ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

PADRÕES DE PROJETOS NO DESENVOLVIMENTO DE APLICAÇÕES J2ME

Trabalho de Conclusão de Curso

Engenharia da Computação

Filipe da Silva Regueira
Orientador: Prof. Dr. Márcio Lopes Cornélio

Recife, 3 de julho de 2006

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

PADRÕES DE PROJETOS NO DESENVOLVIMENTO DE APLICAÇÕES J2ME

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Filipe da Silva Regueira
Orientador: Prof. Dr. Márcio Lopes Cornélio

Recife, 3 de julho de 2006

UNIVERSIDADE
DE PERNAMBUCO

Filipe da Silva Regueira

PADRÕES DE PROJETOS NO DESENVOLVIMENTO DE APLICAÇÕES J2ME

Resumo

Este trabalho apresenta um estudo sobre o desenvolvimento de aplicações J2ME utilizando padrões de projetos. O estudo mostra conceitos sobre padrões de projetos, a tecnologia J2ME e conceitos de Engenharia de Software Experimental utilizados para a criação de um plano de experimento.

Uma pesquisa realizada com alguns engenheiros, de empresas do Porto Digital, mostra padrões de projetos que já estão sendo utilizados em aplicações desenvolvidas em J2ME. Também foi apresentado neste trabalho benefícios do uso de padrões de projetos como, por exemplo: a diminuição dos custos com manutenção, o aumento de legibilidade do código, redução no tempo de desenvolvimento da aplicação e aumento da qualidade interna do produto.

Um estudo experimental realizado com uma aplicação desenvolvida em outro trabalho de graduação apresenta impactos gerados pela implementação de padrões de projetos à aplicação selecionada. O estudo experimental envolveu a implementação dos padrões de projetos *Singleton* e *Factory Method* e a execução de experimentos com quatro versões da aplicação selecionada. A primeira sem a implementação dos padrões, a segunda com a implementação do *Singleton,* uma terceira com a implementação do *Factory Method* e uma última com os dois padrões de projetos. Apesar dos resultados mostrarem uma queda no desempenho geral da aplicação, mostramos benefícios que a implementação de padrões de projetos em aplicações J2ME pode trazer.

Abstract

This work presents an study on the development of J2ME applications using design patterns. We present concepts about design patterns, the J2ME technology and Experimental Software Engineer which is used to create an experience plan.

A study performed with engineers, from Porto Digital companies, shows that design patterns in J2ME applications are already in use. This work presents reasons to use design patterns (e.g. less costs in maintenance, code legibility improvement, reduction in the development time, and better product quality).

An experimental study made with an application developed in another graduating work presents the impacts of using design patterns in the selected application. The experimental study covers the Singleton and Factory Method design patterns implementation and experiments with four versions of the application. The first without design patterns, a second one with the Singleton implementation, a third with the Factory Method implementation and the last one with both design patterns. In spite of the results presents a performance decrease, we show that it's possible to develop J2ME application using design patterns. The results shows that the application with design patterns had a little performance decrease even with a greater number of classes and more lines of code which gives a better maintenance capability.

Sumário

Índice de Figuras

Figura 47. Gráfico com média de desempenho, em milisegundos, de cada funcionalidade analisada para cada implementação.

Índice de Tabelas

Tabela de Símbolos e Siglas

J2ME – *Java 2 Micro Edition*
PDA – *Personal Digital Assistant*
API – *Application Programming Interface*
J2EE – *Java 2 Enterprise Edition*
CLDC – *Connected Limited Device Configuration*
MIDP – *Mobile Information Device Profile*
UML – *Unified Modeling Language*
J2SE – *Java 2 Standard Edition*
CDC – *Connected Device Configuration*
KVM – *Kilo Virtual Machine*
RMI – *Remote Method Invocation*
JDBC – *Java Database Connectivity*
JVM – *Java Virtual Machine*
MHz – Unidade de freqüência igual a um milhão de ciclos por segundo
ROM – *Read Only Memory*
JAR – *Java Archive*
JAD – *Java Application Descriptor*
RAM – *Random Acces Memory*
OEM – *Original Equipment Manufacturer*
URL – *Universal Resource Locator*
QIP – *Quality Improvement Paradigm*
GQM – *Goal/Question/Metric*
NUPEC – Núcleo de Pesquisa de Engenharia da Computação
DSC – Departamento de Sistemas Computacionais

Agradecimentos

Agradeço primeiramente a Deus por toda minha vida e por todas as coisas que ele tem me proporcionado.

