ANÁLISE DE LOCALIDADE DE REFERÊNCIA NA JPC

Formando: Willian Goedert

Orientador: Mauro Marcelo Mattos

Roteiro

- Introdução
- Fundamentação Teórica
- Desenvolvimento
- Conclusões
- Extensões

Introdução

Contexto

- Aplicações atualmente desenvolvidas consomem boa parte de recursos como processamento e memória.

- Para gerenciamento de memória dispõem-se uma memória principal, muita rápida e bastante cara e memória secundária com velocidades e custos baixos.

- Sobre essas deficiências faz-se estudos sobre como aumentar a eficiência para gerenciar o uso de memória a partir de análise de acessos ao uso de memória.

Objetivos

Objetivo Principal

✓ Desenvolver um módulo de análise de localidade de referência de programas que executam na JPC.

Objetivo Específico

✓ Desenvolver um algoritmo de contabilização e registros das informações coletadas.

✓ Desenvolver módulo de visualização das informações coletadas.

Fundamentação Teórica

Java Personal Computer (JPC)

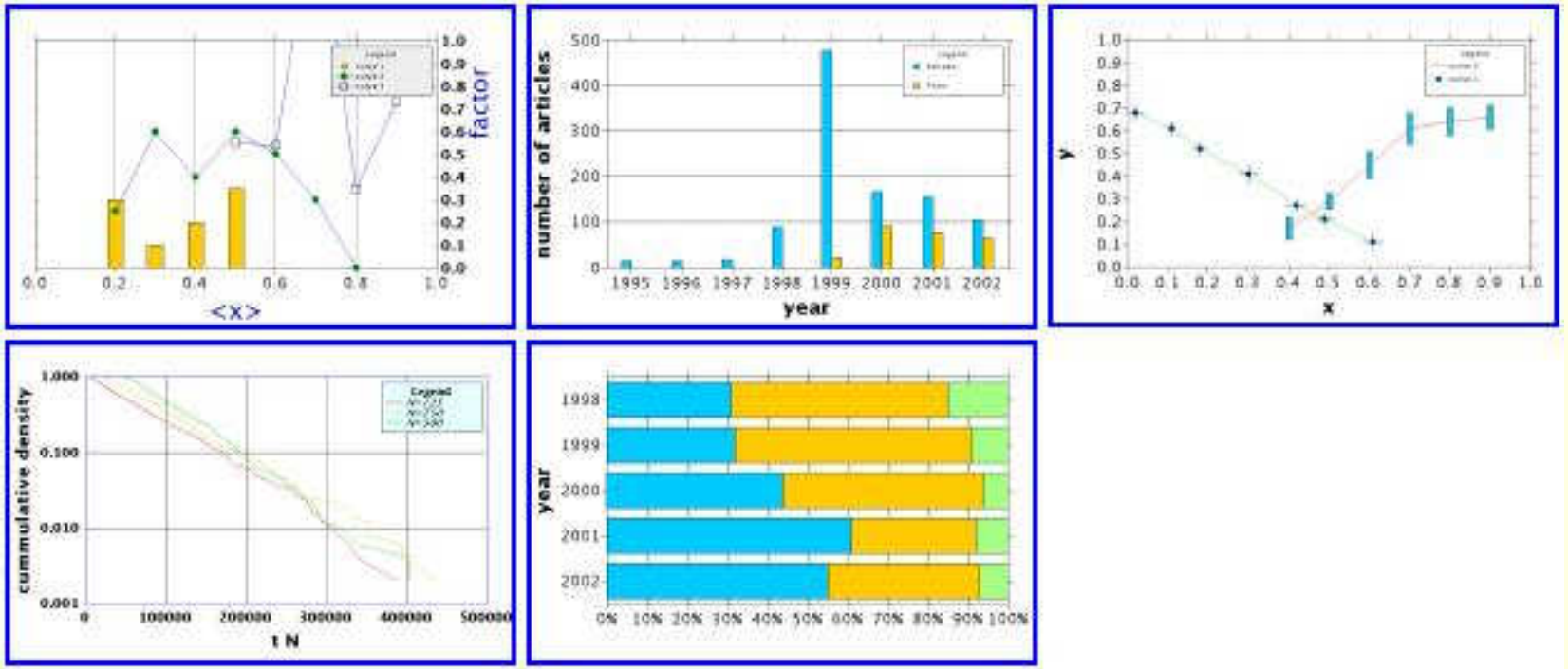
- Desenvolvido desde 2002 por pesquisadores do departamento de física da Universidade de Oxford.
- Sua finalidade é emular um hardware X86 usando puramente a tecnologia Java.
- Compatível com as plataformas Windows, Linux, MacOS e em dispositivos móveis.
- Característica importante é sua segurança por não ter acesso direto ao hardware real da máquina.
- Seu código fonte é *Open Source* e possui pouca documentação sobre ela.

