Desenvolvimento de um Sistema Operacional Orientado a Objetos para uso em Sistemas Embarcados

Rafael de Góes[1], Douglas Renaux[2]

[1]eSysTech – Embedded Systems Technologies
Trav. da Lapa 96, cj. 73 – Curitiba – PR - Brasil

[2]Laboratório de Inovação e Tecnologia em Sistemas Embarcados – LIT
Universidade Tecnológica Federal do Paraná - UTFPR
Curitiba – PR - Brasil

rafael@esystech.com.br, douglas@lit.citec.cefetpr.br

Abstract. *This paper describes the development of an object-oriented real-time kernel for embedded systems from its principles to the results obtained on the first version. Preliminary results on the second version of the kernel are also presented.*

Resumo. *O artigo descreve o desenvolvimento de um kernel para sistemas embarcados que foi projetado utilizando as técnicas de orientação a objetos. São apresentados os motivadores para esta decisão bem como os resultados relativos a primeira versão do kernel e resultados preliminares da sua segunda versão.*

1. Sistemas Operacionais Orientados a Objetos

O termo Sistema Operacional Orientado a Objetos (SO^3) é encontrado na literatura com significados variados. A definição mais simplista (Cunningham and Cunningham 2004) é que o SO^3 tenha sido projetado seguindo as técnicas de orientação a objetos e implementado utilizando uma linguagem orientada a objetos, tal como C++. Já o projeto Renaissance (Russo 1996) tem por meta prover uma base onde aplicações, geradas por diferentes linguagens de programação, possam compartilhar objetos que estejam distribuídos numa rede de máquinas heterogêneas. Cabe ao SO^3 prover os meios de acesso entre estes objetos, permitindo o desenvolvimento de sistemas modulares com baixo acoplamento entre módulos. O Projeto PURE (Beuche 1999) utiliza a orientação a objetos para gerar uma família de sistemas operacionais adaptados às plataformas de hardware específicas, garantindo um elevado grau de eficiência no uso dos recursos, tão limitados nos *deeply embedded systems*. O Projeto EPOS (Frölich 1999) seleciona os módulos adequados do PURE para gerar sistemas operacionais orientados a objeto que sejam direcionados a aplicações específicas.

1.1 Justificativa para o uso de Sistemas Operacionais Orientados a Objetos

A área de Sistemas Operacionais, considerada madura, já possui padrões como POSIX e através do uso da programação estruturada e de linguagens procedimentais, como C, tem-se obtido elevados graus de eficiência computacional e de portabilidade entre

diferentes arquiteturas. Pode-se então questionar o uso da Orientação a Objetos em Sistemas Operacionais.

Sistemas operacionais de uso genérico, monolíticos e com vasta funcionalidade não são apropriados para uso em sistemas embarcados que tipicamente possuem recursos computacionais bem limitados. A tendência nestes casos é a utilização de sistemas operacionais adaptados à arquitetura, à plataforma de hardware e ao domínio da aplicação, para prover apenas a funcionalidade requerida e a máxima eficiência no uso de recursos. A orientação a objetos provê a adaptabilidade necessária, por meio da herança e do polimorfismo, bem como a modularidade e o encapsulamento, incentivando a coesão e reduzindo o acoplamento.

Becker e Pereira (2000) justificam a necessidade de especializar o *kernel* para determinada aplicação visando a redução das necessidades de recursos computacionais. Esta especialização é significativamente facilitada pelo uso da Orientação a Objetos.

1.2 Definição de Termos

Alguns termos, até por razões históricas, têm sido utilizados com significados distintos. Nesta seção, define-se o significado dos termos como foram utilizados no contexto deste trabalho. Estas definições estão em sintonia com Tanenbaum (2000), Ganssle (2003) e Wikipedia (2006).

Processo representa um programa em execução. No contexto dos sistemas computacionais de uso geral atuais, um processo está associado a (1) um espaço de memória virtual protegido do espaço de memória dos demais processos; (2) uma ou mais linhas de execução (*execution threads*) associadas a regiões de pilha, estado do processador (registradores, ...) e contador de programa; (3) descritor de processo incluindo descritores dos recursos (arquivos, canais de comunicação, ...) em uso pelo processo. Neste contexto, o termo tarefa (*task*) é tipicamente utilizado como sinônimo de processo.