À minha maravilhosa mãe (Maria Regueira) pelo carinho, amor, dedicação e compreensão.

Ao meu querido pai (Cezar Regueira Santos) por todos os ensinamentos, proteção e apoio em todos os momentos de minha vida.

Ao meu cuidadoso e grande irmão, Tuca (Arthur Regueira), por todo o apoio, ajuda e incentivo.

A minha eterna irmãzinha do coração, Nininha ou Dra. Marina Genesia da Silva Regueira, a qual sempre está ao meu lado, me acolhendo, nos momentos difíceis e nos momentos de alegria.

À minha linda namorada, Rafaela Nóbrega Pereira, por todo carinho, amor e compreensão ao longo destes 5 anos que estamos juntos.

À minha querida avó, Noeme Regueira, por todo amor, carinho e energia que me proporciona.

Ao meu professor e orientados Márcio Lopes pela ajuda e dedicação para que este trabalho fosse concluído com sucesso e a todos os professores da UPE que participaram da minha formação.

A todos os meus amigos e irmãos, AMIGOSPOLI, por todos os ensinamentos, compreensão, noites de trabalho e momentos de descontração durante estes cinco anos. Especialmente a Diogo Pacheco e Herbert de Menezes por sempre que possível podarem minha pseudo-aleatoriedade.

Aos meus amigos do trabalho por todos os ensinamentos e incentivos para que este trabalho fosse concluído. A todos aqueles que contribuíram para a pesquisa realizada neste trabalho.

Ao C.E.S.A.R. por ter me dado a oportunidade de me transformar no profissional que sou hoje.

Aos meus tio (os) e tia (as), especialmente Renato Tertuliano, a quem tenho grande admiração, por todos os conselhos.

A todos os meus primos, especialmente a Diogo e Fabio, que me acompanham mais de perto, pelos conselhos e momentos de descontração.

Aos meus inesquecíveis amigos da Turma do Mel, pelos anos de amizade que se tornou um grande aprendizado. E a todos aqueles que de certa forma contribuíram para eu chegar até aqui.

Deus, obrigado por colocar todas estas pessoas em meu caminho.

Capítulo 1

Introdução

Em Outubro de 2002, uma pesquisa publicada por Zelos Group [26] estimou que mais de 44 milhões de celulares com suporte à Java [10] foram vendidos em 2002. Isso representa 11% de todos os aparelhos produzidos naquele ano. A pesquisa projetava que em 2007 o número de aparelhos produzidos com Java iria alcançar mais de 450 milhões, o que somará 74% de todos os aparelhos fabricados. Os maiores vendedores de aparelho de celular já adotam Java como parte de suas estratégias para o futuro dos aparelhos. Podemos citar Nokia, Motorola, Siemens, Samsung, LG Eletronics, dentre outros. Grandes operadoras como NexTel, SprintPCS e AT&T, já incluíram em suas redes suporte a aparelhos e aplicações em Java [27].

A plataforma Java 2 Micro Edition (J2ME) [9] provê um ambiente robusto e flexível para o desenvolvimento de aplicações em dispositivos móveis (celulares, PDA's entre outros). Como outras plataformas Java, J2ME possui uma máquina virtual Java e um conjunto de API's (*Application Programming Interface*) definidas pelo Java Community Process [28].

Apesar de todos os benefícios inerentes à plataforma, o projeto de boas aplicações para celulares não é trivial, porque, embora J2ME não seja uma plataforma complexa, ela sofre com as restrições impostas pelos aparelhos móveis, tais como:

- recurso de memória limitado;
- baixa capacidade de processamento;
- interfaces reduzidas, levando-se em conta que os aparelhos celulares possuem normalmente uma tela com pequenas dimensões;
- problemas com transmissão de dados, porque redes sem fio estão sujeitas a mais erros do que as redes com fio;
- curto tempo de vida da bateria.