Biblioteca JCCKit

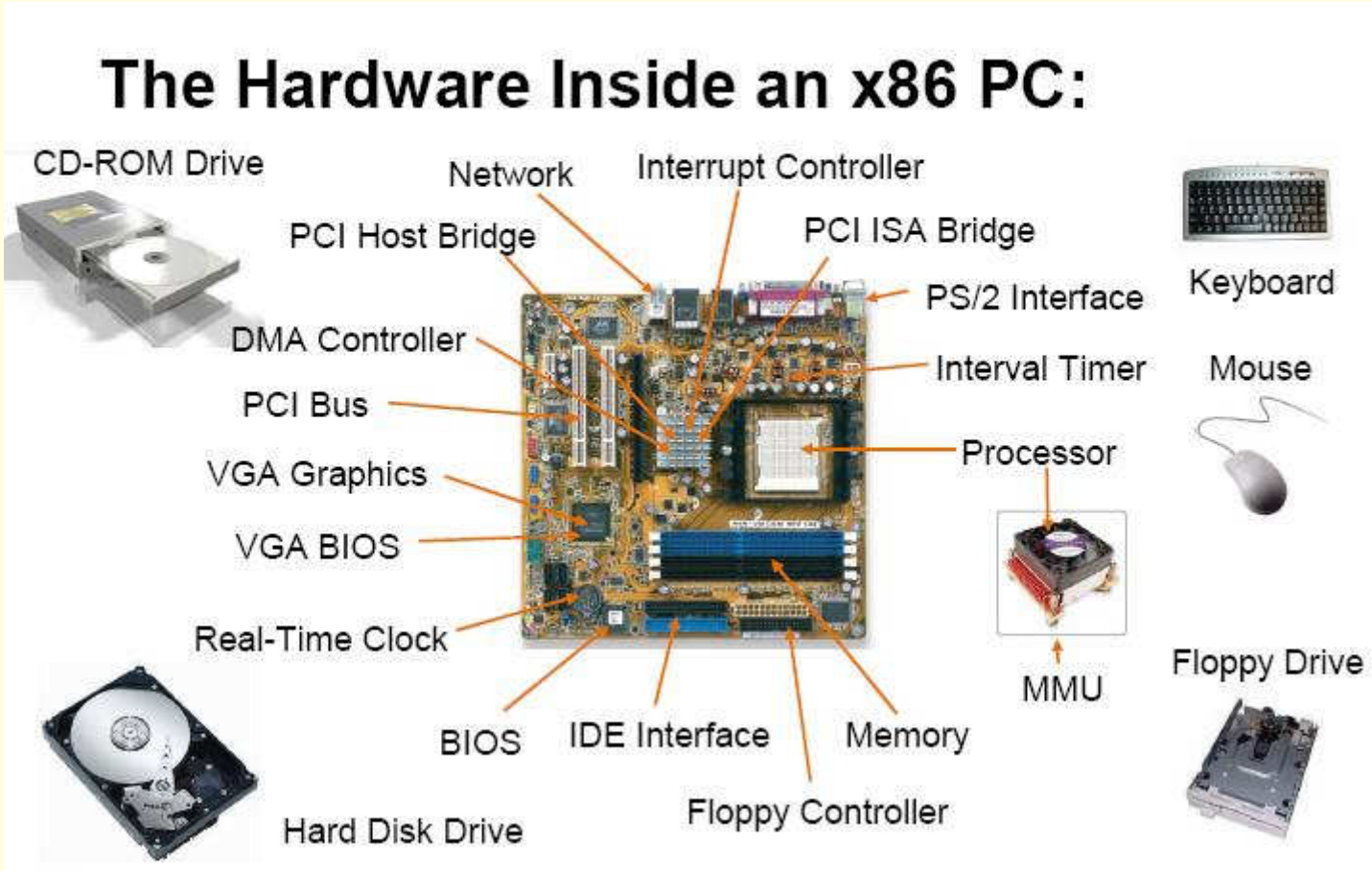
- Para gerar os gráficos foi utilizado uma biblioteca chamada JCCKit.
- Possui código fonte aberto.
- Desenvolvido na linguagem Java.
- Possui classes preparadas para a montagem de gráficos, onde contém métodos para adicionar pontos de dados, para configurar sua apresentação e para desenhar.
- Permite construir gráficos de diversas formas (barras, pontos, linhas).

Biblioteca JCCKit

- Exemplo de tipos de gráficos que podem ser gerados pela JCCKit.



