Faculdade de Tecnologia de Americana
Curso de Processamento de dados

SISTEMA DE TEMPO REAL COM UML

OTÁVIO ANTONIO DOS SANTOS NETO

Americana, SP
2010

Faculdade de Tecnologia de Americana
Curso de Processamento de dados

SISTEMA DE TEMPO REAL COM UML

OTÁVIO ANTONIO DOS SANTOS NETO
otavioasneto@gmail.com

Monografia desenvolvida em cumprimento à exigência curricular do Curso de Processamento de Dados, sob orientação da Prof. Dr. Olivo Bedin.

Americana, SP
2010

BANCA EXAMINADORA

Prof. Dr. Olivo Bedin (Orientador)

Prof[a.] Dr[a.] Acácia de Fátima Ventura

Prof. Irineu Ambrozano

AGRADECIMENTOS

Em primeiro lugar a Deus.

A minha “ajudante” oficial, professora Acácia, sem a qual certamente não teria conseguido concluir esse trabalho.

Não poderia esquecer-me de agradecer ao professor Olivo, que se propões a me orientar, mesmo estando já no meio do caminho, talvez não saiba, mas falar “o trabalho está bom”, foi decisivo para pra mim.

Em especial, a minha colega de classe Elisângela, pela ajuda dispensada.

DEDICATÓRIA

A minha esposa, minha mãe e em especial ao meu pai, pelo pedido.

RESUMO

O presente trabalho de conclusão de curso conceitua sistemas de tempo real, que com o avanço da tecnologia e a necessidade de se criar ferramentas para facilitar e gerar segurança para tarefas críticas e também para as tarefas rotineiras, cada vez mais são utilizados sistemas de tempo real, que trabalham com restrições temporais para cumprimento de tarefas. Os sistemas de tempo real trabalham com verificações constantes do meio controlado, onde, por meio de sensores, obtém informações do meio físico e respondem aos estímulos obtidos de acordo com o programado. Apresenta também a UML, linguagem de modelagem usada para projetar sistemas, a UML, que é uma linguagem unificada de modelagem, uniu três métodos de modelagem, e é voltada para modelagem orientada a objetos.

Palavras Chave: UML, Sistemas, Tempo, Real.

ABSTRACT

The conclusion course work covers the systems in a real time, with the technological advances and the needs to create new tools to get easier and safer the critical and routine tasks, the systems in a real time that work with restrictions times to make the tasks are increasing. The systems in a real time work with a lot of verifications in a control place, and with sensors, it gets the information in a physical place and answers the stimuli obtained in accordance with the programmed. It shows the UML, it's a modeling language used to unified modeling, it unified three methods of modeling, and it's directed to objects modeling.

Keywords: UML, System, Time, Real.

SUMÁRIO

LISTA DE FIGURAS E DE TABELAS

INTRODUÇÃO

Com o avanço da tecnologia e a necessidades que a humanidade tem de criar ferramentas que facilitem e também gerem segurança tanto para as tarefas consideradas críticas como para tarefas simples do cotidiano, as aplicações de sistemas de tempo real são cada vez mais utilizadas. Sistemas com restrições temporais são utilizados desde aparelhos eletrodomésticos como lavadoras aparelhos de DVD entre outros que não demandam uma restrição temporal crítica, até para sistemas de defesas militares, controles de tráfego, controle de usinas nucleares, hidrelétricas, etc.. Independente da utilidade, os Sistemas Operativos de Tempo Real (SOTR) foram desenvolvidos exatamente para dar segurança e previsibilidade a essas tarefas (FARINE, FRAGA e OLIVEIRA, 2000).

A partir do exposto o presente estudo tem por **objetivo geral** estudar o sistema de tempo real, visando compreender sua estrutura e o seu funcionamento, também busca apresentar uma ferramenta eficiente para modelar sistemas de tempo real, que no caso do presente estudo, será mostrada a UML. O estudo não abordará todas as funcionalidades da UML, mas tem por objetivo apresentar a ferramenta, que é particularmente preparada para modelagem de sistemas de tempo real. Já os **específicos** são: Levantamento bibliográfico em sites academicamente reconhecidos e livros; Fazer a leitura e resumo dos textos e apresentar algumas ferramentas da UML.

O tema se **justifica** por sua utilidade e importância no trabalho industrial, e em muitos outros aspectos da vida cotidiana, buscando compreender o sistema para futuramente aplicar o conhecimento.

Metodologia: O primeiro capítulo define o que é um sistema de tempo, mostrando as características e divisões do sistema, bem como a sua estrutura e funcionamento.

Já o segundo capítulo apresenta um breve resumo e mostra algumas ferramentas da UML para modelagem de sistema com o objetivo principal de fazer conhecida a linguagem.

SISTEMAS DE TEMPO REAL

1.1 DEFINIÇÃO DE SISTEMA DE TEMPO REAL

Segundo Sommerville (2003, p.242), um sistema de tempo real é um sistema no qual o correto funcionamento depende dos resultados que o sistema oferece para determinada ação, e do tempo de resposta em que os resultados são oferecidos. O autor descreve ainda que um sistema de tempo real leve é aquele em que o sistema em questão será degradado se os resultados não forem disponibilizados de acordo com os requisitos de tempo especificados pelo sistema, e um sistema de tempo real rígido é aquele cuja operação será incorreta se os resultados produzidos pelo sistema não forem completados dentro das especificações de tempo.

1.2 FUNCIONAMENTO DO SISTEMA DE TEMPO REAL

O funcionamento de um sistema de tempo real, é o de reagir com seu entorno físico, e é feito através de uma interação repetida com o meio físico, respondendo em tempo hábil ou determinado, para cada dado coletado, tempo esse que pode ser determinado na construção do sistema. Por isso, grande parte dos sistemas de tempo real, podem ser definidos como sistemas reativos, pois eles reagem a estímulos oriundos do seu ambiente físico (FARINE, FRAGA e OLIVEIRA, 2000).

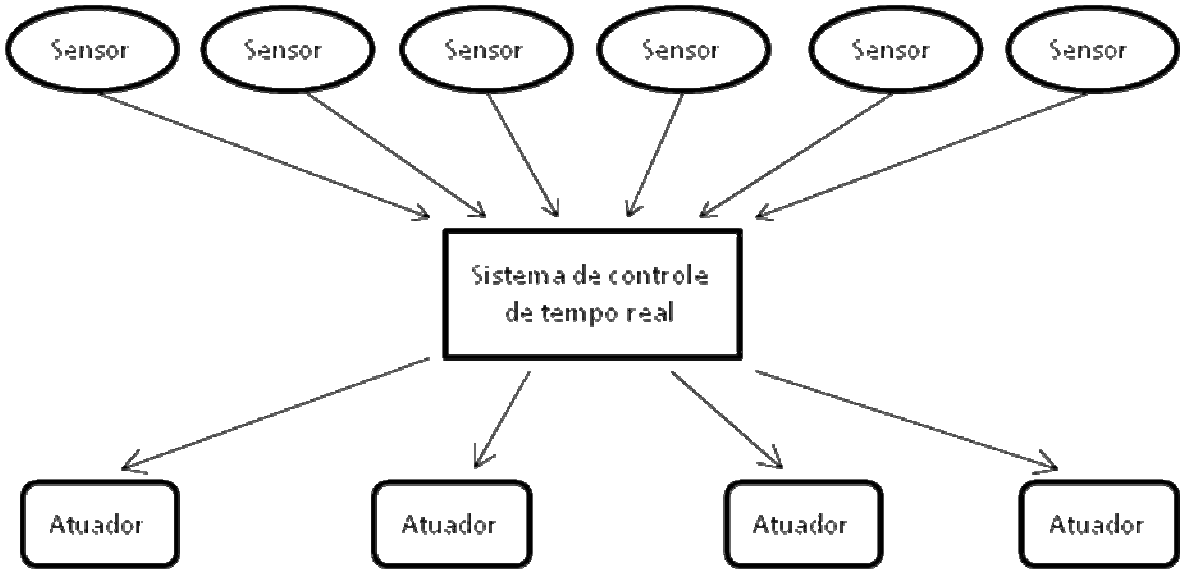
Outra maneira de ver um sistema de tempo real, é comparando com um sistema de estímulos e respostas, em que cada estímulo deve ter uma resposta correspondente, ou seja, o sistema de tempo real é o sistema onde o estimulo recebido pelo sistema, deve ter as respostas associadas a esse estímulo produzidas dentro de um período determinado de tempo (SOMMERVILLE, 2003, p.242).

