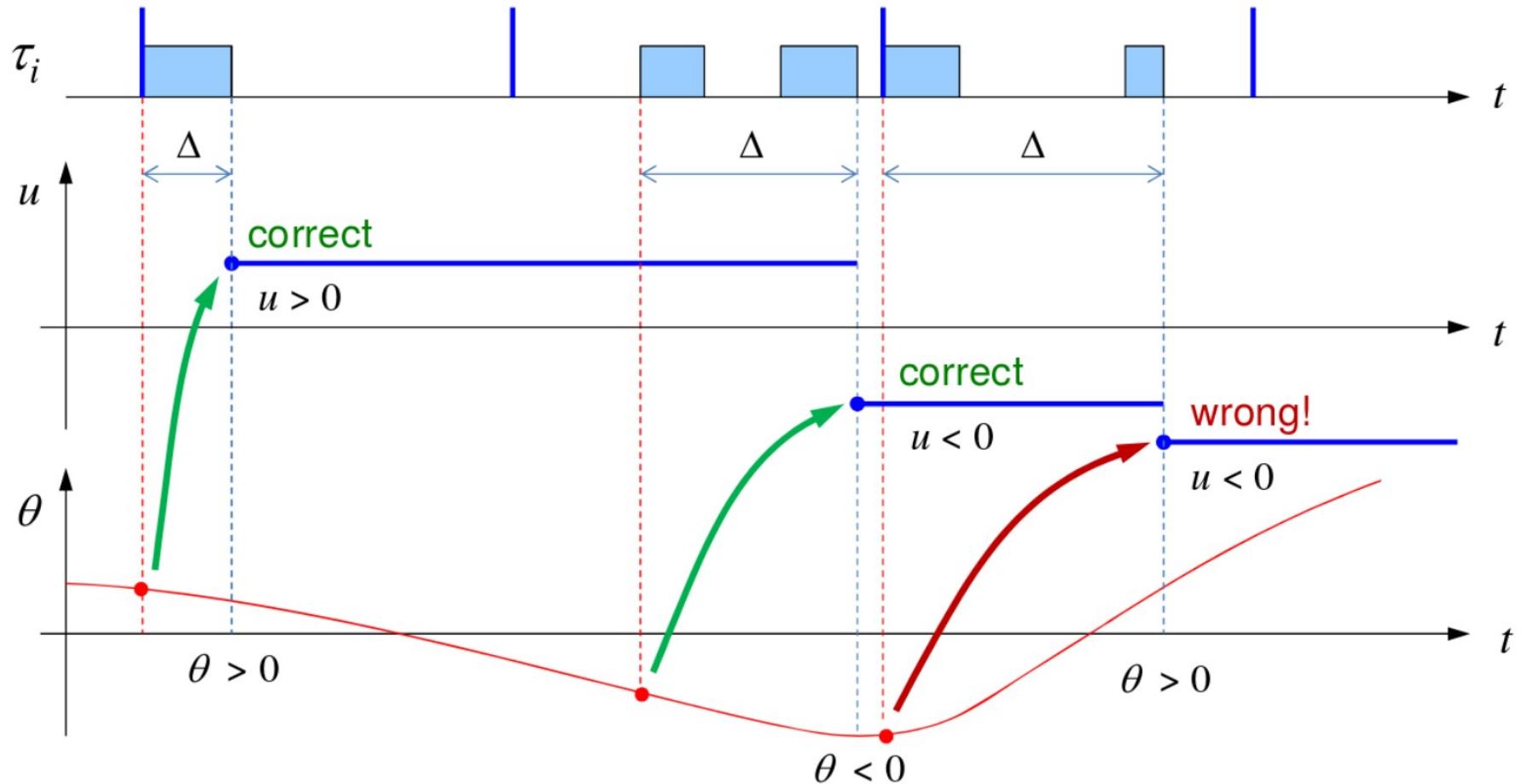
LISHA Introdução a Sistemas de Tempo Real

Os tempos de computação introduzem um atraso não negligeável



LISHA Introdução a Sistemas de Tempo Real

Situação real: atraso variável e jitter



■ Objeção típica

Não vale investir em teoria de tempo real
pois a velocidade do computador aumenta
exponencialmente e as restrições temporais podem
eventualmente desaparecer

■ Resposta

- Dada uma velocidade de computação arbitrária, devemos sempre garantir que as restrições temporais sejam atendidas.
- Testar não é suficiente



Introdução a Sistemas de Tempo Real

■ Fontes de não determinismo

- Arquitetura
 - cache, pipelining, interrupções, DMA
- Sistema Operacional
 - Escalonamento, sincronização, comunicação
- Linguagem
 - Falta de suporte explícito para tempo
- Metodologias de projeto
 - Falta de técnicas de análise e verificação



Introdução a Sistemas de Tempo Real



- Tipicamente a abordagem de projeto de sistemas de tempo real errada, usa técnicas empíricas:
 - Programação assembly
 - Temporização através de timers dedicados
 - Controle através de programação de driver
 - Manipulação de prioridade



Introdução a Sistemas de Tempo Real



■ Quais as desvantagens?

- Programação tediosa que depende fortemente do conhecimento do programador
- Entendimento do código é muito difícil
- Manutenção do software é complicada
- Milhares/Milhões de linhas de código
- Entender o código já escrito demora mais que reescrevê-lo
- Reescrevê-lo é muito caro e bugs irão acontecer
- Dificuldade de verificar as restrições temporais sem um suporte do SO e da linguagem
- Impossível certificar

■ Implicações

- Essa maneira de se programar um sistema de tempo real é muito perigosa
 - Pode funcionar em várias situações, mas o risco de falhas é alto
 - Quando o sistema falha é difícil entender o porquê
- ## ■ Conclusão: **baixa confiabilidade e alto custo**



Introdução a Sistemas de Tempo Real

■ Alguns acidentes devido ao software

- Task overrun during LEM lunar landing
(<http://njjnetwork.com/2009/07/1202-computer-error-almost-aborted-lunar-landing/>)
- First flight of the Space Shuttle (synch)
- Ariane 5 (overflow)
- Airbus 320 (cart task)
- Airbus 320 (holding task)
- Pathfinder (reset for timeout, priority inversion)



Introdução a Sistemas de Tempo Real

- Algumas lições aprendidas
 - Testes, embora necessários e importantes, permitem apenas uma verificação parcial do comportamento do sistema
 - Previsibilidade deve ser melhorada no nível do sistema operacional
 - O sistema deve ser projetado para ser tolerante a falhas e tratar situações de sobrecarga
 - Sistemas críticos devem ser projetados usando suposições pessimistas



Introdução a Sistemas de Tempo Real

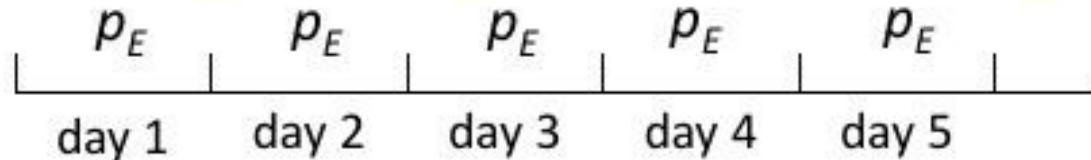


- E lembre-se da Lei de Murphy
 - Se algo pode dar errado, vai dar errado
 - Se o sistema parar de funcionar, ele o fará na pior hora possível
 - Cedo ou tarde a pior combinação de circunstâncias irá acontecer

■ Provando a lei de Murphy

Let p_E be the probability for event E to occur in a day

What is the probability for E to occur in n days?



prob. of E not occurring in **1** day

$$q_E = 1 - p_E$$

prob. of E not occurring in **n** days

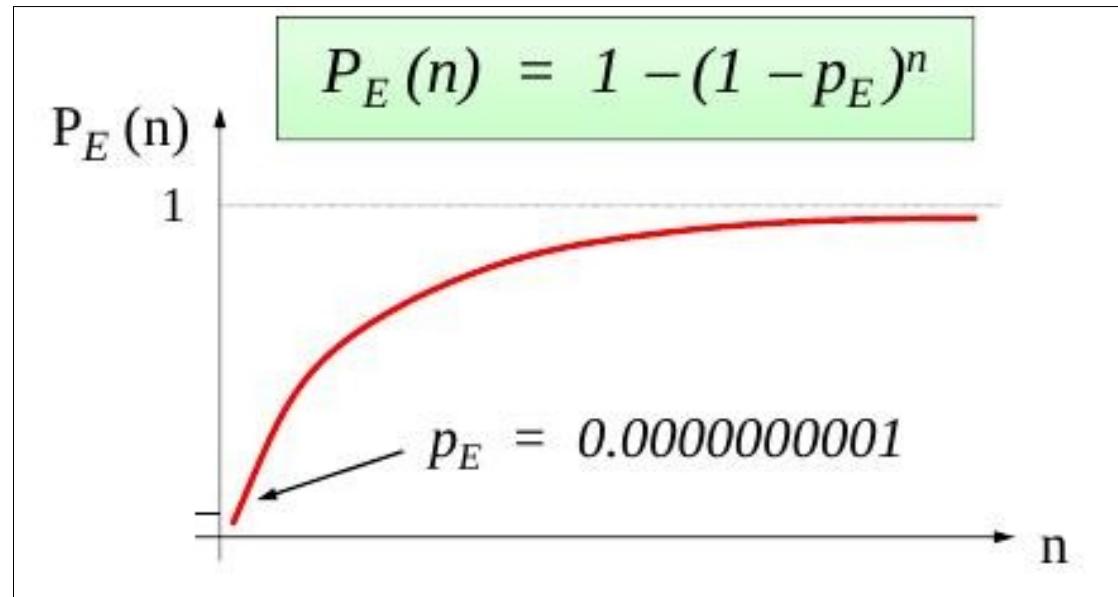
$$Q_E(n) = (1 - p_E)^n$$

prob. of E occurring in **n** day

$$P_E(n) = 1 - Q_E(n)$$

LISHA Introdução a Sistemas de Tempo Real

■ Provando a lei de Murphy



Se algo pode dar errado (não importa o quanto P_E é pequeno), irá acontecer. A probabilidade de E acontecer em um longo intervalo tende a 1.

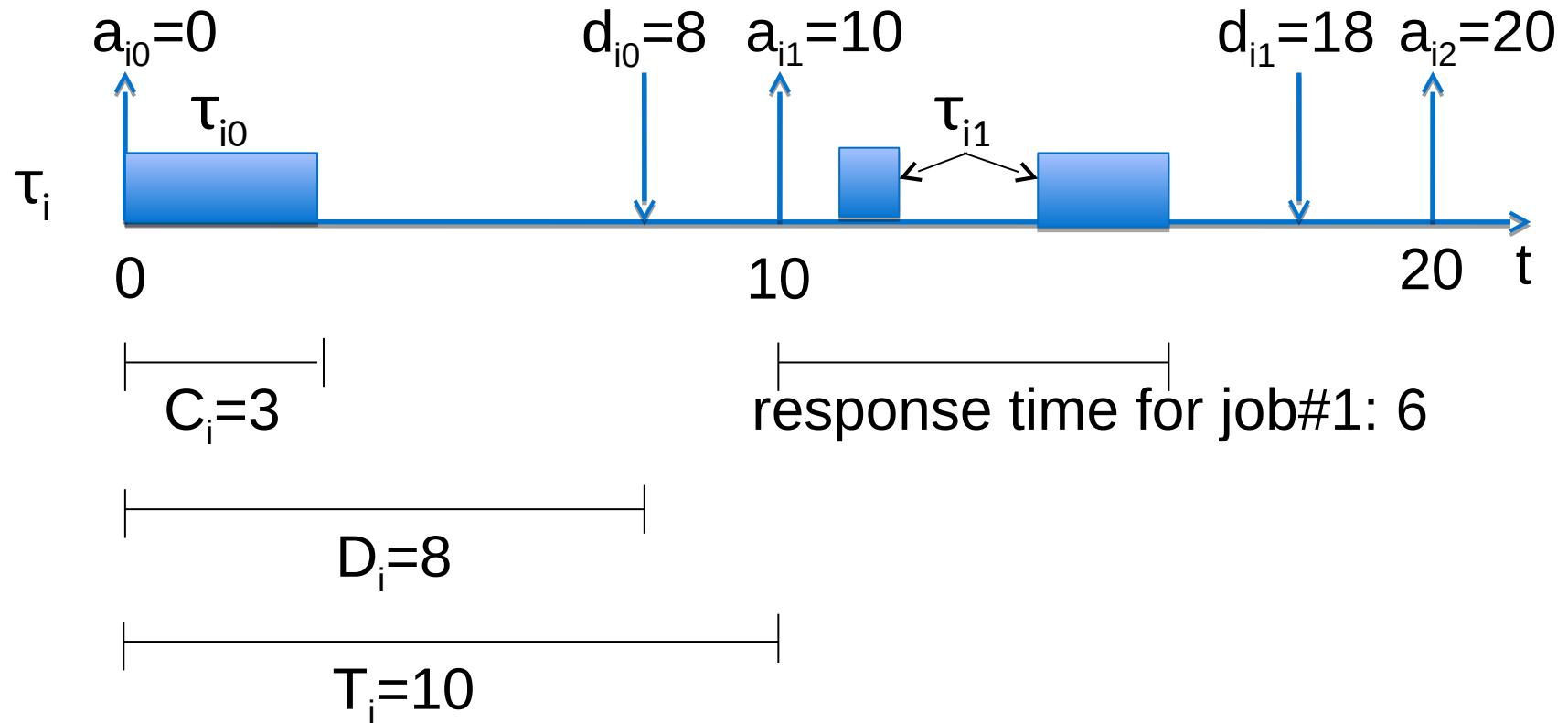


Introdução a Sistemas de Tempo Real

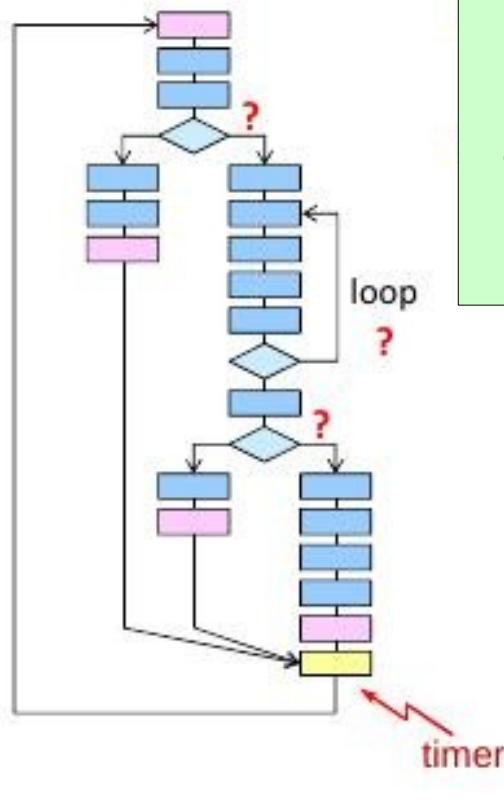


- Discussão?
- Comentários?

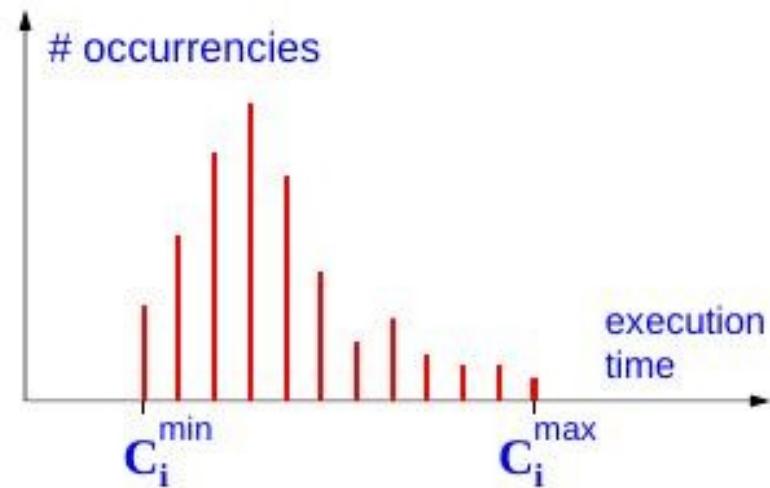
Modelo de Tarefas Periódicas



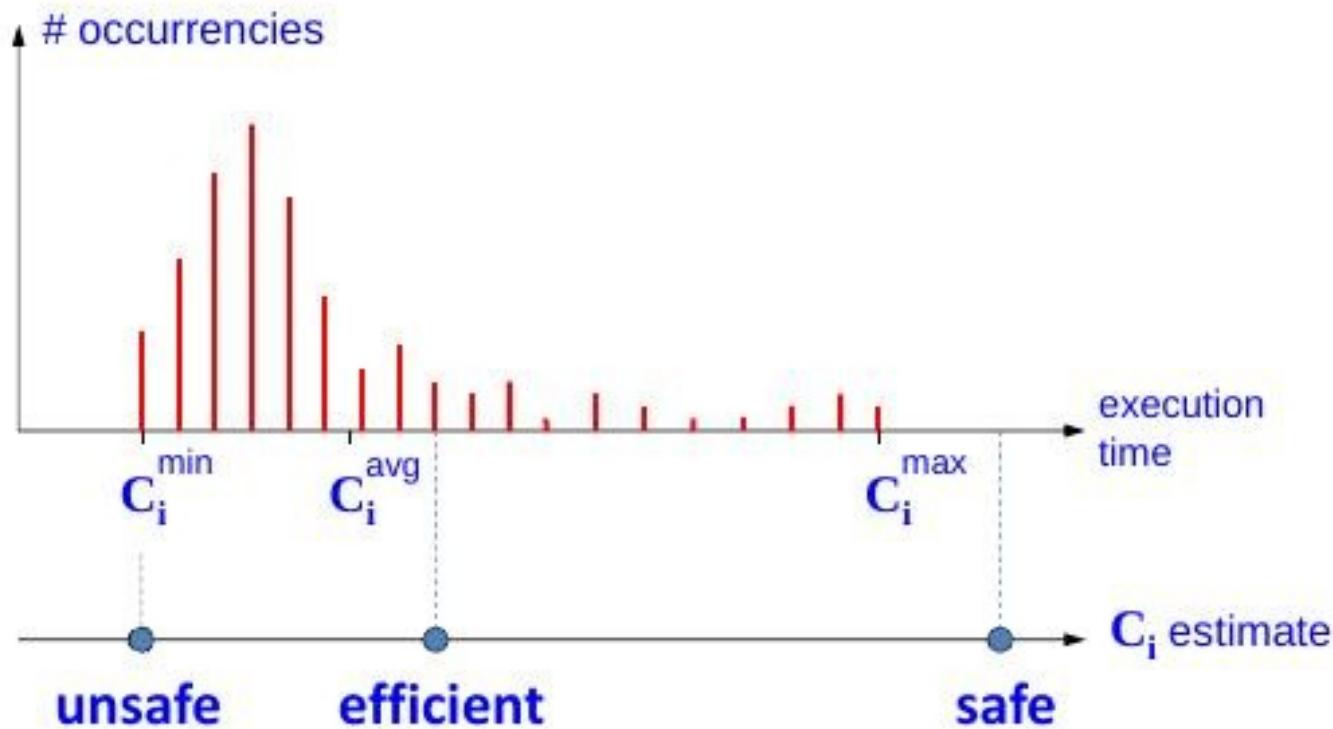
Estimando o Pior Caso



- Cada job opera em diferentes dados e tem caminhos diferentes
- Mesmo para o mesmo dado, o tempo de computação depende do estado do processador (cache, prefetch queue, preempções)



Previsibilidade x Eficiência



- Existe toda uma teoria de escalonamento e sincronização de tarefas tempo real

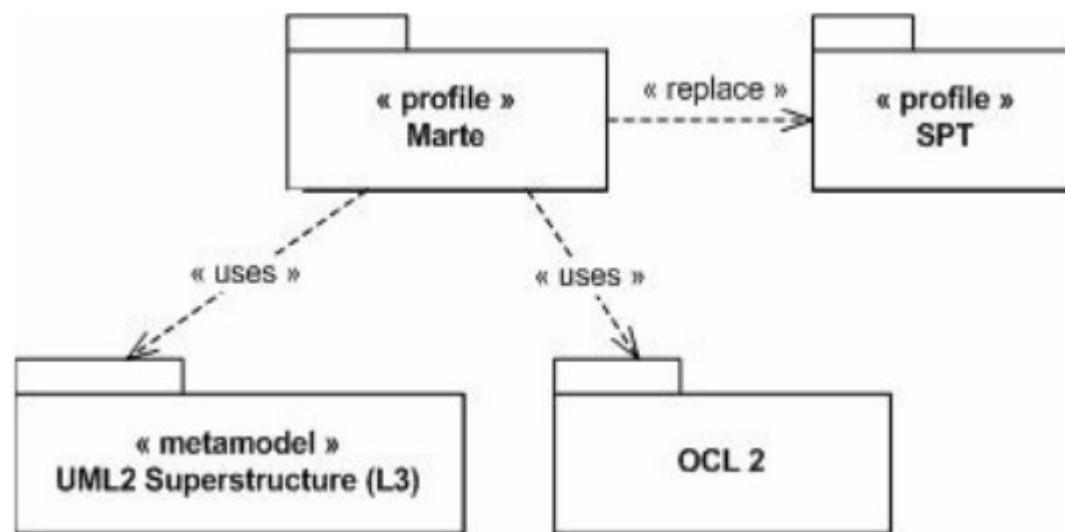
Exemplo: cobertura de código

- TestWell Code Coverage Tool
- Vídeo gravado pela Bruna e adicionado no Moodle
 - Exemplo também para a parte de teste de cobertura de código

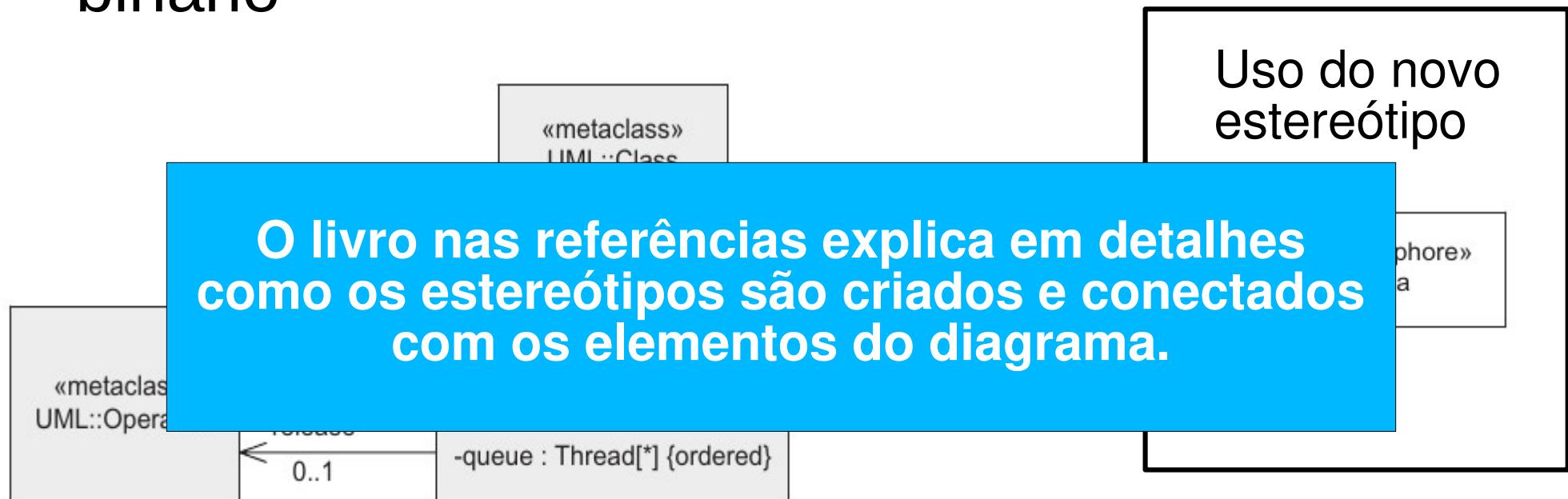
Agenda

- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- Modelagem de requisitos não-funcionais
- Expressão dos dispositivos de hardware
- Exemplos
- Exercícios

- Perfil UML para desenvolver sistemas embarcados e de tempo real
- MARTE - Modeling and Analysis of Real-Time and Embedded systems
- É o sucessor do perfil SPT (Schedulability, Performance and Time)



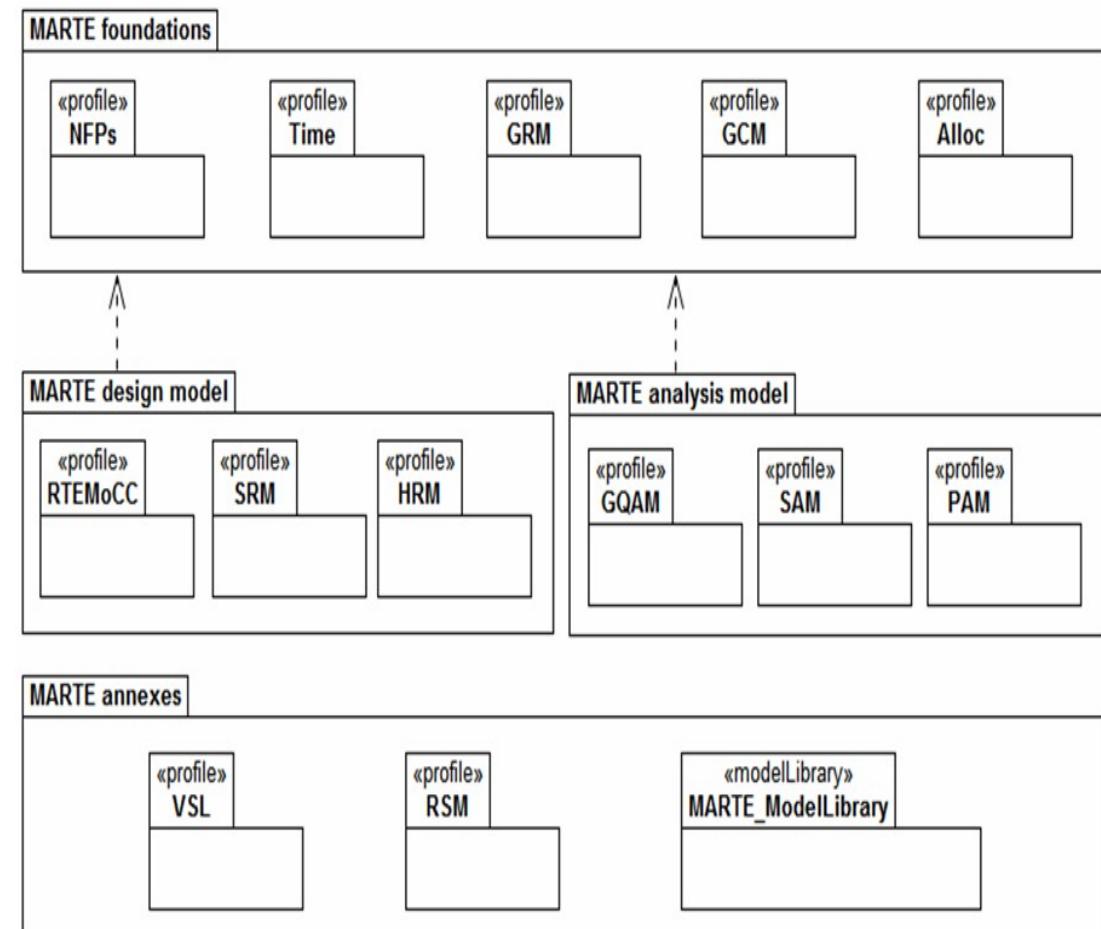
- É baseada em metamodelo
 - Usa diagrama de classes para representar o próprio modelo
- O ponto central é o uso de estereótipos
- Ex: Criando um novo estereótipo Semáforo binário