Enquanto os S.O. de uso geral precisam prover mecanismos de suporte a multi-usuários, segurança, proteção, e mecanismos de execução concorrente de programas (processos) que competem entre si pelo uso de recursos computacionais, os sistemas computacionais embarcados limitam-se a prover mecanismos de execução concorrente de atividades cooperativas, ou seja, visando objetivos comuns. Sistemas Embarcados, geralmente, executam sobre plataforma de hardware com recursos escassos e tipicamente não dispõem dos recursos para implementação dos mecanismos de memória virtual. As atividades concorrentes de um Sistema Embarcado são, portanto, executadas por *threads*, neste contexto chamadas de tarefas. Evitaremos o uso do termo tarefa para evitar confusão com seu uso. O foco deste artigo é em um *kernel* para Sistemas Embarcados, sem uso de memória virtual, onde as atividades são executadas por *threads*.

Um RTOS (*Real-Time Operating System*) é o *kernel* de um S.O. que provê mecanismos para a execução concorrente de *threads* que compartilham o mesmo espaço de endereçamento (área de código, constantes, variáveis globais e memória de alocação dinâmica), tendo cada *thread* a sua própria pilha, estado do processador (incluindo registradores) e contador de programa. Um RTOS escalona as *threads* e provê mecanismos de comunicação, sincronização, temporização e exclusão mútua. Para dar

suporte aos sistemas embarcados operando em tempo-real, um RTOS deve ter tempos de execução determinísticos e definidos dos seus serviços além de permitir que o desenvolvedor configure o escalonamento do sistema.

Embora a maioria dos sistemas operacionais tenham sua origem como monolíticos, a estrutura adotada atualmente é a chamada *micro-kernel*, onde um módulo denominado núcleo (*kernel* ou núcleo operacional) implementa apenas a funcionalidade essencial e vários outros módulos com funcionalidades adicionais são agregados a este. Os sistemas embarcados, que tipicamente possuem restrições severas de capacidade computacional e de armazenamento, se beneficiam muito desta estrutura por utilizarem o núcleo e apenas os módulos adicionais necessários para cada caso. Em Oliveira (2003) as principais funcionalidades de um núcleo operacional são descritas, bem como, considerações sobre escalonamento de tarefas e restrições temporais.

Seguindo o conceito de *micro-kernel*, também são propostos os *nano-kernel*, o *pico-kernel* e finalmente o *exokernel*, com funcionalidade cada vez mais reduzida, ao ponto em que o *exokernel* se limita a realizar a multiplexação do hardware. Um exemplo de *exokernel* é o MIT *Exokernel Operating System* (Engler, Frans Kaashoek, and O'Toole 1995).

1.3 Requisitos de um Sistema Operacional Orientado a Objetos

No contexto deste trabalho um SO[3] deve:

- ter sido concebido, projetado e implementado de acordo com os princípios de orientação a objetos, tendo, portanto, um elevado grau de modularidade, de adaptabilidade, através de mecanismos como herança e polimorfismo;
- ter um projeto bem estruturado que permita a sua evolução contínua;
- permitir, de forma rápida e simples, a criação de variantes que se adaptem às diferentes plataformas computacionais, arquiteturas e domínios de aplicações;
- permitir que seja composto a partir de componentes disponíveis com funcionalidades bem definidas de forma a prover apenas as funcionalidades necessárias em cada aplicação;
- possuir interfaces genéricas em seus módulos de forma a poder se adaptar aos diferentes módulos de hardware de cada plataforma física;
- ser eficiente no uso dos recursos computacionais, incluindo desempenho elevado e o uso reduzido de memória.

2. Objeto – A Família de Núcleos Operacionais X (*X Real-Time Kernel*)

A partir dos requisitos listados anteriormente, o LIT[1] está desenvolvendo em parceria com a eSysTech – Embedded Systems Technologies, uma empresa de base tecnológica

[1] Laboratório de Inovação e Tecnologia em Sistemas Embarcados da UTFPR (ex-CEFET-PR)

incubada na IINCEFET[2], uma família de núcleos operacionais orientados a objeto. O primeiro membro desta família é o XL-ARM7TDMI v1.0, disponível comercialmente desde 2004. A segunda versão deste *kernel* está em desenvolvimento até Setembro 2006.

