Extensão da UML para Suporte ao Projeto de RTOS

Douglas P. B. Renaux[1,2]

[1]Departamento de Eletrônica – Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Sete de Setembro 3165 – Curitiba – PR

[2] eSysTech – Embedded Systems Technologies
Travessa da Lapa 96, cj 73 – Curitiba - PR

douglasrenaux@utfpr.edu.br

Abstract. *UML is an extensible language that can be adapted to specific needs of domain areas, thus, increasing the semantic contents of its models. This paper presents a proposal of an UML extension, in the form of an UML profile, that supports the modeling of domain knowledge for real-time operating systems, typically used in the development of embedded systems.*

Resumo. *A UML foi concebida para ser uma linguagem extensível, de forma a se adaptar às necessidades específicas de diversas áreas e permitir a elaboração de modelos com maior conteúdo semântico. Neste artigo apresenta-se uma extensão à UML que suporta o processo de desenvolvimento de sistemas operacionais de tempo real, tipicamente utilizados em sistemas embarcados. A utilização desta extensão é ilustrada com exemplos envolvendo o desenvolvimento de um RTOS que está em fase de elaboração.*

1. Objetivo

A Linguagem Unificada de Modelagem (UML) [Rumbaugh 04] foi desenvolvida na década de 90 para homogeneizar as notações utilizadas pelas metodologias e ferramentas de modelagem da época. A UML tem semântica rigorosamente definida, por meio de um metamodelo, e permite a modelagem tanto dos aspectos estruturais (estáticos) como dos aspectos comportamentais (dinâmicos) de um sistema.

A proposta da UML é de uma notação padronizada que pela sua generalidade poderia ser utilizada na modelagem de qualquer tipo de sistema e em particular nos sistemas de *software*. A conseqüência óbvia desta generalidade, e do elevado nível de abstração da linguagem, é a falta de especialização em determinadas áreas de desenvolvimento e, portanto, da ausência dos conceitos específicos destas áreas. A UML trata desta questão permitindo que a linguagem seja estendida e especializada por intermédio de Perfis (*Profile*). Um Perfil para uma determinada área define um conjunto de estereótipos (*stereotypes*) especializados e respectivas propriedades e valores atribuídos (*tagged values*) associados, bem como restrições (*constraints*).

O objetivo é o desenvolvimento de um Perfil de UML para a área de Sistemas Operacionais. Trata-se de um trabalho em andamento que tem por escopo inicial os núcleos operacionais (*kernel*), também conhecidos por RTOS (*Real-Time Operating System*), utilizados em sistemas embarcados operando com restrições temporais.

Justifica-se a necessidade de um Perfil de UML para esta área para criar modelos especializados que contemplam os inúmeros conceitos específicos desta área. A presença destes conceitos especializados permitirá a construção de modelos mais ricos, i.e., com maior expressividade. Esta notação extendida não se aplica apenas aos desenvolvedores de RTOS, mas a todos aqueles que continuamente evoluem estes sistemas, desenvolvendo *drivers* para novos dispositivos, realizando manutenção e a acrescentando novos serviços ao *kernel*. Aplica-se também aos desenvolvedores de aplicações, e no estágio atual em particular aos desenvolvedores de aplicações embarcadas, que podem passar a utilizar modelos especializados para os conceitos específicos desta área.

Justifica-se, ainda, pela importância da área de desenvolvimento de *software* embarcado e pelo uso crescente de RTOS nos sistemas embarcados, como forma de reduzir a complexidade do desenvolvimento de *software* embarcado e melhorar a modularidade e compartimentalização deste.