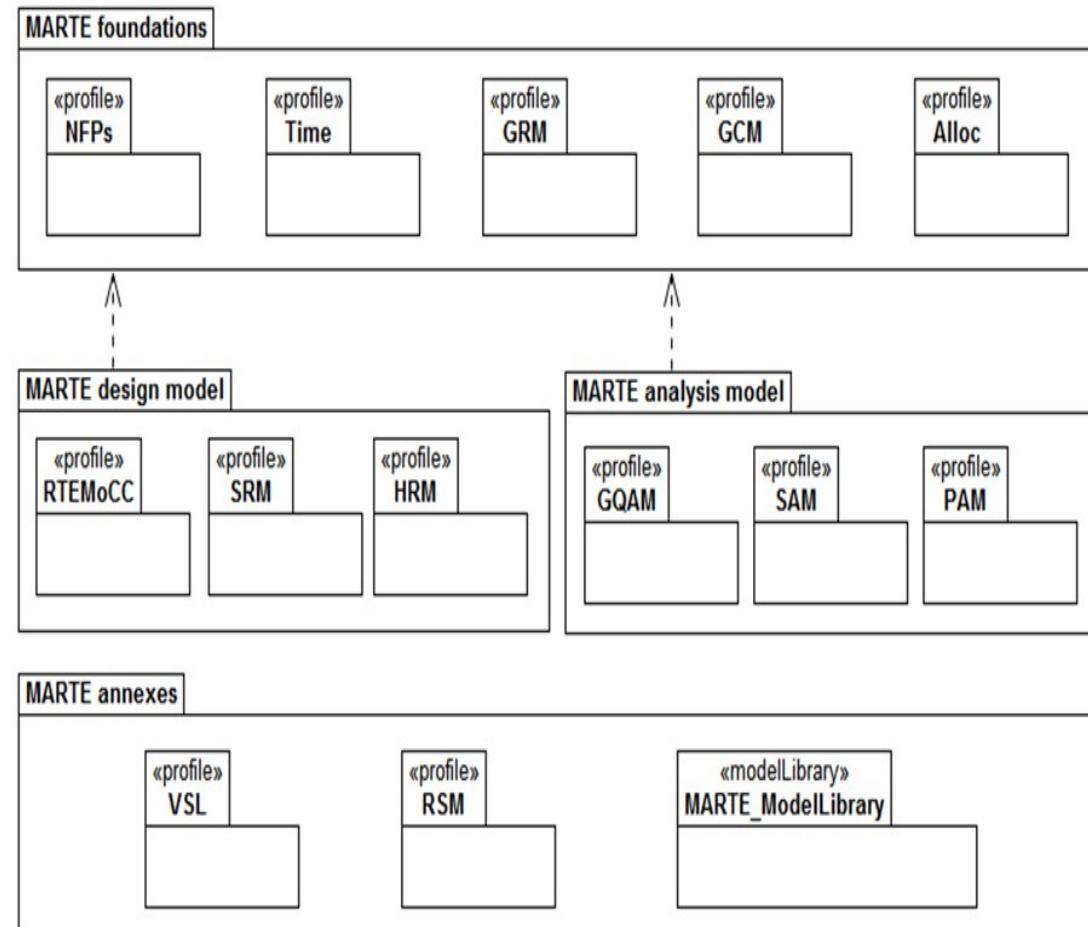
■ Os benefícios em usar o perfil são

- Uma maneira comum de modelar hardware e software em um Sistema Embarcado e de Tempo Real
 - Melhora a comunicação entre as equipes
- Interoperabilidade entre ferramentas usadas para especificação, projeto, verificação e geração de código
- Promover a construção de modelos que possam ser usados para fazer previsões quantitativas a respeito das características relacionadas a sistemas embarcados e de tempo real, considerando HW e SW

- Estruturado em 2 linhas principais:
 - Modelar as características de sistemas TR e Embarcados
 - Análise das propriedades do sistema
- Usado nos estágios de especificação, projeto e verificação e validação
- Profiles definem conjuntos de estereótipos
 - Especificação é longa (800 páginas)

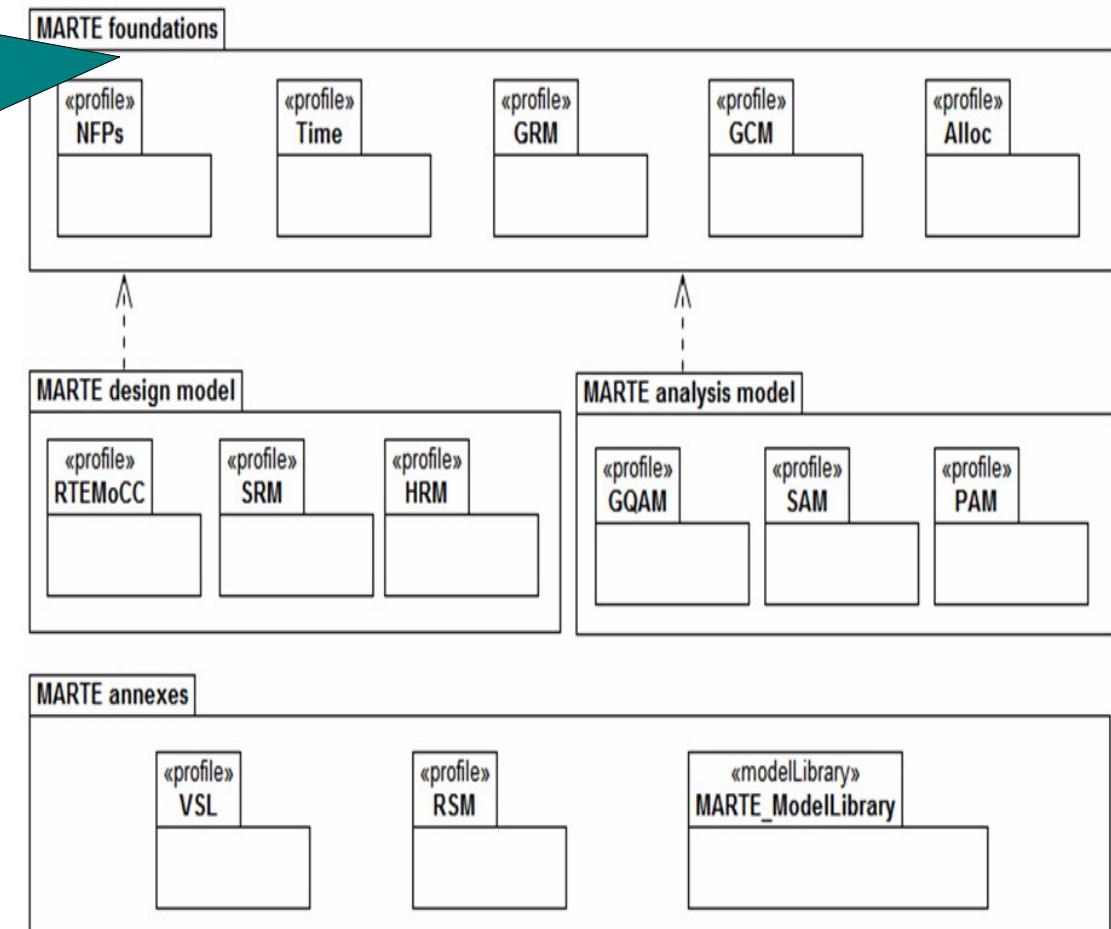


- NFP: non-functional properties
- GRM: generic resource modeling
- GCM: generic component model
- Alloc: allocation modeling
- RTEMoCC: RTE Model of Computation and Communication
- SRM: software resource modeling
- HRM: hardware resource modeling
- GQAM: generic quantitative analysis modeling
- SAM: schedulability analysis modeling
- PAM: performance analysis modeling
- VSL: value specification language
- RSM: repetitive structure modeling



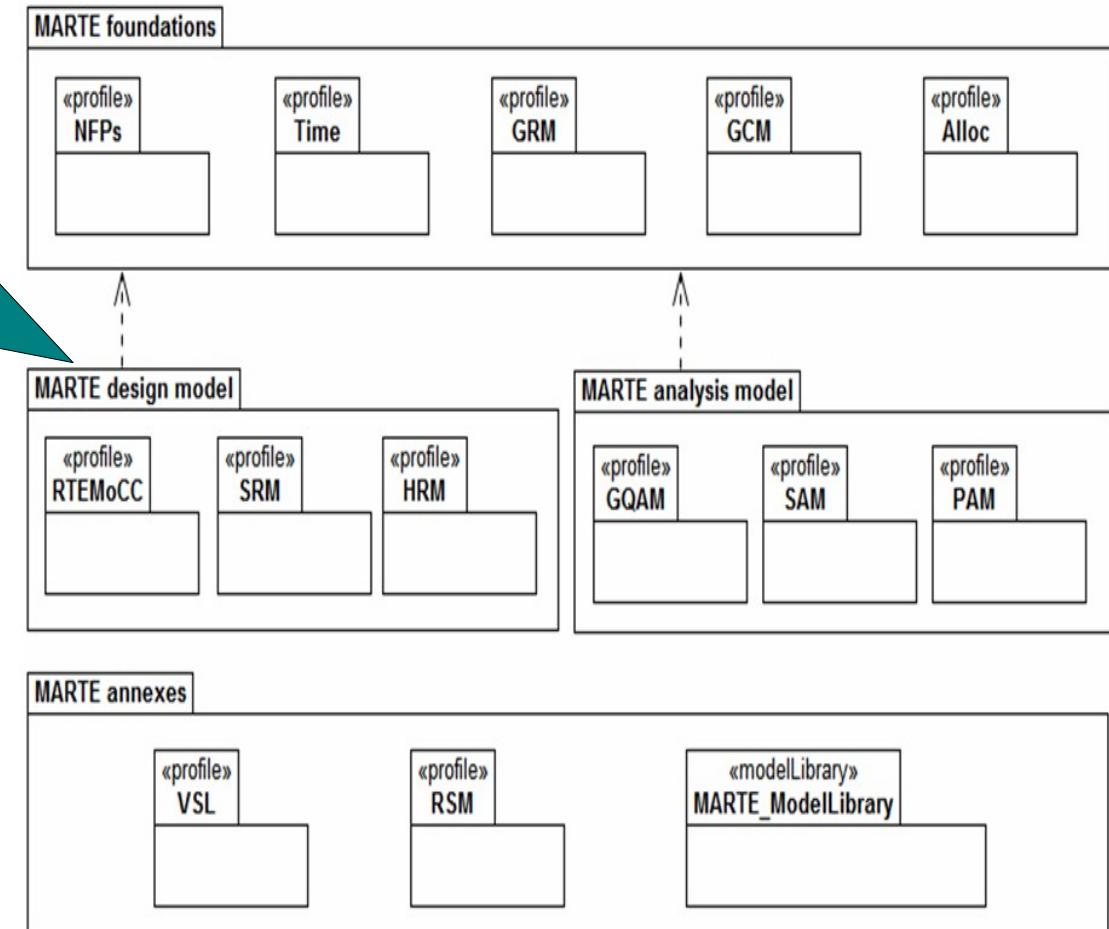
Conceitos bases
refinados tanto para a
modelagem, quanto
para a análise.
Modelagem genérica
com GRM e GCM

Ex: <<resource>>

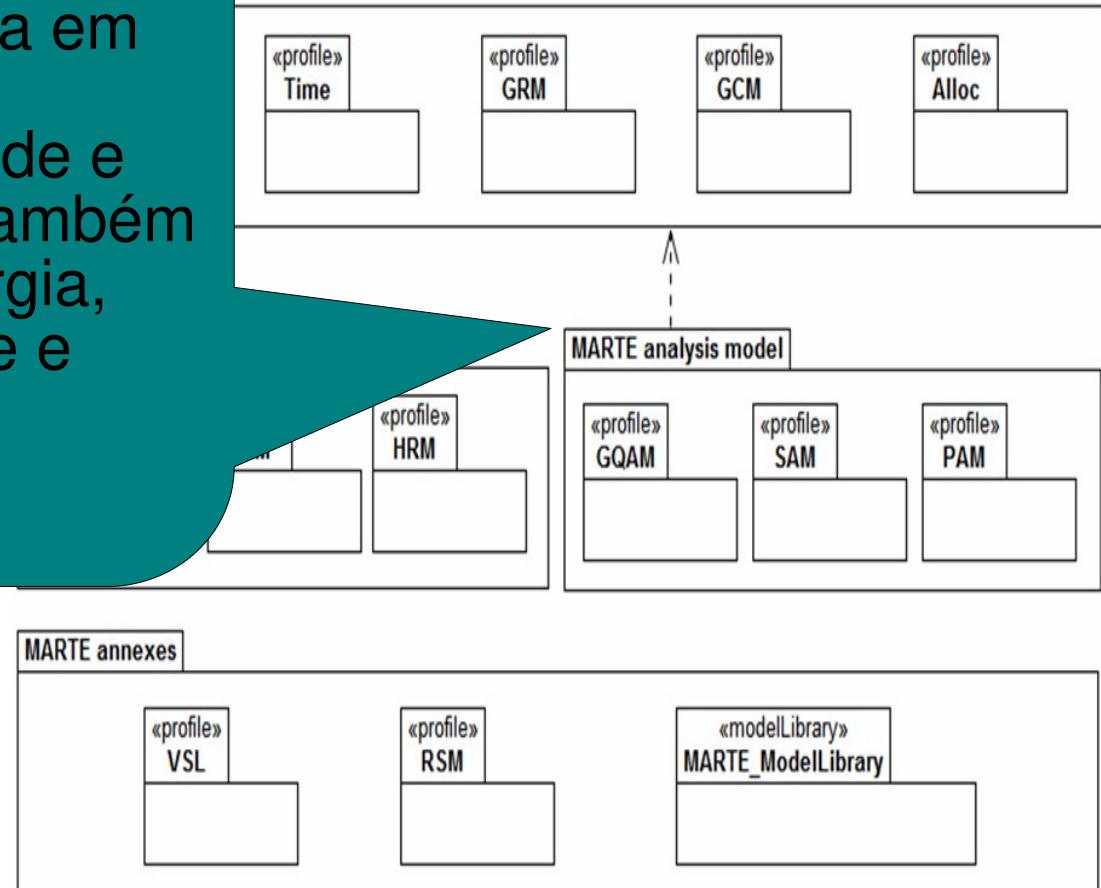


Modelagem detalhada
(SRM e HRM)
tanto de SW quanto de
HW

Ex: <<SwResource>>
<<HwResource>>

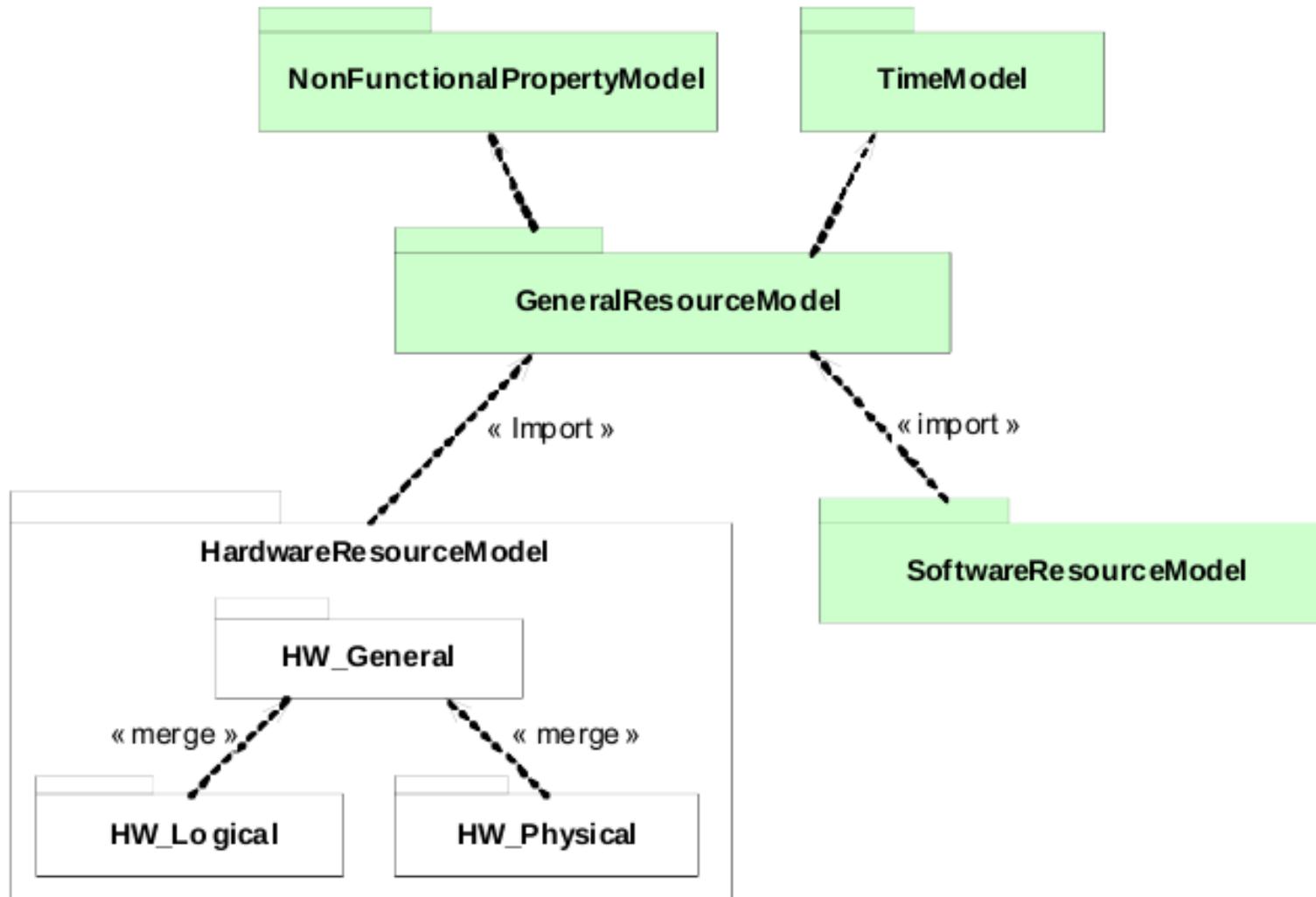


Análise baseada em software (escalonabilidade e desempenho) e também memória, energia, confiabilidade e segurança

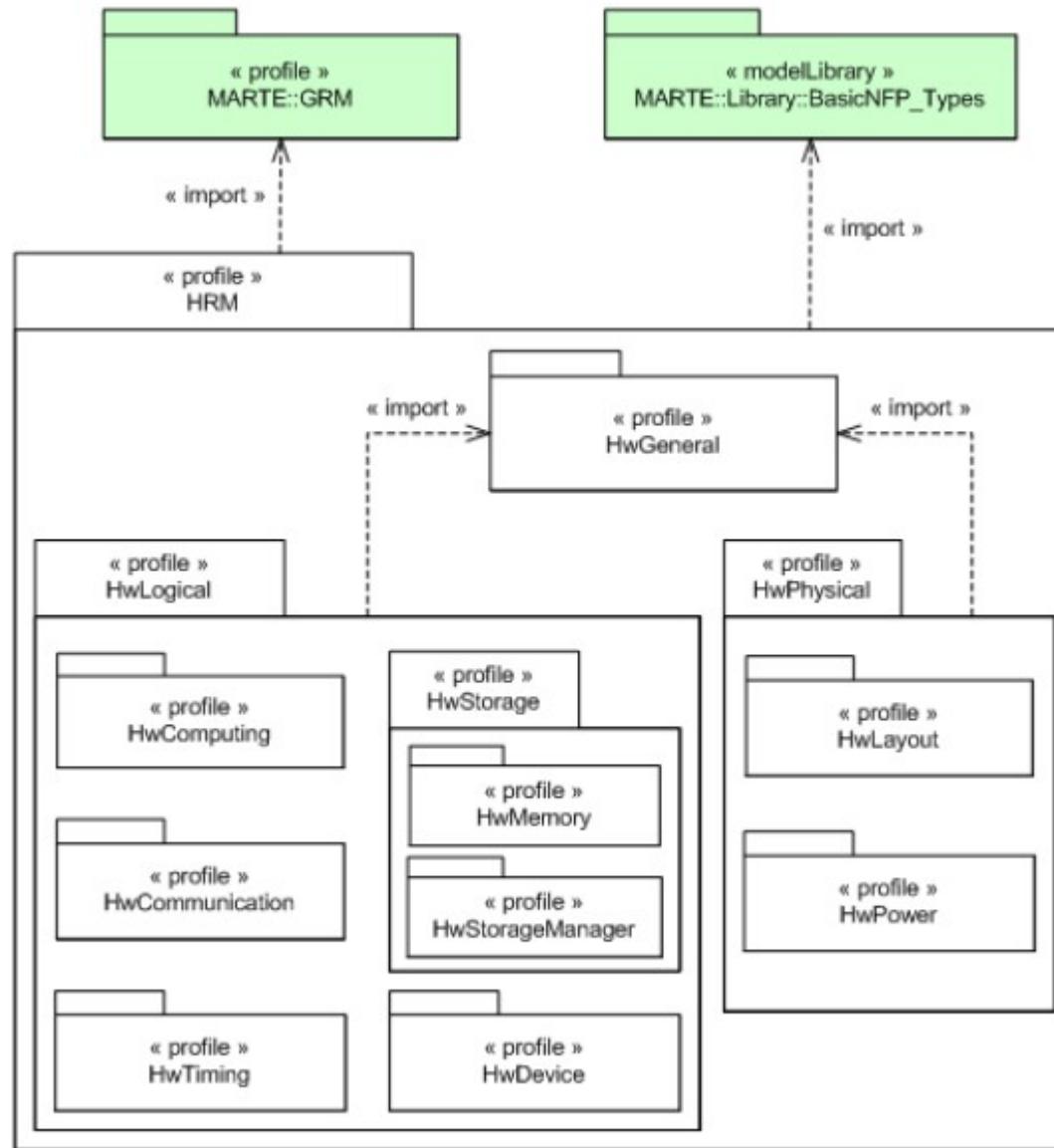


- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- **Resumo dos principais pacotes da UML-MARTE**
- Modelagem de requisitos não-funcionais
- Expressão dos dispositivos de hardware
- Exemplos
- Exercícios

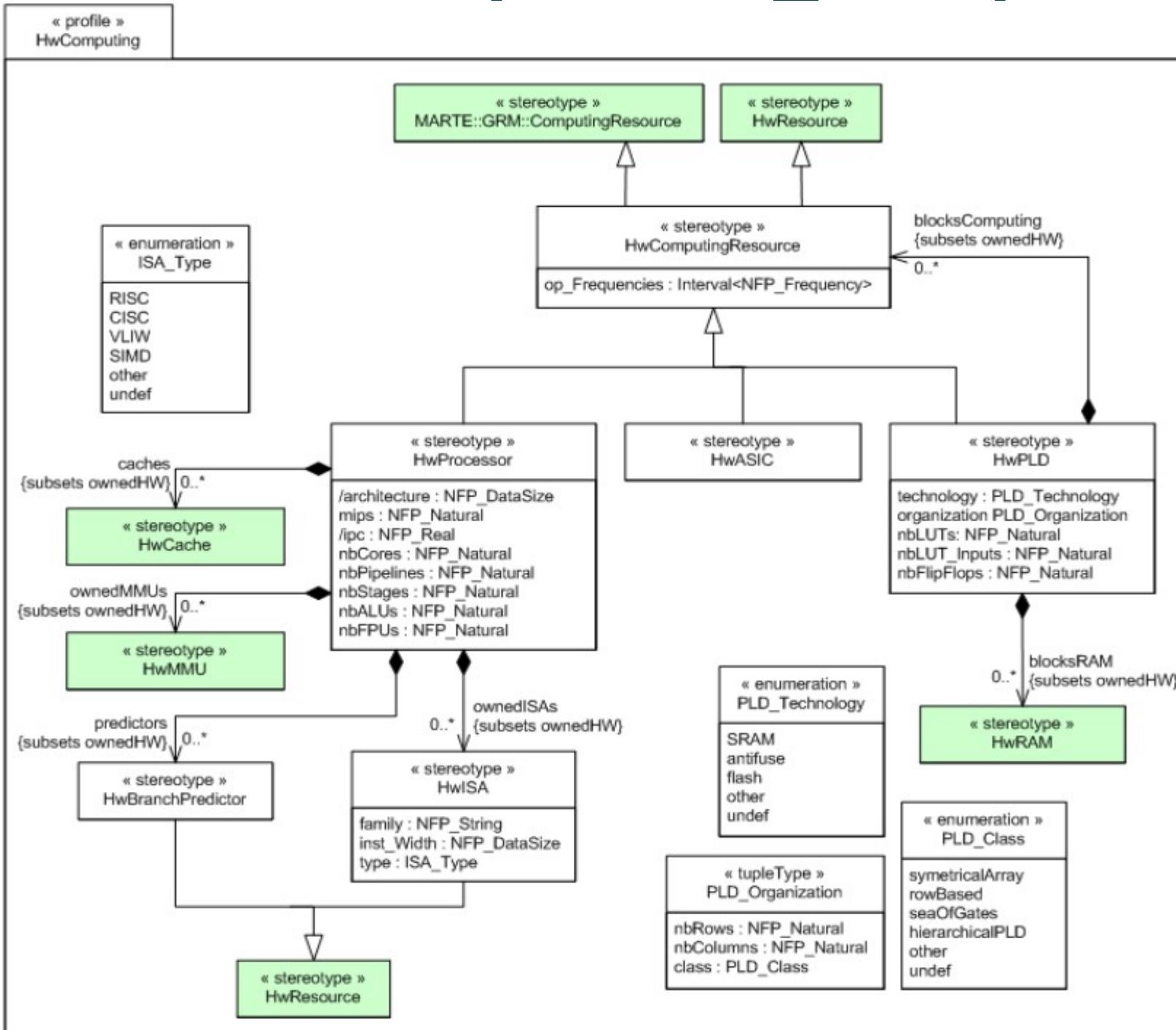
Profile Hardware Resource Modeling (HRM)



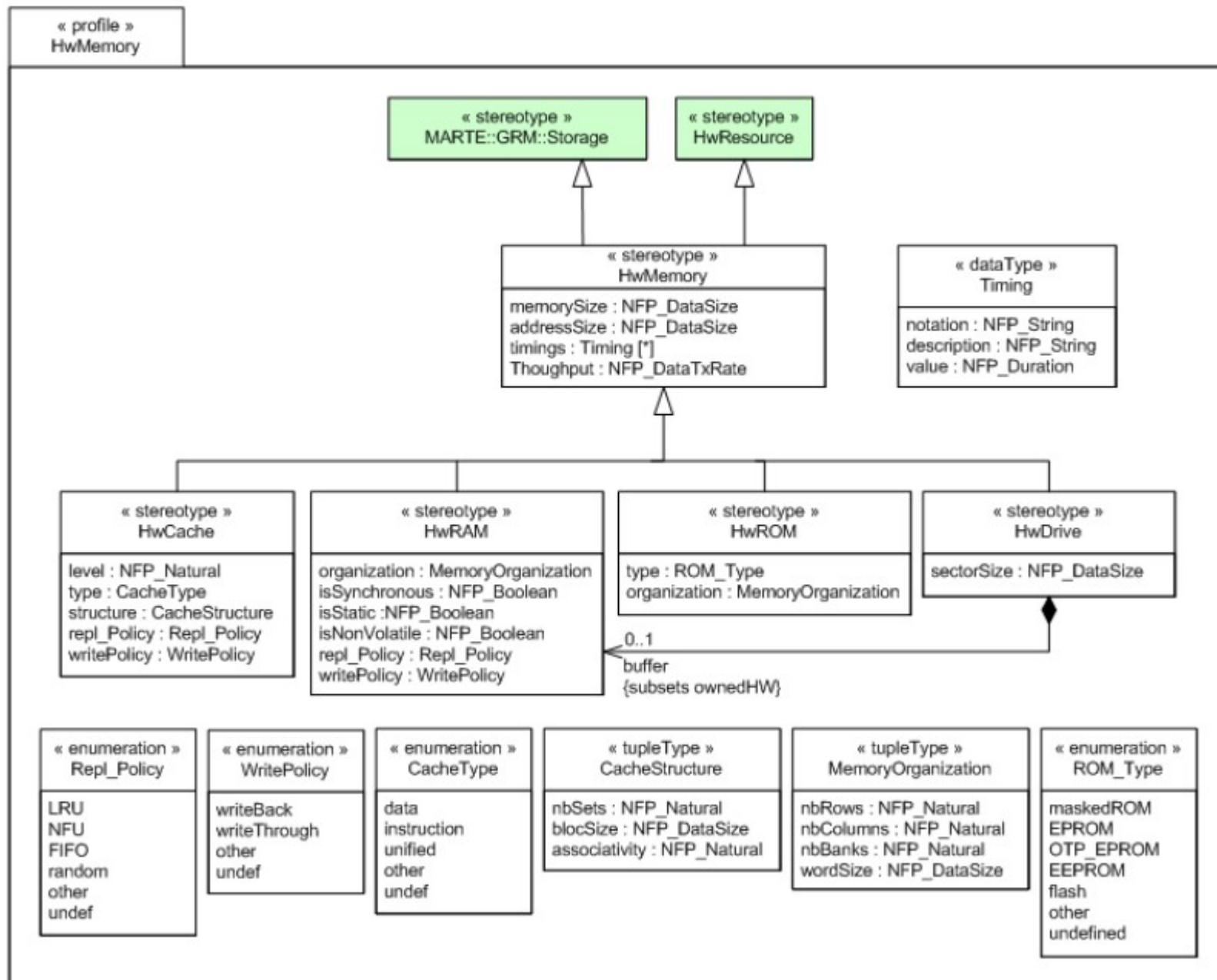
Profile Hardware Resource Modeling (HRM)