2.1 Estrutura

O *X Real-Time Kernel*, na sua primeira versão, é formado por diversos módulos com funcionalidades bem definidas. A estrutura adotada tem as seguintes características:

- Cada módulo foi concebido levando em conta o princípio de coesão elevada e baixo acoplamento. A interface de cada módulo é genérica permitindo que inúmeras implementações de cada módulo, adaptadas às diferentes arquiteturas, periféricos e necessidades específicas dos domínios de aplicação, possam ser utilizadas. O uso extensivo de herança e polimorfismo suportaram esta decisão. A biblioteca de *device drivers* (DDLX), apresentada na Seção 2.3, ilustra este ponto.
- Esta estrutura modular simplifica significativamente a portabilidade do *kernel* por restringir as alterações a módulos específicos.

Os módulos que compõem o *kernel* XL-ARM7TDMI estão listados na Tabela 1:

Tabela 1- Módulos do *kernel* XL-ARM7TDMI v1.0

Módulo	Descrição
xl	*Micro-kernel*: implementa a gestão de tarefas, comunicação e sincronização, e serviços de tempo.
shell	Um *shell* que permite acesso do usuário às informações sobre cada tarefa a partir de um terminal ASCII. Este *shell* permite que o usuário registre funções que podem ser ativadas do terminal.
trace	Implementa a funcionalidade de trace para a monitoração de eventos do *kernel* e/ou da aplicação.
message queue	Implementa a filas de mensagens.
semaphore	Implementa diversos tipos de semáforos.
arm7tdmi	Módulo com as rotinas dependentes da plataforma.
DDLX	*Device driver library for X*: coleção de módulos que implementam as rotinas de acesso aos diferentes periféricos disponíveis.
HDLC	Implementa os protocolos HDLC e SDLC
USB	Implementa uma pilha USB (*host* e *device*)
TCP/IP	Implementa uma pilha TCP/IP

[2] Incubadora de Inovações Tecnológicas do CEFET-PR

2.2 Funcionalidade

O *kernel* X disponibiliza seus serviços para a aplicação através dos métodos de um objeto global denominado *os*. Os principais serviços disponíveis estão apresentados na Tabela 2:

Tabela 2 - Serviços do *kernel*

Classif.	Método	Descrição
Thread Management and Scheduling	**CreateThread**	Criação de uma tarefa
	KillThread	Término de uma tarefa
	GetTId	Consulta ao identificador de uma tarefa
	Yield	Liberação do processador para outra tarefa
	SuspendThread	Mudança de estado de uma tarefa para suspenso
	ResumeThread	Mudança de estado de uma tarefa suspensa para pronto
Time Services	**GetTime**	Leitura do relógio do kernel
	MsgAt	Solicitação de envio de mensagem em hora programada
	PeriodicMsg	Solicitação de envio de mensagem periódica
	ClearMsg	Cancelamento de solicitação de envio de mensagem
	SleepFor	Suspensão temporário da tarefa
Serviços de Comunicação Síncrona e Assíncrona	**Send**	Envio (síncrono) de mensagem
	Receive	Recepção de mensagens síncronas e assíncronas
	Reply	Resposta à mensagem síncrona
	Put	Envio (assíncrono) de mensagem
	CheckForMsg	Verificação de recebimento de mensagem
Serviços de Interrupção	**RegisterISR**	Cadastra uma rotina como *handler* de interrupção
	UnregisterISR	Cancela o cadastro de *handler* de interrupção
	MaskHWInt	Mascara um pedido de interrupção
	UnmaskHWInt	Desmascara (libera) um pedido de interrupção
	ConfigHWInt	Configura um pedido de interrupção
	InterruptLock	Bloqueia as interrupções
	InterruptUnlock	Desbloqueia as interrupções

2.3 DDLX – A Biblioteca de Device Drivers

Um dos aspectos interessantes do *kernel* X é a estruturação da biblioteca de *device drivers*, que ilustra os principais conceitos da estruturação do *kernel* (Figura 1). Nesta figura observa-se, na raiz da estrutura, a interface padronizada a ser adotada por todos os tipos de *device drivers* que compõem a DDLX. Esta estrutura vai sendo paulatinamente refinada a cada nível, formado por grupos cada vez mais especializados. Tipicamente, o *kernel*, as aplicações, os módulos de *middleware* e até os demais *device drivers* acessam a DDLX apenas através das interfaces definidas nos níveis mais altos da estrutura, conseqüentemente, alterações nos níveis inferiores da estrutura, em função de adaptações a novas plataformas, não são percebidas fora da DDLX.