Sommerville (2003, p.242), define ainda que os estímulos são divididos em duas classes, sendo que uma é a dos estímulos periódicos, onde um sensor conectados ao sistema é verificados dentro de uma faixa definida de tempo e desse modo ele reage de acordo com o valor passado pelo sensor. A outra divisão é dos estímulos aperiódicos, que são os estímulos que não têm uma verificação previsível,

esses estímulos são tratados por um mecanismo que interrompe o computador, de outra maneira, quando um processo necessita de uma resposta do sistema, ele interrompe o funcionamento do sistema, manda um estímulo e aguarda uma resposta em tempo hábil (SOMMERVILLE, 2003, p.242).

Seguindo ainda a definição de Sommerville (2003, p. 242), temos que os estímulos periódicos são normalmente gerados por sensores conectados diretamente ao sistema de tempo real, onde os sensores informam ao sistema as condições do hardware controlado, as respostas a essas informações são enviadas a atuadores conectados ao hardware, que terão uma ação no equipamento de acordo com a resposta enviada pelo sistema.

Figura 1: Sistema de Tempo Real periódico

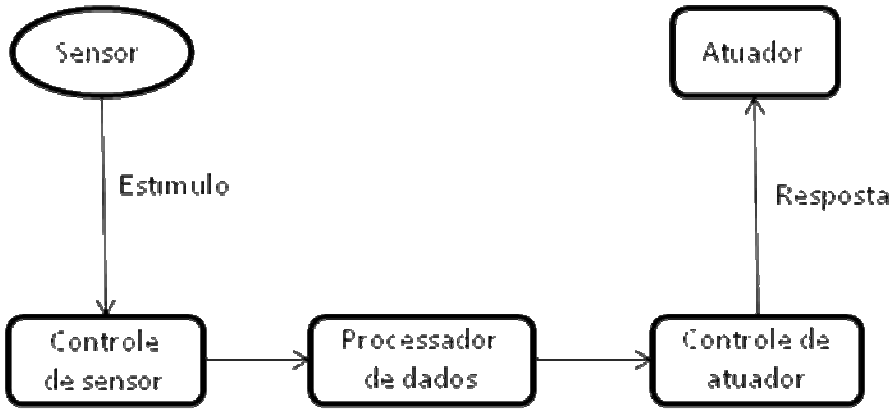


Fonte: SOMMERVILLE, 2003, p.242.

Já para os sistemas de tempo real que respondem a estímulos aperiódicos, esses estímulos sempre indicam alguma condição excepcional, como uma falha de hardware, e é necessário que o sistema seja organizado de maneira não seqüencial, transferindo o controle do sensor e do atuador para um manipulador apropriado, dessa maneira temos um sistema com processos concorrentes e cooperantes, onde parte do sistema de tempo real é dedicado apenas ao controle e gerenciamento desses processos. Nesse modelo temos um processo de gerenciamento para cada tipo de sensor um processo de gerenciamento para cada tipo de atuador, em que os

processos computacionais calculam as respostas requeridas para estímulos recebidos pelo sistema e os processos do atuador gerenciam a ação do atuador (SOMMERVILLE, 2003, p.243).

Figura 2: Sistema de tempo real aperiódico.

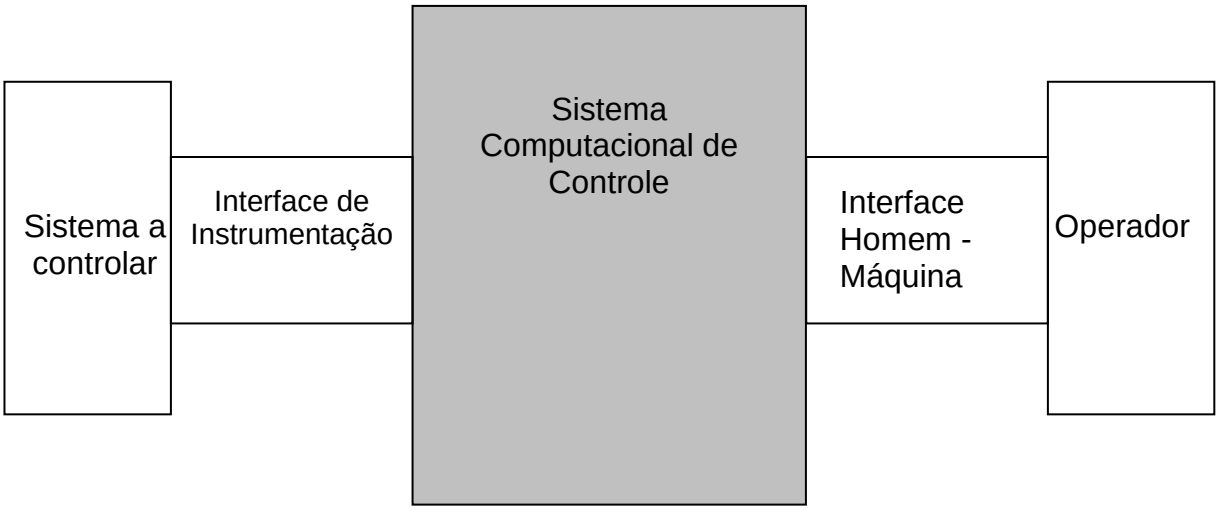


Fonte: SOMMERVILLE, 2003, p.243.

1.3 ESTRUTURA, CARACTERÍSTICAS E CLASSIFICAÇÕES DE UM SISTEMA DE TEMPO REAL

Para Farine, Fraga e Oliveira (2000), um sistema de tempo real típico tem a sua estrutura dividida basicamente em um console para o operador do sistema, o sistema em si e o ambiente que está sendo controlado:

Figura 3: Sistema de Tempo Real.



Fonte: FARINE, FRAGA e OLIVEIRA, 2000.

Para Alcântara (acesso em: 24/03/10), os sistemas de tempo real têm como características principais:

Requisitos temporais – Os sistemas de tempo real trabalham com tempos determinísticos para a correta funcionalidade do sistema, por esse motivo, existem deadlines no sistema, e as medidas de tempo para execução de tarefas são expressas em valores fixos e exatos, pois atrasos na execução são considerados falhas de sistema;

Concorrência na execução de tarefas – Como o sistema reage com seu entorno físico e a obtenção de dados é feita através de sensores instalados, e para cada dado recolhido do meio físico o sistema precisa responder corretamente e em tempo correto, é preciso fazer um escalonamento das tarefas, para que o sistema consiga responder a todos os sinais em tempo real;

Confiabilidade e tolerância a falhas – O sistema tem que funcionar corretamente e em um ciclo de tempo completo, porém como podem ocorrer falhas no sistema, o sistema de tempo real tem que ser capaz identificá-las e gerenciá-las;

Tamanho e complexidade – Como os sistemas de tempo real trabalham com situações reais, ele processa informações complexas e que podem variar

constantemente, por isso, para poder atender a todas as possíveis variações, eles são extremamente grandes e complexos;

Manipulação de números reais – Por se tratar de dados obtidos de situações reais, os sistemas trabalham com comparação de dados de entrada e saída para gerar possíveis sinais de erros, por esse motivo, o sistema precisa estar habilitado para trabalhar com matemática complexa (ALCÂNTARA, acesso em: 24/03/10).

O autor divide ainda, os sistemas em tempo real, inicialmente em dois modelos, os sistemas chamados de *Soft Real Time* e os sistemas *Hard Real Time*.

Os sistemas *Hard Real Time*, são sistemas nos quais falhas temporais de respostas não são aceitáveis, pois podem acarretar prejuízos catastróficos e risco à vida humana, sistemas que controlam usinas nucleares, marca-passos, sistemas de freios de veículos ou aeronaves, entre outros, entram nessa classificação, por isso, nos sistemas *Hard Real Time* as restrições temporais determinantes, e qualquer resposta fora do tempo previsto são consideradas falhas no sistema.

Nos sistemas *Soft Real Time*, as restrições temporais não são determinantes e atrasos nas respostas são aceitáveis e tratadas pelo sistema, pois não costumam acarretar prejuízos ou riscos a vida. Nessa categoria, podemos incluir sistemas que controlam eletrodomésticos, alguns bancos de dados entre outros.

Mas como pode ocorrer dos sistemas não ficarem evidenciados em nenhumas das duas categorias, existe também outra categoria, a categoria dos sistemas *firm real time*, onde os sistemas aceitam um atraso na resposta, mas se esse atraso for muito longo ele será considerado uma falha. Um exemplo são alguns sistemas médicos, em que atrasos de alguns segundos não acarretam prejuízos a vida humana, mas esse segundos precisam ser limitados e se a respostas ultrapassar a tolerância de tempo se tornam falhas inadmissíveis.