2. Contexto

A identificação da necessidade de um Perfil para RTOS se deu durante o desenvolvimento da versão 2.0 do X Real-Time Kernel [Esystech 07]. A primeira versão deste *kernel* foi desenvolvida durante o período de incubação da empresa eSysTech na Incubadora de Inovações Tecnológicas do CEFET-PR, atual Universidade Tecnológica Federal do Paraná. O hospedeiro da incubação, o Laboratório de Inovação e Tecnologia em Sistemas Embarcados (LIT), teve forte influência no desenvolvimento do X, pelo seu histórico anterior de desenvolvimento de um *kernel* de tempo real, denominado PET[1]. O PET foi inicialmente desenvolvido para uso acadêmico, para suporte às disciplinas do CPGEI (Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial), e como plataforma de desenvolvimento utilizada pelos alunos de mestrado e doutorado. Sua robustez e confiabilidade levaram o PET ao uso industrial, em áreas como centrais telefônicas públicas e equipamentos de automação comercial e industrial. O X Real-Time Kernel[2] foi o resultado de uma evolução do PET, envolvendo a Orientação a Objetos, a Componentização e a Portabilidade entre Plataformas.

3. Mecanismos de extensão da UML

Um Perfil de UML (*UML Profile*) é uma variante de UML especializada em alguma área. Dentre os perfis adotados pela OMG (*Object Management Group*) [OMG 08] estão o SysML (Systems Modeling Language), para especificação, análise, projeto e verificação de sistemas complexos; o *UML Profile for Schedulability, Performance and Time* (RT-UML) para modelagem dos aspectos de tempo, escalonamento e desempenho em sistemas em tempo real; e vários outros como o MARTE (*Modeling and Analysis of Real-time and Embedded Systems* – em desenvolvimento) e o *UML Profile for Systems on a Chip*. Um Perfil estende o metamodelo da UML [OMG 06] especializando-o para uma área. Além dos perfis adotados pela OMG, vários outros perfis são desenvolvidos

[1] Software registrado no INPI (Instituto Nacional de Propriedade Industrial).

[2] Software registrado no INPI (Instituto Nacional de Propriedade Industrial).

para atender à demandas específicas, a exemplo da área de Programação Orientada a Aspectos [Aldawud 01].

O objetivo dos perfis de UML é de oferecer uma linguagem customizada para certos domínios sem alterar a semântica da UML. Desta forma, um perfil é simplesmente um conjunto coerente de elementos especializados do metamodelo (estereótipos) com valores atribuídos e regras/restrições. Os estereótipos são usados para representar conceitos específicos de um domínio de forma simples e leve.

Um estereótipo tipifica uma classe ou objeto. Ao fazê-lo, define um conjunto de propriedades que devem ser especificadas para uma classe ou objeto em particular, utilizando valores atribuídos, bem como um conjunto de regras, ou restrições, que devem ser seguidos. Um estereótipo pode ser especificado textualmente, pela notação <<NomeEstereótipo>> ou através de um símbolo gráfico ou ícone, o que em muitas situações facilita a leitura do diagrama.

Valores atribuídos (*tagged values*) são apresentados na forma { propriedade = valor }. Cada estereótipo define um conjunto de propriedades que devem ser preenchidas pelas instâncias (classes e objetos) daquele estereótipo.

Já uma restrição é uma regra, de determinado domínio, que é aplicada sobre elementos / conceitos daquele domínio.

O Perfil RT-UML define várias dezenas de estereótipos e suas respectivas propriedades e restrições. Dentre os estereótipos definidos encontramos: <<RTAction>>, <<RTclkInterrupt>>, <<RTdelay>>, <<RTevent>>, <<RTnewTimer>> e <<RTreset>>. É interessante notar que os dois últimos estereótipos se aplicam a operações e não a classes.

Outra característica da UML, que a nosso ver é bastante poderosa na modelagem mas pouco utilizada pelos desenvolvedores, é a possibilidade de acrescentar compartimentos a uma classe, além dos três compartimentos padrões que contém nome, atributos e operações. Estes compartimentos podem ter um nome associado (*named compartment*) para facilitar o entendimento.

4. Extensão Proposta

A tabela a seguir apresenta os estereótipos propostos neste trabalho, de acordo com o modelo apresentado em [Douglass 04]

Estereótipo	Aplicável à	Propriedades	Descrição
<<thread>>	Função ou operação	Period WCExecutionTime	Esta função/operação é a rotina principal (root) de uma thread. As características da thread estão descritas como propriedades.
<<monitor>>	Classe ou Módulo de SW.		A implementação desta classe / módulo garante que as operações/funções associadas são mutuamente exclusivas .
<<function>>	Classificador		Usado principalmente na notação detalhada (ver próxima seção) para identificar que este classificador se refere a uma operação de uma classe ou a uma função.