Embora os sistemas operacionais convencionais (estruturados) também tenham obtido um algum grau de portabilidade, a implementação típica nestes casos é através de inúmeros ponteiros e *strings* de configuração, dificultando significativamente a manutenção e a evolução do código. Este é, portanto, um exemplo onde as técnicas de orientação a objetos demonstram sua superioridade.

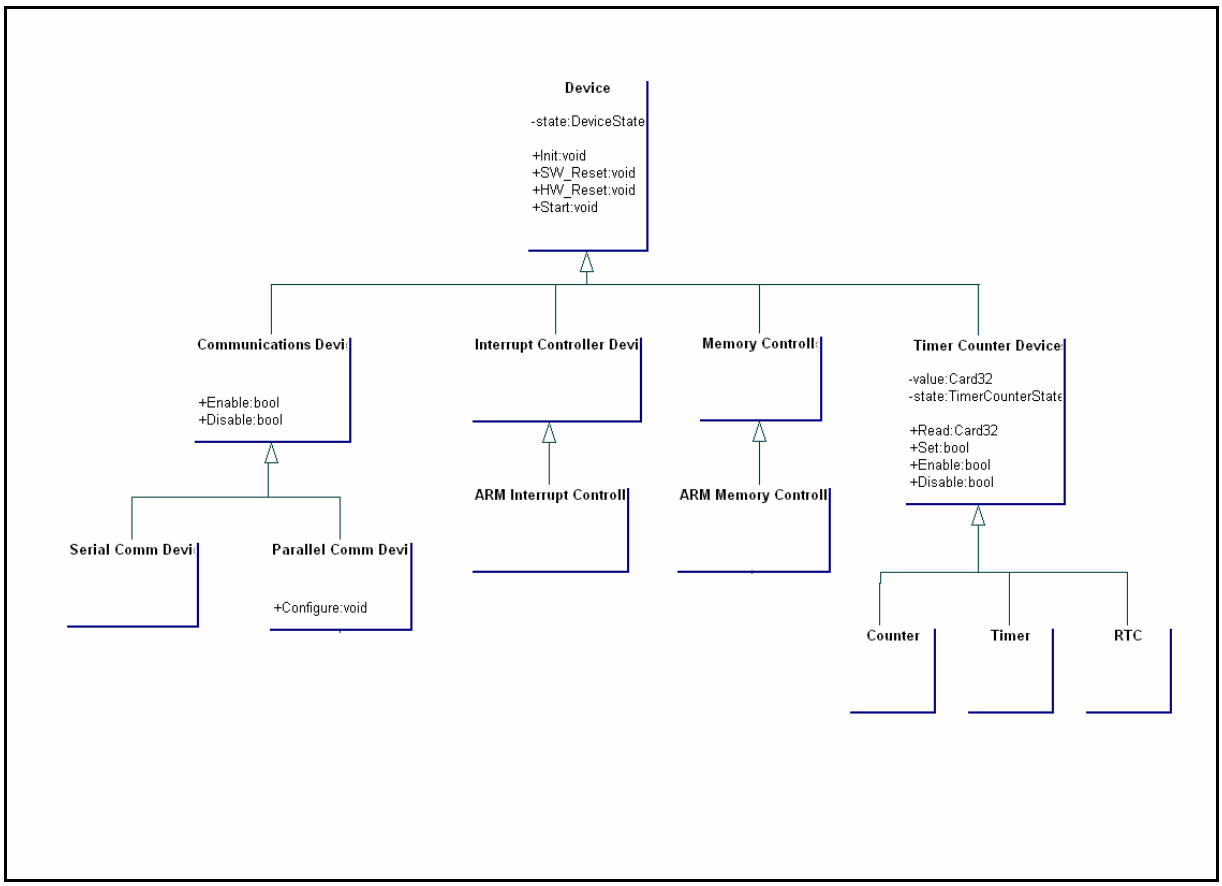

Figura 1 - Estrutura da DDLX (vista parcial)

3. Resultados (XL-ARM7TDMI v1.0)

O primeiro membro da família do *kernel* X é o XL-ARM7TDMI que foi desenvolvido para a arquitetura ARM7TDMI. É uma arquitetura RISC de 32-bits bastante utilizada em sistemas embarcados. Seu desempenho está na faixa de 20 a 100 MIPS.

