Um Sistema de Arquivos para o EPOS

Hugo Marcondes[1] **, Antônio Augusto M. Fröhlich**[1] **,**
Marcelo Trierveiler Pereira[1]

[1]Laboratório de Integração Software Hardware – UFSC
Campus Universitário - CP 476 - 88040-900 Florianópolis - SC

{hugom,guto,trier}@lisha.ufsc.br

Abstract. *With the growth of the market of embedded systems the support to a file system is more present in the requirements of applications. This work presents a modeling of file systems, following the methodology of Application Oriented System Design, providing the necessary abstractions so that operational system* EPOS *offers support to file systems.*

Keywords: *operating systems, embedded systems, file systems, software components.*

Resumo. *Com o crescimento do mercado de sistemas embutidos é cada vez mais presente nos requisitos das aplicações o suporte a um sistema de arquivos. Este trabalho apresenta uma modelagem de sistemas de arquivos, seguindo a metodologia de desenvolvimento de sistemas orientados à aplicação, provendo as abstrações necessárias para que o sistema operacional* EPOS *ofereça suporte a sistemas de arquivos.*

Palavras-chave: *sistemas operacionais, sistemas embutidos, sistema de arquivos, componentes de software.*

1. Introdução

Embora grande parte das aplicações embutidas se destinam a efetuar apenas o controle de sistemas maiores, armazenando seus dados em memória RAM do sistema, cada vez se torna mais presente em nossas vidas, aplicações embutidas de maior complexidade e de maior valor agregado, que necessitam armazenar seus dados em um sistema de arquivos. Exemplos de tais aplicações são os tocadores de MP3, câmeras digitais e computadores de bolso (PDA e PALMTOPS).

Por outro lado, sistemas de suporte ao tempo de execução de sistemas embutidos devem fornecer à aplicação somente os requisitos estritamente necessários a sua execução, evitando assim *overheads* desnecessários em sua natureza dedicada. Pensando nisso, [Fröhlich 2001] apresenta uma metodologia para o desenvolvimento de sistemas denominado *Application Oriented System Design* (AOSD). A partir deste trabalho o sistema operacional EPOS (*Embedded Parallel Operating System*) foi concebido com o intuito de demonstrar as idéias propostas em [Fröhlich 2001].

Este trabalho se baseia na modelagem do domínio de sistemas de arquivos seguindo os conceitos abordados no trabalho de [Fröhlich 2001], identificando as famílias de componentes de software que irão integrar o sistema de arquivos no EPOS.

2. Sistema de Arquivos

”Toda aplicação necessita armazenar e recuperar informações” [Tanenbaum and Woodhull 1997], para isso o sistema operacional fornece à aplicação um sistema de arquivos, que define a forma estrutural física dos dados que são armazenados de forma persistente em um dispositivo, permitindo assim que diversos arquivos coexistam e se organizem de forma lógica para a visão do usuário.

Arquivos são uma forma de abstração fornecida pelo sistema operacional, para permitir que os processos possam armazenar informações sob entidades exclusivas e persistentes, em um meio de armazenamento de dados e sobre essas entidades aplicar diversas operações (leitura, escrita), que podem estar sendo influenciadas por aspectos (compartilhamento, segurança, etc).

Assim cabe ao sistema operacional, definir uma forma de armazenar essas informações e definir uma maneira de mapear essas entidades em um meio de armazenamento. Esse mapeamento é efetuado através de estruturas denominadas ”meta-dados”, ou seja, dados que descrevem outros dados. Opcionalmente, essas entidades denominadas arquivos, podem ser nomeadas, de forma que as mesmas possam ser acessadas e referenciadas através de nomes. Nesse caso uma coleção adicional de meta-dados deve ser utilizada para fazer esta nomeação e formar um subsistema de diretórios estando este geralmente agregado ao próprio sistema de arquivos. Embora [Tanenbaum and Woodhull 1997] cite a importância deste mecanismo ao sistema de arquivos, um subsistema de diretórios não é estritamente essencial no contexto de sistemas embutidos, uma vez que os arquivos podem ser acessados através de uma referência a sua posição (índice em uma tabela de descritores de arquivos, por exemplo).