Não podemos escrever aplicações para dispositivos móveis da mesma forma que aplicações para computadores de mesa ou servidores são desenvolvidas, visto que estes possuem alto poder de processamento, grande capacidade de memória e enorme poder na transmissão de dados na rede. Tendo em vista as limitações apresentadas referentes à

plataforma J2ME pode-se perceber que é necessária uma atenção extra ao projetar uma aplicação para celular.

Padrões de projeto apresentam soluções para problemas recorrentes [8]. Aplicações J2ME também podem obter benefícios a partir do uso de padrões de projeto. Por exemplo, por possuir uma estrutura melhor, devido ao uso de padrões, a manutenção pode ser facilitada, reduzindo custo. É importante observar que alguns padrões de projeto podem gerar problemas para aplicações, por exemplo, alguns deles diminuem o acoplamento entre as funcionalidades do sistema gerando um maior número de chamadas a funções, aumentando o tempo de resposta do sistema. Os padrões por si só não garantem o sucesso. Este trabalho tem como motivação verificar quais são, de fato, os impactos do uso de padrões de projeto em aplicações J2ME.

1.1 Objetivos e Metas

O objetivo geral deste trabalho é realizar um estudo do uso de padrões de projeto no desenvolvimento de aplicações J2ME. Diferentemente da plataforma Java 2 Enterprise Edition (J2EE) [25], que possui vários catálogos de padrões de projetos escritos por desenvolvedores e arquitetos experientes, a plataforma J2ME [9] não possui catálogos de padrões. Tendo em vista que poucos são os padrões de projetos publicados para J2ME, uma meta deste trabalho é selecionar alguns padrões de projetos descritos nos catálogos de padrões da plataforma e verificar a possibilidade de aplicá-los em sistemas desenvolvidos usando a plataforma da Java para dispositivos móveis.

Os objetivos específicos são identificar, em algumas empresas do Porto Digital [16], quais padrões realmente estão sendo utilizados; compreender quais benefícios tais padrões podem trazer para projetos e mostrar quais problemas podem ser evitados e quais podem ser gerados com o uso de padrões.

Com a intenção de saber quais dos padrões estudados são mais utilizados e quais trazem mais benefícios para as aplicações desenvolvidas em um domínio especifico, um questionário foi elaborado e distribuído para engenheiros de algumas empresas do Porto Digital.

Uma aplicação, desenvolvida em J2ME, foi escolhida para ser alvo de experimentos, os quais envolveram a implementação de alguns padrões de projetos que foram selecionados a fim de auxiliar no entendimento das vantagens e desvantagens geradas pelo uso de determinado padrão. A partir destes experimentos, métricas foram recuperadas e analisadas, como, por exemplo, o desempenho de uma determinada funcionalidade com e sem o padrão implementado.

1.2 Visão Geral do Trabalho

Este trabalho está organizado da seguinte maneira. No Capítulo 2 abordamos os padrões de projetos, apresentando algumas definições, características, os elementos essenciais e classificação, de acordo com Gamma et al [8].

O Capítulo 3 mostra a plataforma J2ME. Neste capítulo é apresentado uma visão geral da plataforma, a Configuração de Dispositivo Conectado Limitado (CLDC), o Perfil de Dispositivo de Informação Móvel (MIDP) e os principais fundamentos da programação com MIDP.

O Capítulo 4 apresenta os conceitos da Engenharia de Software Experimental. É discutida a descrição geral da experimentação, os tipos de experimentos e como ocorre o processo de experimentação.

No Capítulo 5 é definido o plano experimental, são apresentados os resultados dos experimentos e uma análise comparativa deles.

Por fim, no Capítulo 6, temos as conclusões, as principais contribuições deste trabalho e propostas para trabalhos futuros.