No projeto e implementação do XL-ARM7TDMI tomou-se todo o cuidado para reduzir o uso dos recursos computacionais. Tal cuidado pode ser comprovado através da Tabela 3 que apresenta o tamanho do código de cada módulo. Do ponto de vista de desempenho, foi realizada uma comparação do tempo de chaveamento de contexto do XL-ARM7TDMI com um valor típico (KADAK 2003) – ver Tabela 4. Ambos executando em um ARM7TDMI. A Tabela 5 apresenta exemplos de tempos de execução de serviços do XL-ARM7TDMI.

Tabela 3 - Tamanho do código do XL-ARM7TDMI

Módulo	Tamanho do código (bytes)
xl	*8492 (incluindo semaphore e message queues)*
arm7tdmi	2544
shell	6340
trace	1280

Tabela 4 - Comparação de tempos de chaveamento de contexto para execução em um ARM7TDMI.

kernel	Tempo de chaveamento de contexto (@clock)	Tempo normalizado para 66 MHz
XL-ARM7TDMI	5.5 us @ 66 MHz	5.5 us
AMX	38 us @ 20 MHz	11.5 us

Tabela 5 – Tempos de execução de serviços do XL-ARM7TDMI (@ 66 MHz)

Módulo	Tempo de execução (us)
ClearMsg	2.6
CreateThread	19.1
GetMyTId	0.48
GetTId	17.1
GetTime	1.9
KillThread	2.1
PeriodicMsg	4.6
Put	2.5
Receive	2.5

4. Desenvolvimento do XL-ARM7TDMI v2.0

A versão 2.0 do *kernel* XL-ARM7TDMI acrescenta as seguintes características à versão anterior:

- escalonamento preemptivo;
- suporte a outras versões da arquitetura ARM: ARM9, ARM11 e XSCALE;
- componentização: cada funcionalidade do X será encapsulada em um componente individual. A biblioteca, formada por estes componentes de granularidade muito fina, permite selecionar apenas os componentes

necessários em uma determinada aplicação, reduzindo ainda mais a necessidade de memória.

- *Shell* gráfico: permitirá maior eficiência no uso de recursos no sistema embarcado e melhor ergonomia para o usuário.
- *Trace* gráfico: também permitirá maior eficiência no uso dos recursos do sistema embarcado, oferecerá ao usuário uma interface mais ergonômica e com mais funcionalidades, facilitando o procedimento de depuração e monitoração da operação do sistema embarcado.
- Integração à ferramenta de desenvolvimento, permitindo o acesso ao estado do *kernel* através do ambiente de depuração.

A estrutura da versão 2, ilustrada na Figura 2, é uma evolução da estrutura da v 1.0.