1.4 DESENVOLVIMENTO DE SISTEMAS EM TEMPO REAL

Para Alcântara (acesso em: 24/03/10), existem diversas linguagens para programar sistemas em tempo real, normalmente não são usadas as mesmas linguagens utilizadas para desenvolvimento de sistemas de plataformas, como o C, Delphi, Pascal, etc, devidos a falta de precisão nas tratativas relacionadas com o fator determinístico de tempo que é necessário para a execução correta dos sistemas de tempo real.

Embora não venha a trabalhar com todas as linguagens abaixo nesse trabalho, citarei algumas linguagens utilizadas para o desenvolvimento de sistemas de tempo real: **Real-Time UML** – A UML, é uma linguagem de modelagem para fazer construções e relacionamentos de sistemas complexos, essa linguagem não é usada para escrever os algoritmos dos sistemas de tempo real, mas para modelar o sistema. Essa linguagem de modelagem iniciou-se como uma resposta do Grupo de Modelagem de Objetos (OMG) a uma requisição de um padrão para sistemas orientados a objetos; **OCCAM** – É uma linguagem usada para aplicações científicas e de engenharia, para controle de processos industriais e também para sistemas embarcados. Sendo originalmente desenvolvida para transputer (contração de trasistor e computer), uma arquitetura microprocessada desenvolvida pelo INMOS, essa linguagem suporta concorrência e sincronização de tarefas; **Handel – C** – Parecida com a linguagem C, é projetada para compilação de programas em implementações sincrônicas de hardware. Por ser uma linguagem semelhante a C, desenvolvedores de software podem, em um curto espaço de tempo, aprender a linguagem e implementar projetos de hardware; **ADA** – Desenvolvida pelo Departamento de Defesa Americano no início da decada de 80 (ADA 83) e então revisada para a linguagem ADA 95, foi a primeira linguagem orientada a objetos padronizada internacionalmente. Com o intuito de prover uma solução poderosa e robusta para projetos de aplicações críticas, possui características de modularização que facilitam a construção de sistemas de grande porte, além de fornecer métodos de encapsulamento de informações e fornecer métodos avançados para tratar concorrências de tarefas: as tarefas da ADA podem comunicar entre si implicitamente, através de regiões compartilhadas de memória, ou explicitamente por meio de rendezvous; **Real-Time JavaTM –** Desenvolvida pela Sun

Microsystems, essa linguagem vem sendo muito utilizada em diversos ambientes, desde dispositivos embarcados, até para soluções corporativas. A linguagem foi criada para oferecer uma solução portável, orientada a objetos, além de ser distribuída e segura (ALCÂNTARA, acesso em: 24/03/10).

Para Sommerville (2003, p.245), a linguagem que será usada para programar o sistema de tempo real pode também afetar o projeto do mesmo, por esse motivo, os sistemas de tempo real rígido (ou *Hard Real Time)* são, em alguns casos , ainda programados em linguagem assembly, pois nessa programação prazos curtos de requisitos temporais dos sistemas podem ser cumpridos.

O autor explica também que linguagens de nível, como a linguagem C, também pode ser utilizada, e uma vantagem de usar uma linguagem de baixo nível como a C, é que se pode gerar programas muito eficientes, o único porém é que a linguagem não oferece nenhum princípio para trabalhar com a concorrência ou com o gerenciamento de recursos compartilhados. Como a linguagem se baseia em sistemas operacionais ou em recursos de executivos, existe um maior enfoque nos erros de programação.

Continuando, o autor apresenta ainda as vantagens e desvantagens de se usar as outras linguagens de programação como no caso da ADA, que tem características como execução de tarefas, exceções e cláusulas de representação. Sua capacidade de *rendezvous* é um mecanismo muito bom para sincronizar tarefas, porém a versão inicial não era eficiente para desenvolvimento de sistemas de tempo real rígido, pois não era possível especificar prazos para execução de tarefas, a versão atualizada melhorou esses aspectos da linguagem, pois permitiu uma implementação mais fácil de estruturas protegidas e também permitiu maior controle sobre o escalonamento de tarefas e tempo, mas ainda assim não fornece controle suficiente para sistemas de tempo real complexos.

O autor destaca também a linguagem Java, por ter sido desenvolvida para sistemas integrados de pequena escala, como sistemas de equipamentos domésticos, foram incluídos na linguagem, algumas compatibilidades para processos concorrentes, contudo, como em geral os sistemas de pequena escala

não têm restrições rígidas de tempo, não foi incluída na linguagem recursos para controlar escalonamento de objetos concorrentes (*threads*) ou para especificar qual deva ser executado no momento específico. Essas observações não tornam a linguagem adequada para programar sistemas de tempo real rígido (SOMMERVILLE ;2003, p.245).

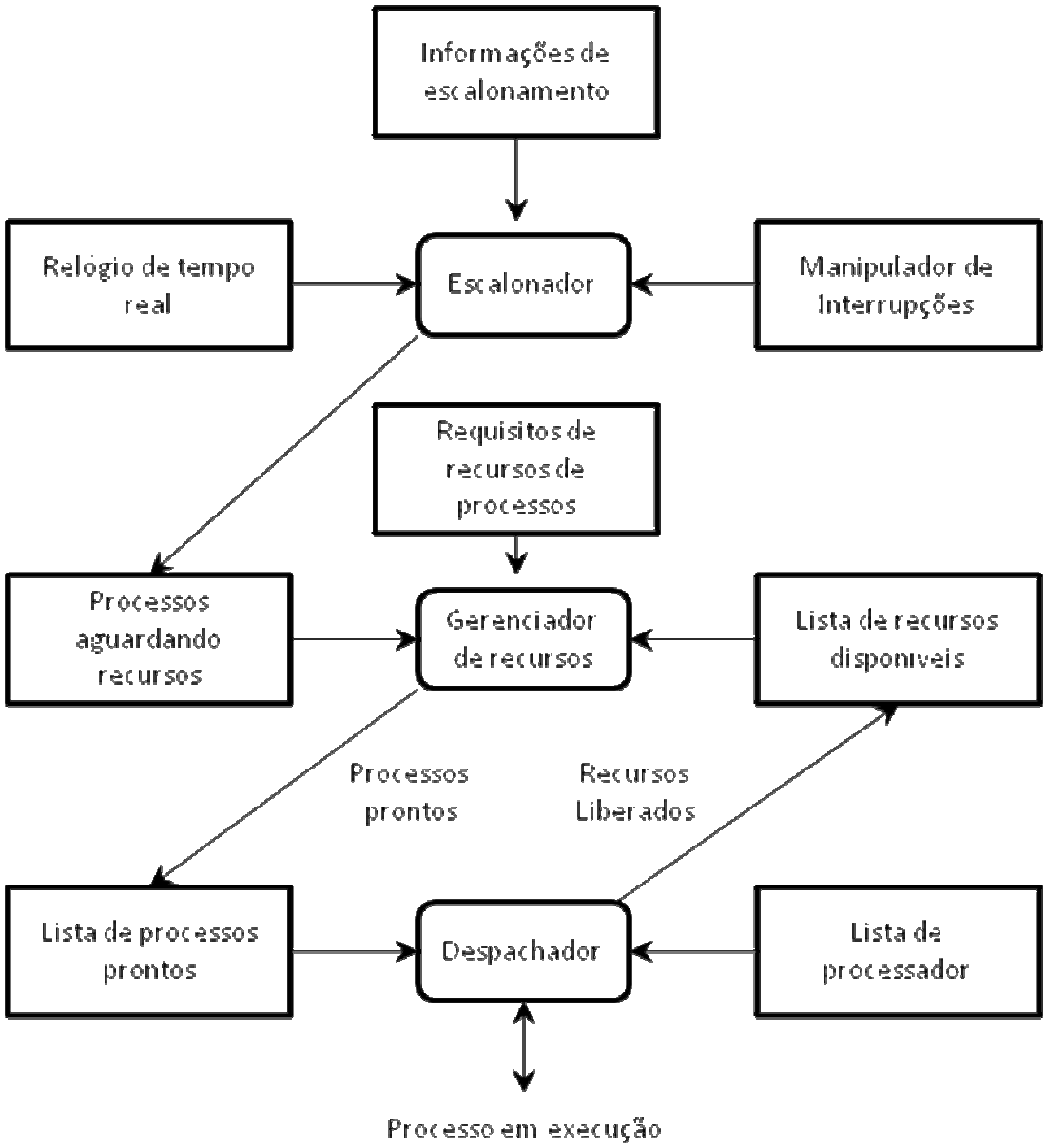
1.5 ESCALONAMENTO DE TEMPO REAL

Nos sistemas de tempo real, onde noções de tempo e concorrência são tratados de forma explícita, os conceitos relacionados ao escalonamento de tarefas, em geral focando mecanismos de prioridades, é o ponto chave na previsibilidade e comportamento dos sistemas de tempo real (FARINE, FRAGA e OLIVEIRA, 2000).