Por sua vez, os meta-dados definidos para realizar a implementação de um sistema de arquivos, podem ser separados nos seguintes grupos:

- **Descritores do sistema:** Descrevem o sistema de arquivos como um todo, definindo a localização dentro do dispositivo de armazenamento de outras estruturas de meta-dados, assim como informações sobre a forma como este dispositivo está segmentado. Geralmente essa estrutura permanece em uma localização fixa do dispositivo, facilitando assim a inicialização do sistema de arquivos. Um exemplo deste tipo de estrutura é o chamado superbloco, existente no sistema de arquivos EXT2.
- **Descritores de arquivos:** Descrevem cada entidade denominada arquivo, existente no sistema, definindo possíveis atributos dos arquivos, como por exemplo datas de último acesso, criação e modificação, permissões de acesso e visibilidade. No descritor do arquivo também é definido a localização dos dados do arquivo, que pode ser efetuado através de alocação contínua, lista de blocos, lista indexada ou lista de nodos indexados [Tanenbaum and Woodhull 1997].
- **Gerenciadores de alocação:** Representam o estado de alocação de todos os recursos disponíveis para o sistema de arquivos, sendo estes referentes a dados de arquivos, ou mesmo a alocação de outras estruturas de meta-dados, como por exemplo os descritores de arquivos.

3. Modelagem de sistemas orientados à aplicação

Segundo a metodologia apresentada por [Fröhlich 2001], o trabalho de modelagem de um sistema orientado à aplicação (AOSD) se inicia na decomposição do domínio em famílias de abstrações independentes de cenário que, através de reusabilidade, podem ser instâncias do mesmo sistema, conforme mostra a figura 1. Através de *configurable features* e aspectos de cenário, a configurabilidade do sistema pode ser alcançada. Tais famílias de abstrações constituem componentes de software que unidos através de um *framework* formam o sistema adaptado à aplicação. Os componentes de tal *framework* são acessados através de *Interfaces Infladas* que garantem a portabilidade do sistema e são o principal objeto de análise dos requisitos da aplicação.

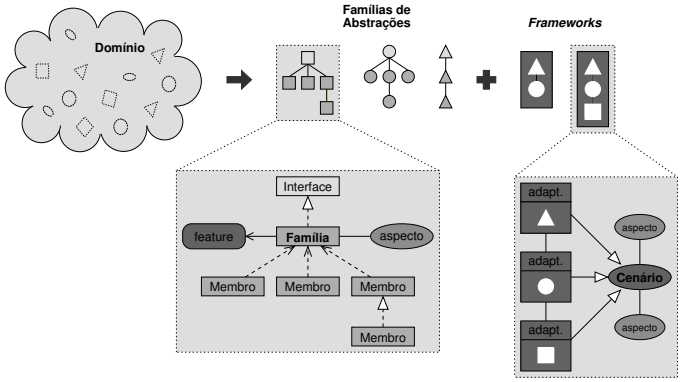

Figura 1. Decomposição de um domínio através da AOSD

A fatoração de um determinado domínio em famílias pode ser dividida em:

- **Mediadores:** componentes de software estritamente dependentes da arquitetura e hardware presentes no sistema cuja principal função é isolar aspectos específicos do hardware, garantindo assim a portabilidade do mesmo [Polpeta and Fröhlich 2004].
- **Abstrações:** modelam os conceitos independentes da arquitetura de hardware e cenário de execução da aplicação.
- **Aspectos:** implementam conceitos ortogonais aos conceitos empregados nas famílias de abstrações tais como compartilhamento, segurança e identificação, e que por sua vez, conforme descrito em [Fröhlich 2001], podem ser aplicados ao sistema, através de adaptadores de cenários.

3.1. O Sistema Operacional EPOS

O projeto EPOS (*Embedded Parallel Operating System*) foca no desenvolvimento automático de sistemas dedicados, permitindo assim, que desenvolvedores se concentrem apenas em suas aplicações. O EPOS se baseia no desenvolvimento de sistemas orientados à aplicação (AOSD) para modelar e implementar componentes tanto de software como de hardware que podem ser adaptados para preencher os requisitos de uma aplicação particular de forma adequada. Adicionalmente, o sistema EPOS fornece um conjunto de ferramentas para selecionar, adaptar e unir componentes em um *framework* específico à aplicação.