Capítulo 2

Padrões de Projetos

Neste capítulo veremos uma introdução aos padrões de projetos e um breve histórico. Estudaremos várias definições de padrões de projeto, suas características e elementos principais. Observaremos uma forma de como podemos descrever um padrão de projeto e finalmente as várias classificações de um padrão de projeto segundo Gamma et. al. [8]

2.1 Introdução

Em Engenharia de Software um padrão de projeto é uma solução que foi utilizada diversas vezes para resolver um determinado problema no projeto de uma aplicação. Padrões de projetos na orientação a objetos mostram as relações e comunicações entre classes e objetos, sem especificar as classes ou objetos finais que estarão envolvidos. Importante lembrar que algoritmos não são padrões de projetos, pois os mesmos resolvem problemas computacionais ao invés de problemas de projeto.

Um projetista experiente sabe que não deve resolver problemas baseado na primeira idéia de solução. O mesmo reutiliza soluções que já funcionaram em projetos passados. Quando uma boa solução é encontrada, o projetista experiente reutiliza a mesma diversas vezes.

2.2 Histórico

Durante a década de 70 um arquiteto chamado Chistopher Alexander iniciou os primeiros trabalhos na área de padrões de projeto. Com o intuito de identificar e descrever as etapas no desenvolvimento de projetos de qualidade, Alexander e seus colegas estudaram diferentes estruturas que foram projetadas para resolver o mesmo problema. Ele

identificou similaridades entre projetos de alta qualidade. Em alguns de seus livros ele usou a palavra "padrão" para se referenciar a estas similaridades. Todos os padrões identificados e documentados por Alexander são puramente arquiteturais e falam a respeito de estruturas como prédios, jardins e vias de automóveis [1].

Em 1987, influenciados pelos estudos de Alexander, Kent Beck e Ward Cunningham aplicaram as idéias de padrões arquiteturais para projeto e desenvolvimento de programas. Eles utilizaram algumas idéias de Alexander para criar um conjunto de padrões para o desenvolvimento de interfaces elegantes em Smalltalk[31], uma linguagem de programação orientada a objeto desenvolvida pela Xérox na década de 70. No mesmo ano eles apresentaram seus resultados na conferência do OOPSLA (*Object-Oriented Programming Systems, Languages, and Applications*) em uma apresentação nomeada *Using Pattern Languages for Object-Oriented Programming* (usando padrões de linguagens para programação orientada a objeto). Nos anos seguintes muitos artigos e apresentações relacionados com padrões foram publicados pelas pessoas da área de programação orientada a objetos.

A popularidade dos padrões de projetos na ciência da computação cresceu após a publicação do livro *Design Patterns: Elements of Reusable Object-Oriented Software* em 1994 por Gamma et al. Os quatro autores, do livro, são referenciados como a gangue dos quatro (*Gang of Four*, GoF). No livro os autores documentaram 23 padrões de projetos que encontraram durante anos de trabalho. Desde então, muitos livros foram publicados mostrando novos padrões de projetos e outras boas práticas para Engenharia de Software.

2.3 O que é um padrão de projeto?

Um padrão de projeto documenta uma solução que foi aplicada com sucesso em vários ambientes para resolver um problema recorrente em um conjunto específico de situações.

2.3.1 Diferentes definições de padrão

Segundo Christopher Alexander: "cada padrão descreve um problema o qual ocorreu diversas vezes em nosso ambiente, e então descreve os principais pontos da solução para aquele problema, de uma maneira na qual podemos utilizar esta solução mais de um milhão de vezes, sem nunca precisar resolver o problema mais de uma vez" [1]. Contudo Alexander estava falando sobre padrões em construções e cidades, o que ele falou também é verdade para padrões de projeto relativos à orientação a objetos. No contexto de orientação a objeto as soluções são expressas em termos de objetos e interfaces ao invés de paredes e portas, mas a idéia principal para ambos os tipos de padrões é uma solução para um problema em um determinado contexto.