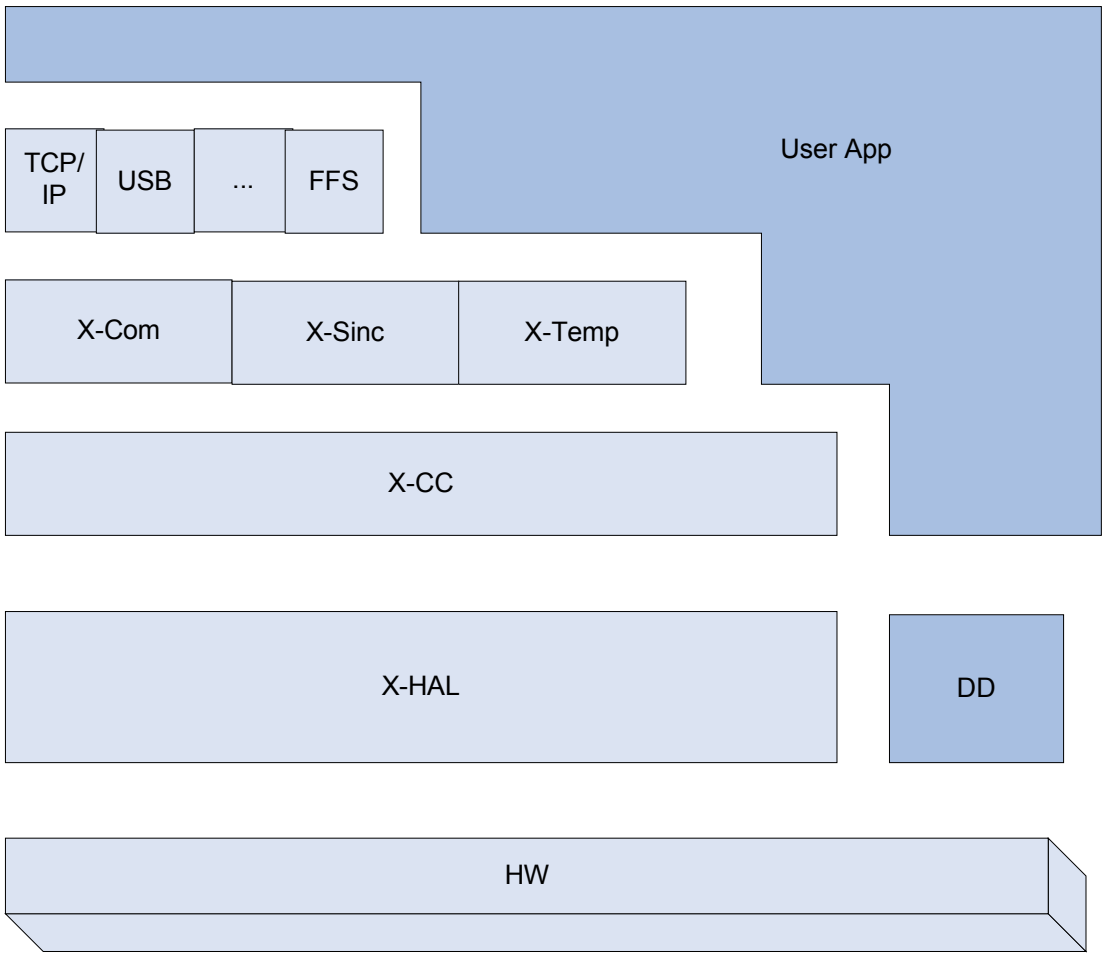

Figura 2 - Estrutura do XL-ARM7TDMI v2.0

A primeira camada sobre o hardware é formada pelo HAL – *Hardware Abstraction Layer*, responsável pelo acesso ao hardware e por isolar as diferenças de periféricos de origens diversas através da implementação de *device drivers* com interfaces padronizadas. O X-HAL é uma evolução da DDLX utilizada na versão 1. *Device drivers*

não contemplados no X-HAL são desenvolvidos pelo usuário do X (bloco DD). A camada seguinte é dos componentes de software do X responsáveis pelo chaveamento de contexto (X-CC). Esta camada é dependente da arquitetura do processador. A terceira camada é composta pelos componentes responsáveis pela comunicação (X-Com), sincronização (X-Sinc) e temporização (X-Temp) dos *threads*. A camada seguinte é formada por módulos de software opcionais, em particular por protocolos de comunicação. A aplicação do usuário é formada por diversas *threads* que podem acessar os serviços disponibilizados em qualquer das camadas.

Esta estrutura tem por metas (1) facilitar a portabilidade para diferentes processadores e (2) fazer uso extensivo dos benefícios decorrentes da componentização fina.

A camada X-HAL é que concentra as alterações necessárias para o porte entre diferentes processadores de uma mesma arquitetura. Já a camada X-CC encapsula os detalhes de cada arquitetura.

A camada X-HAL é formada por um número significativo de componentes de software, cada um especializado em um determinado periférico. Em muitos casos, um determinado periférico físico está associado a diversos componentes do X-HAL, cada um tratando de um diferente modo de operação do periférico ou de diferentes aspectos da sua funcionalidade. Um exemplo seria o temporizador utilizado nos processadores ARM produzidos pela Atmel. Quatro componentes de software podem ser associados a um temporizador: *single-shot* (gera uma interrupção ao final do tempo estabelecido), *periodic* (gera interrupções periódicas), PWM (gera forma de onda PWM com *duty-cycle* e freqüência programáveis), *frequency-counter* (mede *duty-cycle* e freqüencia de um sinal TTL).