<<attribute>>	Classificador		Usado principalmente na notação detalhada para identificar que este classificador se refere a um atributo de uma classe.
<<ISR>>	Operações Funções	IrqPriorityLevel IrqNesting	A função, ou operação (método) de uma classe, é utilizada como rotina de atendimento à interrupção. A propriedade IrqPriorityLevel tem um valor numérico correspondente ao nível de prioridade daquela interrupção. A propriedade IrqNesting é falsa se outras interrupções estiverem bloqueadas durante a execução desta ISR, e verdadeira se o aninhamento de interrupções é permitido.
<<IST>>	RootFunction de uma Thread RootOperation de uma classe ativa	ThPriorityLevel	A função, ou operação de uma classe ativa, é utilizada como função principal (RootFunction) de uma Thread. Esta thread tem por papel ser a Interrupt-Service-Thread para alguma interrupção. A propriedade ThPriorityLevel tem valor numérico correspondente ao nível de prioridade daquela tarefa no nível base.
<<irq>>	Relacionamento	irqId	Este relacionamento representa o sinal elétrico correspondente ao pedido de interrupção. A propriedade irqId é o identificador desta interrupção.
<<synch_msg>>	Relacionamento	MaxSize	O relacionamento entre estas classes se dá através de mensagens síncronas, implementadas por mecanismos providos pelo RTOS ou pelo *run-time system*. A liberação da tarefa transmissora da mensagem só ocorre quando a tarefa receptora tiver respondido a mensagem.
<<assynch_msg>>	Relacionamento		O relacionamento entre estas classes se dá por meio do envio de mensagens assíncronas. A tarefa transmissora não suspende sua execução devido ao envio da mensagem. A mensagem fica armazenada num buffer até que a tarefa receptora esteja apta a consumi-la.
<<function_call>>	Relacionamento		O relacionamento entre estas classes se dá por meio de chamada de operação ou função. É, portanto, síncrono, i.e., o chamador suspende enquanto o chamado executa. A thread não é suspensa, apenas a função/operação chamadora.
<<atomic>>	Função; Acesso à variável		A implementação da função ou do acesso à variável garante a sua atomicidade, dispensando o chamador de proteger a chamado/acesso de interrupções ou de

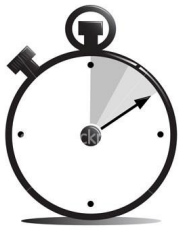
			alterações externas ao fluxo do programa.
<<r>>	Relacionamento		Acesso de leitura à variável.
<<w>>	Relacionamento		Acesso de escrita à variável.
<<rw>>	Relacionamento		Acesso de leitura e escrita à variável.
<<ar>>	Relacionamento		Acesso atômico de leitura à variável.
<<aw>>	Relacionamento		Acesso atômico de escrita à variável.
<<imarw>>	Relacionamento		Acesso atômico de escrita/leitura à variável. A atomicidade é garantida por mascaramento de interrupção.
<<nparw>>	Relacionamento		Acesso atômico de escrita/leitura à variável. Atomicidade é garantida por bloqueio de preempção.
<<msg_queue>>	Classe	Capacity	Esta classe implementa um canal de comunicação unidirecional entre duas threads. Através deste canal a thread transmissora pode enviar pacotes de informação (mensagens) para a thread receptora. A msg_queue tem uma capacidade de armazenamento de mensagens que é configurável.
<<timer>>	Classificador	isPeriodic Duration	Representa uma abstração de um serviço do kernel responsável pelo envio de mensagens assíncronas temporizadas ou pela mudança de estado de uma tarefa (de Suspenso para Pronto). A propriedade lógica isPeriodic é verdadeira quando este temporizador for periódico e é falsa quando este temporizador for *single-shot*. A propriedade Duration representa o duração da temporização ou o seu período.