Examinaremos agora mais duas definições de padrões:
Richard Gabriel [2] fornece uma definição baseada na definição de Alexander aplicada a programas de computador:

"Cada padrão é uma regra de três partes, que expressa uma relação entre um certo contexto, um certo sistema de forças que ocorre repetidamente nesse contexto e uma certa configuração de software que permite que essas forças sejam resolvidas" [2].

Martin Fowler [3] fornece uma definição mais flexível:

"Um padrão é uma idéia que foi útil em um contexto prático e provavelmente será útil em outros" [3].

Como podemos observar muitas são a definições de um padrão, mas todas elas possuem em comum a recorrência de um par problema/solução em um contexto especifico.

2.3.2 Características de um padrão

Abaixo podemos ver algumas características dos padrões de acordo com Alur et al [29]:

- os padrões são observados através da experiência.
- os padrões normalmente são escritos em um formato estruturado.
- os padrões evitam a reinvenção da roda.
- existem padrões de diferentes níveis de abstração.
- os padrões suportam melhorias contínuas.
- os padrões são artefatos reutilizáveis.
- os padrões comunicam as melhores práticas.
- os padrões podem ser utilizados em conjunto para resolverem problemas maiores que, sozinhos, não conseguiriam.

2.3.3 Elementos essenciais de um padrão

Em geral, um padrão possui quatro elementos essenciais, de acordo com Gamma et al [8]:

1. O **nome do padrão** é o que podemos usar para descrever um problema, suas soluções e conseqüências. Ele permite que projetemos em um nível mais alto de abstração, ao invés de descrevermos em detalhes um possível problema e sua solução, utilizamos do padrão. Nomear um padrão facilita a comunicação dos mesmos juntamente com suas vantagens e desvantagens para as outras pessoas. Encontrar um bom nome para um padrão ainda é uma grande dificuldade para quem documenta padrões.

2. O **problema** descreve quando devemos aplicar um padrão. Explica qual o problema e seu contexto. Deve descrever com estruturas de classes ou objetos o que corresponde a um projeto inflexível. Algumas vezes o problema irá incluir uma lista de condições que devem ser conhecidas antes de definir se o padrão deve ou não ser utilizado.

3. A **solução** descreve os elementos que fazem parte da implementação, as relações entre os elementos, suas responsabilidades e colaborações. A solução não deve descrever uma implementação concreta em particular, porque um padrão é como um modelo que pode ser aplicado em diferentes situações. Ao invés disto, um padrão deve prover uma descrição abstrata de um problema e como uma organização de elementos (classes ou objetos) pode resolver ele.

4. As **conseqüências** são os resultados, vantagens e desvantagens de utilizar o padrão. Elas são importantes para avaliar alternativas de implementação e para entender os custos e benefícios de aplicar o padrão.

2.4 Descrevendo um padrão de projeto

Neste tópico veremos como descrever um padrão de projeto. Anotações gráficas, apesar de muito importantes, não são suficientes. Para reutilizar uma decisão de projeto nós devemos registrar as decisões, alternativas, vantagens e desvantagens. Para ajudar a ver a implementação em uso devemos utilizar exemplos concretos.

A seguir veremos um formato consistente de descrever um padrão definido por Gamma et al.[8]

- Nome e classificação do padrão:
  O nome do padrão refere-se à essência do padrão. A classificação do padrão reflete o tipo do padrão, iremos introduzir a classificação do padrão na próxima seção.

- Intenção do padrão:
  Um parágrafo que responda as seguintes perguntas; O que o padrão faz? Qual é a intenção do padrão? Qual problema em particular ele resolve?

- Também conhecido como:
  Outros nomes bem conhecidos para o padrão, se existir.

- Motivação:
  Um cenário ilustrando um problema de projeto e como a estrutura de objetos e classes no padrão resolve o problema.

- Aplicabilidade:
  Em quais situações o padrão pode ser aplicado?
  Descrever exemplos de implementações pobres em que o padrão pode ser aplicado.
  Como você pode reconhecer estas situações?