A seleção de quais componentes do *kernel* devem fazer parte de uma determinada aplicação é realizada de forma automática pelo ambiente de desenvolvimento, durante o processo de compilação e ligação, com apoio de ferramentas específicas.

Os resultados preliminares do desenvolvimento da versão 2.0 indicam que o tamanho dos componentes varia de 8 a 240 bytes. Mais de 100 componentes já foram desenvolvidos e estima-se cerca de 300 componentes quando a implementação da versão 2.0 estiver completa. Os valores de tempo de execução são próximos dos valores da versão 1.0, apresentados na Seção 3.

5. Conclusão

O *kernel* XL-ARM7TDMI foi desenvolvido como parte de uma família de núcleos operacionais fundamentada na orientação a objetos, no princípio da adaptação do *kernel* à plataforma e à aplicação, e com o firme propósito de demonstrar que este paradigma permite a implementação de núcleos com baixo uso de recursos computacionais e facilmente adaptáveis às mais diversas necessidades dos seus usuários.

Uma das características da versão 2.0, em desenvolvimento, é a componentização fina. Desta forma, o *kernel* XL-ARM7TDMI atende aos requisitos das plataformas com baixa disponibilidade de RAM de Flash, como é o caso de inúmeros lançamentos recentes de microcontroladores de 32-bits, sem barramento externo de memória, e com reduzida capacidade de memória interna. As aplicações desenvolvidas para estes microcontroladores não requerem toda a funcionalidade disponível no XL-ARM7TDMI.

A componentização fina permite adequar a funcionalidade do *kernel* estritamente às necessidades da aplicação.

A denominação de família decorre de um grupo de núcleos operacionais em constante evolução: novos membros são continuamente agregados para atender aos novos requisitos dos usuários. O XL-ARM7TDMI é apenas o primeiro deste grupo, mas já foi utilizado no projeto de três produtos: um equipamento de GPS, um computador de bordo e um equipamento de telemetria. Nestes usos, o *kernel* se mostrou robusto, estável e eficiente.

Referências

Becker, L. B. and Pereira, C. E. From Design to Implementation: Tool Support for the Development of Object-Oriented Distributed Real-Time Systems. 12th Euromicro Conference on Real-Time Systems, Stockholm, Sweden, June 2000.

Beuche, Danilo, Guerrouat, Abdelaziz, Papajewski, Holger, Schröder-Preikschat, Wolfgang, Spinczyk, Olaf, and Spinczyk, Ute, On the Development of Object-Oriented Operating Systems for Deeply Embedded Systems - The PURE Project, Proc. of the 2nd ECOOP Workshop on Object-Orientation and Operating Systems (ECOOP-OOOSWS'99), Lisboa, Portugal, June 1999, pp. 27-31.

Cunningham and Cunningham, Inc Object Oriented Operating Systems available at: http://c2.com/cgi/wiki?ObjectOrientedOperatingSystem January, 2004.

Engler, Dawson R., FRANS KAASHOEK, M. and O'TOOLE, James Exokernel: An Operating System Architecture for Application-Level Resource Management Symposium on Operating Systems Principles, 1995, pp. 251-266

Fröhlich, Antônio A. and SCHRÖDER-PREIKSCHAT, Wolfgang EPOS: An Object-Oriented Operating System, *Proc. of the 2nd ECOOP Workshop on Object-Orientation and Operating Systems (ECOOP-OOOSWS'99)*, Lisboa, Portugal, June 1999, pp. 38-43

Ganssle, J. and Barr, M. (2003), Embedded Systems Dictionary, CMP Books, San Francisco.

Gomaa, Hassan Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, 2000.

KADAK AMX Timing Guide and Data for AMX for ARM Multitasking Executive, April 2003. Available at: http://www.kadak.com/manual/amx/thumb/timinggd.pdf

Oliveira, Rômulo S. Aspectos Construtivos dos Sistemas Operacionais de Tempo Real. IV Workshop de Tempo Real (WTR2003), SBRC-SBC, Natal, Maio 2003, pp 11-18.

Russo, Vincent F. The Renaissance Object-Oriented Operating System. available at: http://www.cs.purdue.edu/Renaissance 1996.

Tanenbaum, A. S. e Woodhull, A. S. (2000), Sistemas Operacionais: Projeto e Implementação, Bookman, Porto Alegre, 2ª edição.

Wikipedia (2006), disponível em: http://en.wikipedia.org