Algumas propriedades adicionais mais comuns são:

Propriedade	Aplicável à	Descrição
mutually exclusive	Operações Funções	As operações / funções envolvidas na restrição são mutuamente exclusivas.
preemptable	Thread Função Operação	Esta thread / função / operação tolera preempção.
non-preemptable	Thread Função Operação	Esta thread / função / operação não tolera preempção.

A extensão proposta também define ícones associados a alguns dos estereótipos. São eles:

- Ícone para temporizador (timer):



- Ícone para Message Queue:

Além de utilizar o já conhecido ícone para hardware:

5. Exemplos de Uso

A figura a seguir apresenta uma pequena fração do diagrama de objetos referente à Fila de Temporizadores. Pode-se notar que o objeto tl da classe CTimerList tem os atributos head_time e head, embora os mesmos não estejam listados na posição habitual, em um dos compartimentos da representação gráfica do objeto. Da mesma forma, o objeto tl tem uma operação denominada Insert, cujo parâmetro é do tipo CTimer. A notação denominamos detalhada pois permite que se represente nos diagrama as propriedade e relacionamentos de cada atributo ou operação de classe ou objeto.

Neste diagrama, observa-se que o atributo head_time é atômico, indicando que o mesmo será acessado de forma indivisível (ou seja, a implementação garante que a seqüência de instruções que acessam este atributo não pode ser interrompida). Da mesma forma, a execução da operação Insert( ) também não pode ser interrompida, seja por preempção, interrupção, ou outro motivo.

O atributo head aponta para o objeto tim1. Este, por sua vez, tem um atributo next que aponta para tim2. Como o objeto tim2 é da mesma classe que tim1, não é mais necessário repetir a notação detalhada, portanto, em tim2, o atributo next aparece na sua posição habitual.

Na Figura 2 apresenta-se a recepção de caracteres por uma porta serial. A UART é o hardware, estereótipo representado pelo seu ícone, que gera pedidos de interrupção (<<irq>>). Estes são tratados pela operação UART_isr( ) da classe Serial. Esta operação está estereotipada como <<ISR>>; durante sua execução faz acessos de leitura e escrita ao atributo in_buffer_next_in. A atomicidade destes acessos está garantida pelo mascaramento das interrupções durante a execução da UART_isr( ).