Para facilitar o entendimento, precisamos entender conceitos como executivos de tempo real, o executivo gerencia as alocações de recurso e processos de um sistema de tempo real, interrompendo e iniciando processos apropriados, de modo que os estímulos possam ser manipulados (SOMMERVILLE, 2003, P.247).

Os componentes de um executivo, vão depender do tamanho e da complexidade do sistema de tempo real, mas em geral, incluirão um relógio de tempo real, que fornecerá dados para escalonar os processos periodicamente, um manipulador de interrupções, esse componente administrará os pedidos aperiódicos de serviços, um escalonador, que é o responsável por verificar os processos que podem ser executados e escolher qual deverá ser processado, um gerenciador de recursos, que aloca a memória e recursos de processador apropriados para um processo que esteja programado para execução e um despachador, que é responsável por começar a execução de um processo. Sistemas que fornecem um serviço contínuo, como os de telecomunicações, podem incluir também um gerenciador de configuração, que controla a configuração dinâmica do hardware, onde partes do hardware podem ser tiradas de serviços e novas partes podem ser incluídas, tudo sem interromper o funcionamento do sistema, e ainda um gerenciador de defeitos, que é responsável pela detecção de defeitos tantos do software quando do hardware e tomar as medidas apropriadas para a recuperação a partir desses defeitos (SOMMERVILLE, 2003, p. 247).

Figura 4: Componentes de um executivo de tempo real.



Fonte: SOMMERVILLE, 2003, p. 247.

Outro conceito é a tarefa, ou processo, que é a abstração básica a qual faz parte do escalonamento. Um sistema básico de tempo real é constituído basicamente de diversas tarefas que formam as unidades de processamento seqüencial e concorrem com um ou mais recursos computacionais de um sistema (FARINE, FRAGA e OLIVEIRA, 2000).

1.5.1 RESTRIÇÕES TEMPORAIS

Para Farine, Fraga e Oliveira (2000), os sistemas de tempo real são caracterizados por terem restrições temporais, e essas restrições precisam ser respeitadas. Todas as tarefas precisam ser completadas dentro do limite de tempo especificado, e o não cumprimento da tarefa dentro do tempo determinado define dois tipos de tarefas de tempo real:

Tarefa Crítica – é considerada crítica, a tarefa que se completada após o tempo limite, pode causar danos catastrófico no sistema, ou no ambiente controlado. Uma falha dessas pode causar danos irreversíveis no equipamento ou mesmo perda de vidas humanas;

Tarefa Branda – é considerada branda, a tarefa que se completada após o tempo limite, não causa danos significativos ao sistema ou ambiente controlado, no máximo causa uma diminuição de desempenho no sistema de tempo real.

Os autores apontam ainda outras características das tarefas de tempo real, que são definidas pela freqüência de ativação:

Tarefas periódicas – são as tarefas que são ativadas em uma sequência infinita, sendo que a primeira ativação ocorre na origem dos tempos designados na aplicação (t=0);

Tarefas aperiódicas ou assíncronas – as tarefas são ativadas aleatoriamente, em resposta a um evento externo, ou interno do sistema de tempo real.

Continuando, os autores dizem ainda que as tarefas periódicas devem ser, usualmente, associadas às tarefas consideradas críticas, devido à previsibilidade e regularidade em que elas ocorrem, já as tarefas aperiódicas, por não terem previsibilidade, devem associadas às tarefas brandas.

As outras definições consideradas importantes pelos autores para definir o comportamento temporal de uma tarefa são:

Tempo de computação – É o tempo necessário para completar a execução da tarefa;

Tempo de início – Corresponde ao instante de início do processamento de uma tarefa;

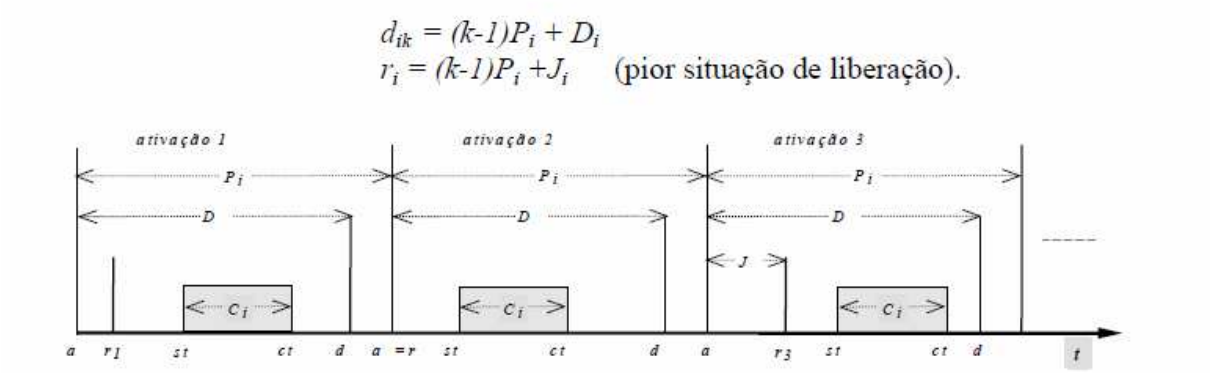
Tempo de término – É o instante de término de execução de uma tarefa no sistema;

Tempo de chegada – É o instante em que o escalonador toma conhecimento da ativação de uma tarefa;

Tempo de liberação – Coincide com o momento da sua inclusão na fila de tarefas prontas para execução (FARINE, FRAGA e OLIVEIRA, 2000).

Pode acontecer, em alguns casos, o tempo de liberação não coincidir com o momento de inclusão da tarefa na lista de tarefas prontas, pois em alguns casos, a liberação pode ser retardada pelo *polling* de um escalonador ou talvez, devido a um bloqueio na recepção da mensagem, o nome dado para essa não coincidência entre a chegada e a liberação da tarefa é chamado de *Release Jitter*, que é a máxima variação dos tempos de liberação das instâncias das tarefas (FARINE, FRAGA e OLIVEIRA, 2000).

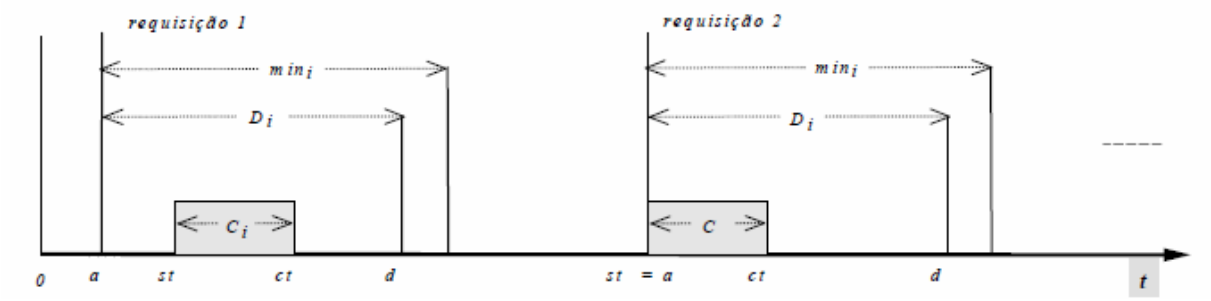
Figura 5: Ativação de uma tarefa periódica



Fonte: FARINE, FRAGA e OLIVEIRA, 2000, Cap. 2, p. 14.

> (...) O comportamento temporal de uma tarefa periódica $T_{i,}$ descrito pela quádrupla (J_i, C_i, P_i, D_i) onde $C_{i,}$ representa o tempo de computação da tarefa, P_i é o período da tarefa, D_i é o tempo limite e J_i é o *Release Jitter* da tarefa que, de certa maneira, corresponde a pior situação de liberação da tarefa. Nessa representação de tarefas periódicas, J_i e D_i são grandezas relativas (intervalos), medias a partir do início do período P_i. O tempo limite absoluto e o tempo de liberação da $k^{ésima}$ ativação da tarefa períodica T_i são determinados a partir dos períodos anteriores.
>
> Na figura é ilustrado alguns dos parâmetros descritos acima. Porém, cada ativação da tarefa períodica (J_i, C_i, P_i, D_i) é definida a partir dos tempo absolutos: os tempos de chegada (a_i), os tempos de liberação(r_i), os tempos de início (st_i), os tempo de termino (ct_i) e os *"deadlines"* absolutos (d_i) (FARINE, FRAGA e OLIVEIRA, 2000, Cap. 2, p. 14).