4. Decomposição do domínio

O sistema de arquivos foi dividido inicialmente em duas visões distintas, a visão do sistema operacional, que enxerga um sistema de arquivos como uma coleção de estruturas que contém meta-dados armazenados em um dispositivo e a visão do usuário que provê a realização das abstrações de arquivos e diretórios, fornecendo acesso ao seus dados para o usuário do sistema operacional. A figura 2 mostra as famílias identificadas no domínio.

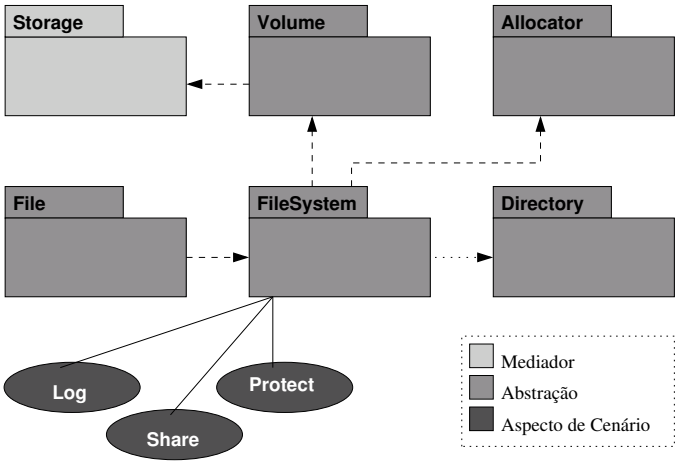

Figura 2. Famílias do Sistema de Arquivos

- **Storage :** Responsável por implementar as funcionalidades de determinado hardware de armazenamento de dados. Esta é uma família de mediadores.
- **Volume :** Responsável pela interface entre o sistema de arquivos e o hardware de armazenamento de dados, implementando o conceito de volumes através de uma tabela de volumes.
- **Allocator :** Responsável pelo gerenciamento dos blocos livres no sistema e utilização da tabela de descritores de arquivos, implementando algoritmos de alocação de espaço sobre as estruturas que fazem esse controle.
- **FileSystem :** Responsável por gerenciar os meta-dados específicos de um sistema de arquivos e algumas das operações que são realizadas no sistema de forma global, como por exemplo a exclusão de arquivos.
- **Share :** Esta é uma família de aspectos e sua função é implementar os algoritmos necessários para que o sistema de arquivos suporte o compartilhamento de arquivos.
- **Protect :** Esta é uma família de aspectos e sua função é implementar os algoritmos necessários para que o sistema de arquivos suporte mecanismos de proteção de acesso aos arquivos.
- **Log :** Esta é uma família de aspectos e sua função é implementar os algoritmos necessários para que o sistema de arquivos suporte a utilização de arquivos baseados em Log [Rosenblum and Ousterhout 1992], como por exemplo o mapa de descritores de arquivos em memória.
- **File :** Implementa as chamadas de sistemas necessárias para criar, acessar e modificar arquivos do sistema.
- **Directory :** Implementa as chamadas de sistema responsáveis por acessar diretórios no sistema de arquivo além de incluir, modificar e excluir entradas no diretório.

4.1. A Família Storage

A família *Storage* visa implementar os mediadores para acesso a um dispositivo de armazenamento de dados persistentes, entre eles, dispositivos ATA, SCSI, memórias FLASH e todos os dispositivos baseados em blocos. Basicamente a interface desta família possui os métodos **getblock** e **putblock**. Alguns membros desta família podem possuir alguns métodos adicionais que implementam funcionalidades específicas do hardware de armazenamento. É o caso de uma memória FLASH que necessita de um método específico para apagar um determinado setor antes de sobrescrever algum dado no mesmo.

Neste trabalho, o *Storage* foi implementado ”emulando” um dispositivo orientado a blocos, sobre a memória RAM do sistema.

4.2. A Família Volume

Esta família implementa uma unidade lógica denominada VOLUME, também conhecida como partição em alguns sistemas. Cada membro desta família encapsula o conhecimento de uma determinada especificação de tabela de volumes. Esta família possui as seguintes funções no sistema:

- Garantir, caso necessário, a tradução de endereços de blocos lógicos para blocos físicos.
- Garantir o acesso de um determinado sistema de arquivos exclusivamente a região que lhe foi destinada.
- Encapsular o conceito de tabelas de volumes.