JPC – Exemplo da Estrutura X86



Gerenciamento de Memória

- É uma técnica utilizada para otimizar o uso da memória, como controlar quais partes da memória estão sendo usadas, alocar memória de acordo com a necessidade de cada processo, liberar memória alocada a um processo quando termina sua execução e efetuar o controle entre memória principal e memória secundária.

Memória Principal

- Conhecida como memória Real ou Física (Ex. Memória RAM).
- Seu tamanho máximo é fixo.
- Ao executar processos o gerenciador de memória localiza um bloco livre cujo tamanho atenda a necessidade do processo.

Memória Secundária

- Dispositivo de armazenamento utilizado como extensão da memória (Ex. HD, pen drive).

Gerência de Memória Virtual

- Separação da memória física da memória lógica, ou seja, desvincular o endereço físico criado pelos programas na memória real.
- Controle de endereçamento “Físico X Virtual” por páginas
  - Política de Busca
  - Política de Alocação
  - Política de Substituição
    - Localidade de referência (endereços próximos)
    - Conjunto de trabalho – working-set (páginas).
- Os programas não estão mais limitados ao tamanho da memória física

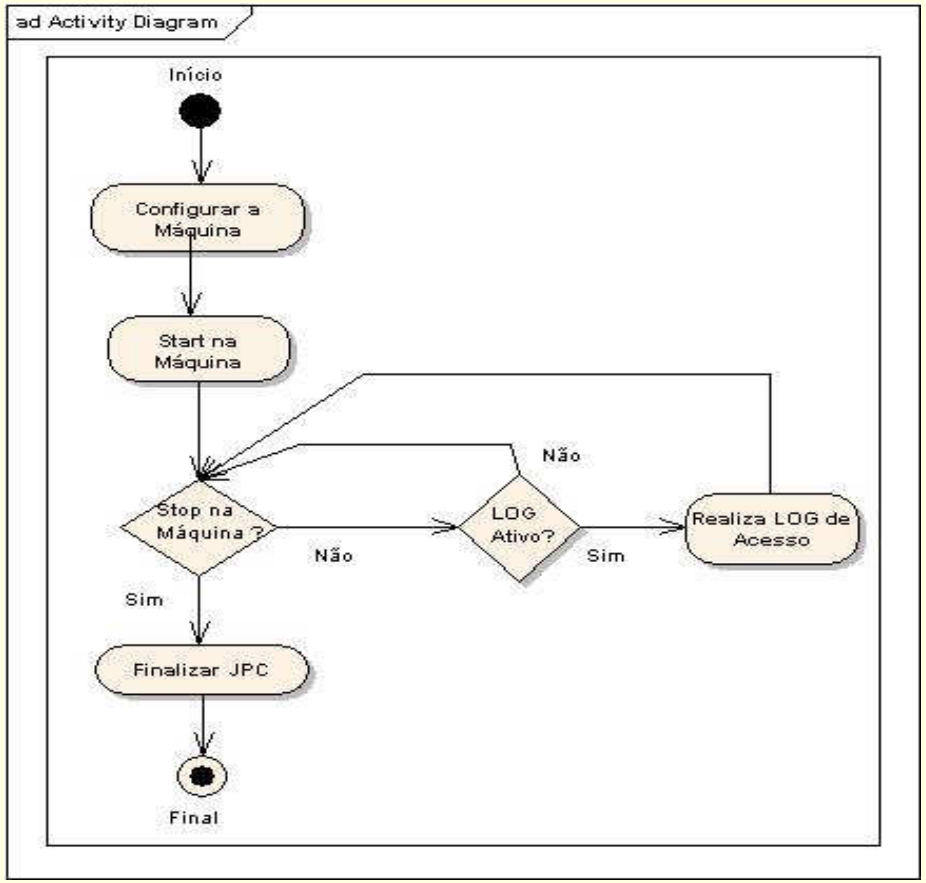
Desenvolvimento

Requisitos

- Implementar um contador de acesso a posições de memória real.
- Implementar um modelo de *log* dos acessos.
- Disponibilizar uma visualização em forma gráfica.