Figura 6: Ativação de uma tarefa aperiódica



Fonte: FARINE, FRAGA e OLIVEIRA, 2000, Cap. 2, p. 14.

> Uma tarefa esporádica é descrita pela tripa ($C_{i,}$ $D_{i,}$ $min_{i)}$ onde C_i é o tempo de computação D_i é o *"deadline"* relativo medido a partir do instante da requisição do processamento aperiódico (chegada da tarefa esporádica) e min_i corresponde ao mínimo intervalo entre duas requisições consecutivas da tarefa aperiódica. A descrição de uma tarefa aperiódica pura se limita apenas às restrições $C_{i,}$ $D_{i.}$. Na figura, a tarefa aperiódica esporádica ($C_{i,}$ $D_{i,}$ min_i) é apresentada com duas requisições. Tomando o tempo de chegada da requisição esporádica 2 como a a_2 o "*deadline"* absoluta desta ativação assume o valor dado por: $d_2 = a_2 + D_i$ (FARINE, FRAGA e OLIVEIRA, 2000, Cap. 2, p. 15).

Sommerville (2003, p. 248), também fala que os estímulos processados por um sistema de tempo real têm diferentes níveis de prioridade, alertando que alguns estímulos, aqueles associados a eventos excepcionais, devem ter, essencialmente, o seu processamento concluído dentro do limite de tempo especificado, mesmo que seja necessário atrasar, com segurança, outros processos menos importantes, por

esse motivo, o executivo para um sistema de tempo real tem que ser capaz de gerenciar no mínimo dois níveis de prioridade de processos:

1 – Nível de interrupção – É o nível de mais alta prioridade, deve ser alocado para eventos que necessitem de uma resposta muito rápida e precisa. Um processo com essa classificação será o de relógio de tempo real;

2 – Nível de relógio – Os processos periódicos, que se repetem com freqüência programada, são alocados nessa classificação (SOMMERVILLE; 2003, p. 248).

Outro nível de prioridade identificado por Sommerville, são os alocados para processos de background, como é o caso de processos de autoverificação, que não tenham a exigência de atender prazos de conclusão em tempo real. O autor considera ainda que dentro de cada um desses níveis de prioridades, diferentes classes de processos podem ser alocados em diferentes níveis de prioriodade, um exemplo apresentado, é que dentro de um processo pode haver várias linhas de interrupção , e uma interrupção feita por um dispositivo muito rápido pode ter que ser previsto o processamento da interrupção a partir de um dispositivo mais lento, para que não haja o risco de perdas de informação.

1.5.2 GERENCIAMENTO DE PROCESSOS

Em um sistema de tempo real, o executivo de tempo real, é quem faz o gerenciamento dos processos, cuida de fazer o gerenciamento dos processos concorrentes, é ele quem escolhe o processo que deve entrar em execução, alocar memória e recursos do processador, além de iniciar a execução do processo (SOMMERVILLE, 2003, p. 248).

Sommerville (2009, p.248), explica que processos periódicos, que precisam ser executados em intervalos de tempo determinados, e o relógio de tempo real do sistema é configurado para fazer o ‘tique’ periodicamente, esse ‘tique’ inicia um processo de nível de interrupção, que então programa o gerenciador de processos para processos periódicos. O processo de nível de interrupção normalmente não é o

responsável pelo gerenciamento de processos periódicos, porque ele precisa ser concluído o mais rápido possível (SOMMERVILLE, 2003, p. 248).

O autor destaca que a lista de processos é examinada pelo escalonador, que escolherá o que deverá ser executado dependendo da prioridade do processo, dos prazos estabelecidos para início e conclusão. Quando acontece de dois processos precisarem ser respondidos ao mesmo tempo, o sistema adia um deles, avaliando se os prazos de conclusão poderão ser atendidos.

A qualquer momento, pode acontecer de ter vários processos, com diferentes prioridades para serem executados, e para decidir qual deverá ser executado, existem duas estratégias fundamentais de escalonamento: O não preemptivo (*non pré-emptive)*, que uma vez escalonado, o processo é executado até ser concluído ou interrompido (como nos casos de ter que aguardar por uma entrada). Essa estratégia causa problemas quando um processo de alta prioridade precisa aguardar um processo de baixa prioridade; O preemptivo (*pré-emptive),* que determina que um processo de prioridade alta pode interromper um processo de baixa prioridade (SOMMERVILLE; 2009, p. 249).

O autor explica que após concluído o processo passa para um lista de “pronto”, e o despachador é chamado, que ao encontrar um processo para ser executado, o faz no processador disponível.

2 UML - LINGUAGEM DE MODELAGEM

Segundo Grady, Rumbaugh e Jacobson (2005), uma linguagem, seja ela qual for, tem a finalidade de fornecer um vocabulário, e regras para combinação deles, com a finalidade de permitir um sentido para algo que se queira comunicar, dessa forma as linguagens de modelagem são modelos padrões que tem a intenção de facilitar a comunicação de algo que se queira modelar.

O autor acrescenta que as regras definidas na linguagem vão indicar como criar e ler modelos, mas não dirá quais modelos deverão ser criados. A tarefa de decidir o que deverá ser feito terá que ser definida pela equipe de desenvolvimento do modelo em questão. Uma vez definida a linguagem que será usada, o fato de se utilizar regras de modelagem, que são padrões definidos e com significados certos, ficará facilitada a comunicação e o entendimento do que se deseja ser feito entre grupos de desenvolvedores, além é claro de facilitar à visualização e a verificação todas as possibilidades existentes para o desenvolvimento de uma ferramenta ou sistema.

Como a linguagem que abortada será a UML, temos um breve resumo da origem da *Unified Modeling Languaje,* que se originou a partir do esforço dos autores de outras metodologias, que são a *Object Modeling Technique* (OMT), de James Rumbaungh, a *Booch Method* (Booch), de Grady Booch, e a *Object-Oriented Software Engineering* (OOSE), de Ivar Jacobson, que em meados de 1990, começaram a trabalhar juntos para unificação de seu métodos, e foi certificada como padrão da industria em 1997, pelo OMG – *Object Management Group* – sendo a partir de então utilizado por várias corporações como padrão para análise e *design* de *software*. A UML é uma linguagem de modelagem, voltada para desenvolvimento de sistemas orientados a objetos, que permite desenhar, planejar, analisar um sistema a ser desenvolvido, além é claro, de documentá-lo após a construção, mas pode ser utilizado, também, para documentar um já existente (GRADY, RUMBAUGH e JACOBSON; 2005.)

Continuando, temos a explicação dada por Furlan **(**1998; p.33**)**, que acrescenta o fato da UML ser muito mais que uma simples padronização em busca

de uma notação unificada, pois contém conceitos que não são encontrados em outros métodos orientados a objetos. A UML é uma linguagem que incorporou técnicas de modelagem de dados como Diagramas de Entidade e Relacionamento, modelagem de negócios, modelagem de objetos e componentes, além de incorporar as idéias de vários outros autores, ou seja, os criadores da UML não desenvolveram a maioria da idéias que compõem a UML, mas em vez disso, eles selecionaram o que havia de melhor no mercado e integraram a UML (FURLAN; 1998, p.33**)**.

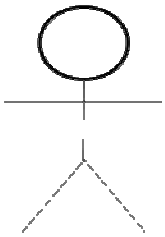
Furlan (1998; p. 33) explica que a linguagem padrão da UML serve para visualizar, documentar e construir artefatos de um sistema e pode ser utilizada em todo o ciclo de desenvolvimento do sistema, independente da tecnologia ou código de programação que será usado para implementar o projeto.

2.1 ENTENDENDO A UML

A UML trabalha com padrões visuais de *design*, não especificamente com um código de programação, portanto ela pode ser usada para desenvolver sistemas em qualquer linguagem. Por exemplo, para entendermos, ou mesmo especificar as funcionalidades globais do sistema, podemos utilizar um modelo chamado de *use-case* **(**FURLAN; 1998; p. 77**)**.

Um use-case é usado para representar uma pessoa, ou mesmo algum mecanismo que dispara uma ação no sistema, esse conceito é conhecido como ator (GRADY, RUMBAUGH e JACOBSON, 2005).

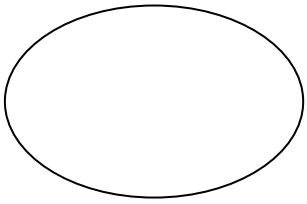
Figura 7: Ator.