Os membros inicialmente identificados nessa família foram o *FlatVolume*, *EposVolume* e *DosVolume*. O membro *FlatVolume* implementa um volume que ocupa todo o espaço de armazenamento do componente *Storage*, ficando a ele atribuída apenas a função de tradução de endereços lógicos para físicos. O membro *EposVolume* implementa uma especificação de tabela de volumes própria do EPOS, visando suprir a necessidade de se conhecer o tamanho do bloco lógico. Em grande parte dos sistemas, o tamanho do bloco lógico está armazenado dentro do próprio sistema de arquivos, com isso é necessário inicialmente, efetuar uma busca no disco para localizar o descritor do sistema, geralmente chamado de superbloco, para se conhecer o real tamanho do bloco lógico, enquanto essa informação já poderia estar armazenada na própria tabela de volumes, simplificando assim o algoritmo de inicialização do sistema de arquivos. O membro *DosVolume* implementa a especificação de uma tabela de volumes do Sistema Operacional DOS.

4.3. A Família Allocator

Esta família é implementada como uma classe utilitária do EPOS, ou seja, seus membros são genéricos para que outras partes do sistema operacional, assim como a própria aplicação, possam utilizá-lo para efetuarem alocação de recursos.

Nesta modelagem, buscou-se encontrar a separação entre o algoritmo de alocação e a estrutura de dado utilizada pelo mesmo para efetuar o gerenciamento dos blocos que estão livres. Através do paradigma de programação genérica, este resultado foi alcançado, utilizando-se o recurso de *templates* da linguagem C++.

Através da definição de uma interface comum foi possível modelar um conjunto de classes que são passadas a classe que implementa o algoritmo de alocação através de um *template*, gerando assim um alocador específico em tempo de compilação do sistema.

Para a implementação do protótipo deste trabalho, foi criada uma classe de dados, chamada *PersistentBitmap* que é responsável por efetuar a leitura do Bitmap[1] do sistema de arquivos no disco e disponibilizá-lo ao algoritmo através de sua interface, assim como manter os dados atualizados no disco, conforme ocorrem as modificações neste Bitmap.

4.4. A Família FileSystem

Sendo esta a família central de um de sistemas de arquivos, cabe a esta efetuar grande parte da implementação de uma determinada especificação de um sistema de arquivos.

Todos os algoritmos necessários para acessar os meta-dados que compõem um sistema de arquivos são realizados nos membros desta. Dessa maneira a implementação dos membros da família *File* se torna genérica, podendo assim ser utilizada independente do sistema de arquivos em questão.

Algumas das operações definidas para arquivos e diretórios são implementadas nesta família, uma vez que o contexto destas operações necessitam modificar diversas estruturas internas ao sistema de arquivos. As operações de exclusão de arquivos e diretórios é um caso típico. Ao excluir-se um arquivo do sistema de arquivos é necessária a atualização de diversos meta-dados no descritor do sistema de arquivos, assim como a desalocação dos blocos que este utiliza entre outros dados. Caso, esta operação fosse definida na interface da família *File*, seria necessário garantir acesso a diversas informações (descritor do sistema de arquivos, alocadores de descritores de arquivos e blocos) que não são pertinentes ao arquivo em si, quebrando-se o encapsulamento de dados da programação orientada a objetos.

Internamente aos membros do sistema de arquivos é definida uma classe denominada *FileDescriptor*, responsável por fornecer acesso aos meta-dados necessários a implementação dos algoritmos presentes na família *File*.

Sendo assim, a família *FileSystem* além de implementar diversos métodos para algumas das operações sobre arquivos e diretórios, ainda implementa uma ”fábrica” de *FileDescriptor* que são utilizados pela família *File*, afim desta poder fornecer o acesso de leitura e escrita aos dados do arquivo. Esta funcionalidade é implementada através do método ***FileDescriptor * open_filedescriptor(index: u32)***.

4.5. A Família File

Esta família é o principal componente desta modelagem, sendo esta responsável pelo acesso efetivo aos arquivos do sistema. Sua modelagem foi efetuada visando ser a mais genérica possível, sem que peculiaridades de uma determinada especificação de sistema de arquivos influencie sua interface. Desta maneira foi possível criar componentes realmente genéricos e que podem ser aproveitados independentemente do sistema de arquivo que está sendo utilizado.