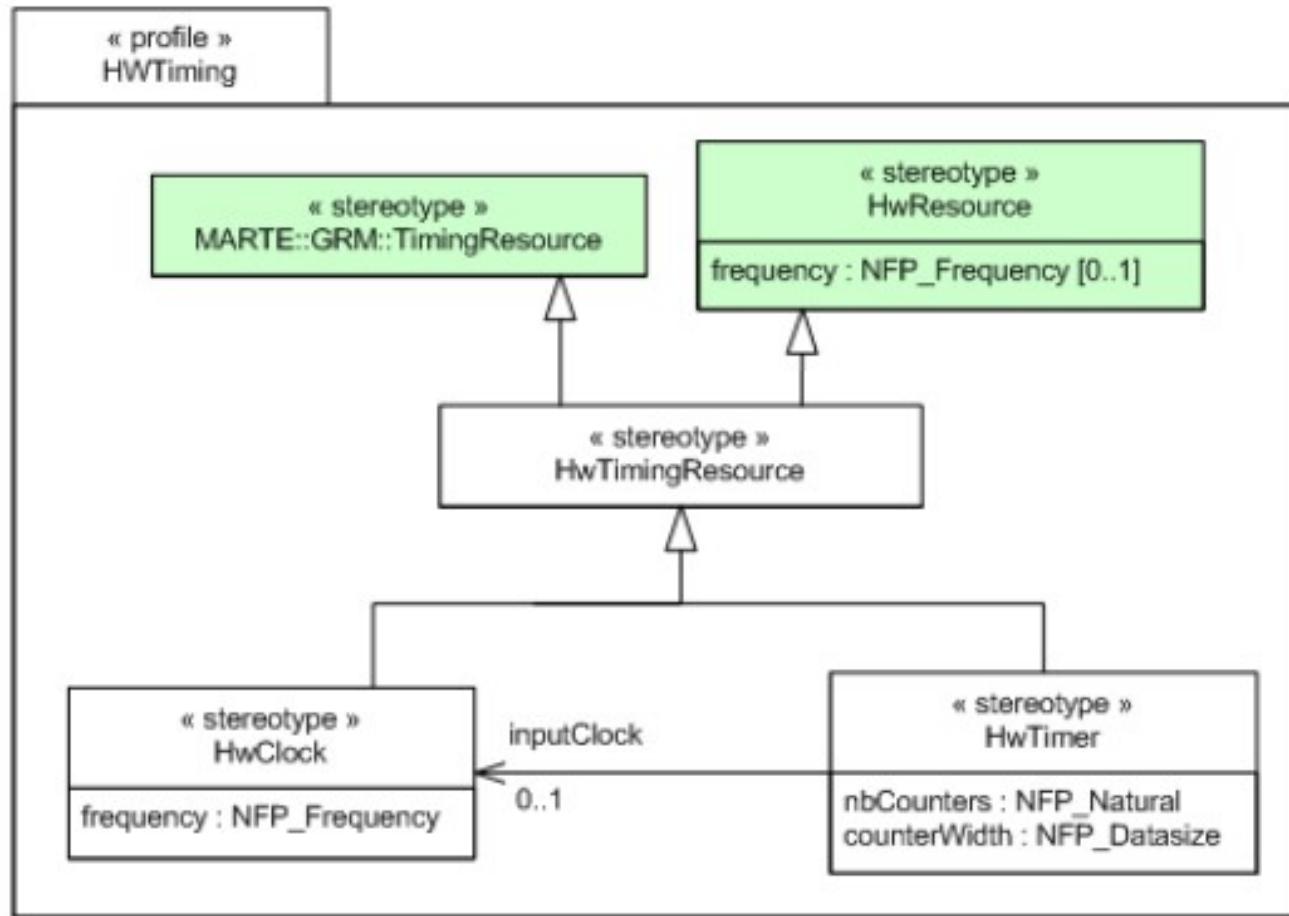
HRM Estereótipos HW_Computing



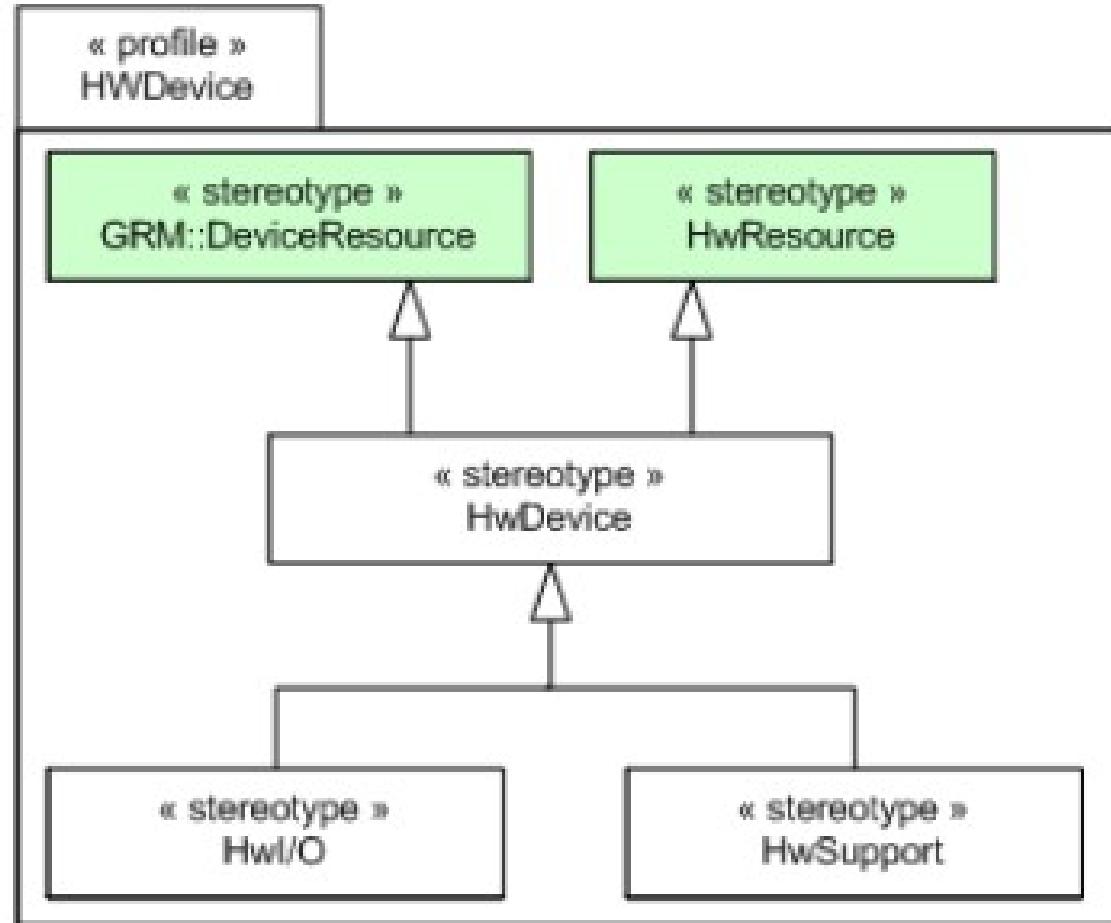
HRM Estereótipos HW_Memory



HRM Estereótipos HW_Timing



HRM Estereótipos HW_Device

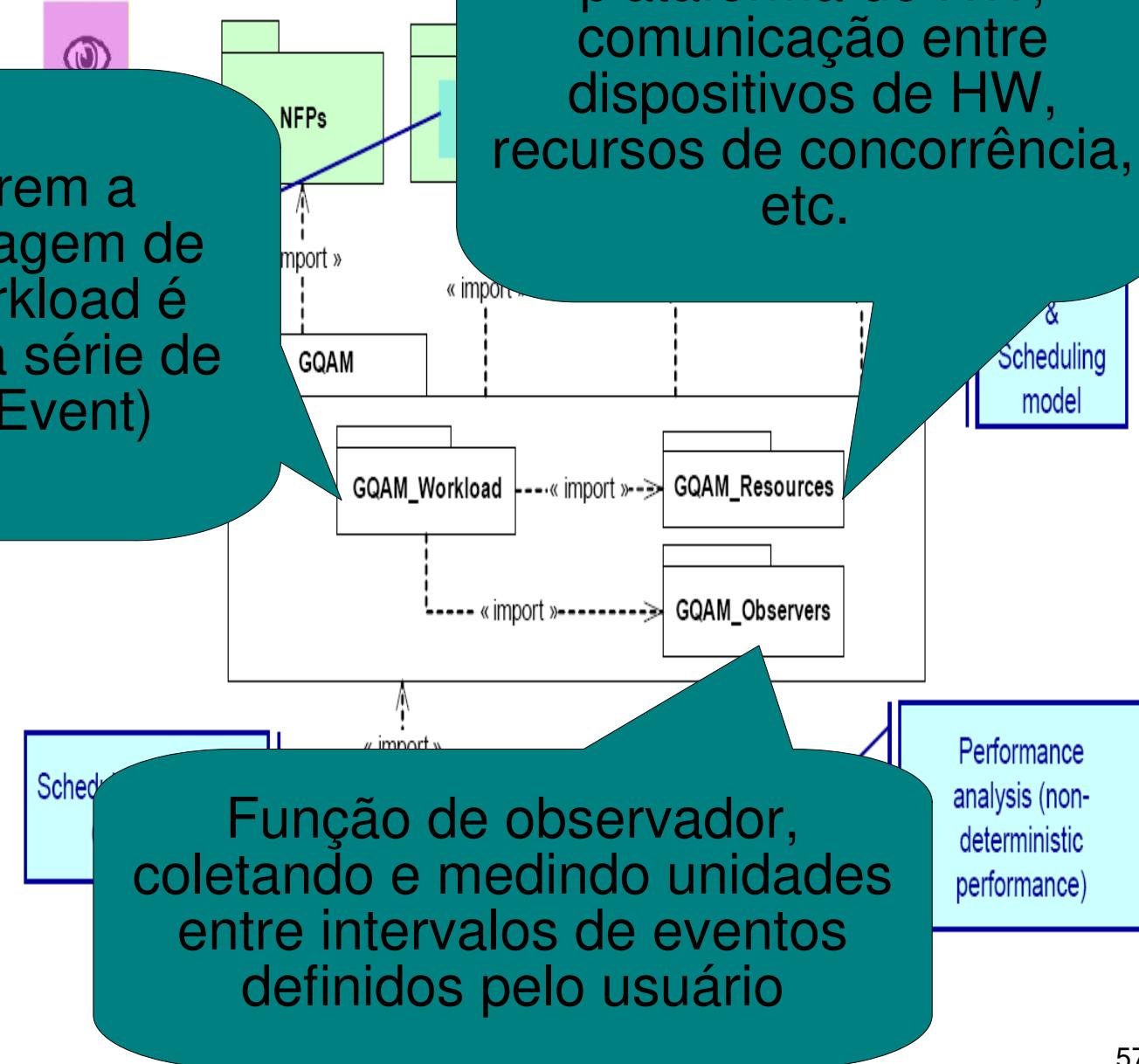


Profile Generic Quantitative Analysis Modeling (GQA)

- Define como o

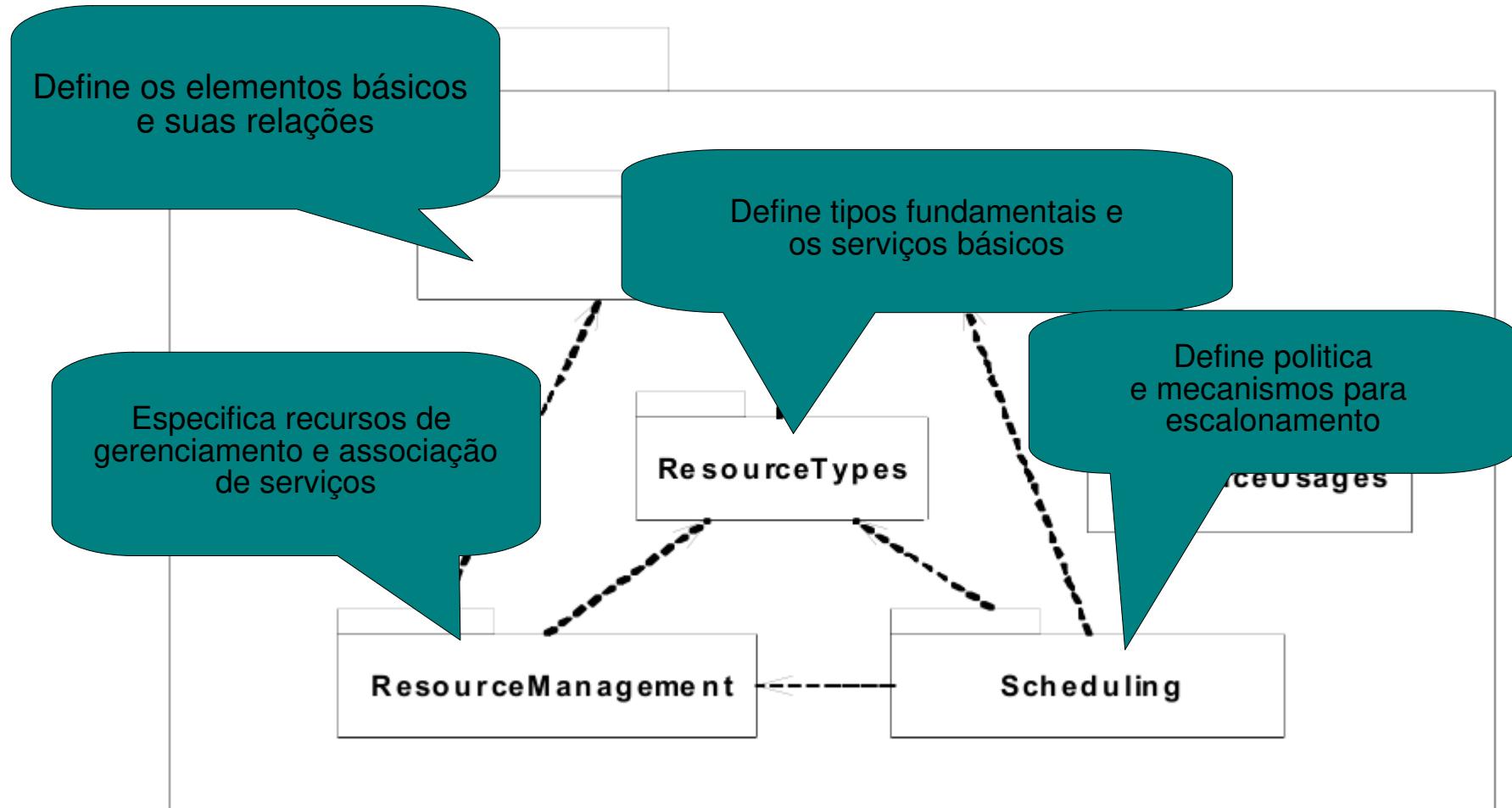
Workloads se referem a situações (e.g. decolagem de um avião). Cada workload é representado por uma série de eventos (WorkloadEvent)

de deadline, utilização de recursos) baseado em entradas NFR (deadlines, QoS)



- Em resposta a um evento, é definido um **BehaviorScenario** no qual é composto por sub-operações (**Steps** – processo em SW ou HW)
- **<<gaAcqStep>>**
 - um Step que adquiri um recurso. Diagramas de máquina de estados, interação, atividades
- **<<gaExecHost>>**
 - define um processador que executa Steps
- **<<gaResourcesPlatform>>**
 - elemento lógico que define uma plataforma
- **<<gaWorkLoadEvent>>**
 - conjunto de eventos relacionados ao comportamento do sistema
- **<<gaWorkloadGenerator>>**
 - gerador de eventos. Diagramas de classes, deployment, sequência, componentes, objetos

- O objetivo deste pacote é oferecer conceitos para modelar uma plataforma geral, para aplicações embarcadas e de tempo real
- O principal conceito deste pacote é o de *Resource*, o qual pode representar uma entidade física ou lógica que oferece serviços
- Pode se fazer o modelo de diferentes plataformas e níveis de detalhes
- O modelo pode ser tanto de hardware (Ex. Memória) ou software (Ex. Sistema operacional de tempo real)



■ <<TimingResource>>

- Representa uma entidade de hardware ou software capaz de seguir os eventos de tempo

**■ <<ConcurrencyResource>> e
<<SchedulableResource>>**

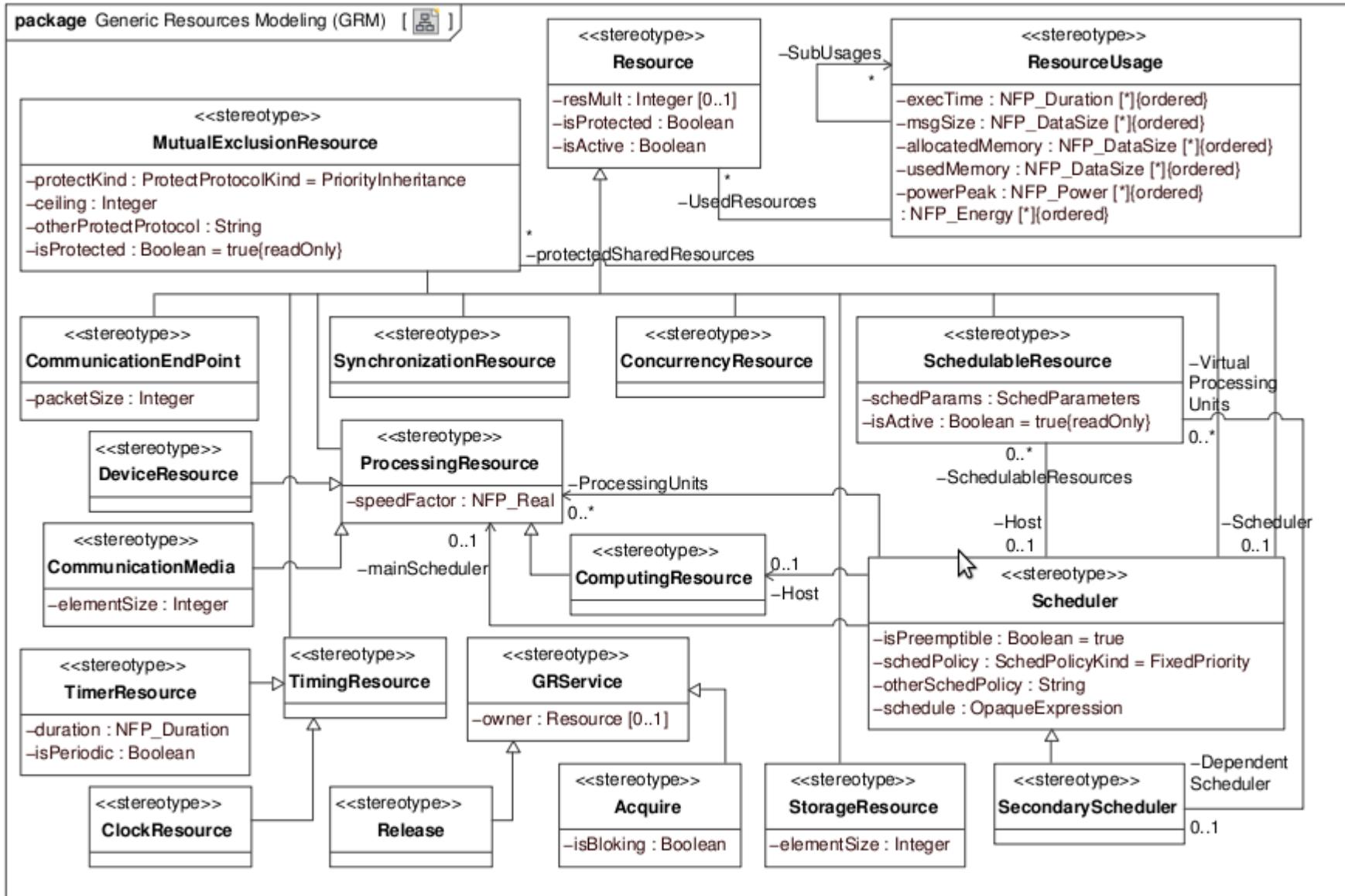
- Representa recursos protegidos e ativos que podem exercer atividades concorrentes

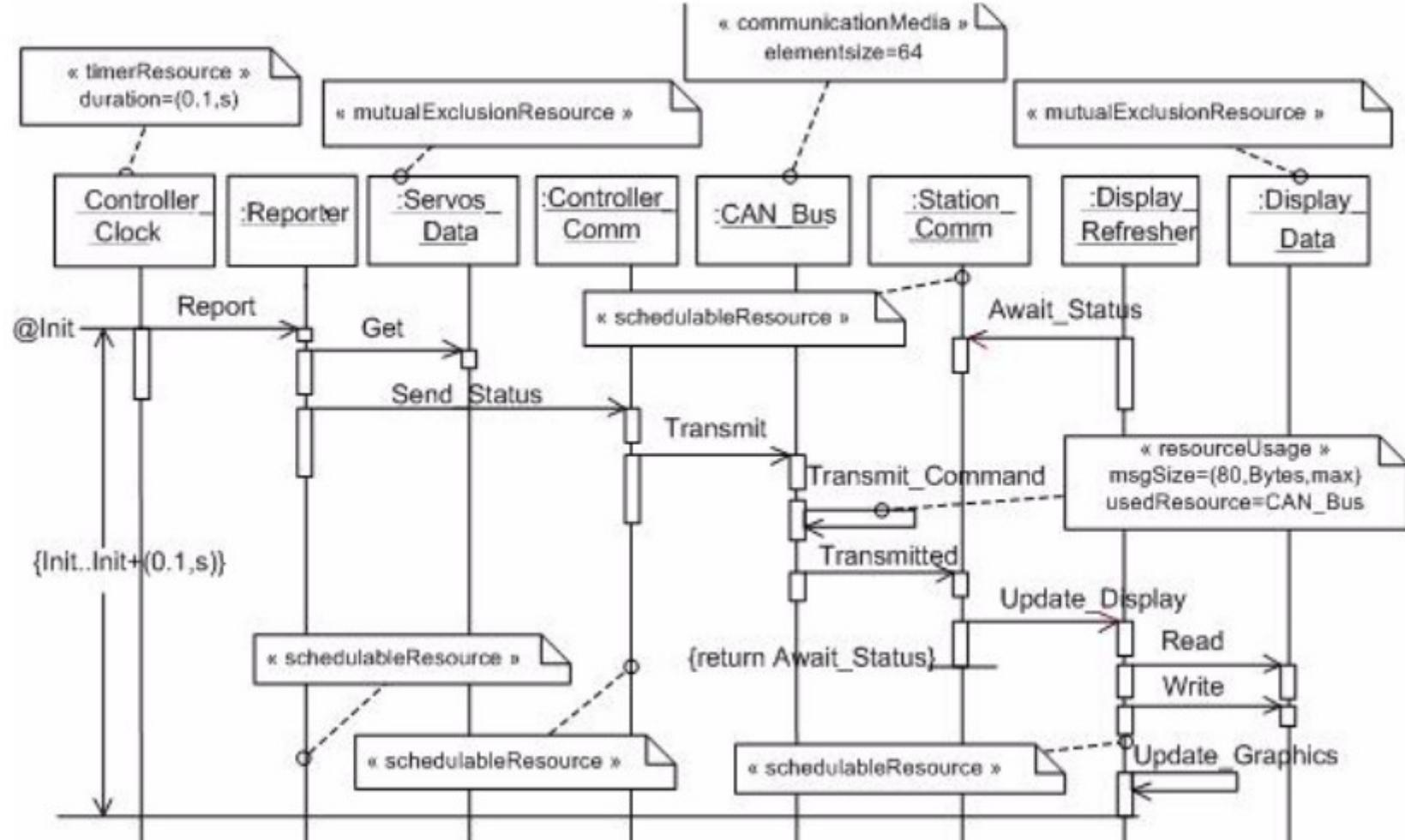
■ <<Scheduler>>

- Coordena o acesso ao <<ProcessingResource>> de um <<SchedulableResource>>

■ <<MutualExclusionResource>>

- Para o controle de acesso concorrente



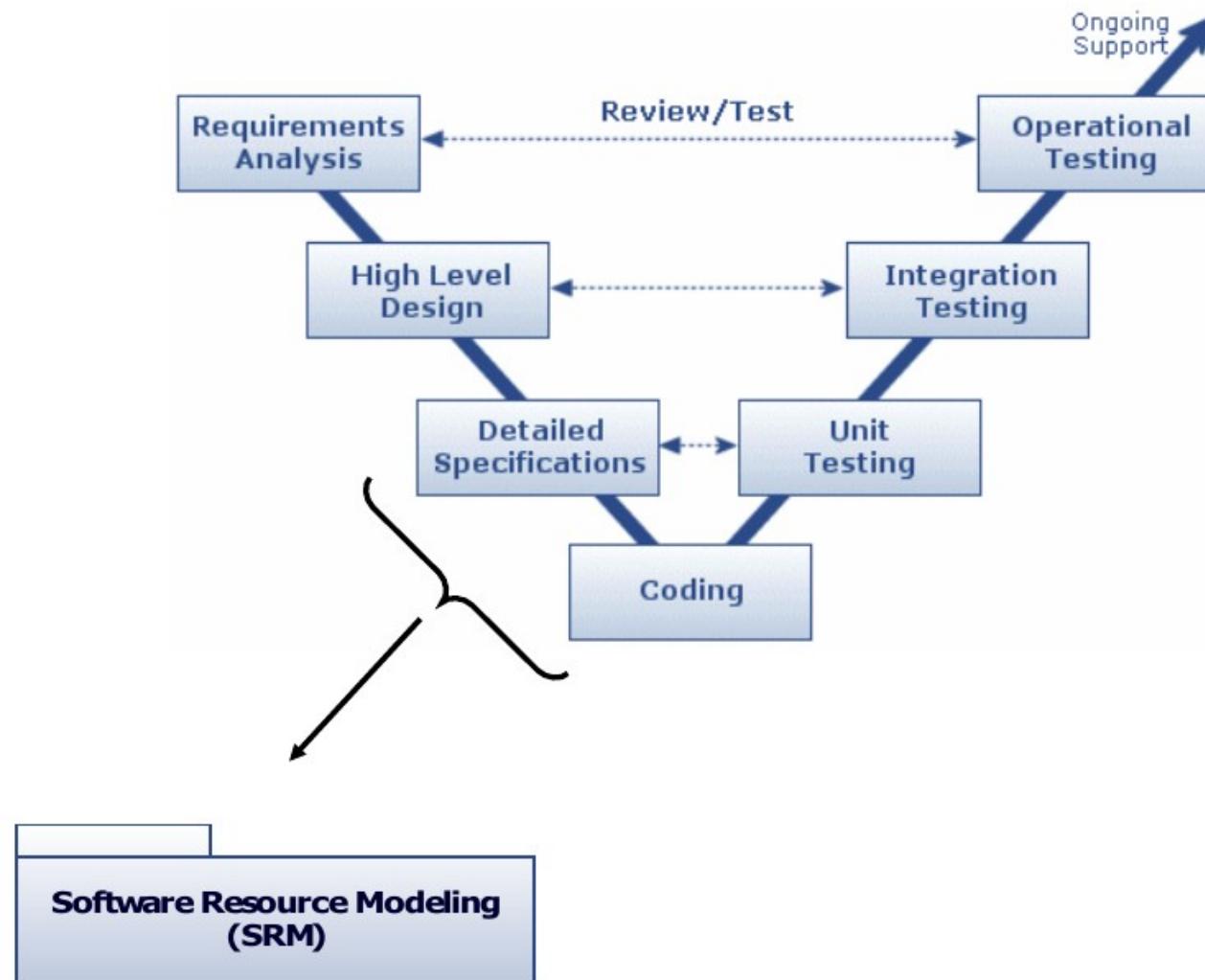


- O perfil permite

- Descrever modelos com múltiplas tarefas (multitask)
- Descrever ferramentas genéricas (geradores de código, transformação de modelos)
- Descrever modelos em uma maneira uniforme e padronizada

Profile Software Resource Modeling (SRM)

- Onde usá-lo dentro dos estágios de desenvolvimento?