Fonte: Grady, Rumbaugh e Jacobson, 2005, p. 231 – adaptado pelo autor.

O símbolo acima representa um ator (*actor*), e é usada para representar um ser humano, dispositivo ou um sistema que interaja com o sistema modelado, e a finalidade é exatamente representar os papeis que os usuários do sistema desempenhar ao interagir com o sistema (GRADY, RUMBAUGH e JACOBSON, 2005, p. 231).

Figura 8: Caso de uso (*use-case)*



Fonte: Grady, Rumbaugh e Jacobson, 2005, p. 230 – adaptado pelo autor.

O símbolo acima representa o caso de uso, e é a descrição das sequência de ações que o sistema executa, incluindo também as variantes, que produzirá um resultado perceptível para o ator (GRADY, RUMBAUGH e JACOBSON, 2005, p.230).

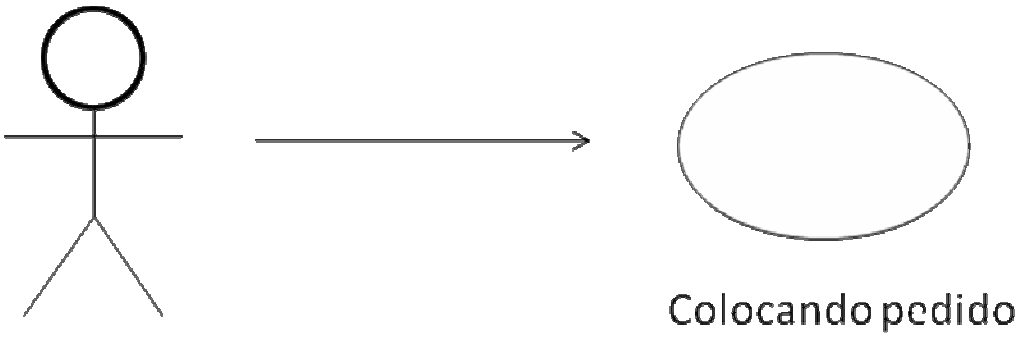
Figura 9: Relacionamento

Fonte: Grady, Rumbaugh e Jacobson, 2005, p.65 – adaptado pelo autor.

A seta é o símbolo[1] representa a interação que existe entre o ator e o caso de uso, ou seja um relacionamento é a conexão entre itens (GRADY, RUMBAUGH e JACOBSON, 2005, p.230).

Figura 10: Relacionamento de um caso de uso.

[1] Existem outros tipos de setas que representam diferentes tipos de relacionamentos, para efeito explicativo, a seta representa o relacionamento entre os diferentes itens e, para casos específicos poderá ter outras representações gráficas. No capítulo 2.2, serão apresentadas outras representações.



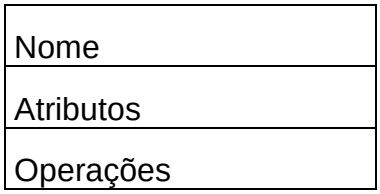
Fonte: Furlan; 1998; p. 77– adaptado pelo autor.

Observando o exemplo acima, temos que o ator representa algo que interage com o sistema, pode ser tanto uma pessoa quanto uma máquina, sensor, equipamento. O ator não é parte do sistema, é usado para representar ações que o usuário do sistema pode realizar ou obter passivamente. Caso de uso representa as ações que o sistema executa, e que produz um resultado para o ator. Ou seja, o *use-case* é uma funcionalidade completa e consistente do sistema que é acionado pelo ator para produzir um resultado de valor para o ator, e o conjunto de todos os casos de uso, representa todas as ações possíveis de utilização do sistema (FURLAN; 1998, p.77).

Objetos: É uma representação abstrata de algo definido, chamado de entidade, e que possui significado para o sistema. Pode tanto representar uma entidade física, conceitual ou de software. Um objeto possui três componentes definidos: Identidade, que é a identificação do objeto, ele é único é representa apenas um objeto; Estado, são os atributos do objeto, representam as informações complementares do objeto e as ligações que o objeto pode ter com outros objetos; Comportamento, é a reação do objeto a estímulos vindo de outros objeto, e é modelado através de mensagens, que representam os resultados visíveis obtidos das operações do objeto em questão (FURLAN; 1998, p. 94).

Classe de Objetos: A classe de objetos é a representação dos atributos, comportamentos e relacionamentos de um grupo de objetos. As classes são consideradas os blocos de construção mais importantes de um sistema orientados a objetos, pois como já foi colocado, a classe é a descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semânticas (GRADY, RUMBAUGH e JACOBSON, 2005, p.230).

Figura 11: Classe de objetos

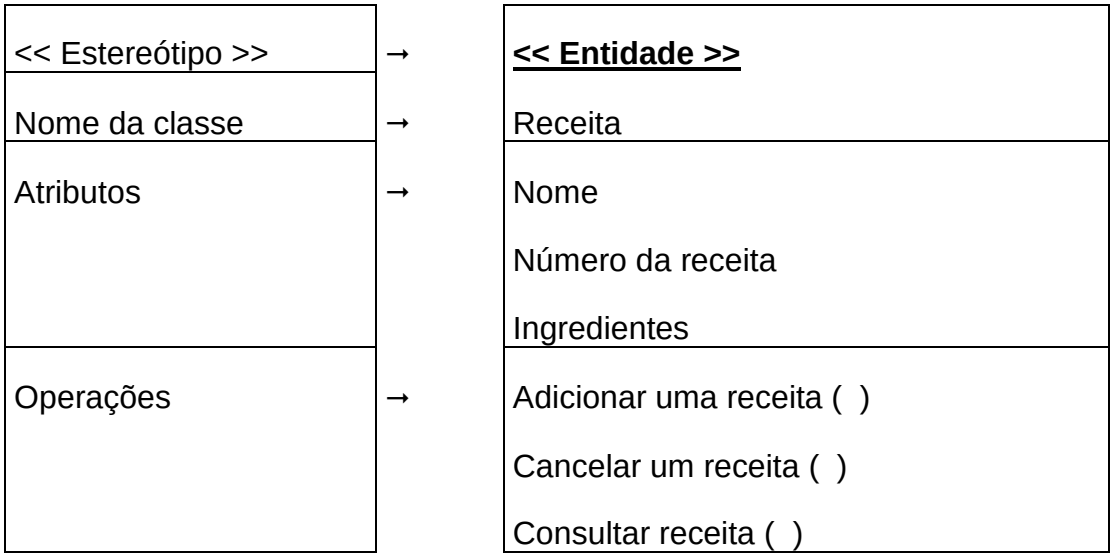


Fonte: Furlan; 1998, p. 100 – adaptado pelo autor.

Estereótipos: Sugerido por Rebecca Wirfs-Brock (apud. FURLAN; 1998, p.87), o uso de estereótipos foi definido para criar uma metaclassificação de elementos na UML, ou seja, a introdução de novos elementos que permitam ao usuário entender a capacidade de modelagem da linguagem. Resumindo, os estereótipos são elementos da modelagem que rotulam os tipos de classes, identificando o tipo e a função da classe no sistema.

A notação em UML para representar estereótipos, é colocando-os entre *guillemets* (<< >>) (FURLAN; 1998, p.88).

Figura 12: Estereótipos



Fonte: Furlan; 1998, p.88 – adaptado pelo autor.

2.2 DESENVOLVENDO SISTEMAS DE TEMPO REAL COM UML

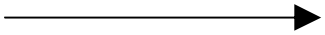
Para Furlan (1998, p. 193), na modelagem de sistemas de tempo real usando UML, é necessária uma extensão à teoria clássica, no sentido de tratar aspectos de comunicação, tratamentos de eventos, simultaneidade e sincronização, além é claro, como já foi comentado, considerar o ambiente físico em que o sistema está integrado.

Furlan acrescenta que um conceito introduzido ao se modelar sistema com execução de tarefas concorrentes é o de classe ativa. A classe ativa significa que ao contrário dos objetos de classe passiva, que aguardam uma mensagem, ou um comando para executar uma ação, os objetos de classe ativa podem tomar a iniciativa de executar ações sem a necessidade de envio de uma mensagem. As classes ativas são frequentemente implementadas através de bibliotecas de classes, que tem uma super classe *ActiveClass.* Na UML, uma classe ativa é mostrada com um retângulo de classe regular desenhado com uma borda espessa, exibindo o estereótipo <<classe ativa>>. Além da diferenciação na demonstração do objeto, existem ícones para serem usados juntos com mensagens, no intuito de indicar como os processamentos de classes ativas concorrentes são sincronizados durante a transferência de mensagens.