- Estrutura:
  Uma representação gráfica das classes utilizadas no padrão.
  A linguagem UML [4] é utilizada para representar graficamente um padrão, ela tem contribuído bastante para o aumento da comunidade de padrões de

projetos. Conceitos como cenários, iterações entre classes, iteração de interfaces de objetos e estados de objetos podem todos ser escritos em UML. Esta linguagem está além do escopo deste trabalho. Existem excelentes referências de UML [4][5][6][7].

- Participantes:
  As classes e/ou objetos que participam da implementação e suas responsabilidades.

- Colaborações:
  Como os participantes colaboram para realizar suas responsabilidades.

- Conseqüências:
  Mostra como o padrão reage e seus objetivos. Apresenta quais são os resultados, as vantagens e desvantagens de utilizar o padrão.

- Implementação:
  Quais armadilhas ou técnicas devem ser evitadas durante a implementação do padrão? Existe algum ponto especifico para alguma linguagem?

- Exemplo de Código:
  Fragmento de código em alguma linguagem para ilustrar como deve ser implementado o padrão.

- Usos conhecidos:
  Exemplos de uso do padrão em sistemas reais.

- Padrões Relacionados:
  Quais padrões estão intimamente relacionados com este?
  Quais são as principais diferenças?
  Este padrão deve ser utilizado com quais outros?

2.5 Classificação dos padrões de projeto segundo Gamma et al [8]

Existem vários tipos de padrões de projetos. Os mesmos estão organizados de uma forma facilitando a busca quando estamos precisando. A classificação também implica no aprendizado mais rápido.

A classificação dos padrões é feita baseada em dois critérios. O primeiro é o propósito, e reflete o que um padrão faz. Segundo Gamma et al os padrões podem ter um propósito de criação, estrutural ou comportamental. O segundo critério é o escopo que pode ser de classe ou de objeto. Veremos cada critério em detalhes.

2.5.1 Classificação segundo o escopo

O mesmo especifica se o padrão é aplicado primariamente a classes ou objetos. Padrões com escopo de classe trabalham com as relações entre classes e subclasses. Esta relação é estabelecida através da herança e é definida em tempo de compilação. Como exemplos destes padrões temos: *Adapter* e *Interpreter* [8].

Padrões com escopo de objeto trabalham com relações entre objetos, e estas relações podem ser alteradas em tempo de execução, sendo assim mais flexível. A maioria dos padrões definidos por Gamma et al possuem escopo de objeto, exemplo: *Singleton* e *Facede* [8].

2.5.2 Classificação segundo o propósito

A seguir veremos cada um dos propósitos em detalhe:

De Criação

Os padrões de criação ajudam a desenvolver sistemas independentemente de como os objetos são criados, compostos e representados. Um padrão de criação com o escopo de classe utiliza herança para variar a classe que é instanciada, enquanto que um padrão de criação com o escopo de objeto delega o processo de instanciar para outro objeto.

Existem duas características em comum a estes padrões. A primeira é que eles encapsulam o conhecimento de qual classe concreta o sistema utiliza. A segunda é que eles escondem a forma de como as instancias das classes são criadas. Com o uso dos padrões de criação, o máximo que o sistema fica sabendo sobre os objetos são suas interfaces, gerando assim muitas flexibilidades em qual objeto está sendo criado, quem criou, como é criado e quando. As configurações podem ser estáticas (especificadas em tempo de compilação) ou dinâmicas (em tempo de execução).

Estrutural

São padrões de projeto que facilitam o projeto, identificando maneiras simples de entender o relacionamento entre entidades.

Padrões estruturais estão envolvidos em como as classes e objetos estão compostos.

Padrões estruturais com escopo de classe utilizam herança para compor interfaces ou implementações. Um exemplo simples é como herança múltipla permite que duas ou mais classes sejam representadas através de uma.

Padrões estruturais com escopo de objeto descrevem formas de compor objetos para realizar novas funcionalidades.

Existe uma flexibilidade que é adicionada com a composição de objetos, a habilidade de mudar a composição em tempo de execução, o que é impossível apenas com herança (é definida em tempo de compilação).