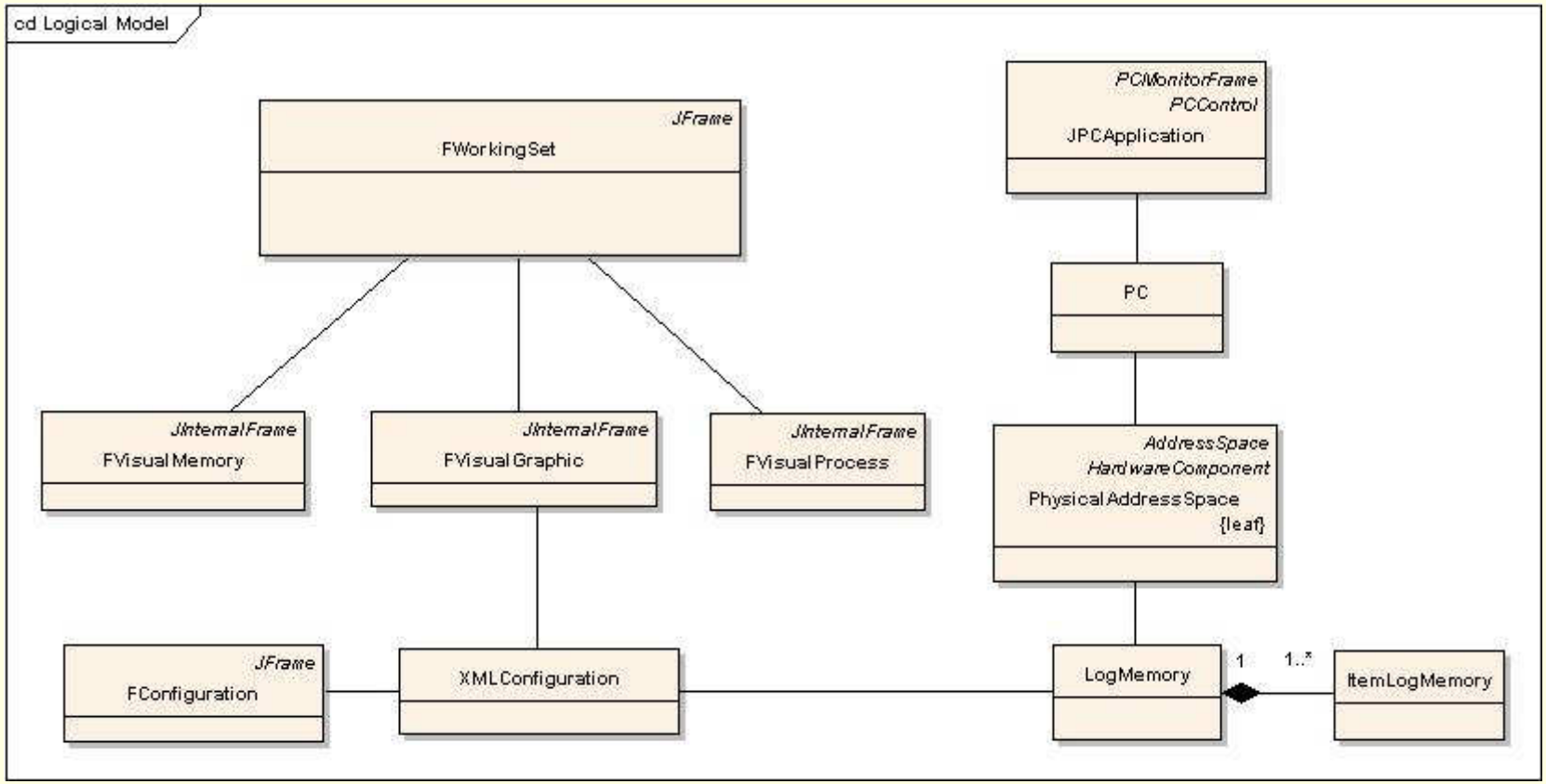
Especificação

Diagrama de Atividades



Especificação

Diagrama de Classes



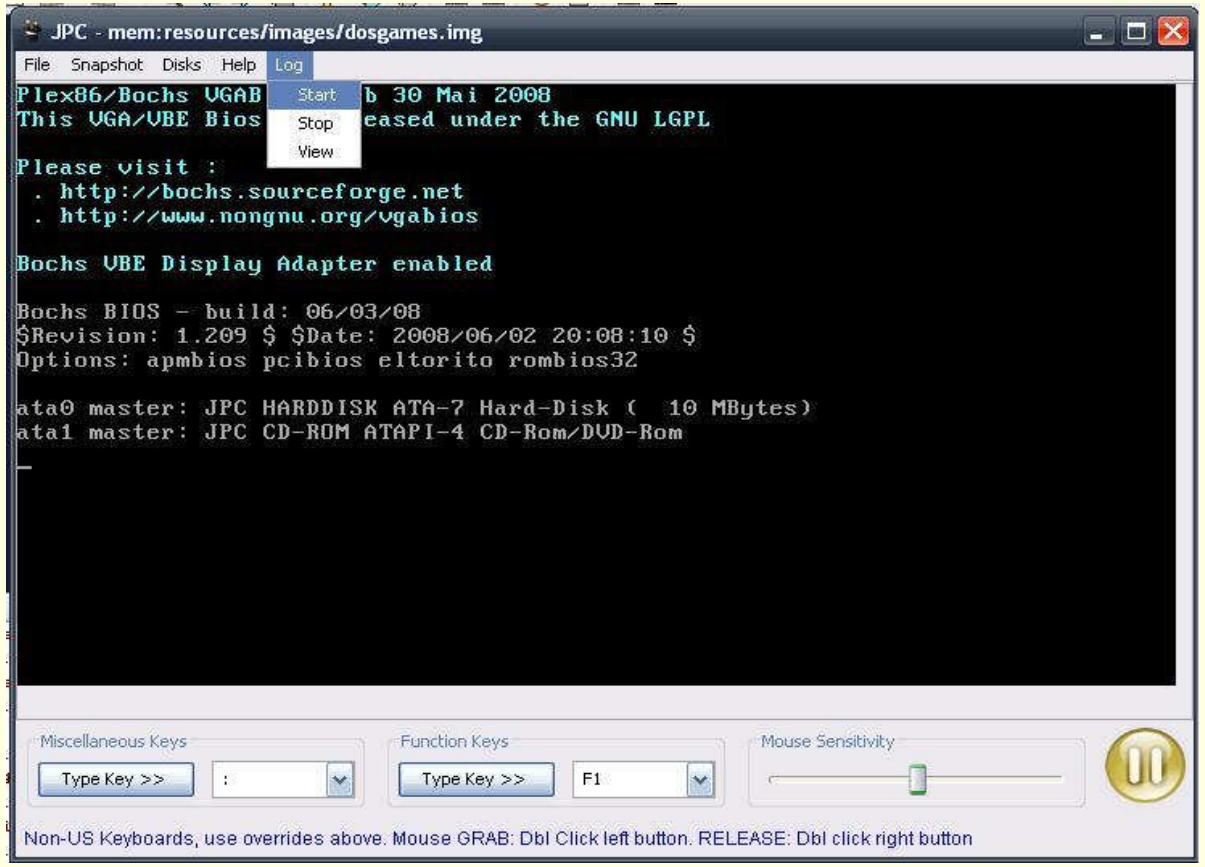
Implementação

Técnicas e Ferramentas Utilizadas

- Implementado na forma de estrutura de classes utilizando a linguagem de programação Java.
- Construção de diagramas utilizando a ferramenta Enterprise Architect.