A seguir veremos as diferentes variações de ponta de flecha, que podem ser usadas para indicar os diferentes tipos de fluxos de mensagens (FURLAN; 1998, p.195):

Ponta de flecha sólida preenchida: Chamada de procedimento síncrona ou outro fluxo aninhado de controle, onde a sequência aninhada inteira é completada antes da retomada da sequência acionadora. Em um sistema de tempo real, com concorrência de tarefas, uma mensagem síncrona representa uma linha de controle com semântica de espera, dessa forma o remetente esperará por tempo indeterminado pelo destinatário aceitar a mensagem antes de continuar seu processamento. Ela pode ser usada para chamada de procedimentos comuns e também com objetos concorrentes ativos quando um deles envia um sinal e espera pela conclusão de uma sequência aninhada de comportamento;

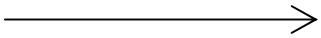
Figura 13: Ponta de flecha sólida preenchida



Fonte: Furlan; 1998, p.195.

Ponta de flecha fina: Fluxo de controle simples, normalmente assíncrono. Tem a função de mostrar como um controle é passado de um objeto para outro sem descrever qualquer detalhe sobre a comunicação, é usado quando os detalhes da comunicação são desconhecidos ou não são considerados relevantes no diagrama e também para indicar o retorno de uma mensagem assíncrona ao ser desenhada uma seta do objeto que manipula ao objeto solicitante (FURLAN; 1998, p.195);

Figura 14: Ponta de flecha fina



Fonte: Furlan; 1998, p.195.

Meia ponta de flecha fina: Significa um fluxo de mensagem assíncrona para mostrar o envio de uma mensagem com semântica de nenhuma espera, ou seja, o remetente envia a mensagem e continua imediatamente em processamento sem esperar pelo destinatário reconhecer sua prontidão em receber a mensagem. Essa flecha é usada para indicar explicitamente uma mensagem assíncrona entre dois objetos em uma sequência procedural (FURLAN; 1998, p.195);

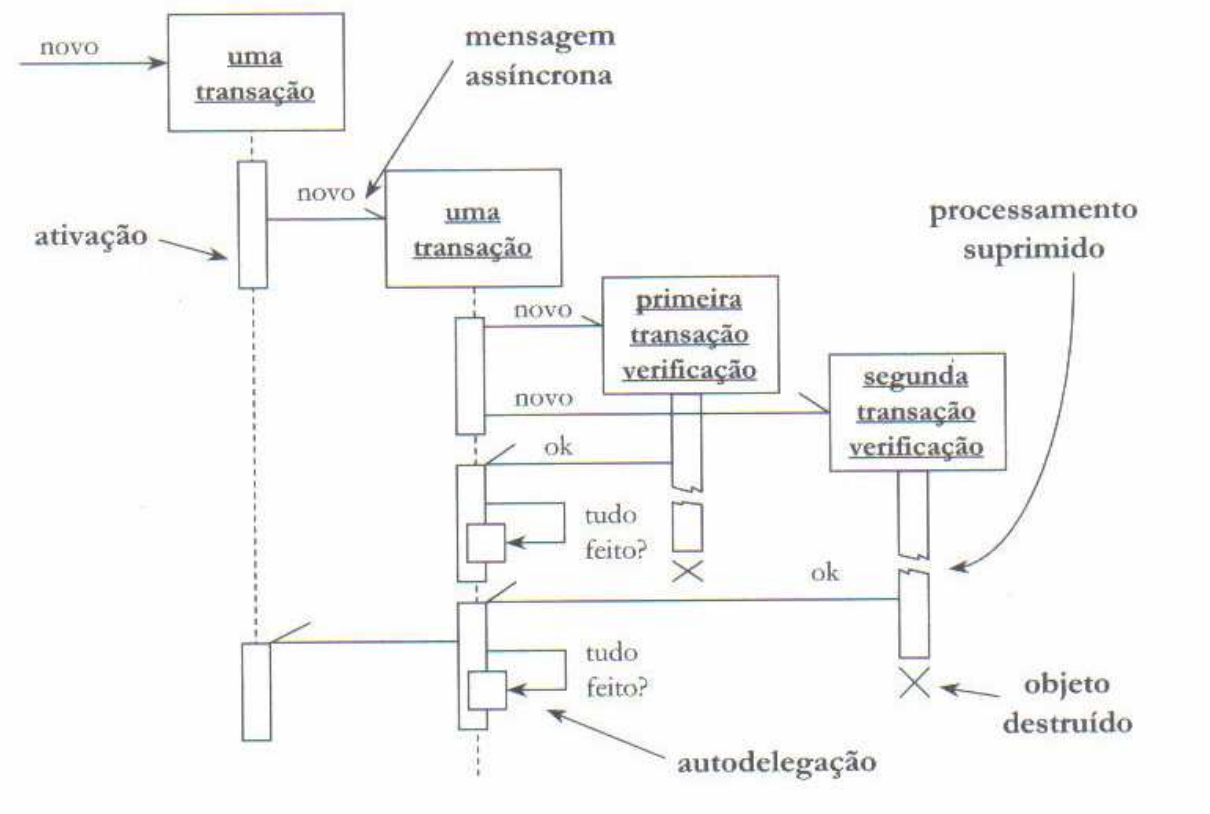
Figura 15: meia ponta de flecha fina.

Fonte: Furlan; 1998, p.195.

Mensagem de intervalo: Indica que o remetente esperará pelo destinatário estar pronto para a mensagem até um período fixo de tempo antes de abortar o processo de transmissão da mensagem e continuar com seu processamento;

Mensagem emperrada: Significa que se o destinatário da mensagem não estiver imediatamente pronto para aceitar a mensagem, o remetente aborta a mensagem e continua o processamento (FURLAN; 1998, p.195).

Figura 16: Mensagem assíncrona

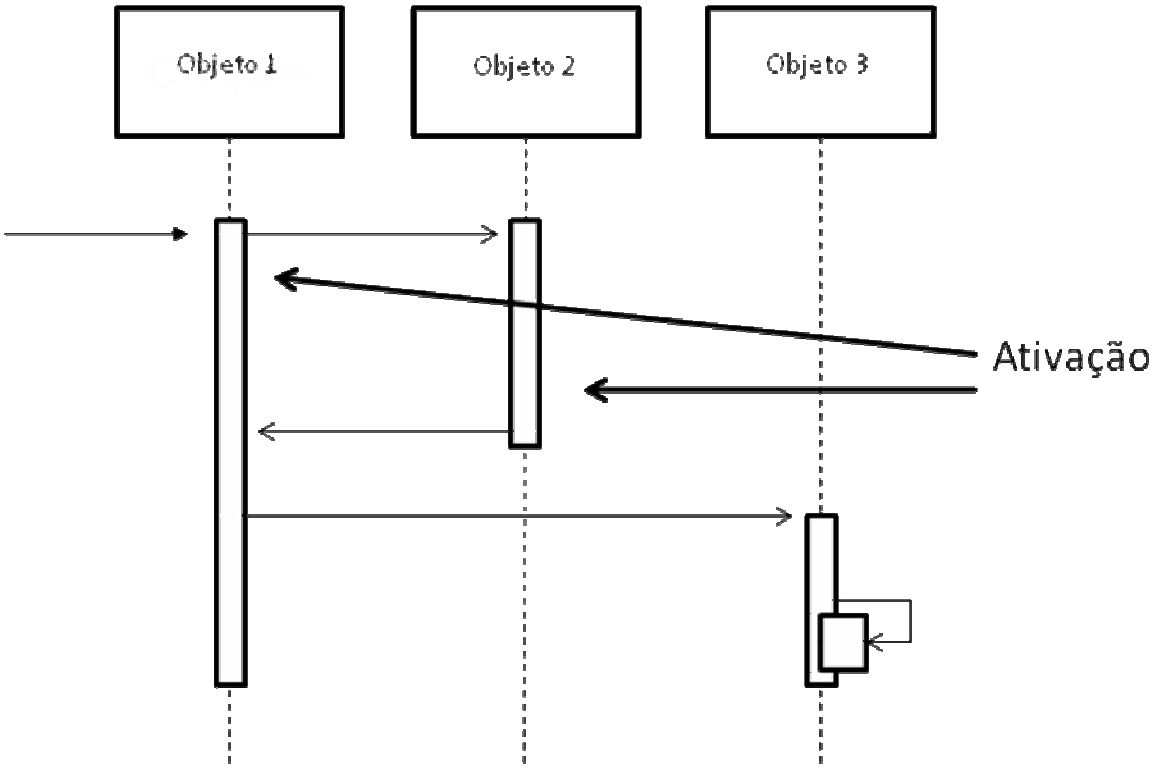


Fonte: Furlan; 1998, p.195

Ativação: Segundo Furlan (1998, p.195), define a ativação como a execução de uma ação. Uma ativação determina a janela de tempo na qual um objeto está executando diretamente uma ação ou através de um procedimento subordinado. É exibida com retângulo cujo topo é alinhado com seu tempo de iniciação e cuja parte inferior é alinha com seu tempo de conclusão.