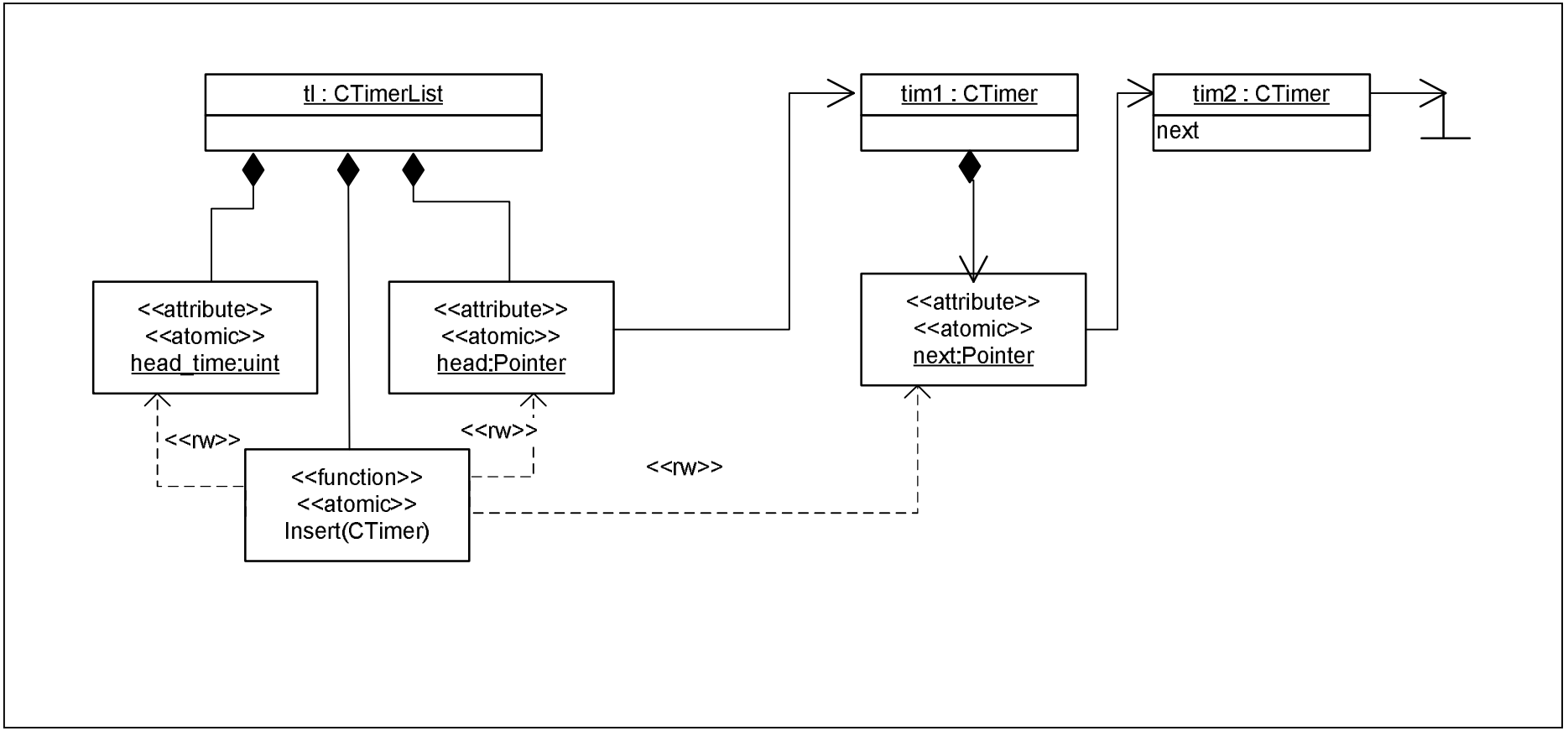


Figura 1 - Diagrama de Objetos de uma Fila de Temporizadores

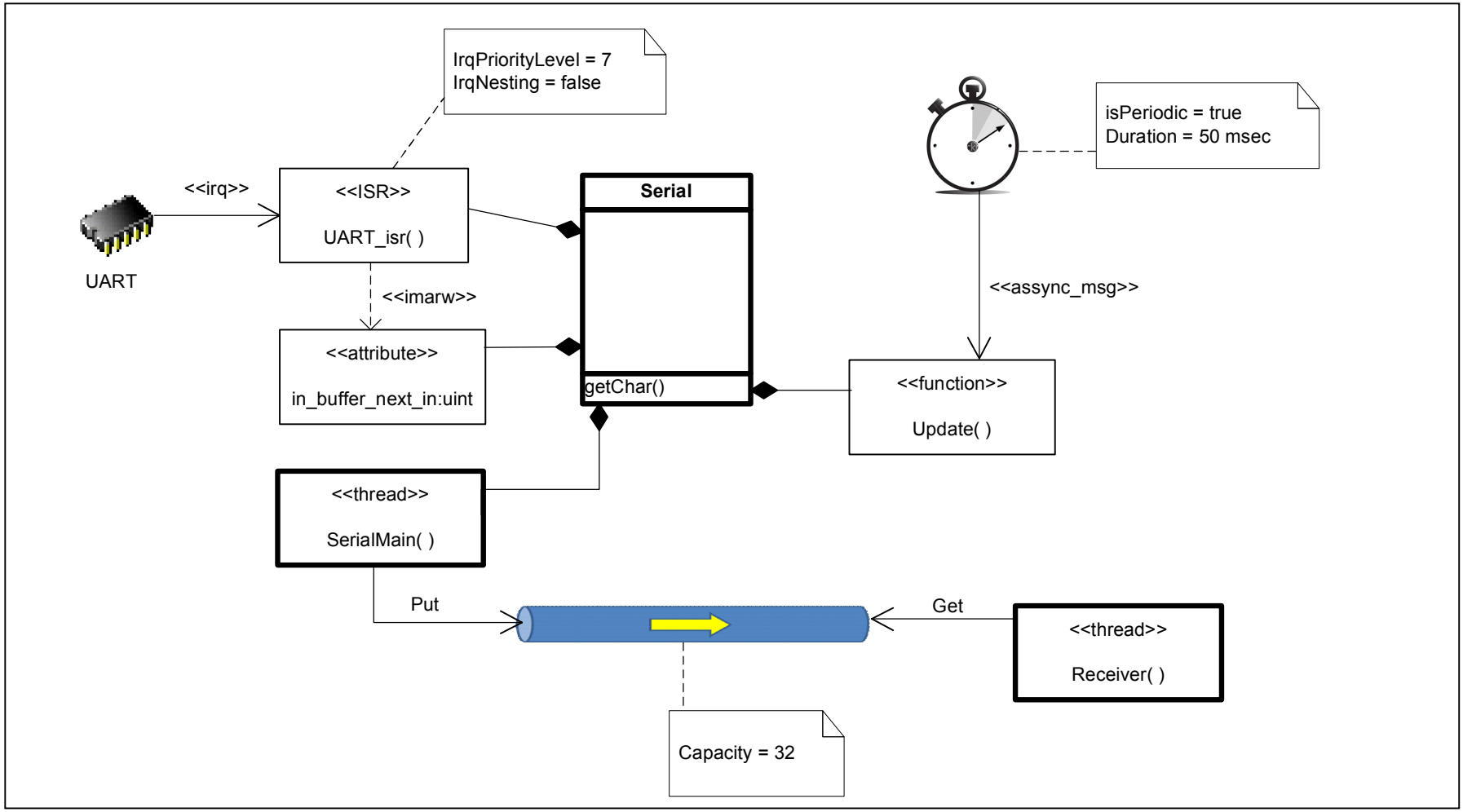


Figura 2 –Interação entre tarefas

A cada 50 milisegundos uma mensagem assíncrona ativa a operação Update( ) da classe Serial. A classe Serial é uma classe ativa e, portanto, tem uma thread associada, cuja função principal é SerialMain( ), uma operação da classe Serial. Esta função envia os bytes recebidos através de uma MessageQueue, representada pelo ícone do tubo. A direção do fluxo das mensagens pela MessageQueue é dada pela seta no tubo. A thread SerialMain deposita mensagens na MessageQueue por meio da associação Put enquanto a thread Receiver as recebe pela associação Get.

6. Considerações Finais

A extensão proposta, na forma de um Perfil de UML, embora ainda não concluída, tem se mostrado de grande utilidade no desenvolvimento de um núcleo operacional de tempo real, tanto no desenvolvimento do núcleo propriamente dito, como no desenvolvimento de *device drivers* e de aplicações embarcadas. O uso deste Perfil em diferentes projetos, irá definir necessidade adicionais, que poderão ser acrescentadas na forma de novos estereótipos, propriedade e restrições.

Referências

[Aldawud 01] Aldawud, Omar, Elrad, Tzilla and Bader, Atef, "A *UML* Profile for Aspect Oriented Modeling", *OOPSLA 2001 - Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa Bay, Florida - Outubro, 2001.

[Douglass 04] Douglass, Bruce Powel (2004) “Real-Time UML: Advances in the UML for Real-Time Systems”, Addison-Wesley, USA.

[Esystech 07] eSysTech “Manual do Usuário: X Real-Time Kernel”, Agosto 2007.

[OMG 06] Meta Object Facility Core Specification version 2.0, OMG, January 2006.

[OMG 08] Object Management Group. www.omg.org. Acessado em Fev 2008.

[Rumbaugh 04] Rumbaugh, James; Jacobson, Ivar; Booch, Grady (2004) “The Unified Modeling Language Reference Manual”. Addison-Wesley.