- Ambiente de desenvolvimento NetBeans.
- Log gravado em arquivos de texto com extensão log.

Operacionalidade da Implementação

- Foi acoplado ao projeto JPC um menu para efetuar o controle de *log*.



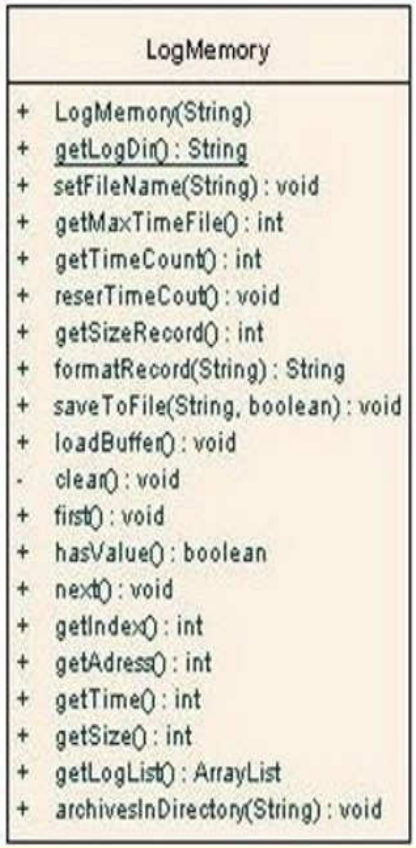
Zoom

Operacionalidade da Implementação

- Desenvolvido rotina de geração de *log*.