Esta família possui os membros *ContinuousFile*, *RandomFile* e *CircularFile*. A

[1]Mapa de bits - um array de bits, onde cada bit representa o estado de cada posição do alocador

principal diferença entre estes membros, se dá na forma como o arquivo é acessado pelo usuário.

O membro *ContinuousFile* implementa um arquivo onde os dados são alocados de forma contínua no disco, seu acesso pode ser efetuado de forma randômica, contudo um tamanho inicial para o arquivo deve ser especificado no momento de sua criação e toda vez que o mesmo necessitar crescer em tamanho, uma tarefa custosa de realocar e copiar todo o arquivo para uma nova área contínua do disco pode ser efetuada.

O membro *RandomFile* implementa a idéia mais comum de arquivos que temos atualmente, um arquivo que pode ser alocado por blocos não consecutivos no disco, possuindo acesso randômico a estes.

O membro *CircularFile* implementa um tipo de arquivo bem específico e geralmente utilizados em arquivos de log. A idéia básica para este arquivo é que o mesmo possui um tamanho fixo, e da mesma maneira que no membro *ContinuousFile*, pode ser aumentado ou reduzido através de uma operação custosa ao sistema, contudo este arquivo não possui um final. A partir do momento que é efetuada uma escrita no último bloco alocado a este, novos dados são armazenados no começo do arquivo, sendo que os dados que anteriormente estavam no início do arquivo são perdidos. Este tipo de arquivo pode ser utilizado com o objetivo de se manter apenas uma certa quantidade de dados recentes em um arquivo, garantindo também que o mesmo não irá crescer e ocupar um espaço maior que o estipulado pelo usuário ao arquivo, durante sua criação, ou através do redimensionamento do mesmo explicitamente pela chamada do método ***u32 resize(u32 size)***.

4.6. A Família Directory

Esta família é responsável pela implementação do recurso de nomes a arquivos. Conforme visto no capítulo 2. um sistema de arquivos não precisa necessariamente possuir um nome para representá-lo dentro do conjunto de arquivos que formam o sistema. Contudo caso a especificação necessite de uma nomeação de seus arquivos, isto é realizado pelos membros desta família.

Note que estes membros não precisam ser utilizados apenas para implementações de sistemas de arquivos. Outros componentes que possuam a característica fundamental de um sistema de diretórios (tradução de um nome para algum outro tipo de informação) podem utilizar este componente para realizar tais funções. Exemplos de sistemas que utilizam o conceito de diretórios são sistemas de tradução de nomes na Internet, através do protocolo *Domain Name Server (DNS)* e sistemas de banco de dados através do protocolo *Lightweight Directory Access Protocol (LDAP)*.

Basicamente, para cada especificação de sistema de arquivos, haverá um membro correspondente que implementa a especificação de diretórios do mesmo. Embora não seja uma regra, a grande maioria dos sistemas de arquivos implementam o sistema de diretórios como arquivos especiais presentes no disco. Com isso, cada sistema de arquivos irá definir uma formatação específica para o seu arquivo de diretório, e daí a necessidade de existir membros específicos a cada implementação.

Um sistema de diretórios é definido como um conjunto de entradas, cujo o conteúdo pode variar de acordo com a aplicação do mesmo, e a estas são associados

nomes que devem ser considerados como chaves do sistema de diretórios. Em um mesmo diretório não é possível existir duas entradas com o mesmo nome.

Uma estrutura interna a cada membro da família *Directory* define o conteúdo de cada entrada do sistema de diretórios, sendo esta implementada internamente a cada membro da família na definição da classe *Entry*.

5. Protótipo implementado

O protótipo implementado para a validação da modelagem é um sistema de arquivos simples armazenado na própria memória do sistema. Na figura 3 pode-se observar a organização de suas estruturas de meta-dados.