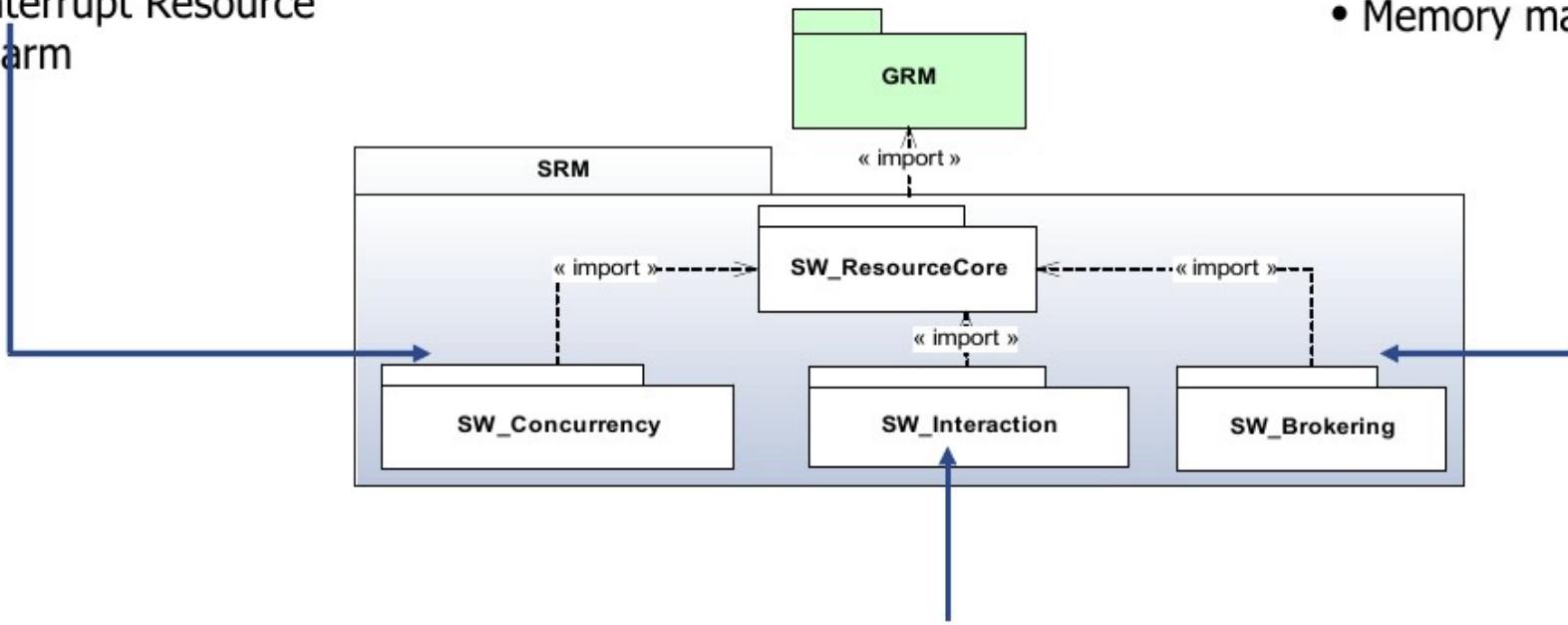


Concurrent execution contexts:

- Schedulable Resource (Task)
- Memory Partition (Process)
- Interrupt Resource
- Alarm

Hardware and software resources brokering:

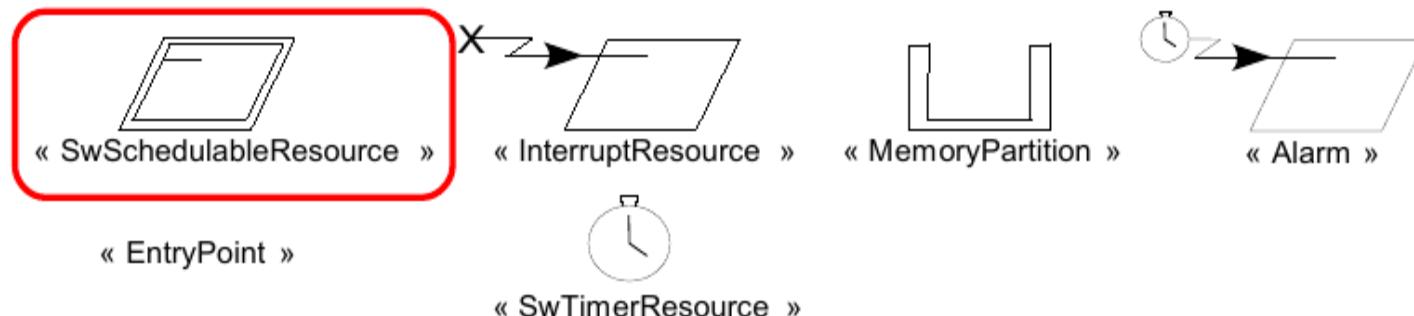
- Drivers
- Memory management



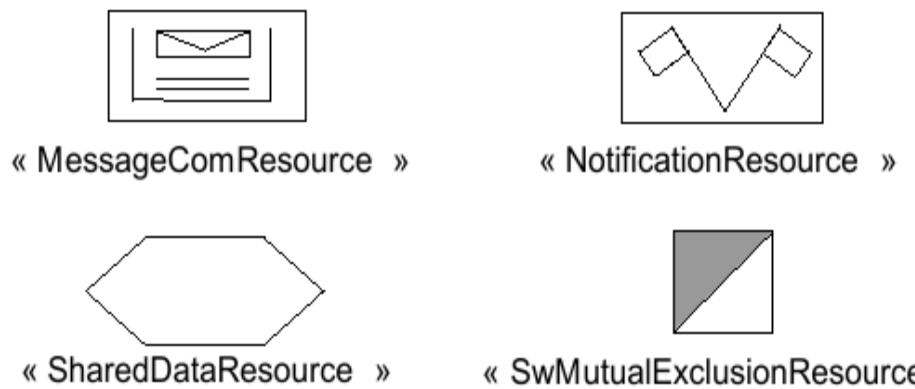
Interactions between concurrent contexts:

- Communication (Data exchange)
 - ✓ Shared data
 - ✓ Message (Message queue)
- Synchronization
 - ✓ Mutual Exclusion (Semaphore)
 - ✓ Notification (Event mechanism)

SRM::SW_Concurrency



SRM::SW_Interaction

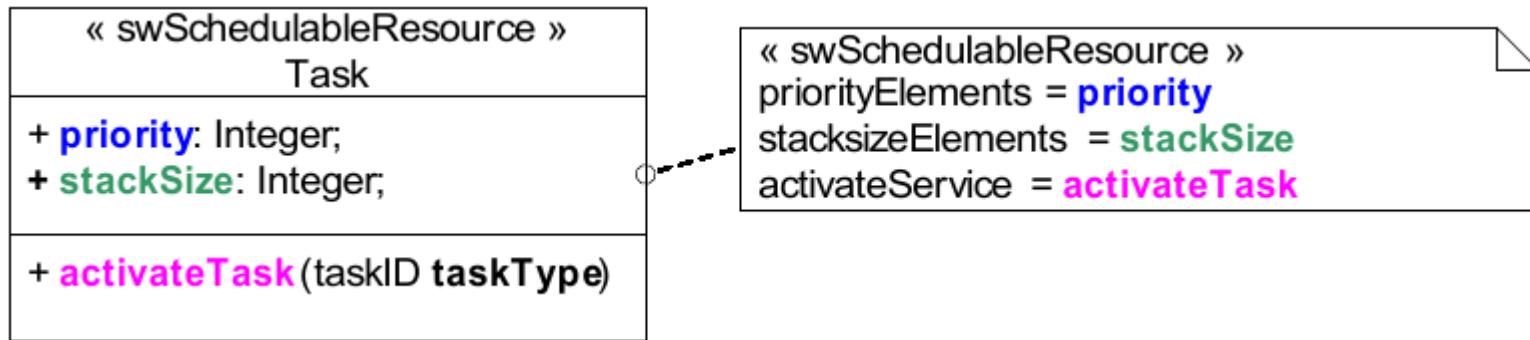


SRM::SW_Brokering

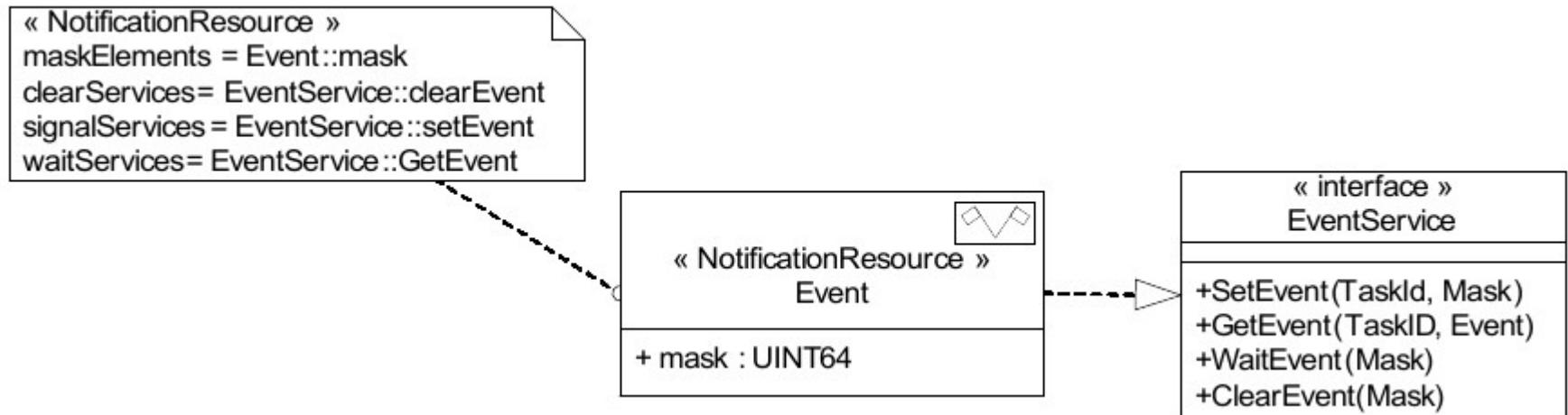


■ <<SwSchedulableResource>>

- Semântica: recurso no qual executa concorrentemente a outro recurso
- Periódico ou aperiódico
- Principais características
 - Tem um entry point (código para executar)
 - Tem um espaço de endereçamento
 - Tem propriedades: prioridade, deadline, período, tamanho da pilha (StackSize)
 - Provê serviços: Activate, Resume, Suspend, ...

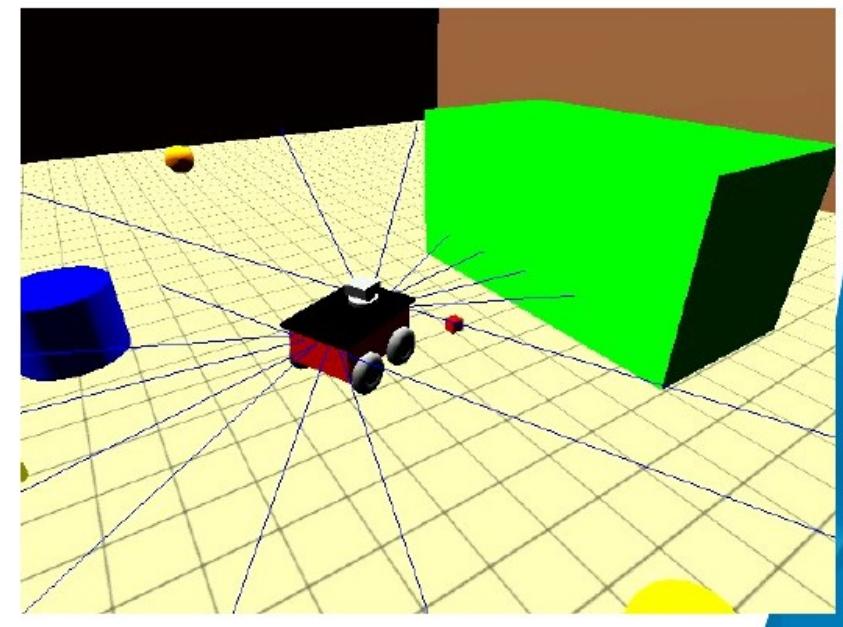


- O evento é um meio de sincronização
- Pode iniciar uma transição de estado de tarefas para ou do estado de espera (waiting)
- Propriedade específica: Mask
- Serviços específicos
 - SetEvent
 - WaitEvent
- O perfil define NotificationResource
 - Suporta o controle de fluxo notificando a ocorrência de condições para recursos que estão esperando o evento (ligando ao entry point)



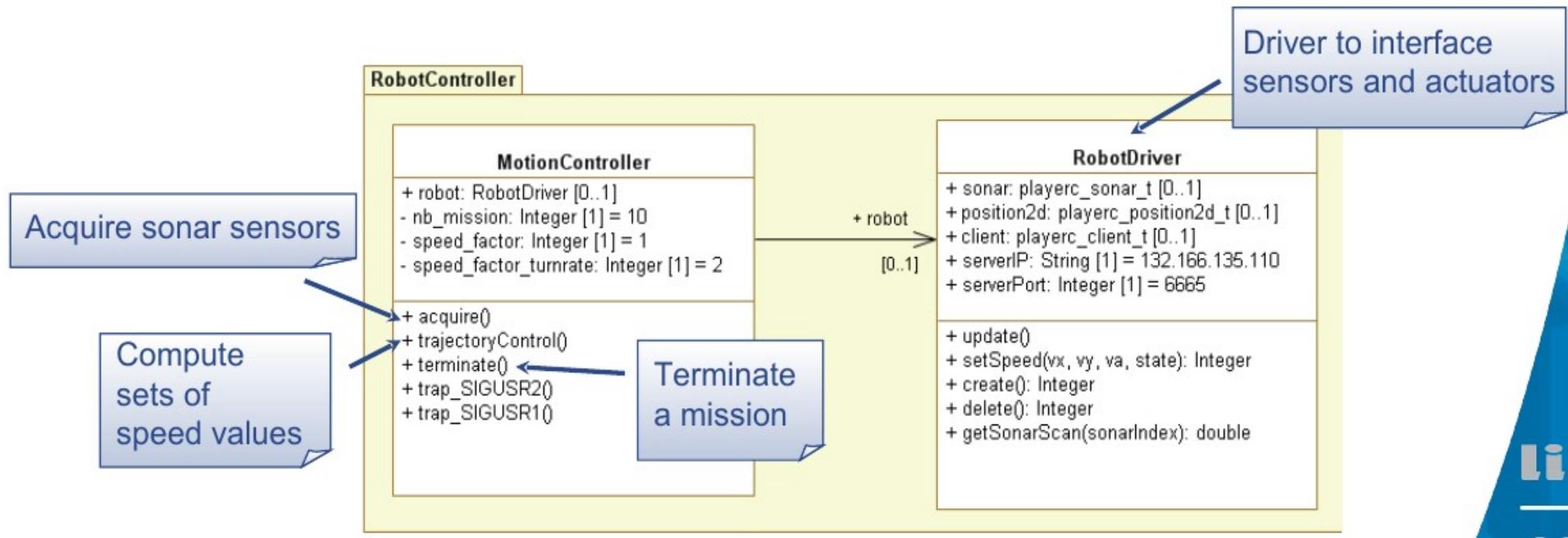
LISHA Exemplo: software para controlar um robô simples

- Objetivo é projetar o sistema de controle para um robô móvel autônomo
- Robô
 - Tem 4 rodas
 - Uma câmera
 - 8 sensores ultrassônicos
- O controlador
 - 2 tarefas periódicas
 - Aquisição: acessa os sensores a cada 1 ms
 - Controle de trajetória: seta a nova velocidade a cada 4ms
- Suporte de execução OSEK/VDX
 - RTOS Trampoline (<http://trampoline.rts-software.org>)



LISHA Exemplo: software para controlar um robô simples

■ Projeto da aplicação

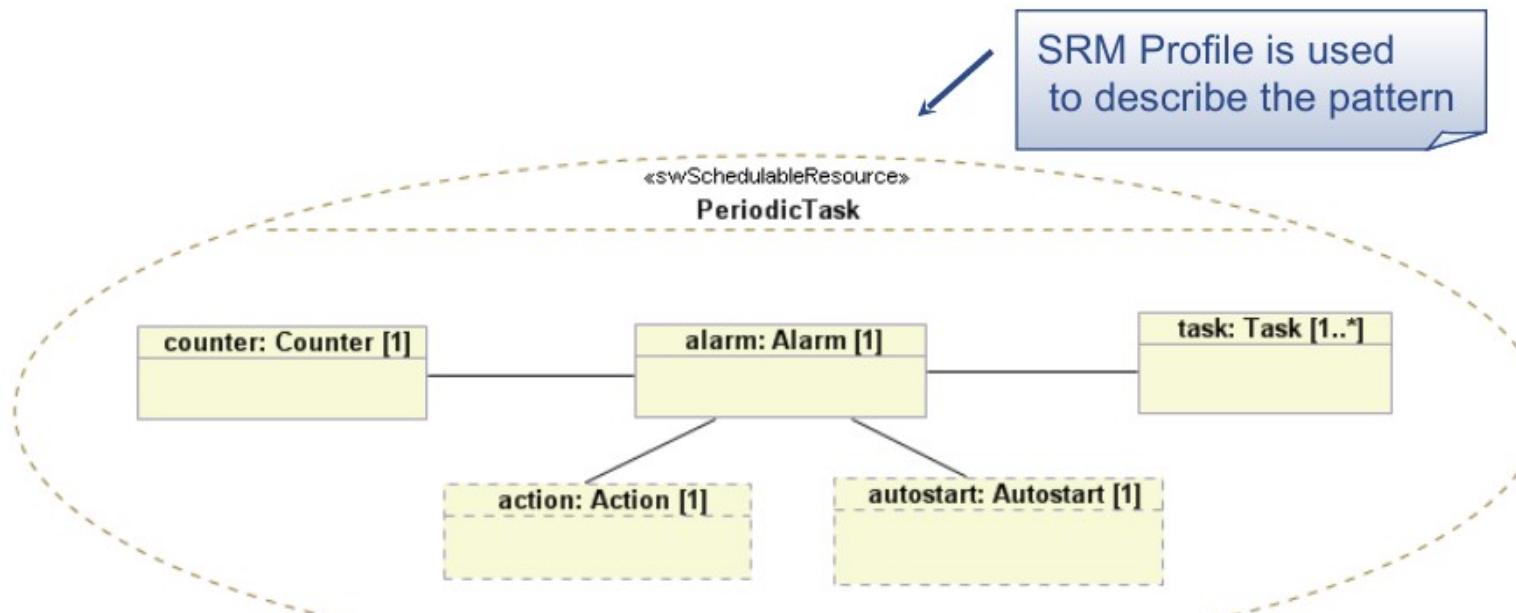


Exemplo: software para controlar um robô simples

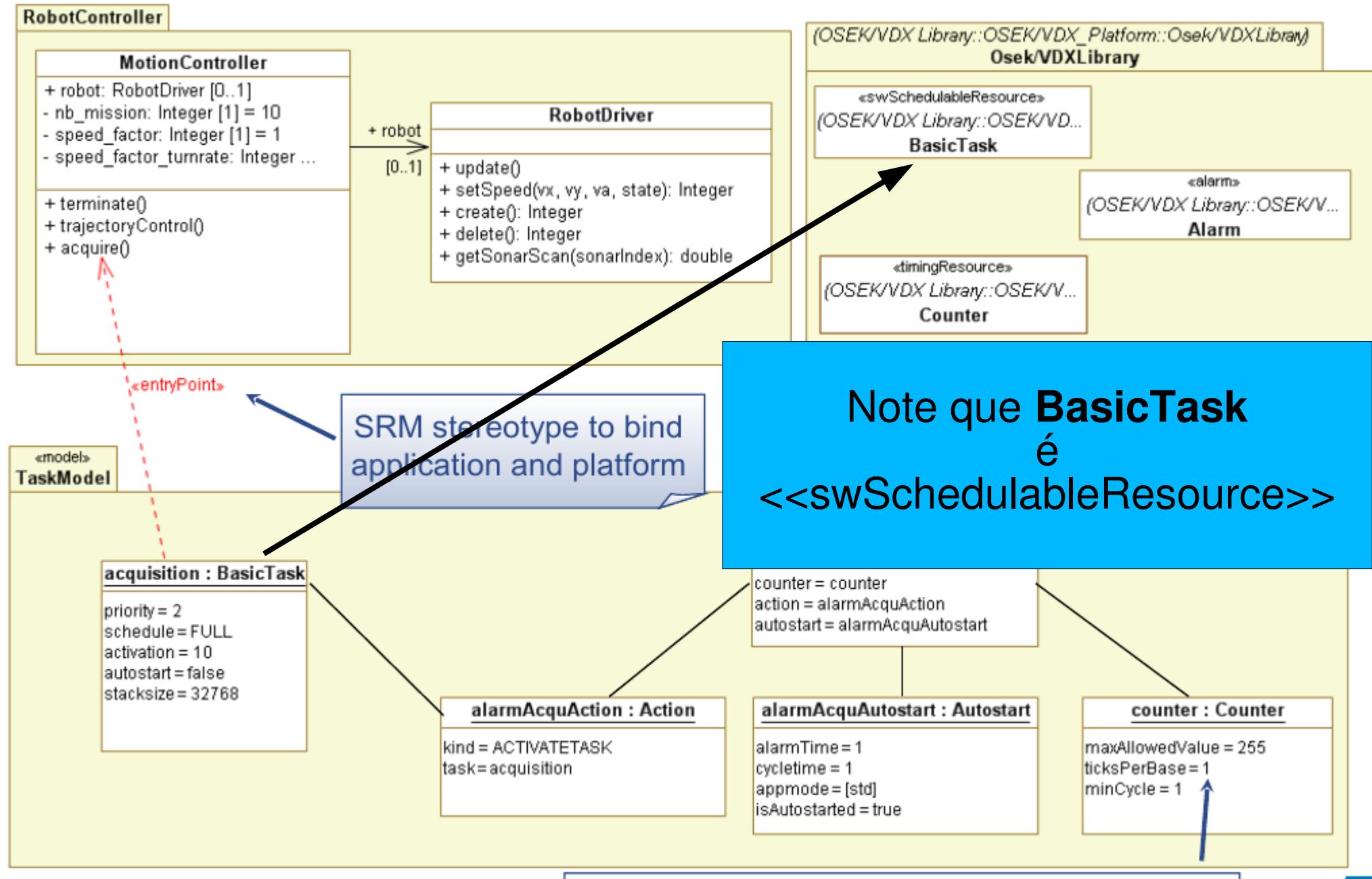
- As duas tarefas periódicas:
 - acquire() com período de 1ms
 - trajectoryControl() com período de 4ms
- A classe RobotDriver não tem tarefas

Exemplo: software para controlar um robô simples

- Padrão para tarefas periódicas do OSEK/VDX
 - Um OSEK/VDX counter
 - Controle do período da tarefa
 - Task OSEK/VDX
 - Entry point para tarefas periódicas
 - Um OSEK/VDX Alarm
 - AutoStart: acionado pelo counter
 - Action: ativa a tarefa



Exemplo: Modelo de tarefas



- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- **Modelagem de requisitos não-funcionais**
- Expressão dos dispositivos de hardware
- Exemplos
- Exercícios



Modelagem de requisitos não-funcionais

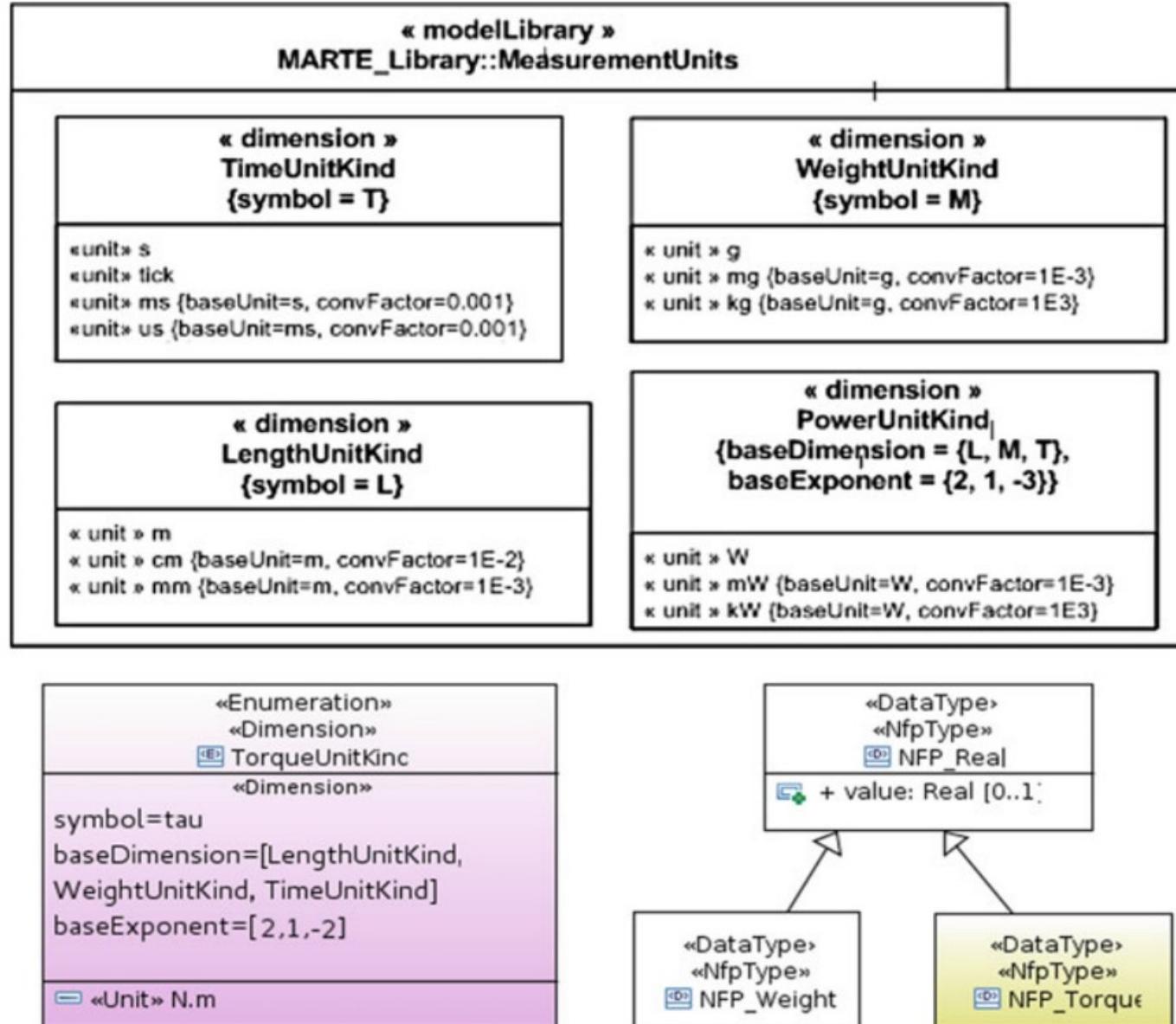
- O perfil NFP tem mecanismos para descrever aspectos quantitativos e qualitativos das propriedades do sistema
- Define um conjunto de unidades, dimensões e quantidades
- O NFP vem acompanhado com uma linguagem chamada Value Specification Language (VSL) que define a sintaxe usada nas expressões das propriedades não funcionais



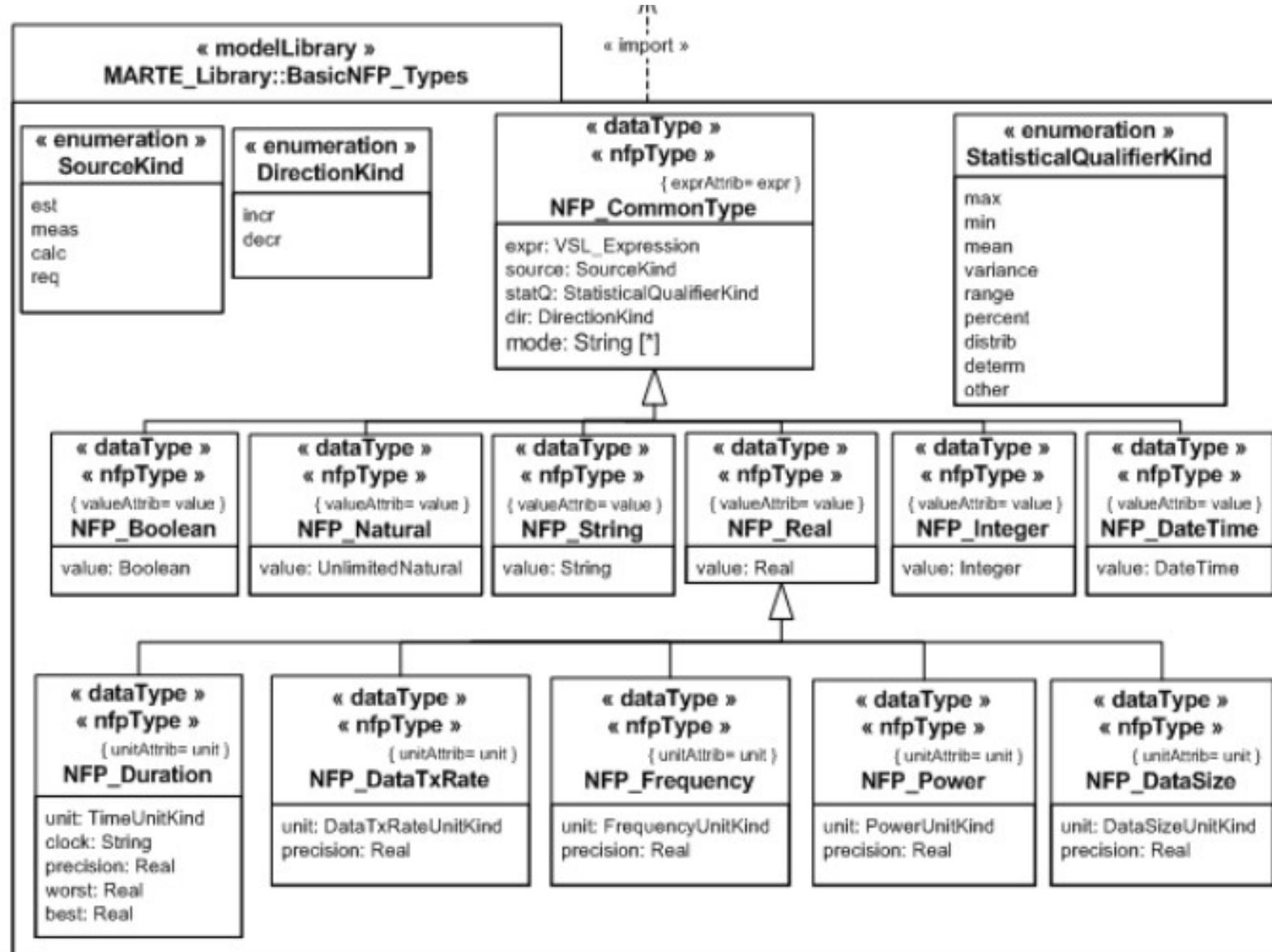
Modelagem de requisitos não-funcionais

- O perfil define tipos prontos para uso que devem ser importados quando relevantes ao projeto
 - Relacionados com o sistema internacional de unidades (SI)
- Ex: dimensão de Tempo define segundos, ms, us, tick
- O perfil também define tipos básicos
 - NFP_Real
 - NFP_Integer
 - NFP_DateTime
 - NFP_String
 - NFP_Boolean