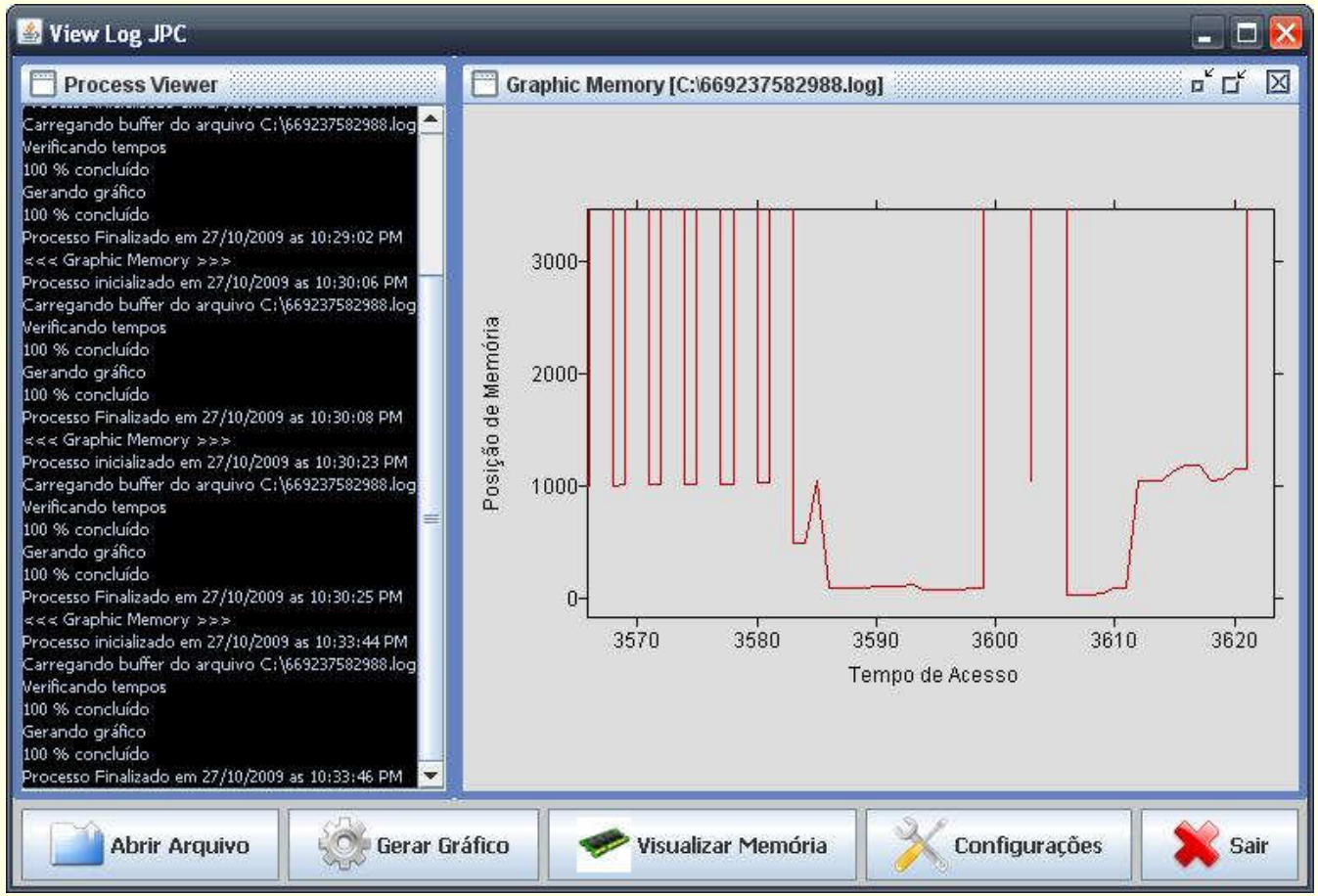
```
w001048560t000000001w001040475t000000002r001040530t000000003
```

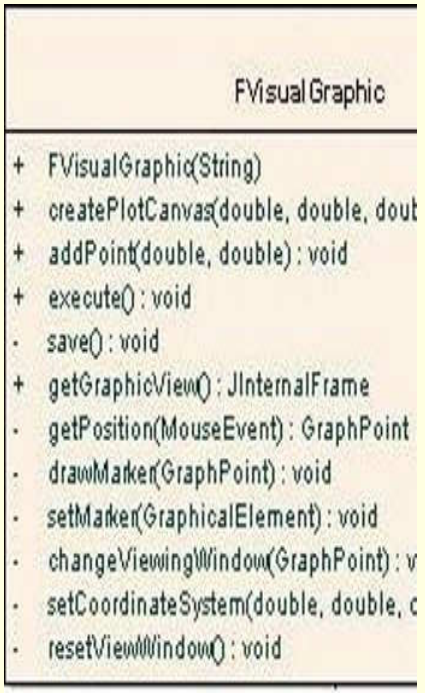
```
protected Memory getReadMemoryBlockAt(int offset) {
//*************************************************************************************************************
// willian.begin of block in 2009 Sep 19
//*************************************************************************************************************
    // Verificar se o flag de geração de log esta ativo
    if  ((offset > 0) && (isLogActived())){
        try {
            // Adicionar 'r' (reader) - Informação de leitura no endereço memória
            String logBlock = "r";
            // Pegar posição da memória acessada e formatá-la
            logBlock += logMem.formatRecord(String.valueOf(offset));
            // Adicionar 't' (timer) - Informação do tempo que foi acessado
            logBlock += "t";
            // Pegar tempo de acesso da memória e formatá-la
            int time = logMem.getTimeCount();
            logBlock += logMem.formatRecord(String.valueOf(time));

            // geração de arquivos de log com tamanho parametrizado
            if  (time > logMem.getMaxTimeFile())
                resetLog();

            // Salvar log no arquivo
            logMem.saveToFile(logBlock, true);
        } catch (IOException ex) {
            Logger.getLogger(PhysicalAddressSpace.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
//*************************************************************************************************************
// willian.end of block in 2009 Sep 19
//*************************************************************************************************************
```