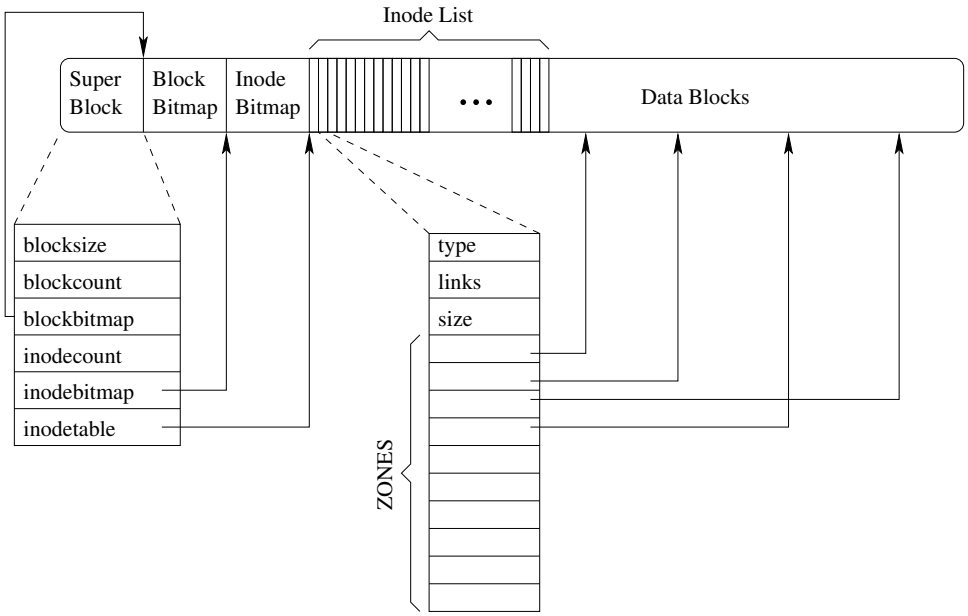

Figura 3. Estrutura geral da especificação implementada

No início do dispositivo é armazenado o descritor do sistema de arquivos (*superblock*), que contém basicamente ponteiros para outras estruturas de dados como os bitmaps de alocação, início da tabela de descritores de arquivos, além de algumas outras informações sobre o sistema de arquivos, como por exemplo, o tamanho do bloco lógico.

Os descritores de arquivos possuem um atributo relativo ao tipo de arquivo, além de um contador que informa o número de entradas de diretório que apontam para o mesmo. A referência aos blocos de dados pertencentes ao arquivo é efetuada através de nodos indexados com referências simples, ou seja, o nodo aponta diretamente aos blocos lógicos do arquivo. Cada descritor pode referenciar até 10 blocos, sendo assim, o tamanho máximo do arquivo é limitado a 10 vezes o tamanho do bloco lógico, definido no momento que o sistema é formatado.

A figura 4 apresenta a abertura e leitura de um arquivo, no sistema implementado, mostrando a interação dos componentes implementados.