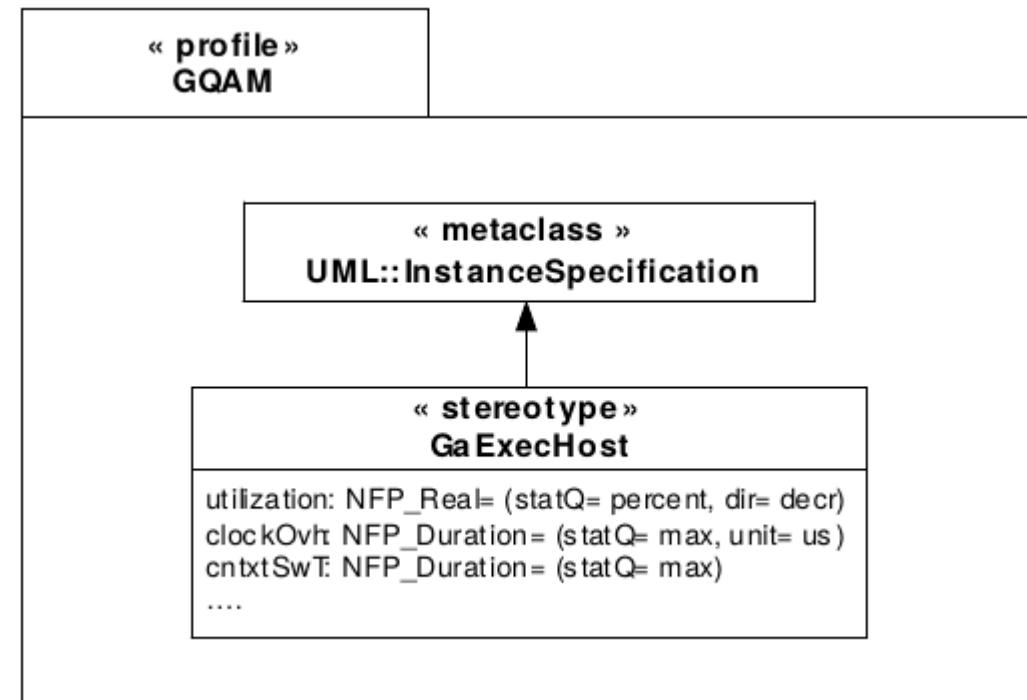
Exemplos de dimensões e extensões



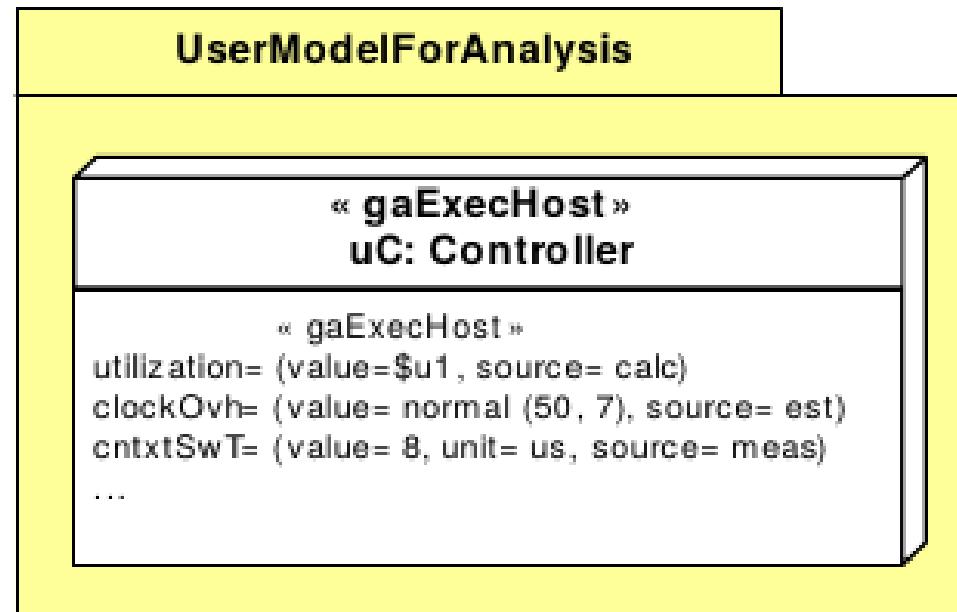
Exemplos dos tipos básicos



- <<gaExecHost>> é um recurso em execução com anotações para análise
- Tem algumas propriedades como utilização, tempo de troca de contexto (cntxtSwT) e clock overhead
- StatQ = statistical qualifier



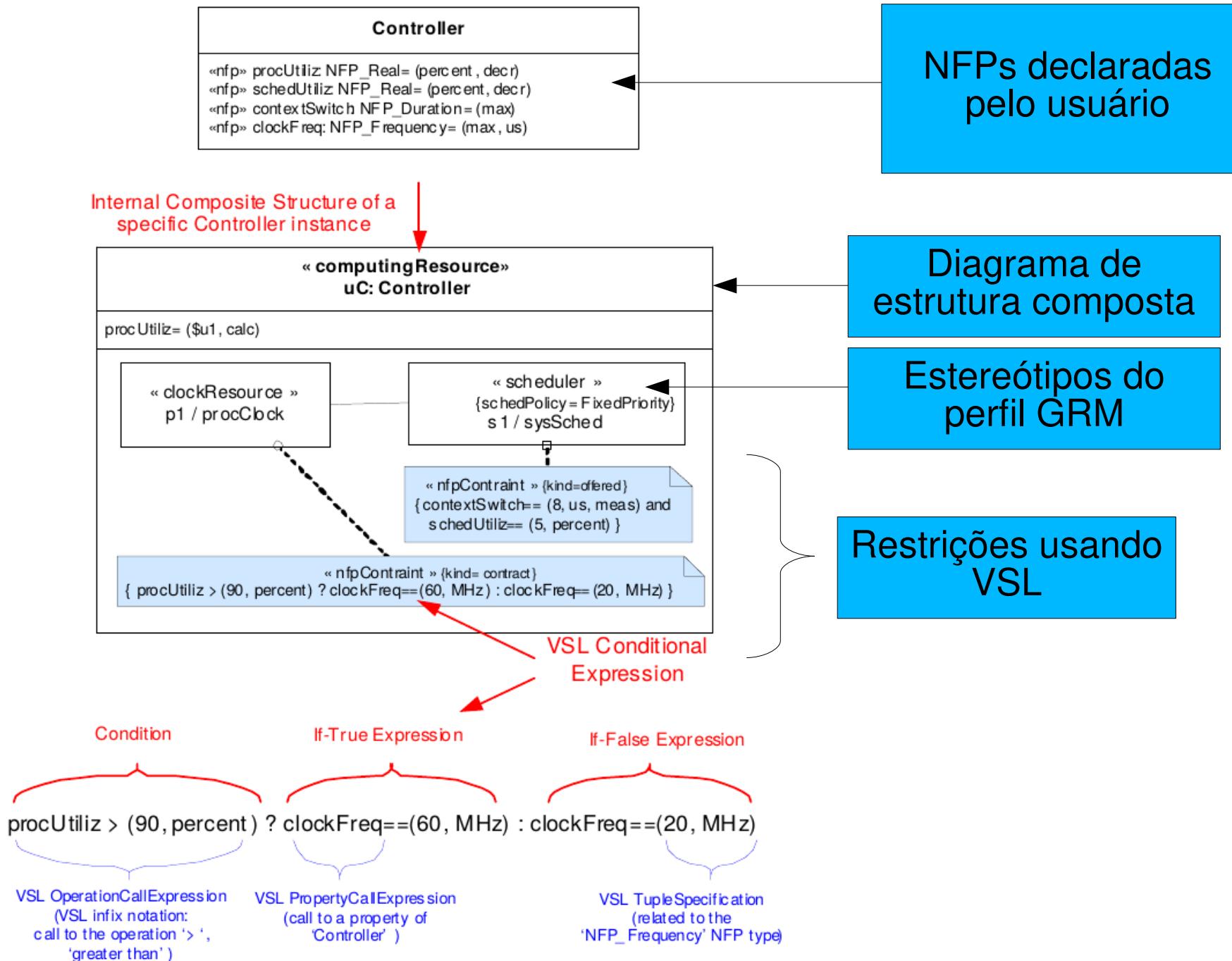
- ClockOvh é formado por um valor e source
 - Normal(50,7) é uma operação que chama uma distribuição de probabilidade normal
 - Utilização usa uma variável \$u1 (variáveis indicam para as ferramentas de análise que os valores devem ser computados e retornados para o modelo UML)



■ NFP Constraints

- Segundo mecanismo para anotar os modelos UML com restrições
 - Restrições definem as expressões de relação entre dois termos
- ## ■ As restrições são escritas usando a notação VSL

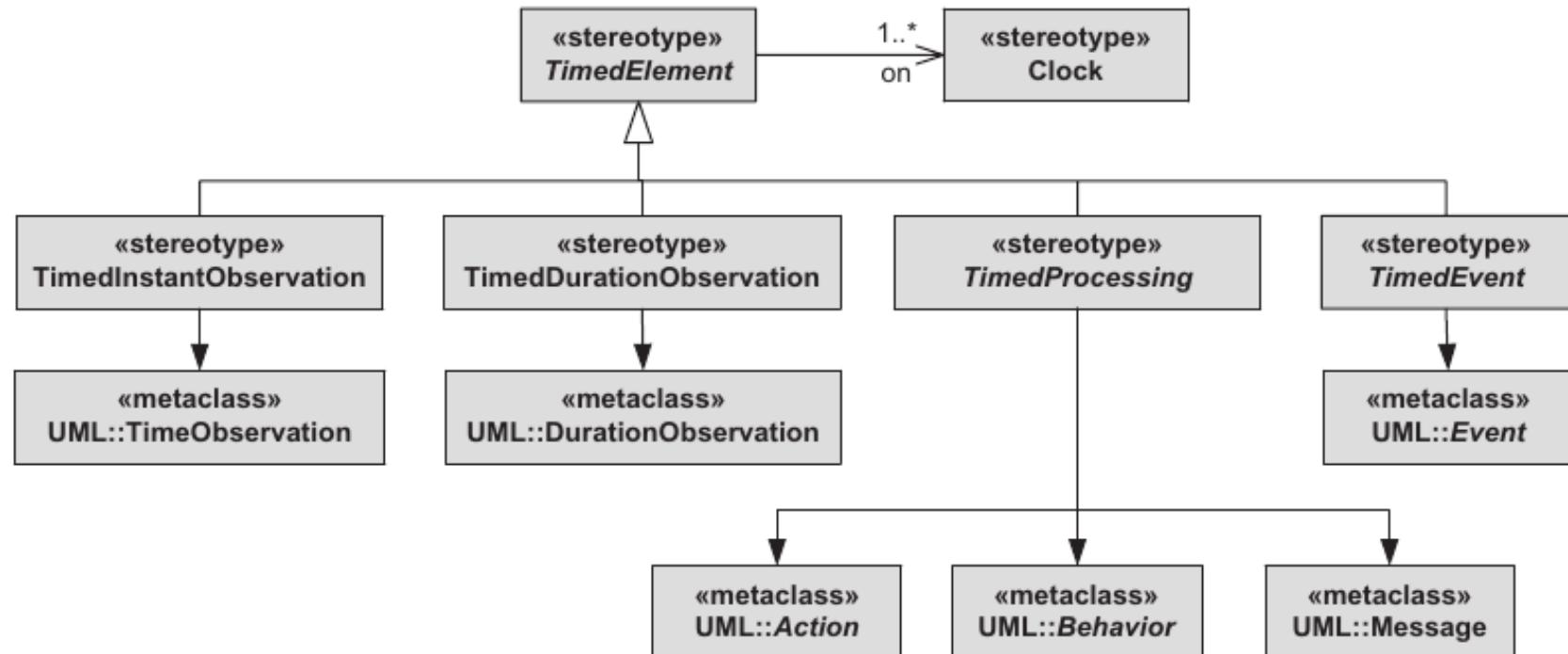
Exemplo de restrições



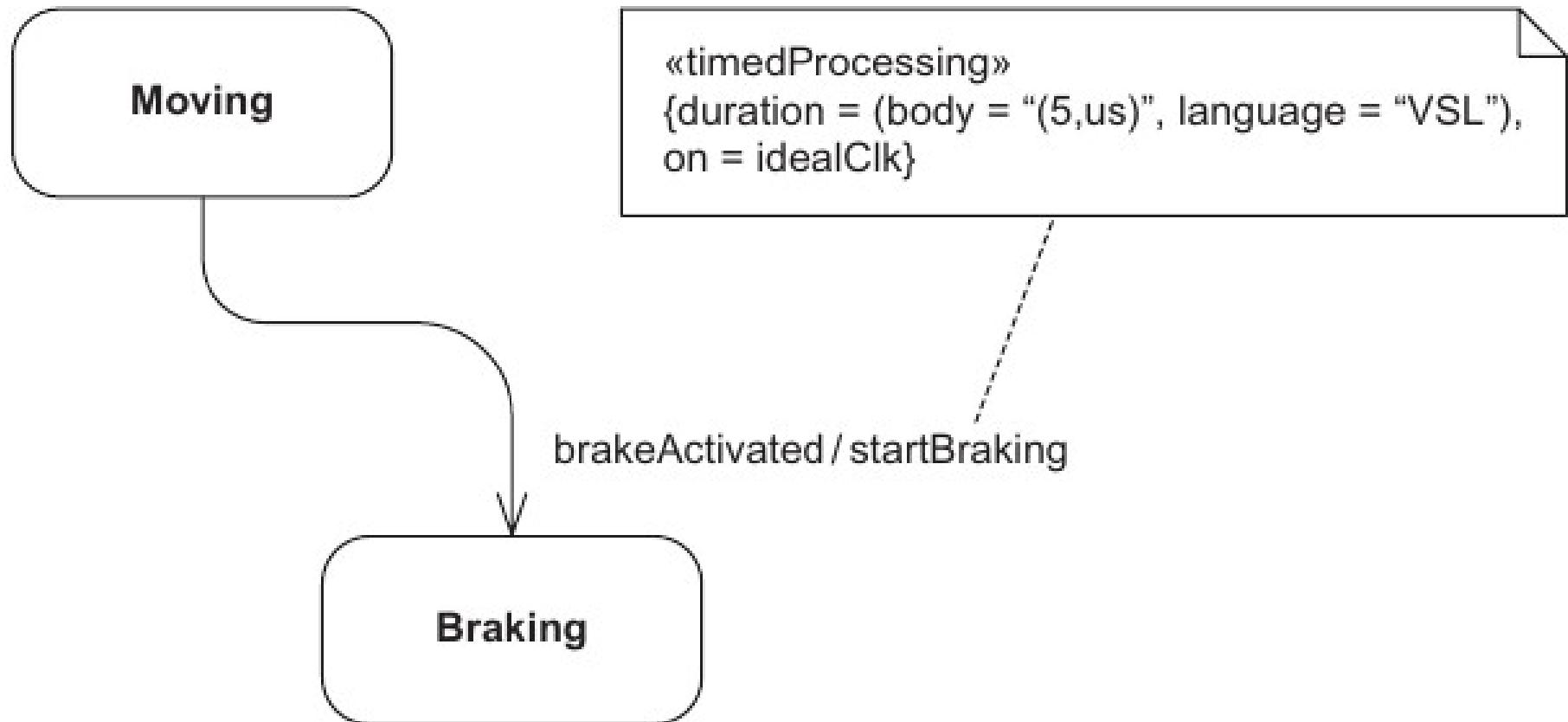
Modelando Tempo

- O tempo pode ser físico (contínuo ou discreto)
- Pode ser lógico, relacionando-se com as definições de clock do usuário
- É representado na UML-MARTE por uma coleção de Clocks
 - Cada clock especifica um conjunto ordenado de instantes (i.e, sequência de ocorrências de eventos)
- Perfil Time

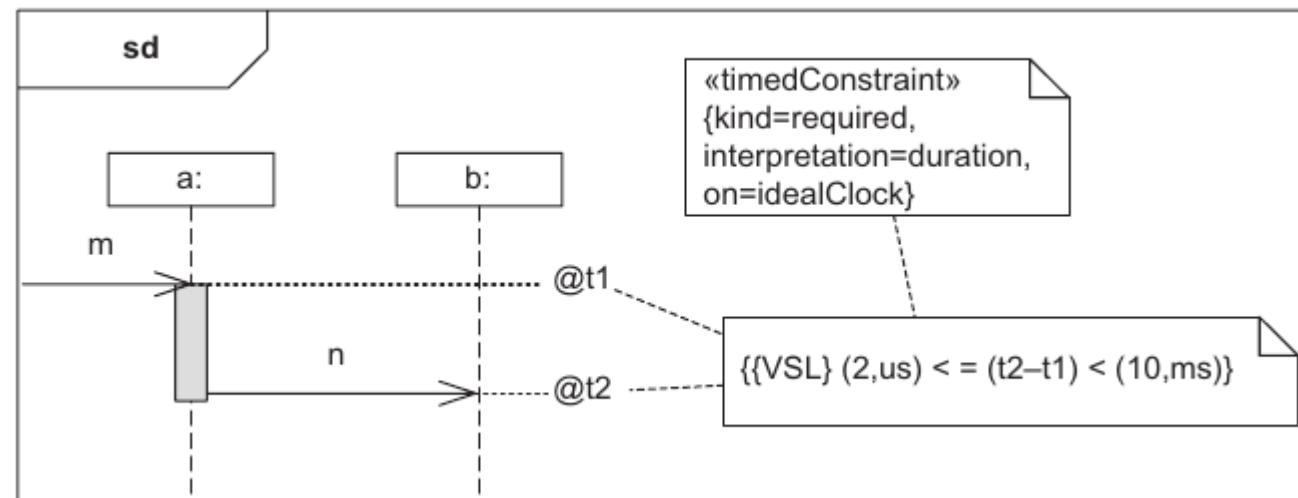
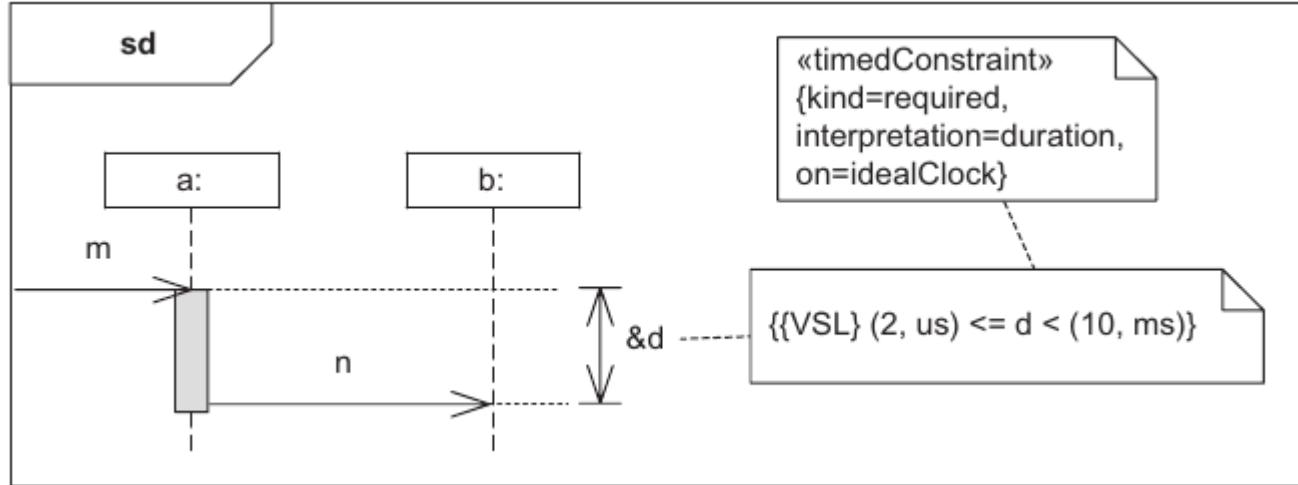
- **TimedInstantObservation**: instâncias que ocorrem em um ponto de tempo particular
- **TimedDurationObservation**: modelar intervalos de tempo entre eventos
- **TimedProcessing**: modelar tempo de computação
- **TimedEvent**: modelar tipos de eventos que ocorrem em um ponto de tempo, tal como timer periódico



Exemplo



Restrições de tempo



- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- Modelagem de requisitos não-funcionais
- **Expressão dos dispositivos de hardware**
- Exemplos
- Exercícios

- A melhor forma de expressar os dispositivos de hardware é usando os estereótipos do perfil HRM
- Basta incluir o estereótipo desejado no diagrama da UML para representar o hardware
- Veja na especificação do perfil HRM quais são os parâmetros atrelados ao estereótipo usado

Exemplos: processador

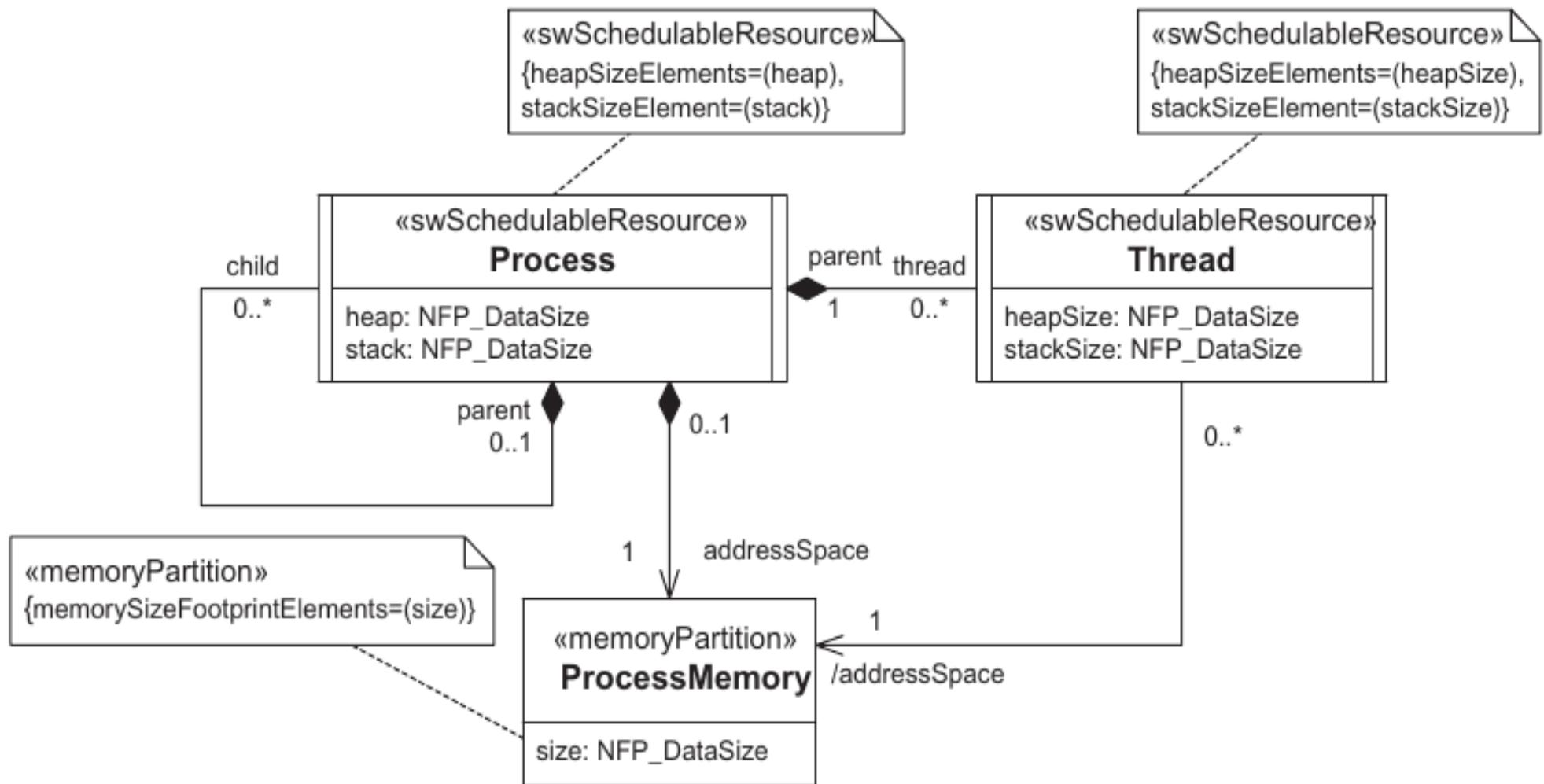
```
«hwComputingResource»  
Multiprocessor  
(description = "ZX432",  
 speedFactor = 3.5,  
 resMult = 4}
```

```
«expressionContext»  
ARM_Cortex_A9_explored_freqs  
{$ZynqARMCortex9_FREQ=(667,766,866},MHz)}
```

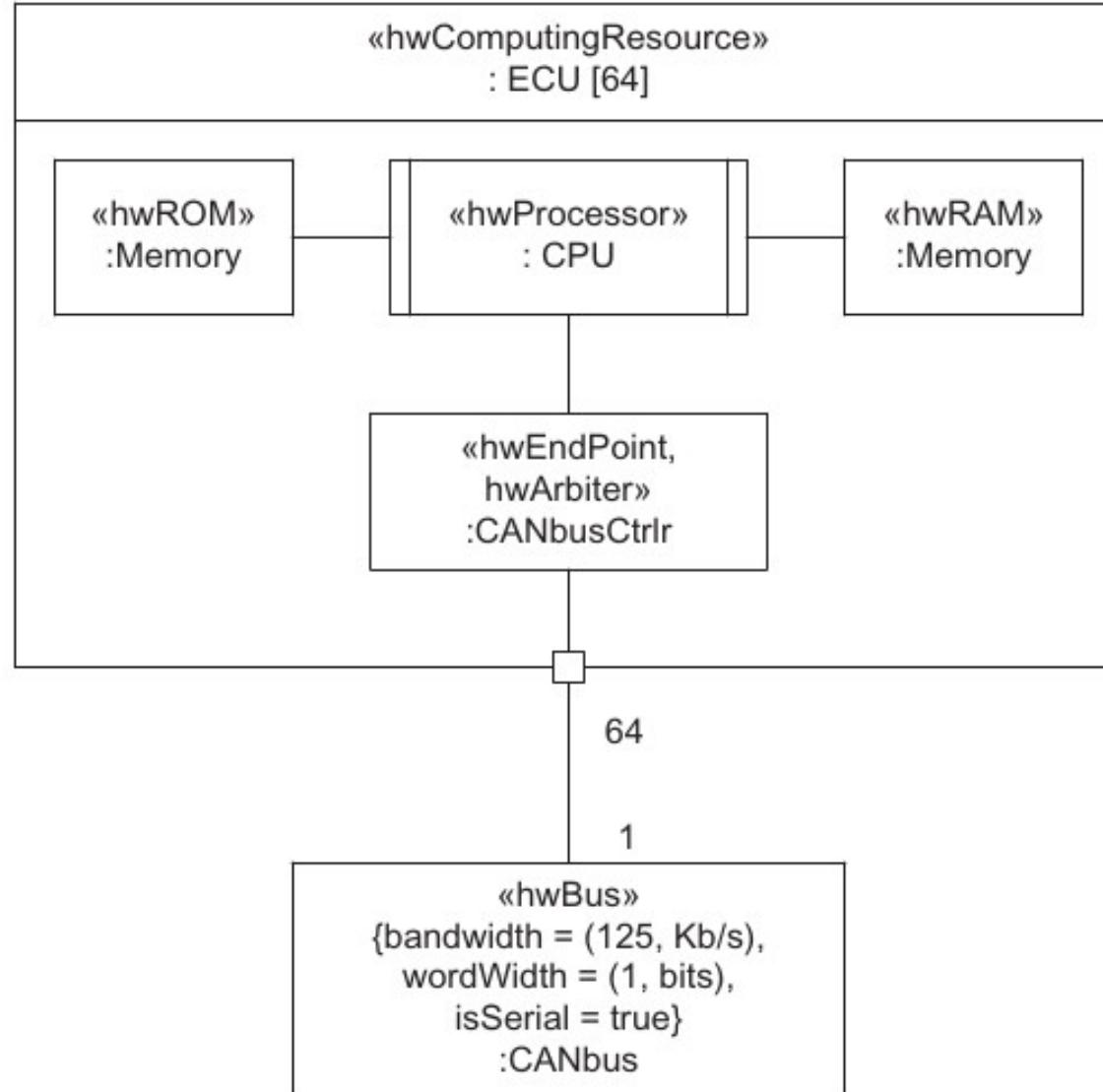
```
«hwProcessor»  
«Component»  
ARM_Cortex_A9
```

```
«HwProcessor»  
frequency=$ZynqARMCortex9_FREQ=(866,MHz)
```