No caso de objetos simultâneos, cada um com sua própria linha de controle, uma ativação delimita a duração que cada objeto está executando uma operação.

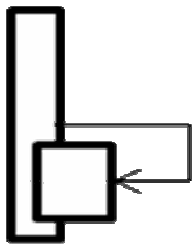
Figura 17: Ativação de objeto



Fonte: Furlan; 1998, p.196 – Adaptado pelo autor.

Autodelegação: Autodelegação ou chamada reflexiva é uma técnica utilizada em algoritmos para mostrar que uma operação chama a si própria. A mensagem autochamada é sempre síncrona e é marcada como tal no diagrama de sequência (FURLAN; 1998, p. 196)

Figura 18: Autodelegação



Fonte: Furlan; 1998, p.196

3 CONSIDERAÇÕES FINAIS

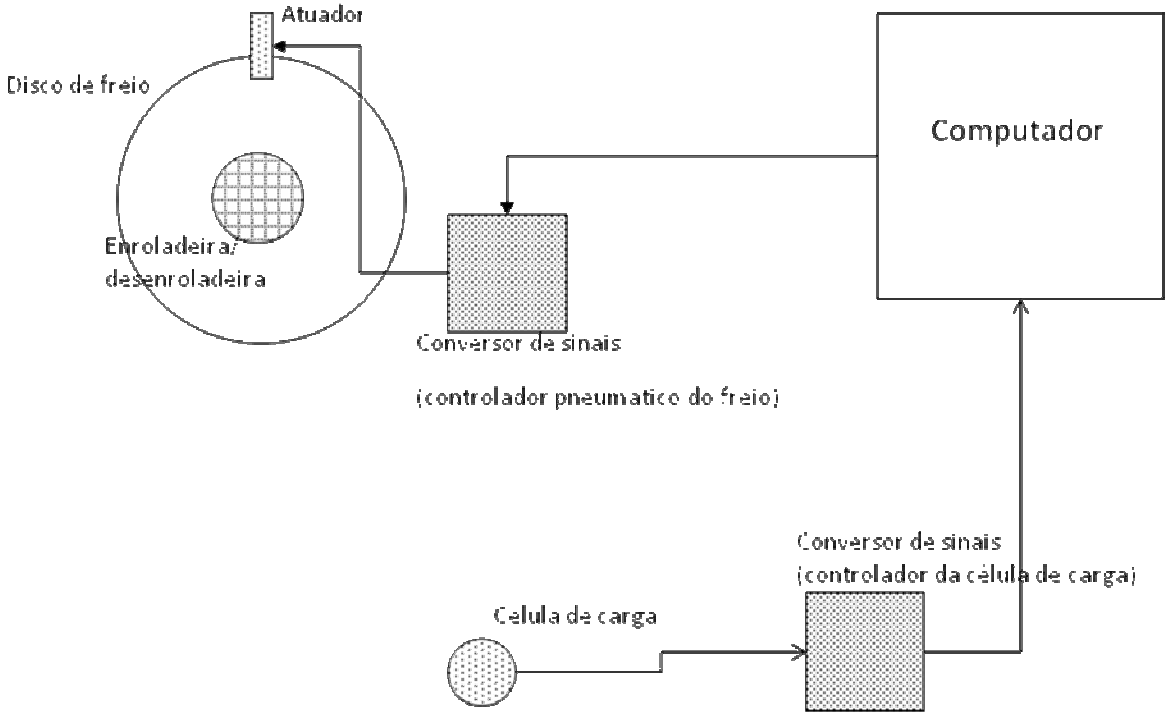
Sistemas de tempo real estão sendo cada vez mais utilizados, e as suas aplicações não se limitam apenas a máquinas ou equipamentos domésticos, mas ele pode ser utilizado para satisfazer necessidades de um sistema confiável em diversas áreas.

Para demonstrar o grande uso desse tipo de sistema, temos um exemplo básico, que é um sistema de controle de tensão de enrolamento, muito utilizado em máquinas industriais que trabalham com a fabricação de produtos que precisam ser enrolados ou desenrolados, como por exemplo, fitas adesivas, rolos de papeis, entre outras operações que podem ter o sistema adaptado.

O funcionamento pode ser descrito da seguinte maneira: O sistema tem que trabalhar com uma tensão pré-definida pelo usuário do sistema, que pode ser definida em quilogramas força. O sistema precisa de sensores, chamados de células de carga, que analisam a carga imposta no sistema e assim trabalham enviando sinais para o sistema diminuir ou aumentar a pressão do atuador nos discos de freio (o sistema trabalha em conjunto com um sistema de freio, embora não tenha um funcionamento totalmente igual a um sistema de freio de um automóvel, ele tem um funcionamento que pode ser colocado como “parecido”, pois uma das funções da célula de carga é não deixar o freio travar o sistema). O sistema tolera atrasos nas respostas do freio, mas não pode permitir que o atraso seja muito grande, portanto, seguindo as definições apresentadas no presente resumo bibliográfico, podemos classificar o sistema como *firm real time*.

Para o console do operador, teremos um gráfico que será atualizado constantemente, mostrando a força imposta no sistema naquele instante.

Figura 18: Visão geral do Sistema



Fonte: Autor (Adaptação de um sistema real de um equipamento de produção de fita adesiva)

Observando que a célula de carga analisa os dados obtidos através da passagem do material pelo sensor, nesse caso o sensor é aplicado em um cilindro, que ao girar pela passagem do material, fornece dados para o sistema da célula de carga comparar com os valores pré-estabelecidos pelo operador do sistema.

Analisando as informações acima, mesmo sendo um sistema simples, já podemos destacar a complexidade de trabalho do sistema, o que torna inviável, senão impossível, tentar controlar o equipamento com um software comum, que não utiliza as ferramentas de interação dos sistemas de tempo real.

Outro fato que podemos elencar, e a complexidade encontrada na hora de projetar e modelar tal sistema, pois sistemas de tempo real precisam ser altamente confiáveis e a prova de falhas. Por isso a necessidade de se utilizar uma ferramenta de modelagem adequada e adaptada para trabalhar com sistemas de tamanha complexidade, no caso do presente estudo, embora existam outras ferramentas, foi apresentado o UML, que possui ferramentas específicas para modelar e facilitar a programação de sistemas de tempo real.

Embora o uso da UML exija conhecimento e estudo para ser bem aplicado, e o trabalho de modelagem seja mais demorado, devido as exigências da linguagem, e para um perfeito entendimento é necessário que as pessoas que irão trabalhar no desenvolvimento do sistema também tenham conhecimento da linguagem de modelagem, é perfeitamente justificado que se use tal linguagem para modelagem de sistemas de tempo real, pois de acordo com a presente pesquisa, sistemas de tempo real são altamente complexos e a UML permite que sejam modelados com grande precisão.

4 REFERÊNCIAS BIBLIOGRÁFICAS

ALCANTARA, Otávio. Nome do Artigo. Disponível em: **http://www2.eletronica.org/artigos/eletronica-digital/sistemas-embutidos-em-tempo-real.** Acesso em: 24 mar. 2010. 20h48.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Citação**: NBR-10520/ago - 2002. Rio de Janeiro: ABNT, 2002.

_____. **Referências**: NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.

FARINE, Jean-Marie, FRAGA, Joni da Silva, OLIVEIRA, Rômulo Silva de. Sistemas de Tempo Real. 2000. Disponível em: **http://www.das.ufsc.br/~romulo/#livrospublicados.** Acesso em: 17 mar 2010. 19h52.

FURLAN, José Davi. **Modelagem de Objetos através da UML –** *the Unified Modeling Language.* São Paulo: Makron Books. 1998.

GRADY, Booch, RUMBAUGH, James e JACOBSON, Ivar. **UML:** Guia do Usuário. 2ª ed. Rio de Janeiro: Elsevier. 2005.

SOMMERVILLE, Ian, **Engenharia de Software,** tradução André Maurício de Andrade, revisão técnica Kechi Hirama. São Paulo: Pearson Addison Wesley. 2003, 6ª. Edição.