Implementação

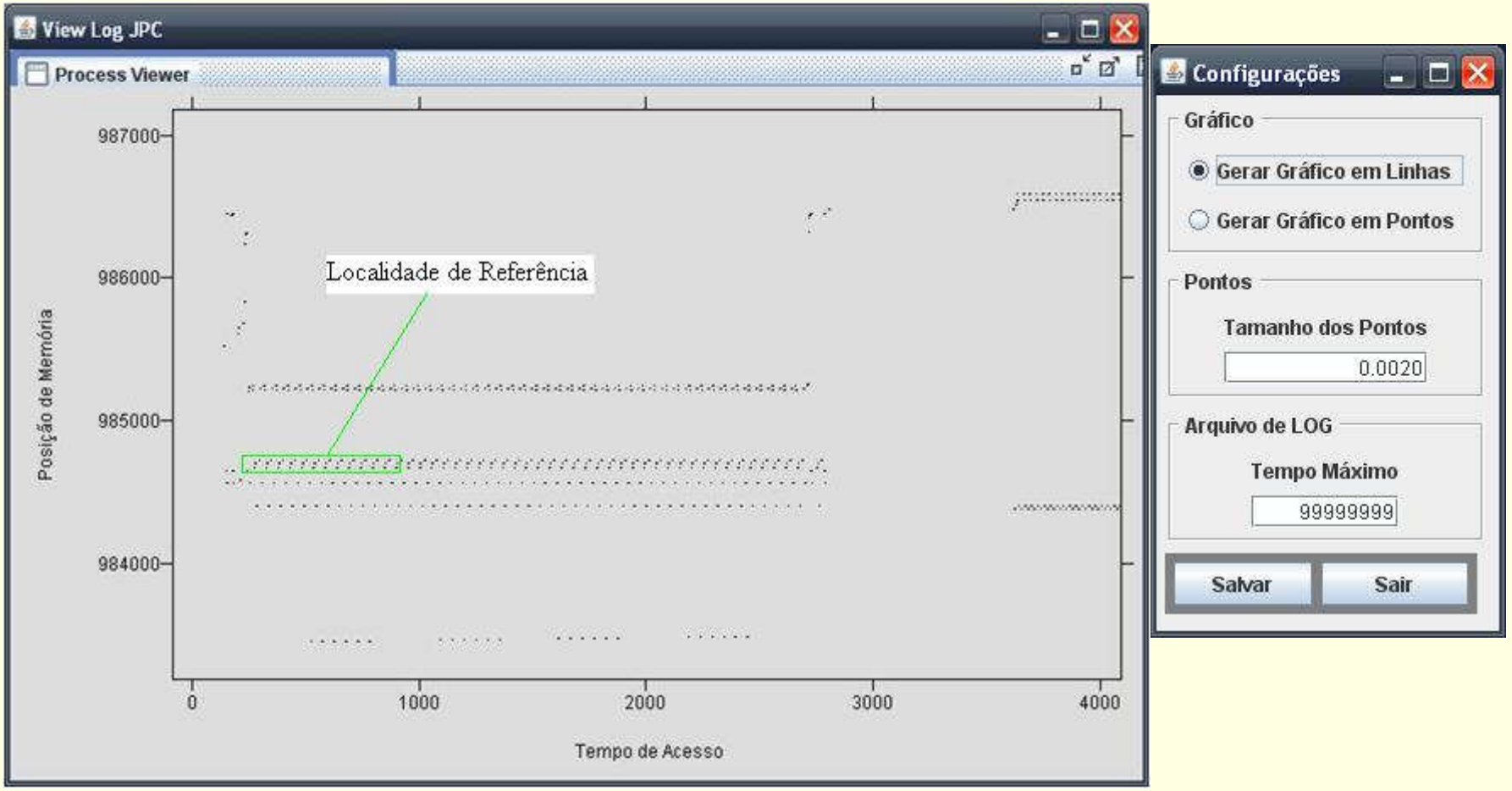
- Visualização das informações de *log*.