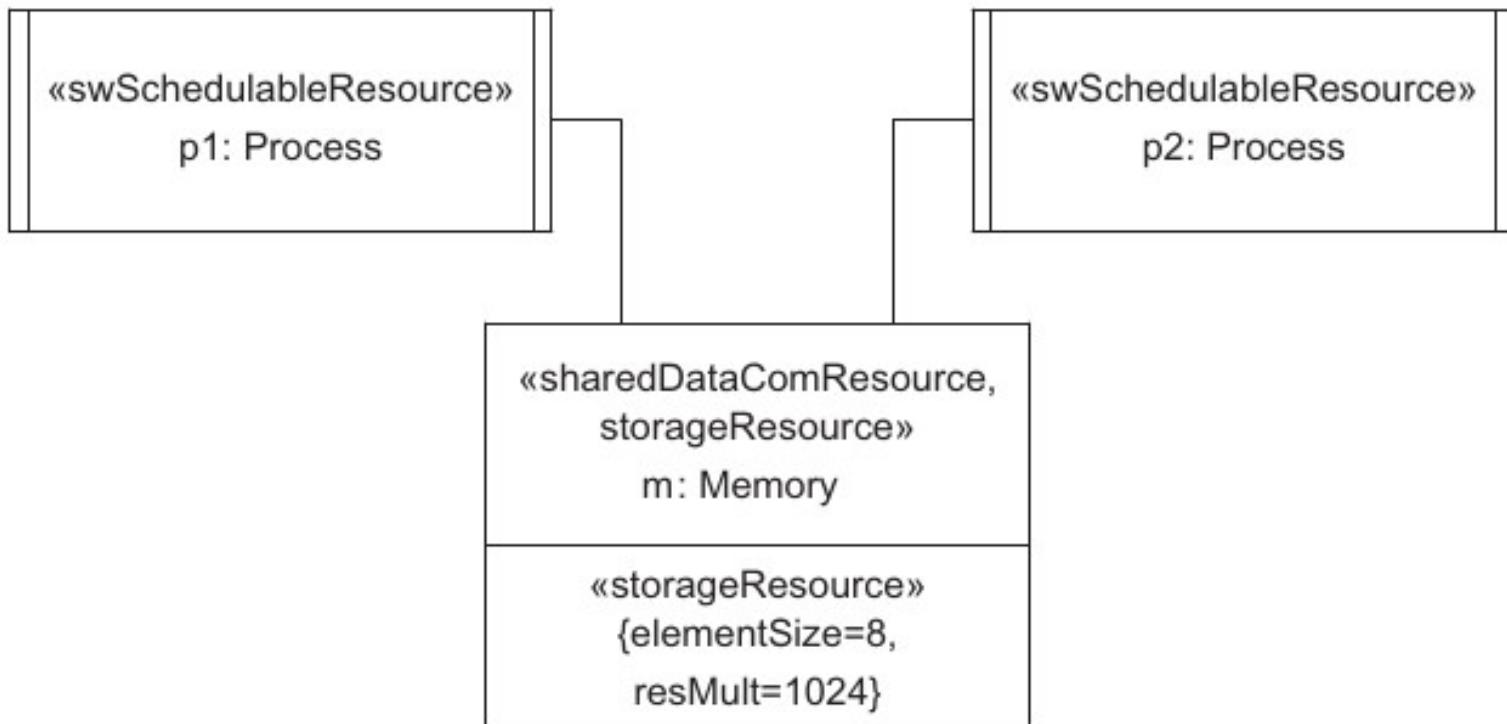
Exemplos: Processo Posix com Threads e Espaço Compartilhado



Exemplos: memória e barramento



Exemplos: comunicação via memória compartilhada

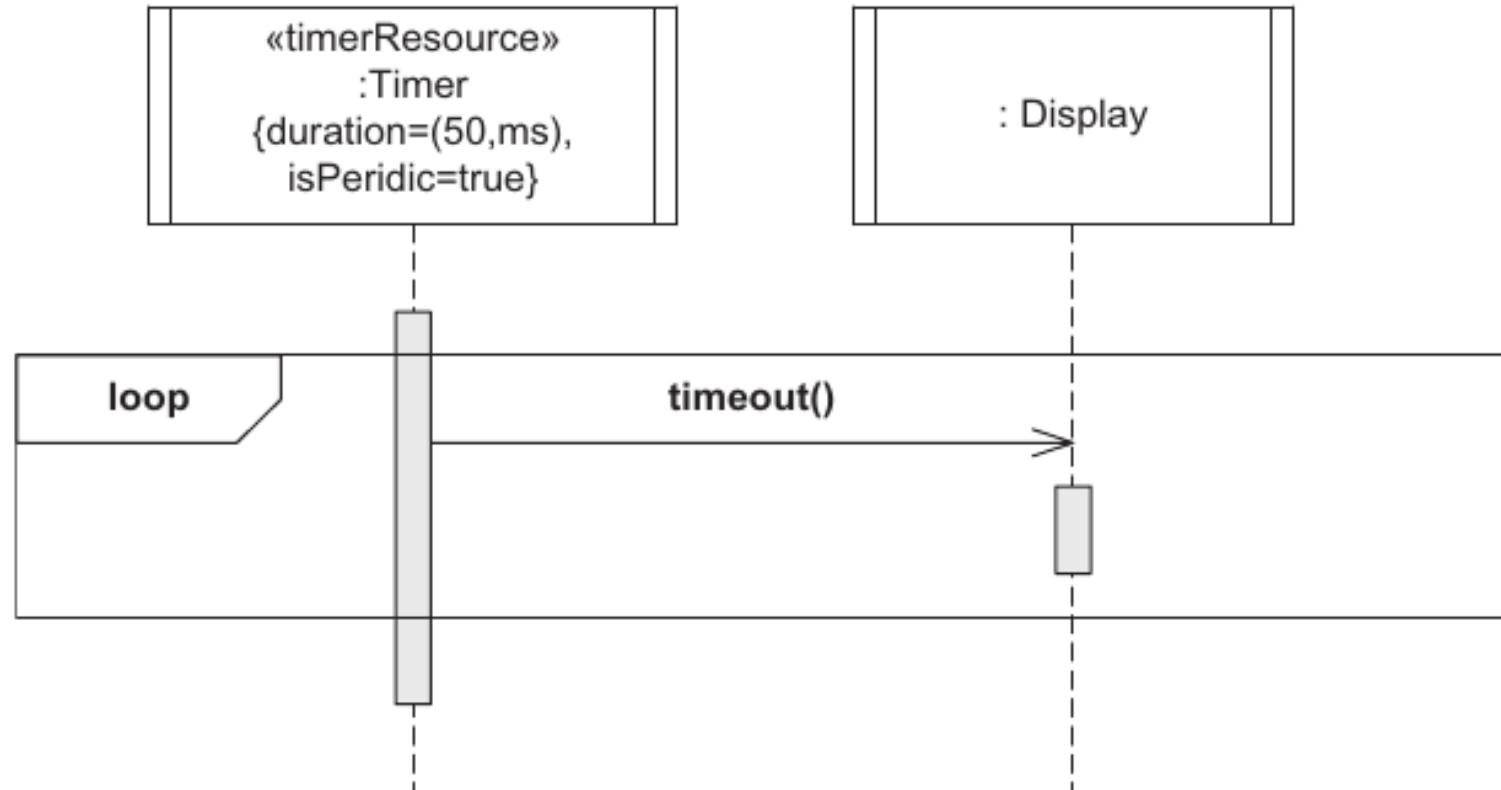


Semáforo em Hardware

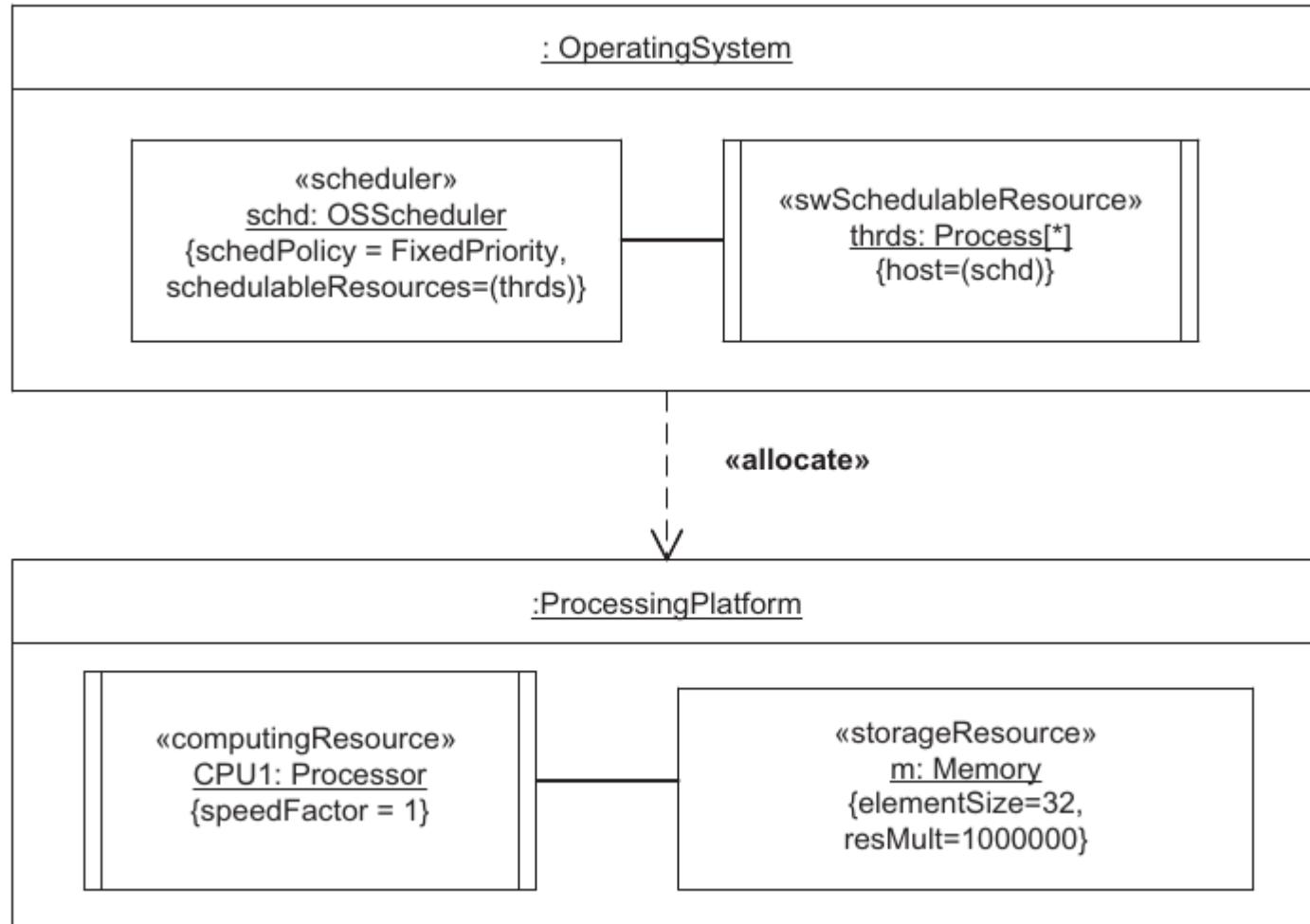
```
«hwDevice,  
mutualExclusionResource»  
MySemaphoreClass
```

```
«mutualExclusionResource»  
{protectKind=NoPreemption}
```

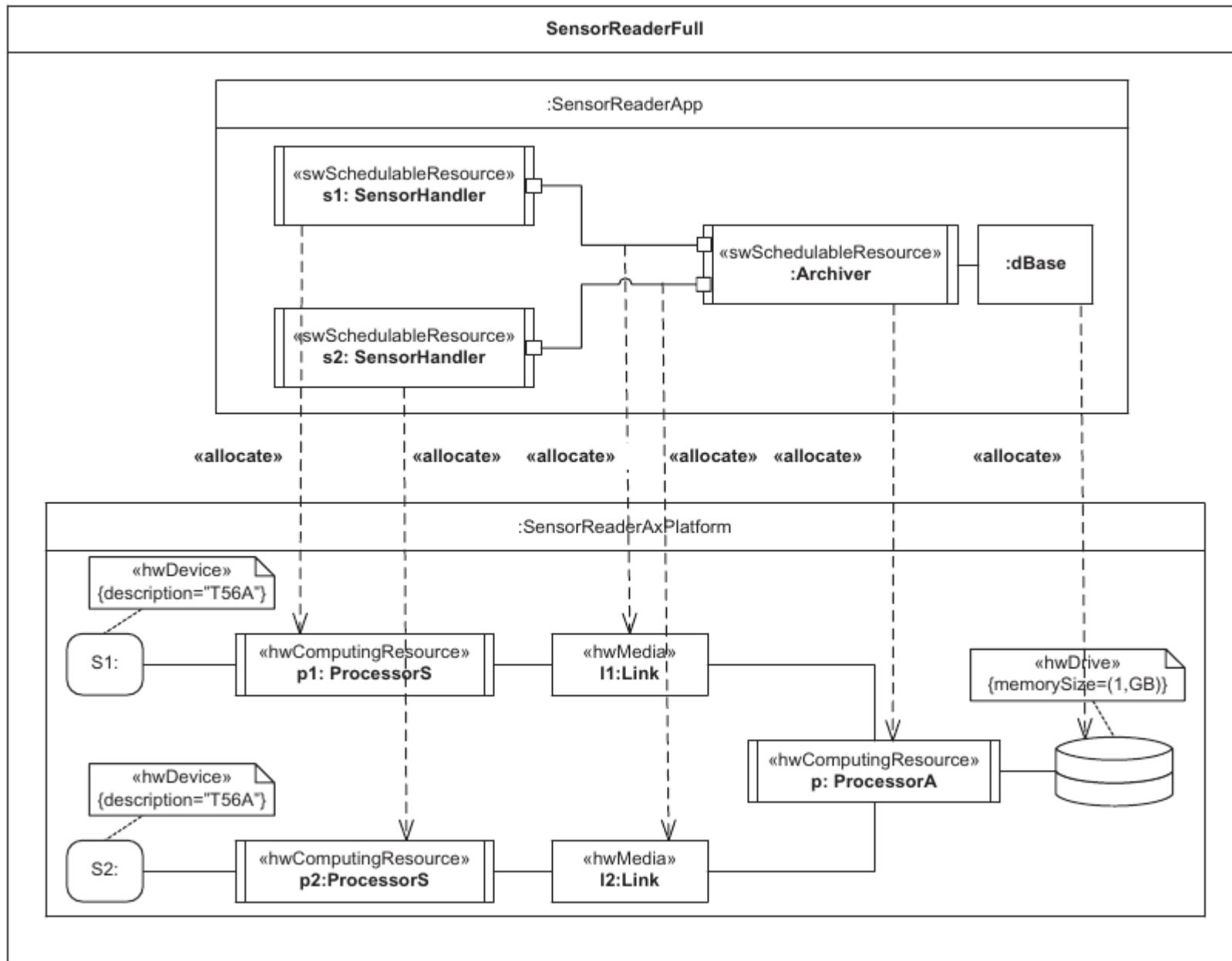
Exemplos: evento periódico



Exemplos: combinando software com plataforma



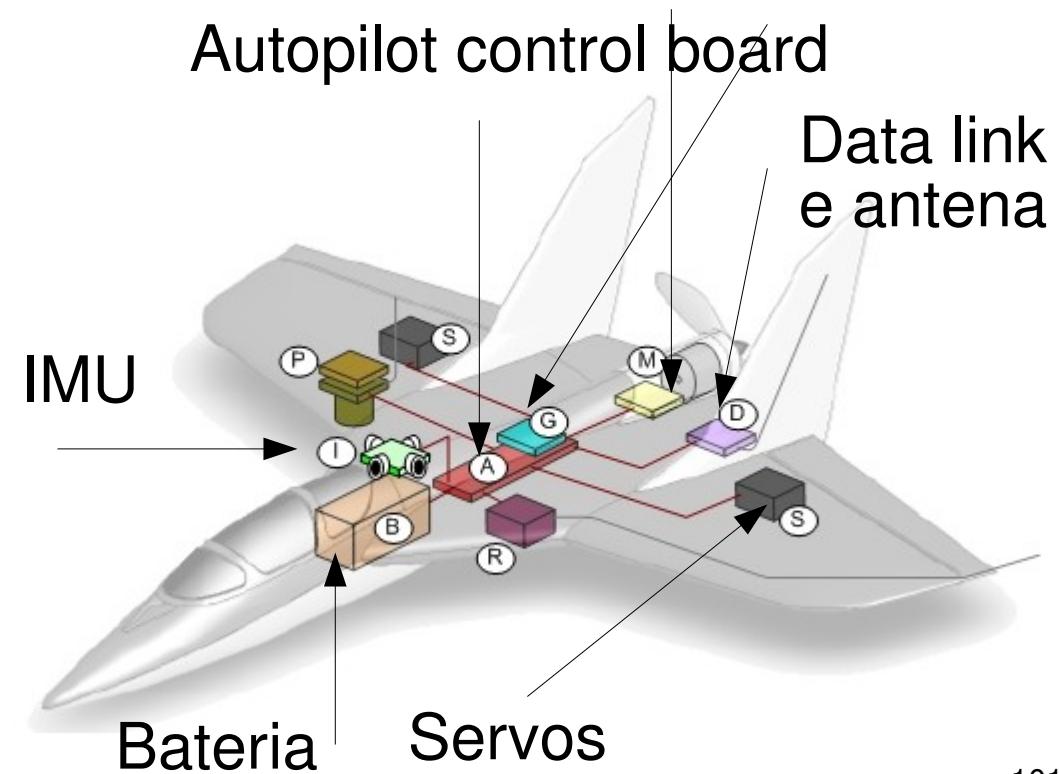
Exemplos: aplicação de leitura de sensor com a plataforma



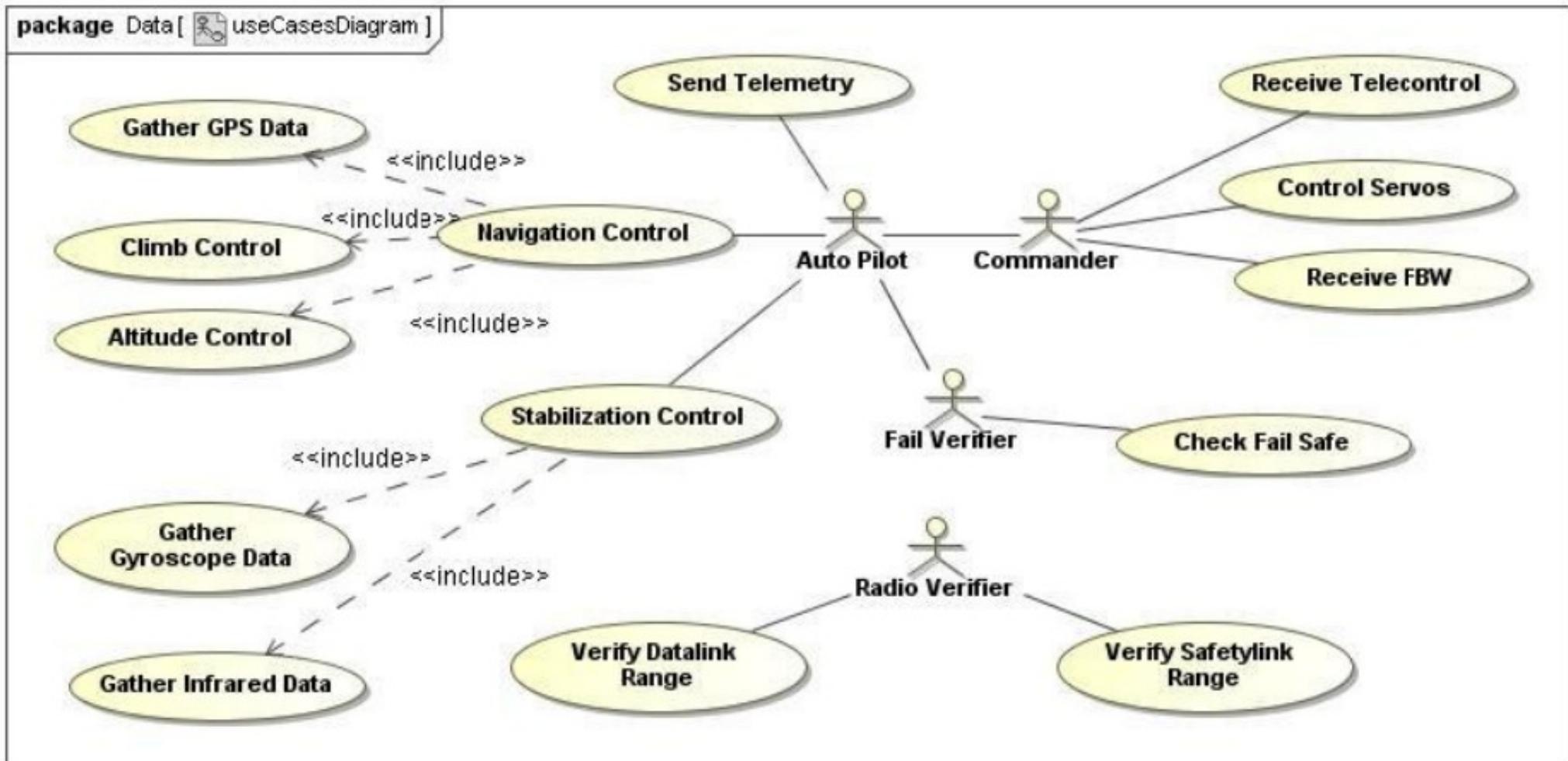
- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- Modelagem de requisitos não-funcionais
- Expressão dos dispositivos de hardware
- **Exemplos**
- Exercícios

Exemplo: UAV paparazzi

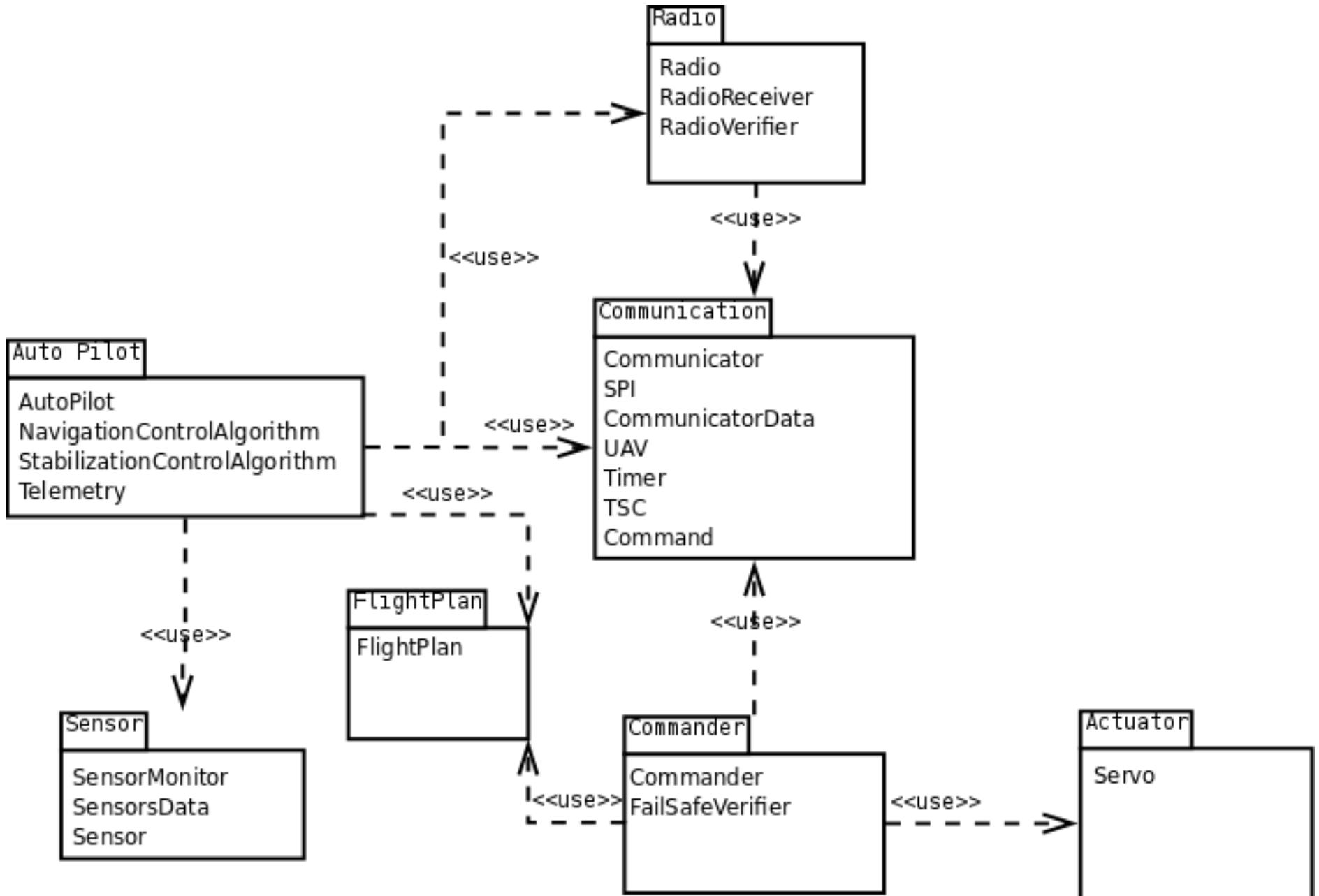
- Disponível em
<http://wiki.paparazziuav.org/wiki/Overview>
- Sistema completo de hardware e software de código aberto para Unmanned Aircraft Systems (UAS)



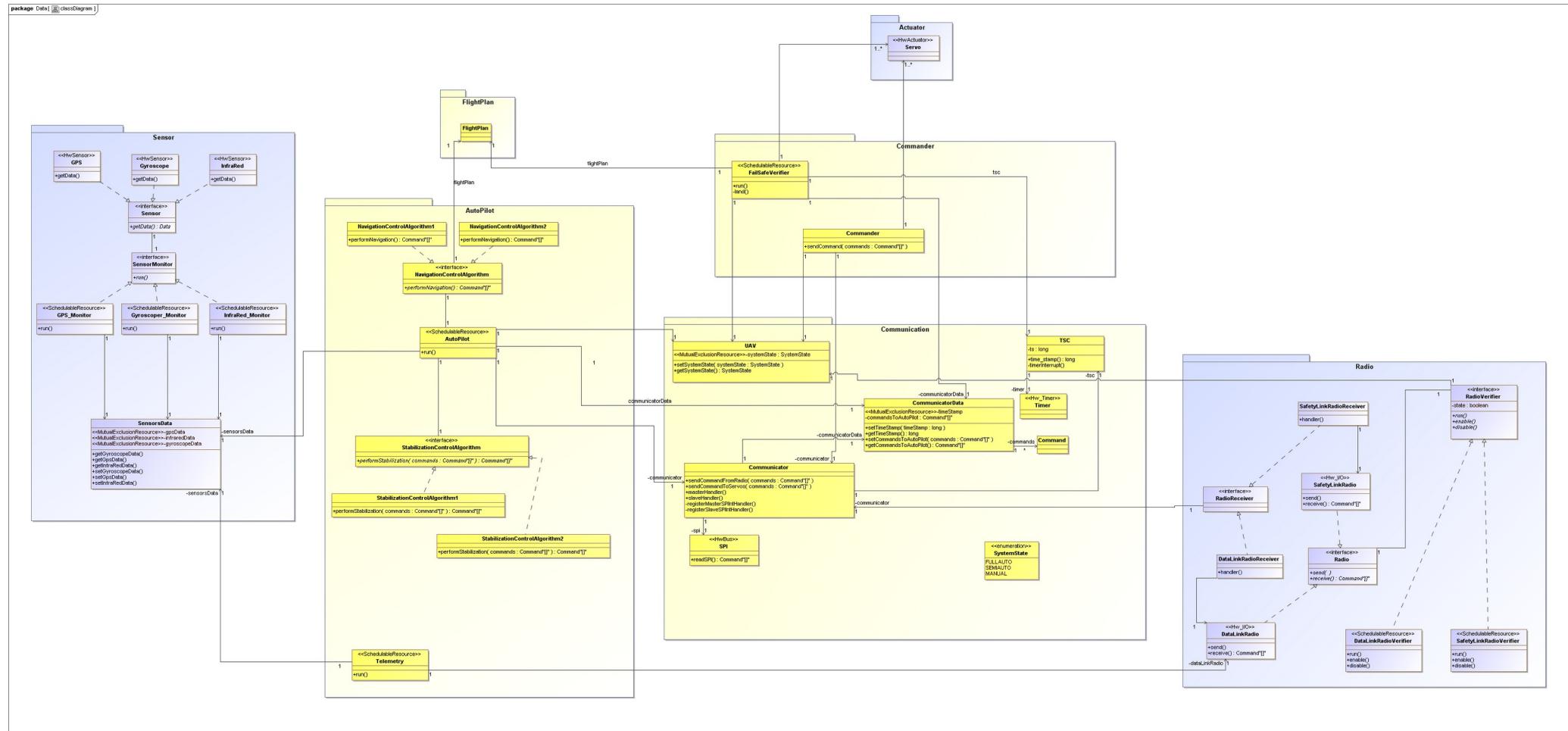
Exemplo: UAV paparazzi



Exemplo: UAV paparazzi



Exemplo: UAV paparazzi

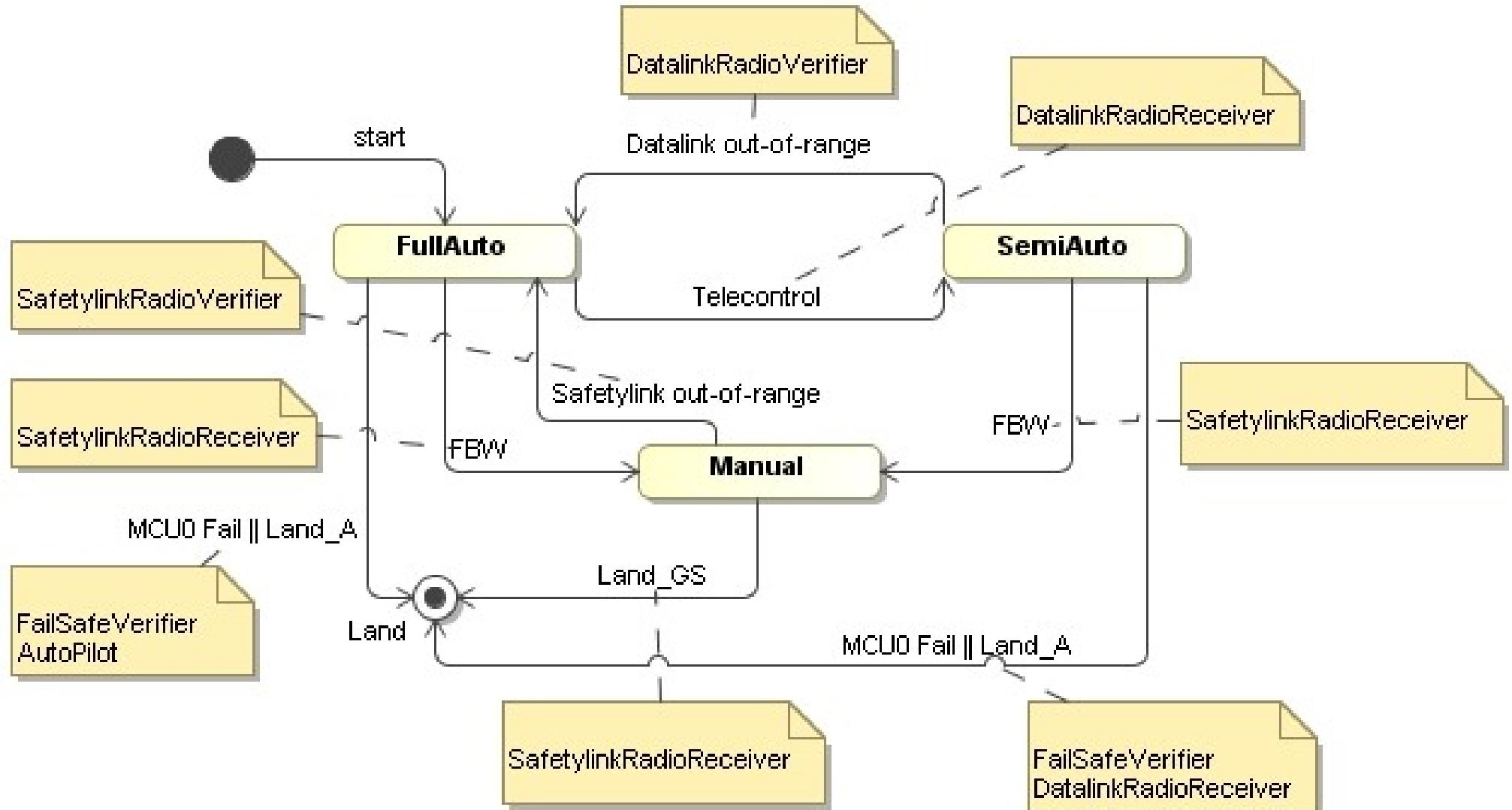


Exemplo: UAV paparazzi

Tabela 4.1: Mapeamento das classes do sistema com os microcontroladores.