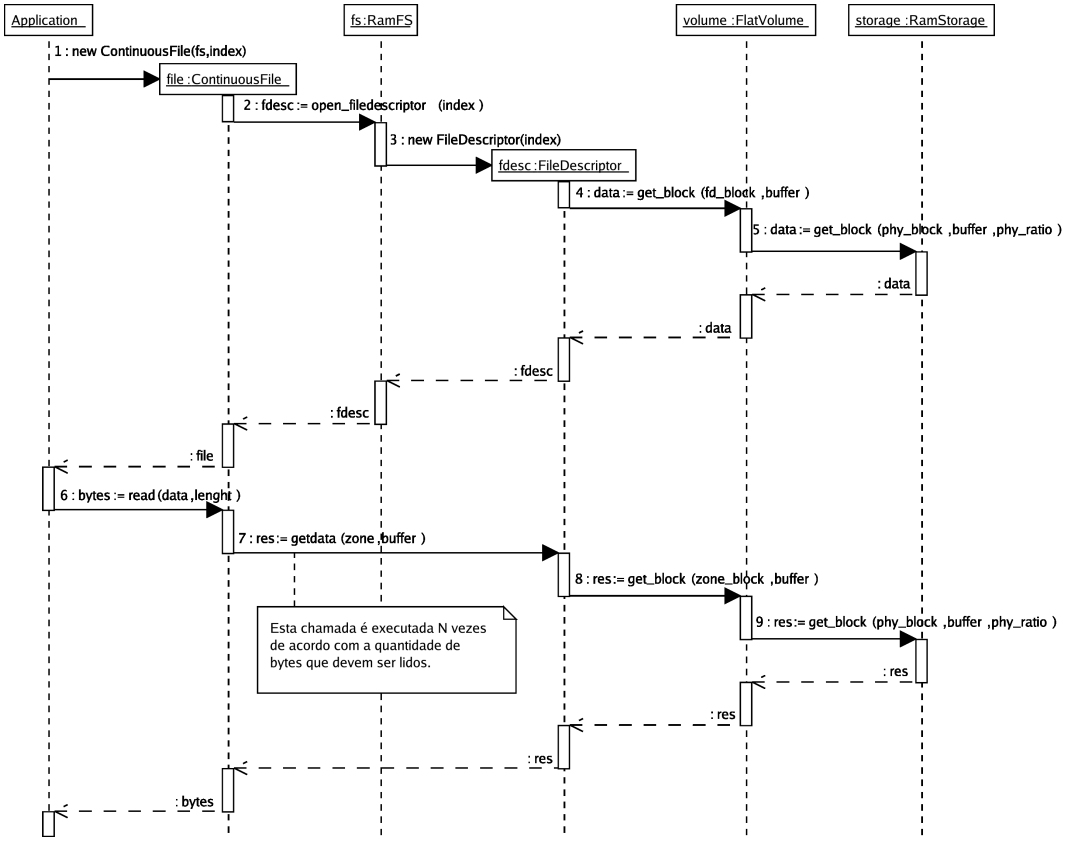

Figura 4. Diagrama de sequência da abertura e leitura de um arquivo.

6. Conclusões

Um sistema de arquivos básico em memória RAM foi implementado dentro do sistema EPOS, oferecendo a possibilidade da aplicação armazenar e recuperar informações de uma mídia estruturada de forma organizada, mostrando-se viável a implementação de sistemas de arquivos baseados em componentes de software, através da metodologia de projeto de sistemas orientados à aplicação.

Através do uso de aspectos de cenários da AOSD, foi possível criar uma modelagem, onde os recursos não utilizados pela aplicação podem ser retirados da implementação, diminuindo assim o *overhead* na mesma. O reúso também é favorecido por tal modelagem, devido a divisão de responsabilidades evidentes na independência dos algoritmos presentes nos membros da família *File* e *FileSystem*, assim como a separação entre os dados utilizados pelos algoritmos de alocação de recursos e sua implementação, através do uso de meta-programação estática (*templates* da linguagem C++).

Os resultados preliminares do protótipo desenvolvido geram sistemas com imagens do tamanho de cerca de 35 Kbytes, sendo que o código objeto dos componentes de sistemas de arquivos selecionados para o protótipo totalizam 19.348 bytes. O tempo de escrita no protótipo implementado é cerca de 6 a 8 vezes maior que o tempo de escrita de dados diretamente na memória RAM do sistema, consequência do tratamento de dados realizado pelo sistema de arquivos.

Referências

Barr, M. (1999). *Programming Embedded Systems in C and C++*. O'Reilly.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2 edition.

Eckel, B. (2000). *Thinking in C++, Volume 1: Introduction to Standard C++*. Prentice Hall, 2 edition.

Fröhlich, A. A. M. (2001). *Application-Oriented Operating Systems*. GMD - Forschungszentrum Informationstechnik, 1 edition.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Johnson, R. E. and Foote, B. (1988). Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35.

Larman, C. (2001). *Applying UML and Patterns*. Unknown, second edition.

Marwede, P. (2003). *Embedded System Design*. Kluwer Academic Publishers.

OMG (1999). *OMG Unified Modeling Language Specification*. OMG.

Polpeta, F. V. and Fröhlich, A. A. (2004). Hardware mediators: A portability artifact for component-based systems. In Yang, L. T., Guo, M., Gao, G. R., and Jha, N. K., editors, *EUC*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280. Springer.

Rosenblum, M. and Ousterhout, J. K. (1992). The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52.

Smolik, T. (1995). An object-oriented file system: an example of using the class hierarchy framework concept. *ACM SIGOPS Operating Systems Review*, 29(2):33–53.

Stroustrup, B. (1997). *The C++ Programming Language*. Addison-Wesley, 3 edition.

Tanenbaum, A. S. and Woodhull, A. S. (1997). *Operational Systems - Design and Implementation*. Prentice Hall, 2 edition.