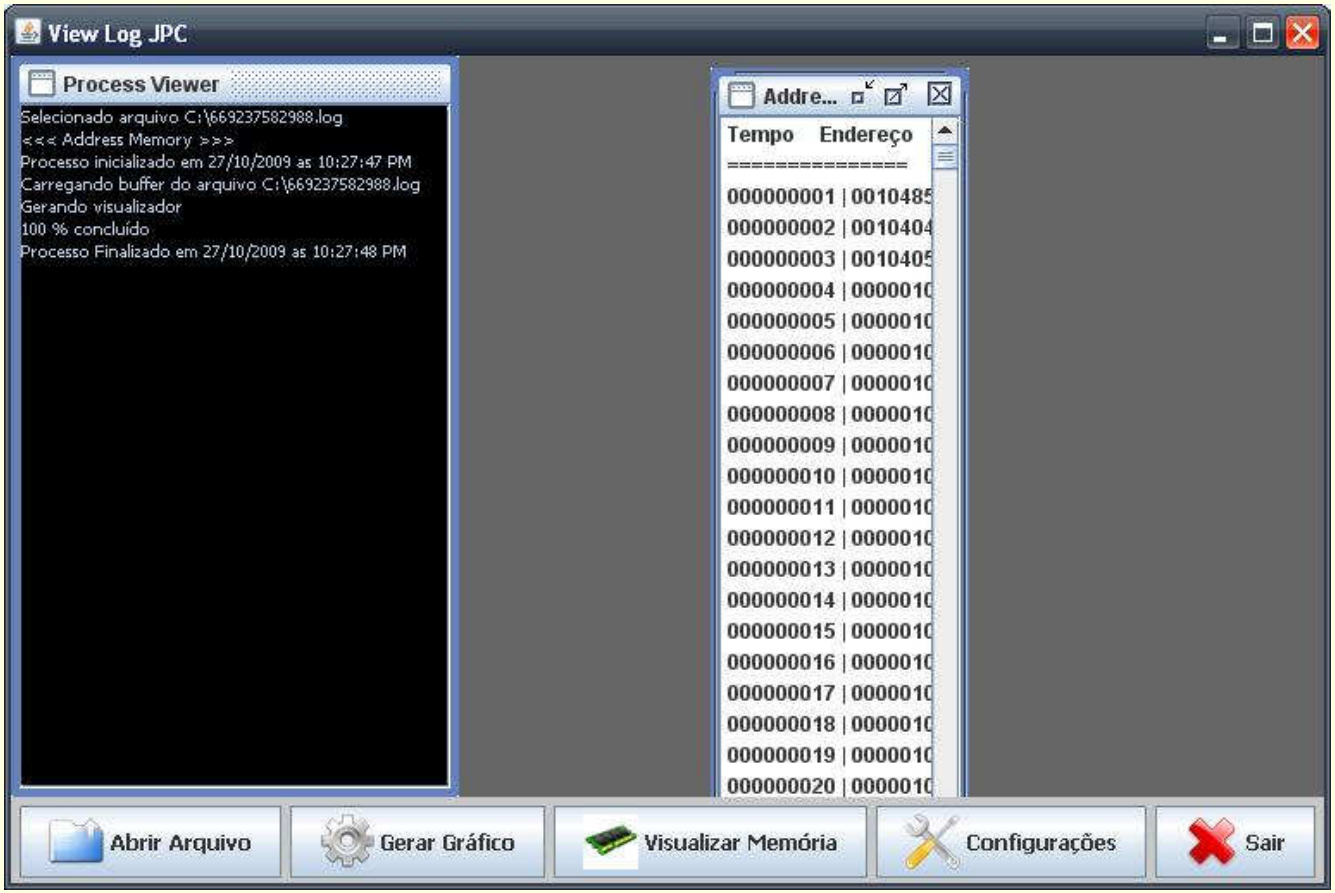
Implementação

- Configurações da visualização.



Implementação

- Visualização do contador de tempos.



Conclusão

Conclusão

- Foram alcançados os objetivos em desenvolver o módulo de análise de localidade de referência.
- Obteve-se um conhecimento aprofundado sobre gerenciamento de memória e de como implementar de forma visual os acessos.

Extensões

Extensões

Sugestões para trabalhos futuros:

- Destacar no gráfico gerado os padrões de acesso a memória.
- Implementar a geração e visualização das instruções executadas para cada *log* gerado pela JPC.
- Plotar os pontos relativos a acessos de leitura e escrita com cores diferentes.
- Implementar a geração de log e um contador de endereços que geram *page fault.*

Demonstração do Projeto