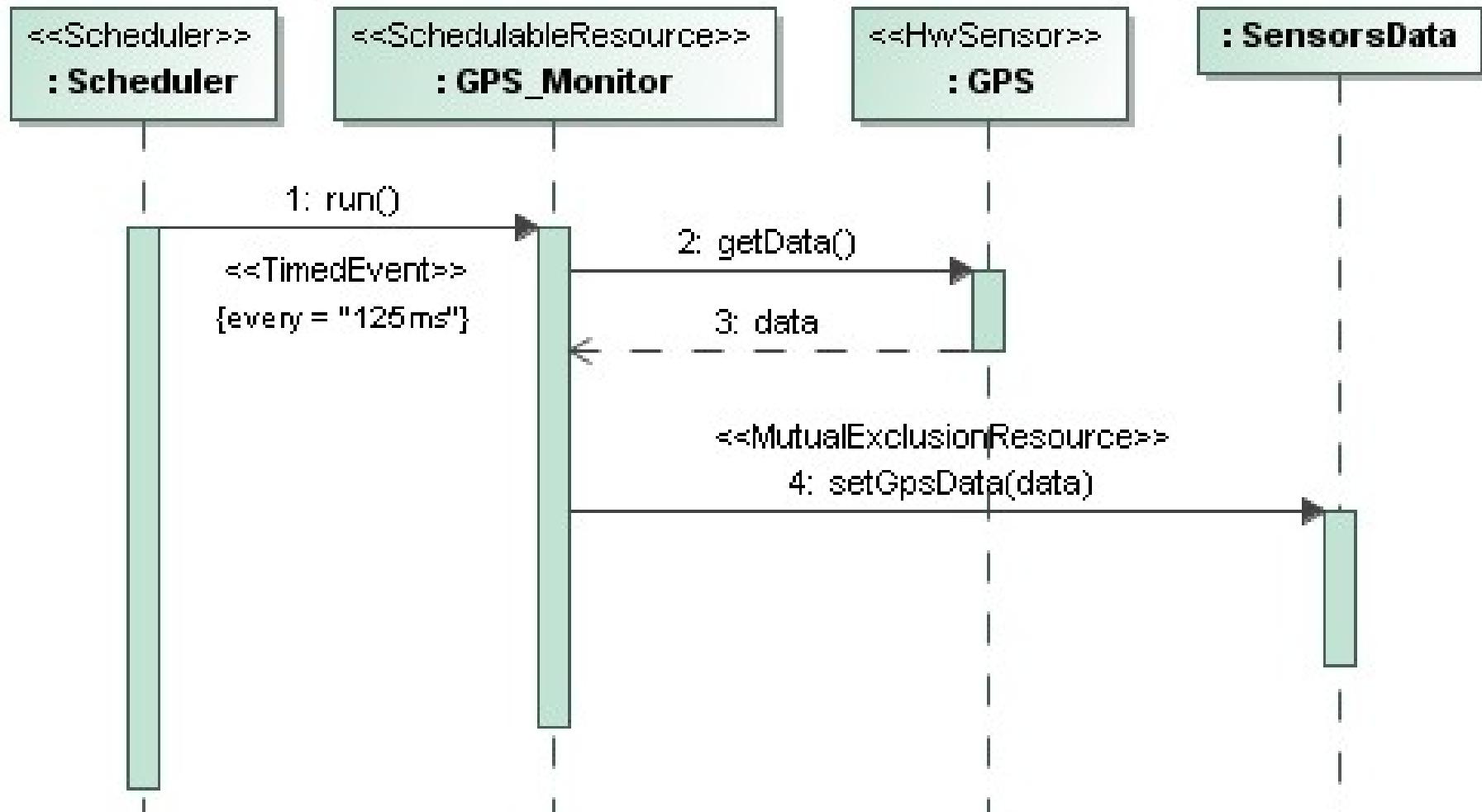
MCU0	MCU1
AutoPilot	Commander
NavigationControlAlgorithm	Servo
StabilizationControlAlgorithm	FailSafeVerifier
FlightPlan	FlightPlan
DataLinkRadio	SafetyLinkRadio
DataLinkRadioReceiver	SafetyLinkRadioReceiver
DataLinkRadioVerifier	SafetyLinkRadioVerifier
SensorMonitor	TSC
SensorsData	Timer
<i>Communicator</i>	<i>Communicator</i>
<i>SPI</i>	<i>SPI</i>
<i>Command</i>	<i>Command</i>
<i>UAV</i>	<i>UAV</i>
<i>CommunicatorData</i>	<i>CommunicatorData</i>
Sensor	-
Telemetry	-

Exemplo: UAV paparazzi



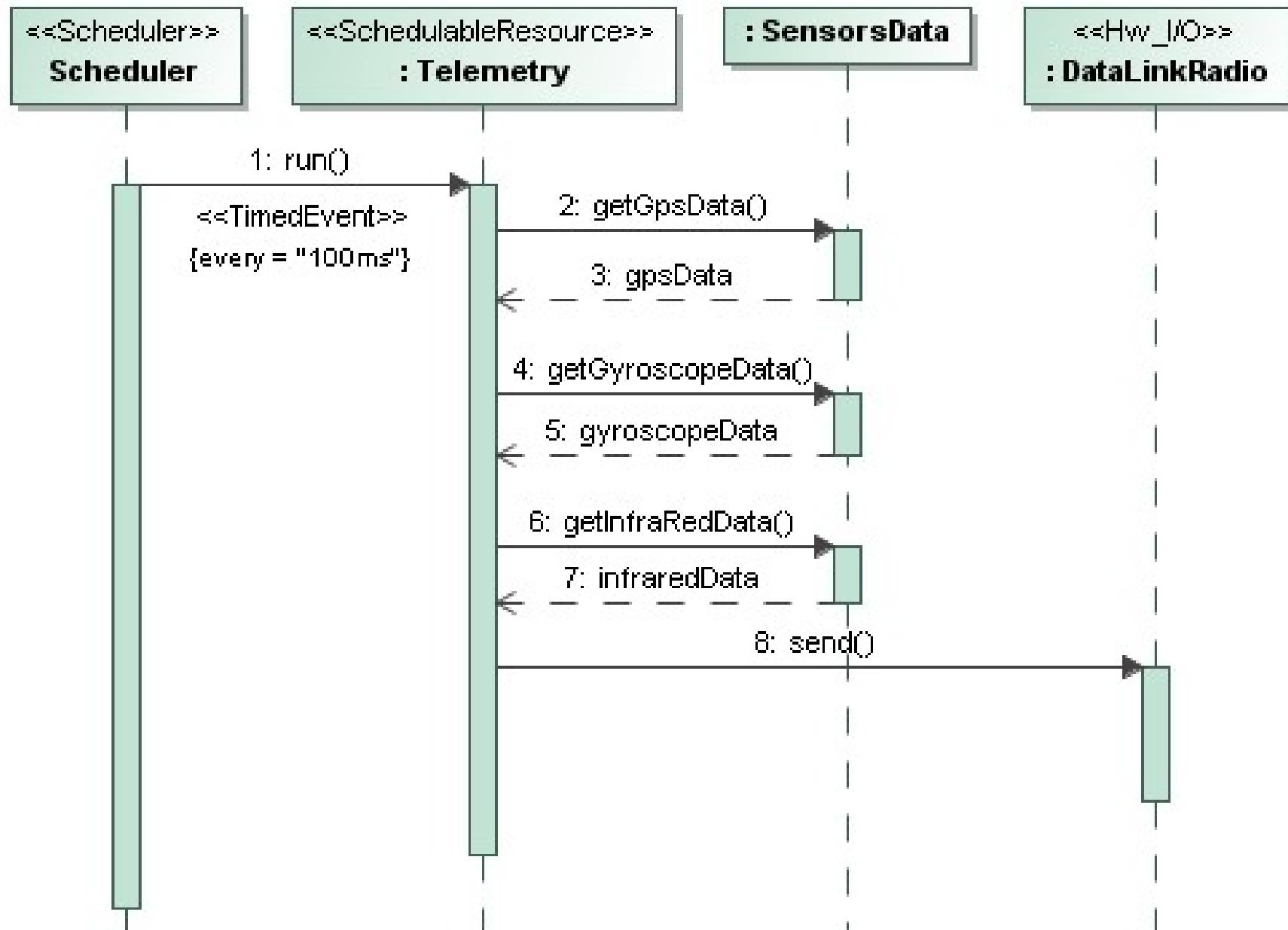
Exemplo: UAV paparazzi

Monitoramento dos sensores



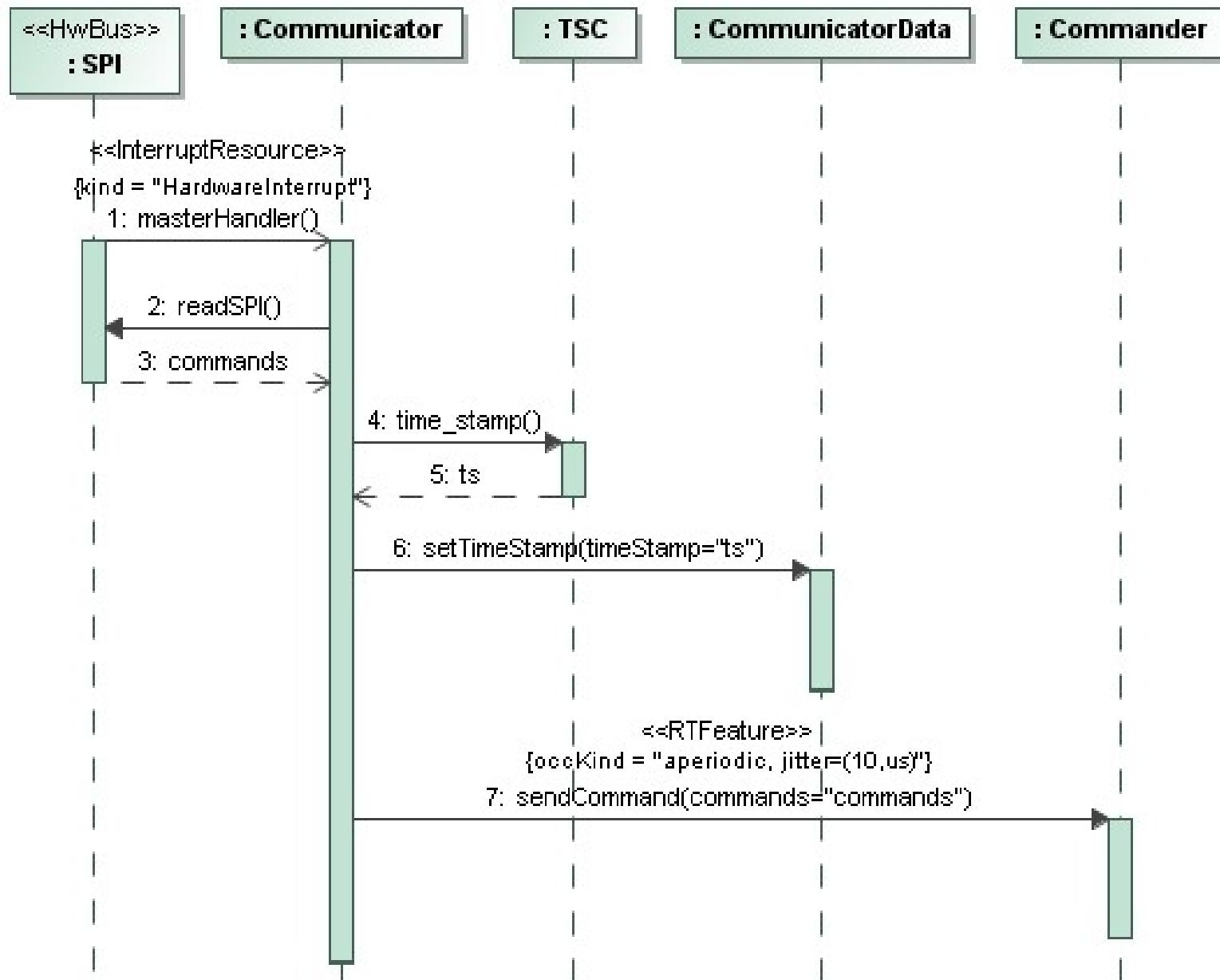
Exemplo: UAV paparazzi

Telemetria



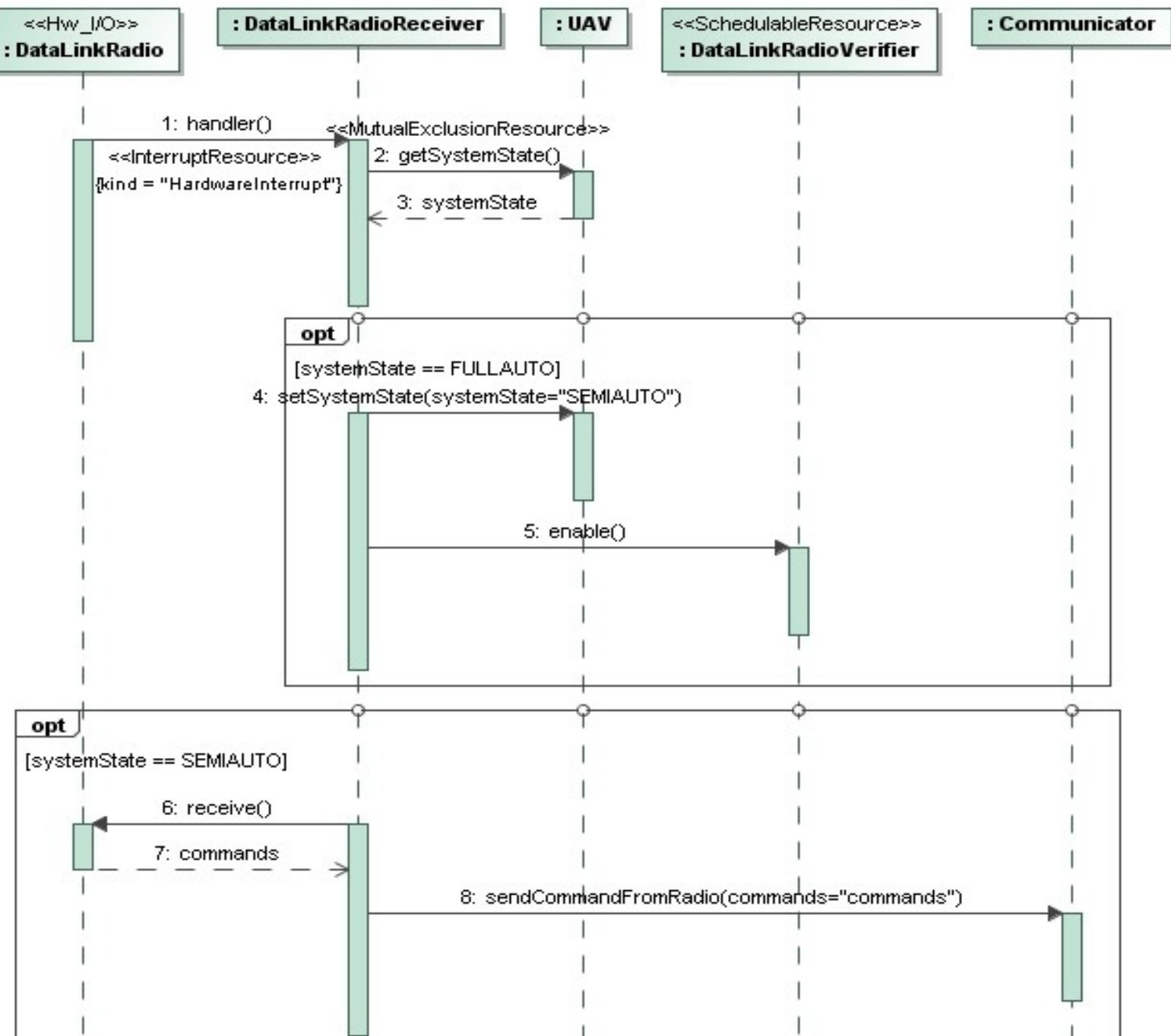
Exemplo: UAV paparazzi

Comunicação entre os MCUs



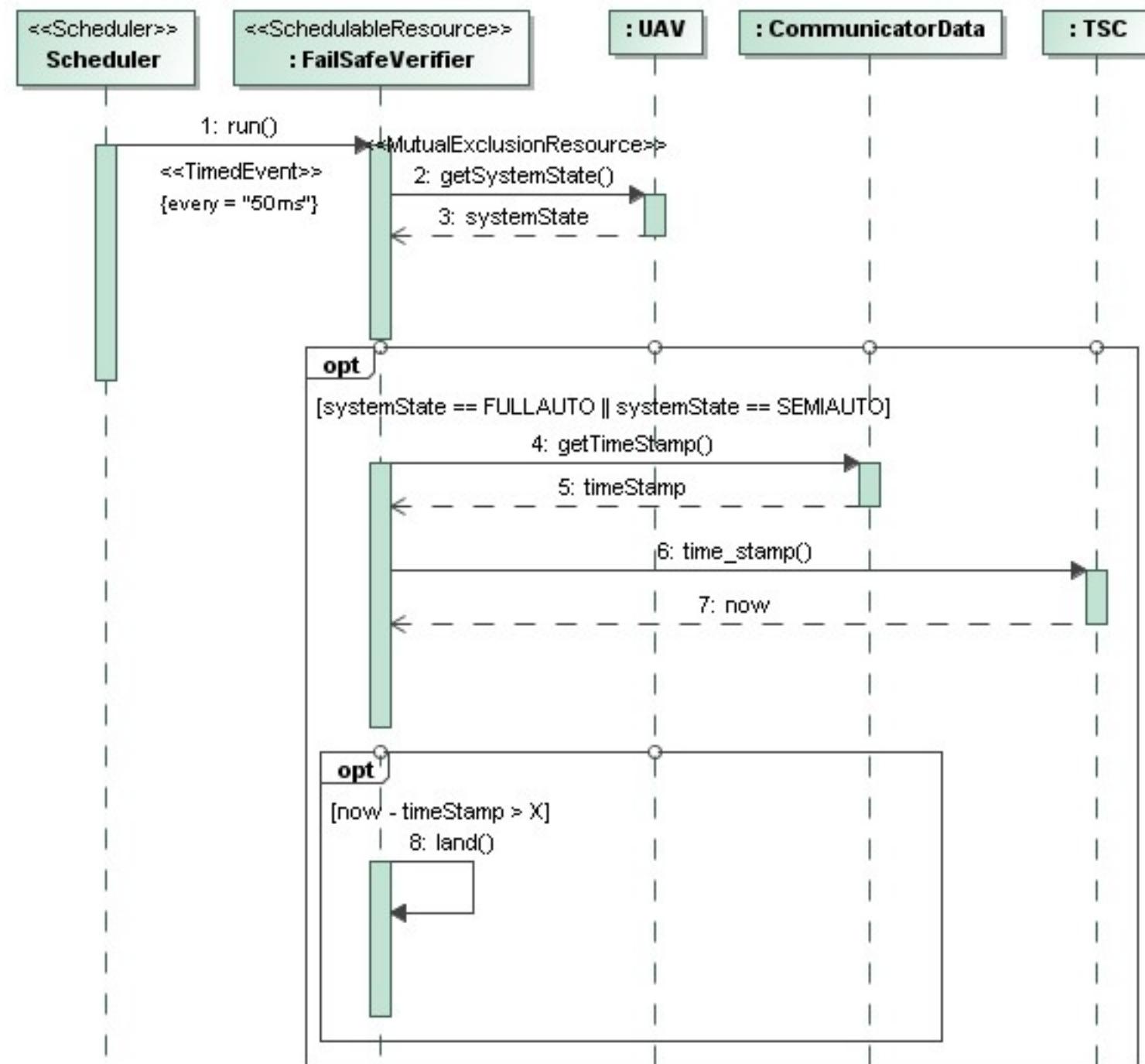
Exemplo: UAV paparazzi

Recepção
dos dados



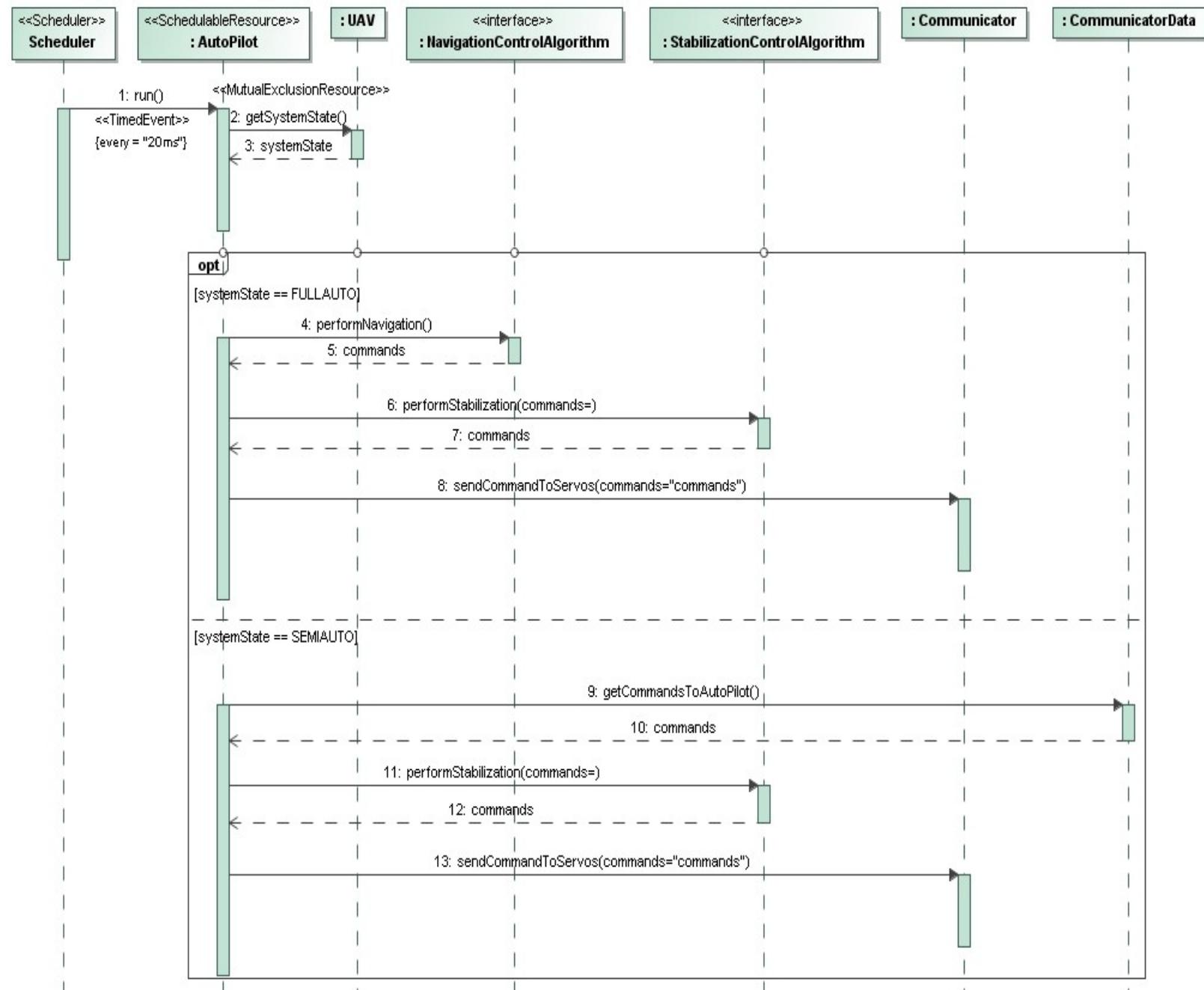
Exemplo: UAV paparazzi

Verificação de falhas



Exemplo: UAV paparazzi

Piloto
automático



- Introdução a Sistemas de Tempo Real
- Introdução a UML-MARTE
- Resumo dos principais pacotes da UML-MARTE
- Modelagem de requisitos não-funcionais
- Expressão dos dispositivos de hardware
- Exemplos
- Exercícios

Exercício: sistema de estacionamento autônomo

- A proposta é construir um sistema que realize o estacionamento autônomo de um carro convencional, considerando que a vaga de estacionamento esteja localizada paralelamente ao carro.
- O sistema deve identificar a vaga disponível e deve ser capaz de dirigir o carro até a vaga encontrada
 - Requisitos detalhados em
<https://repositorio.ufsc.br/xmlui/handle/123456789/93134>

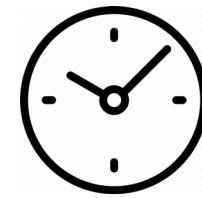
Exercício: sistema de estacionamento autônomo

■ Requisitos

F1 Início/Parada do sistema	
Descrição: O sistema deve ser explicitamente ativado pelo motorista para entrar em modo de operação.	
Nome	Restrição
NFR1.1 - Velocidade Máxima	Para iniciar o sistema a velocidade deve ser mantida $\leq 20\text{Km/h}$.
F2 Procura por Vaga (<i>real-time operation</i>)	
Descrição: Quando ativado, o sistema deve iniciar a procura por uma vaga adequada enquanto o veículo move-se para frente.	
Nome	Restrição
NFR2.1 - Dimensões do Veículo	O sistema deve ser adaptável a veículos de dimensões diferentes.
F3 Estacionamento (<i>real-time operation</i>)	
Descrição: O motorista pode ativar o início de manobra somente com a vaga encontrada. O sistema controla a velocidade e a direção do veículo.	
Nome	Restrição
NFR3.1 - Distância Máxima	O veículo não deve estar a mais do que 20m da vaga.
NFR3.2 - Parada com intervenção do motorista	O sistema deve ser interrompido quando o motorista tocar a direção.

Exercício: sistema de estacionamento autônomo

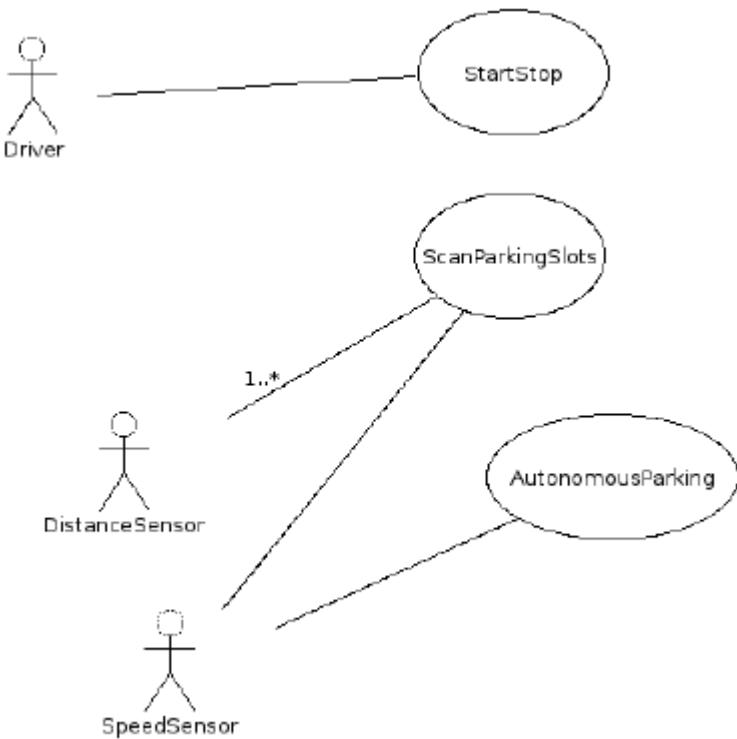
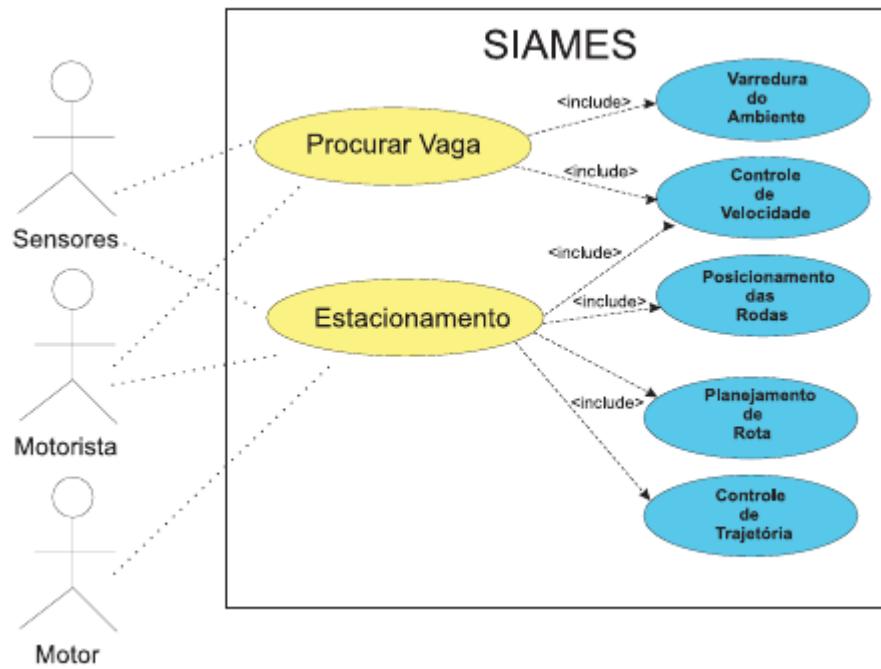
- Faça o diagrama de casos de uso do sistema



10 minutos

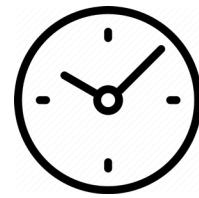
Exercício: sistema de estacionamento autônomo

- Faça o diagrama de casos de uso do sistema



Exercício: sistema de estacionamento autônomo

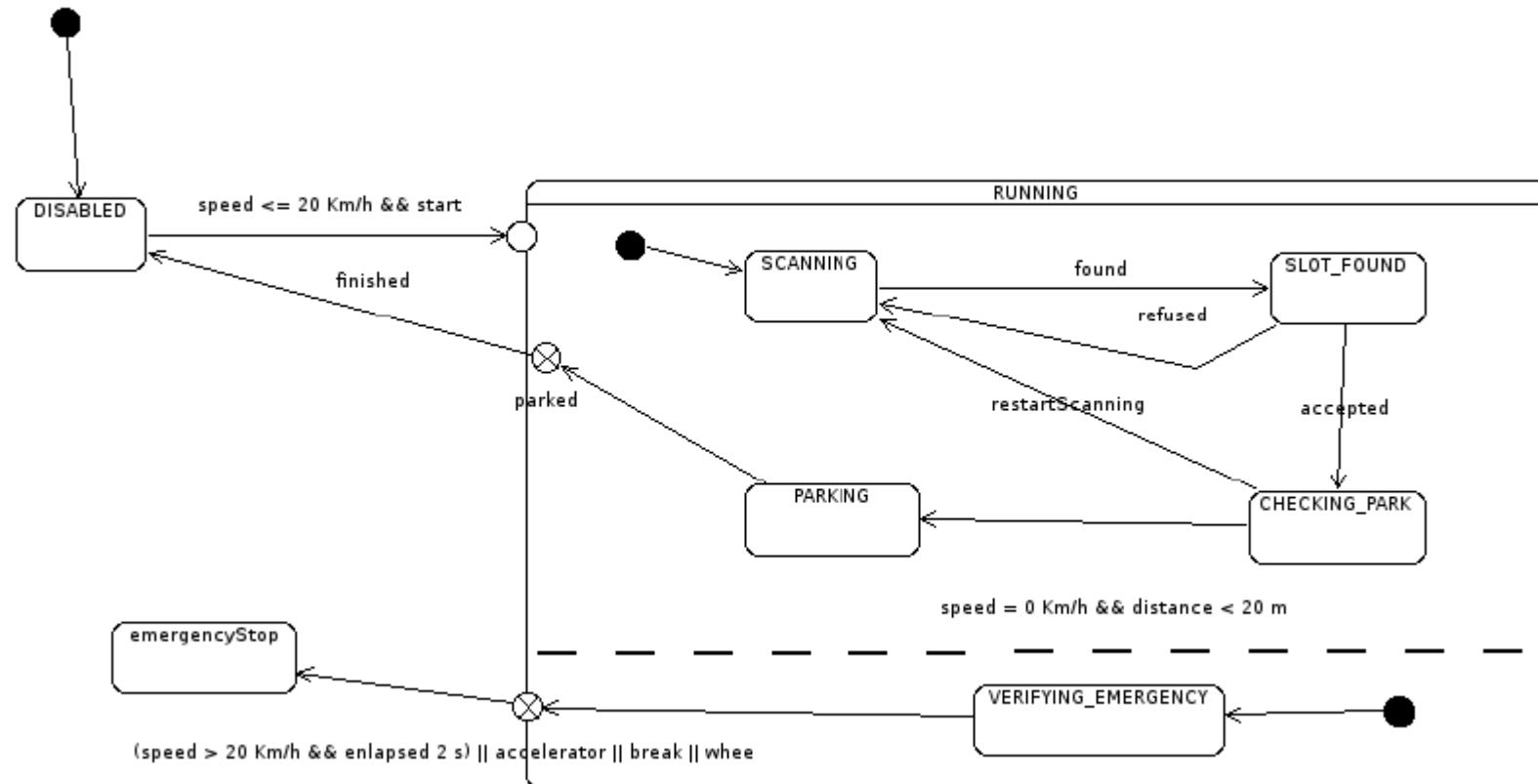
- Faça o diagrama de máquinas de estados do sistema



15 minutos

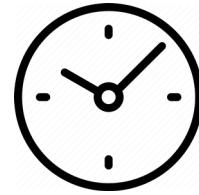
Exercício: sistema de estacionamento autônomo

- Faça o diagrama de máquinas de estados do sistema



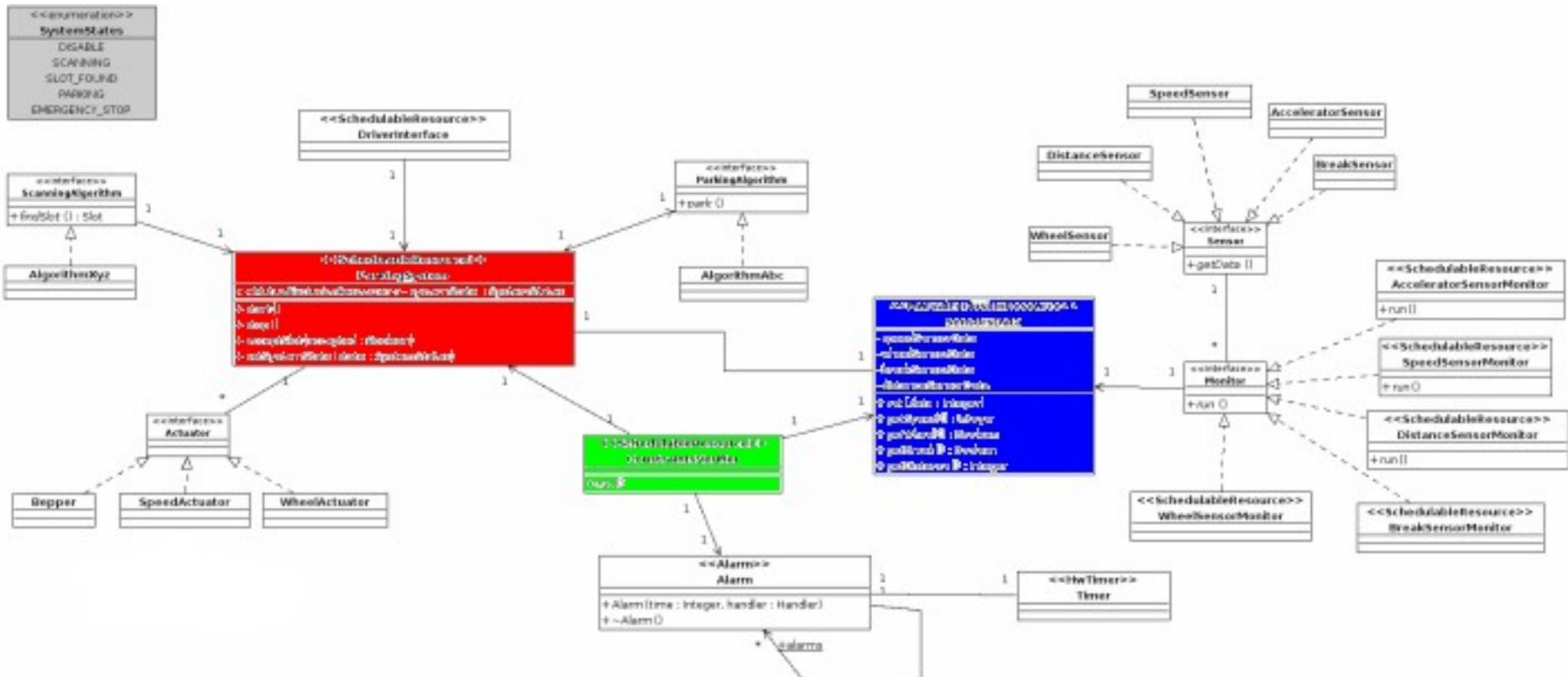
Exercício: sistema de estacionamento autônomo

- Faça o diagrama de classes do sistema utilizando também os estereótipos do perfil UML-MARTE

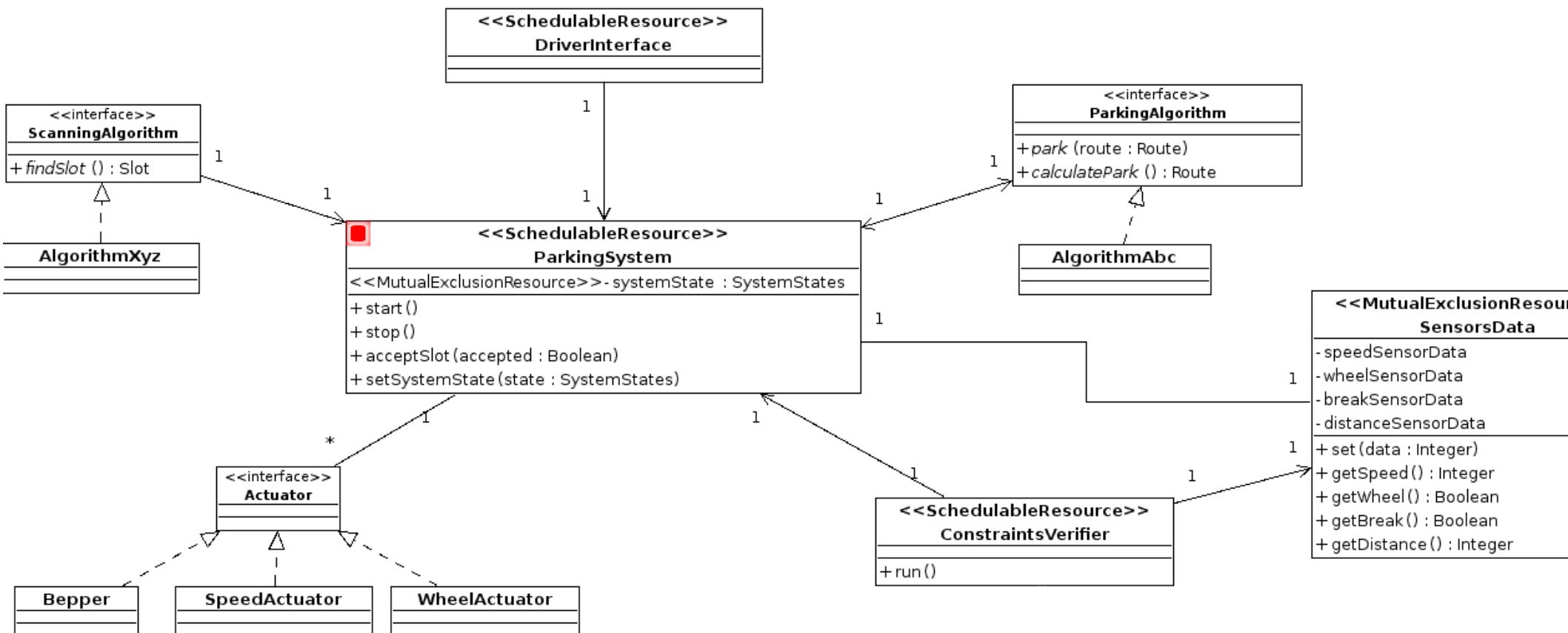


20 minutos

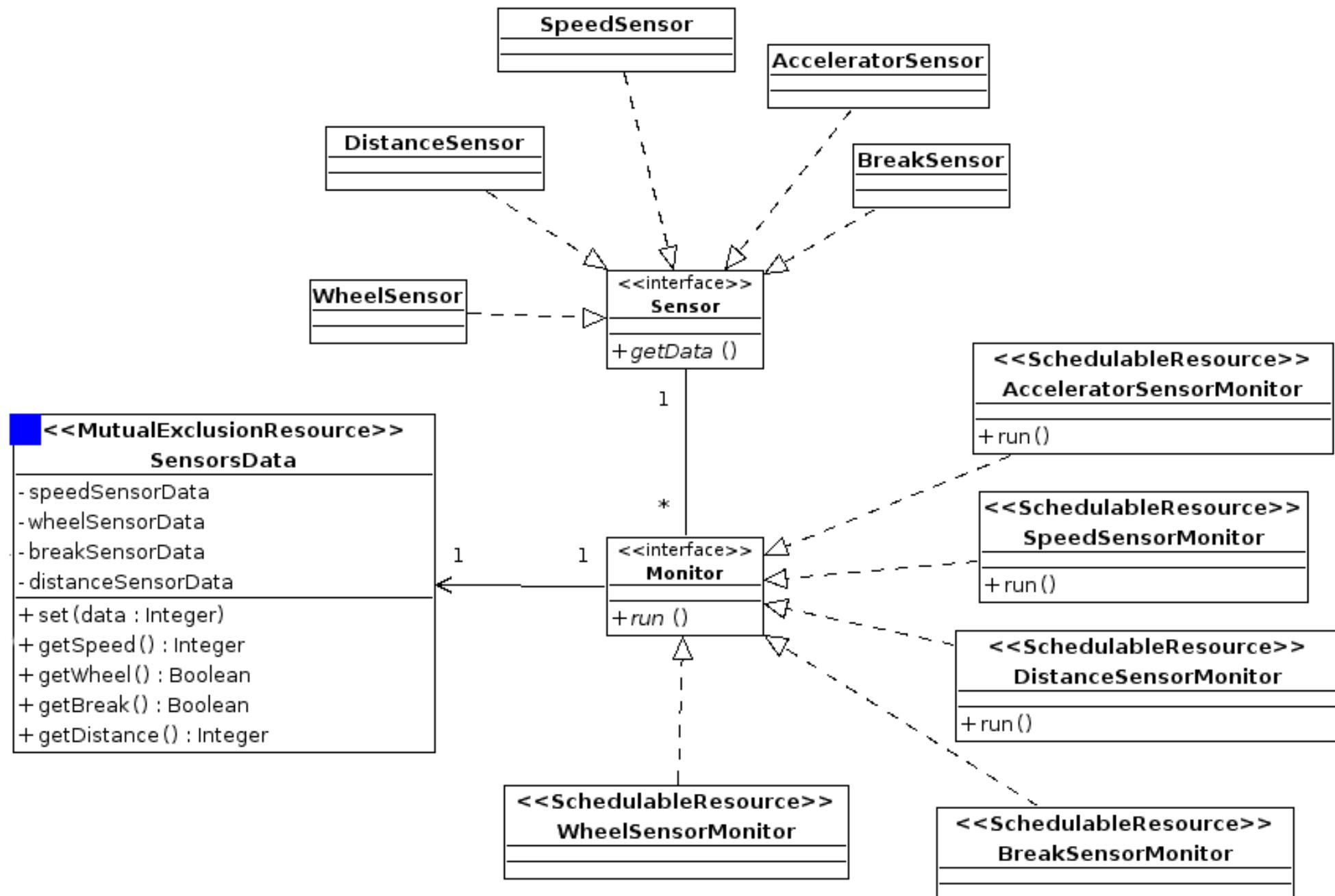
Exercício: sistema de estacionamento autônomo



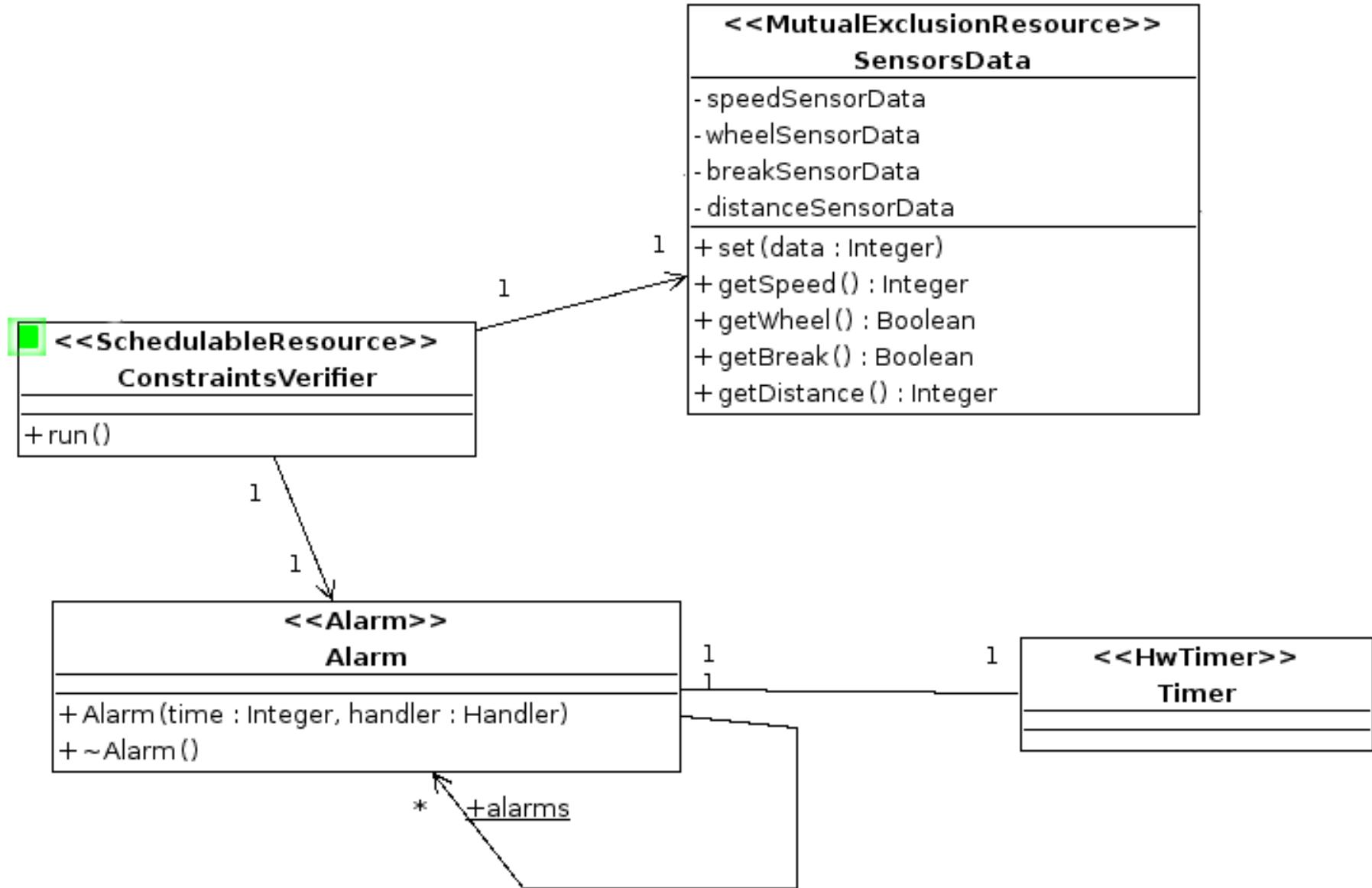
Exercício: sistema de estacionamento autônomo



Exercício: sistema de estacionamento autônomo

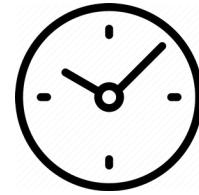


Exercício: sistema de estacionamento autônomo



Exercício: sistema de estacionamento autônomo

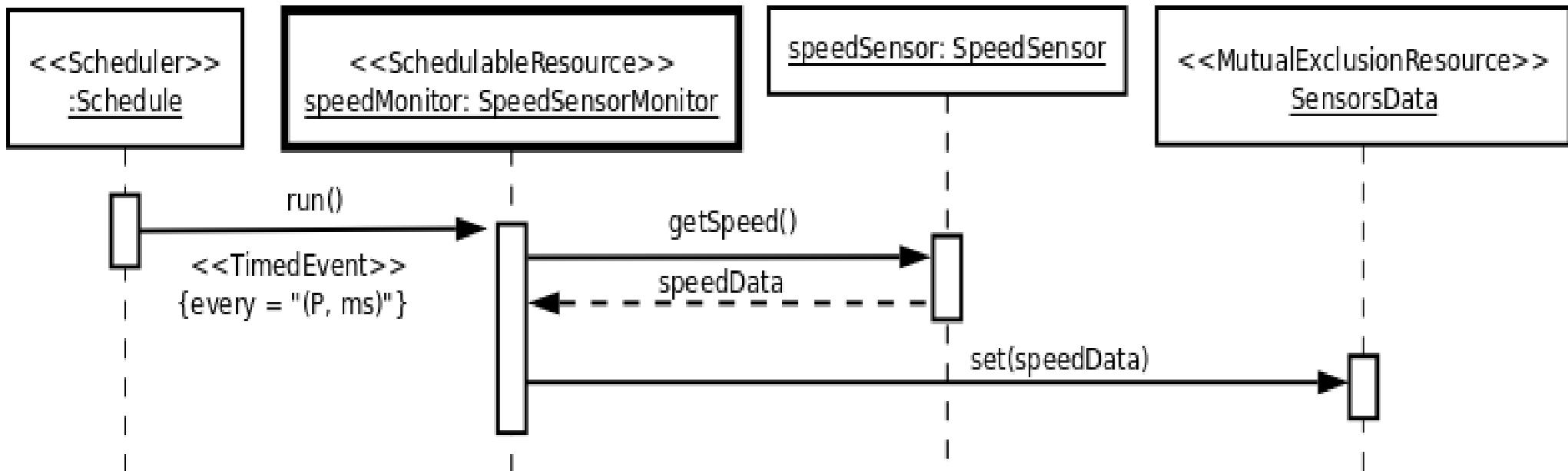
- Faça diagrama de sequência para o monitoramento dos sensores utilizando os estereótipos da UML-MARTE



15 minutos

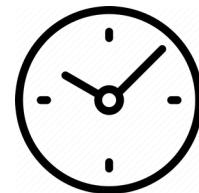
Exercício: sistema de estacionamento autônomo

- Faça diagrama de sequência para o monitoramento dos sensores utilizando os estereótipos da UML-MARTE



Exercício: sistema de estacionamento autônomo

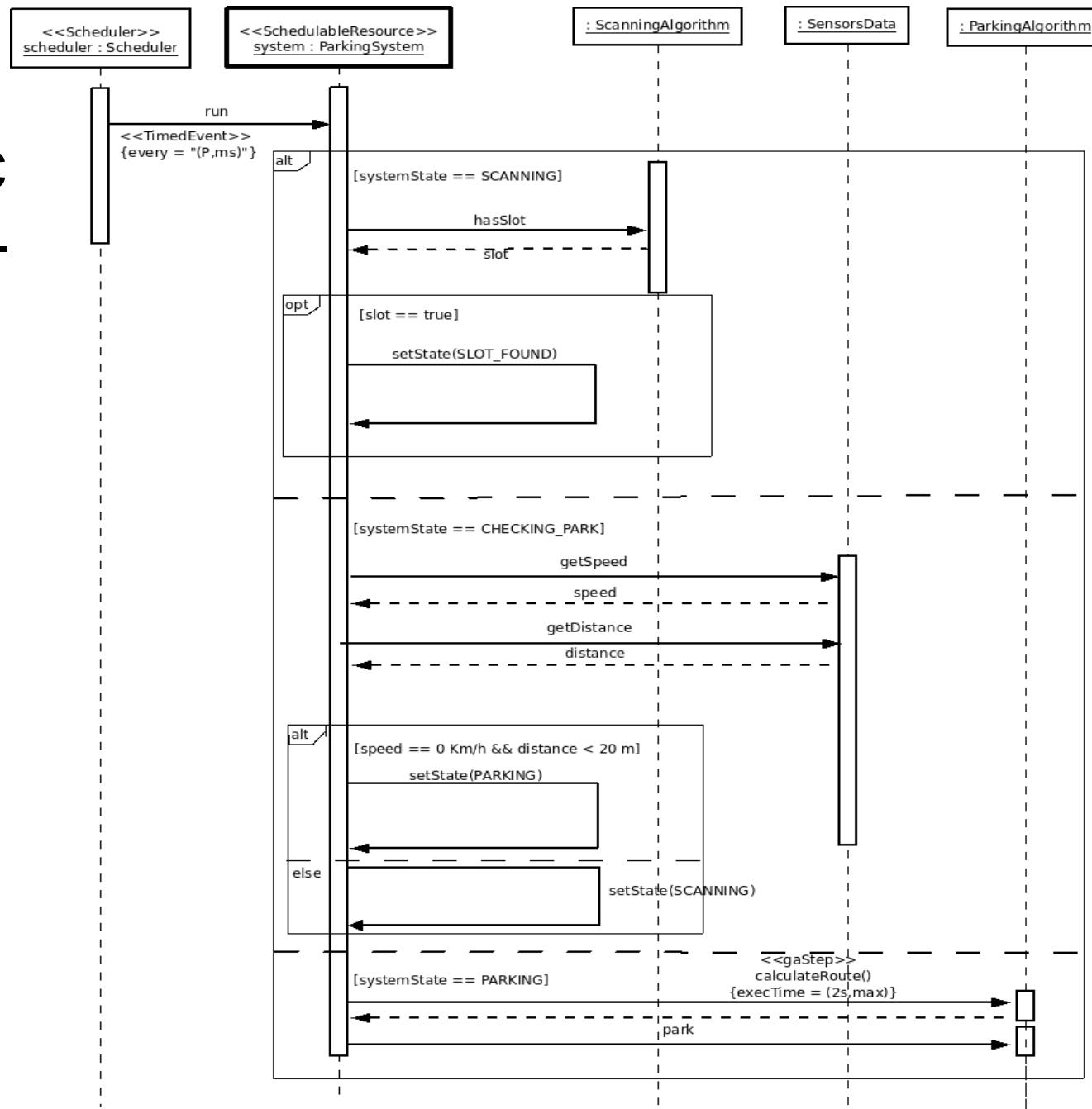
- Faça diagrama de sequência para o controle do estacionamento utilizando os estereótipos da UML-MARTE



15 minutos

Exercício: sistema de estacionamento autônomo

- Faça estac UML-



itrole do
is da

Obrigado!



Referências

- Bran Selic e Sébastien Gerar. Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE – Developing Cyber-Physical Systems. 2013
- <https://www.omg.org/omgmarte/>
- https://www.omg.org/news/meetings/workshops/Real-time_WS_Final_Presentations_2008/Tutorials/00-T5_VanZandt-Mraida_Part2.pdf