Padrão Comportamental

Padrões com propósito comportamental estão envolvidos com algoritmos e a atribuição de responsabilidades entre objetos. Padrões comportamentais com escopo de classe utilizam herança para distribuir comportamento entre as classes, enquanto que os com escopo de objeto utilizam composição de objeto.

2.6 Resumo

Neste capítulo observamos o quanto é importante o uso de padrões. Vimos que os padrões de projetos em Engenharia de Software foram baseados em padrões para construções desenvolvidos por Chistopher Alexander.

Verificamos várias definições do que é um padrão de projeto, entre elas podemos citar que um padrão de projeto é uma solução que foi útil em um contexto e provavelmente será útil em outros contextos. É através da re-utilização desta solução que projetistas se tornam especialistas.

Sabemos que muitas são as etapas envolvidas na descrição de um padrão de projeto, entre elas podemos citar: nome e classificação do padrão; intenção do padrão; motivação; aplicabilidade; estrutura (podendo ser representada por UML); participantes; colaborações; conseqüências; implementação; exemplo de código; usos conhecidos e padrões relacionados.

Mostramos que um padrão pode ser classificado baseado em dois critérios, primeiro vimos o critério de escopo que pode ser de classe ou de objeto; o segundo critério foi o de propósito. Um padrão de projeto pode ser classificado como tendo um propósito de criação, estrutural e comportamental.

Os padrões de projeto podem acelerar o processo de desenvolvimento de software provendo soluções provadas e testadas. O uso de padrões melhora a legibilidade do código. A maioria dos desenvolvedores sabe apenas como aplicar uma determinada técnica de implementação para resolver um problema especifico. Diferentemente de um padrão de projeto estas técnicas não foram criadas para resolverem um conjunto de problemas. Para isto, os padrões de projeto provêm soluções genéricas e documentadas em um formato genérico.

Capítulo 3

Java 2 Micro Edition

A primeira versão da linguagem Java [10] começou com a frase "*Write Once, Run Anywhere*" (Escreva uma vez, execute em qualquer lugar). O objetivo era criar uma linguagem com a qual os desenvolvedores pudessem escrever o código apenas uma vez e executa em qualquer plataforma que tivesse uma máquina virtual. Inicialmente Java tinha como objetivo criar aplicações para sistemas embarcados.

Desde o lançamento da linguagem, em 1995, o objetivo principal mudou significativamente. A linguagem aumentou o alcance das máquinas-alvo, inicialmente sistemas embarcados e computadores pessoais. Dois anos após o lançamento o time de desenvolvimento responsável pela tecnologia, criou uma nova edição da linguagem, a Java 2 Enterprise Edition (J2EE) [25].

Com o aumento na quantidade de aparelhos portáteis (celular, assistente pessoal digital entre outros) as pessoas envolvidas com a tecnologia Java viram mais uma grande oportunidade no mundo dos negócios e, para alcançá-la, lançaram a edição *Micro Edition* da linguagem Java. Conhecida como Java 2 *Micro Edition* (J2ME) [9] atinge desde máquinas ligadas à TV habilitadas para Internet até telefones celulares.

Conheceremos neste capítulo a plataforma de desenvolvimento J2ME a qual será utilizada no estudo de caso. Veremos as edições da plataforma Java, o porquê de utilizar a tecnologia J2ME, estudaremos as configurações e os perfis que existem na linguagem e o ambiente de desenvolvimento.

3.1 Visão geral da Plataforma J2ME

Nesta Seção será mostrada uma visão geral da tecnologia incluindo as configurações e os perfis descritos. Alguns motivos para utilizarmos também serão visualizados juntamente com uma visão geral da arquitetura definida para a plataforma J2ME.

Dentre os dialetos de Java, destacamos três: J2SE, J2EE e J2ME. A *Standard Edition* ou J2SE foi a primeira a ser criada e foi projetada para o desenvolvimento de aplicações para computadores pessoais e estações de trabalho. Com o avanço da Internet e das aplicações empresariais foi criada a *Enterprise Edition* ou J2EE, projetada para o desenvolvimento de aplicações baseadas em servidor (máquinas com alto poder processamento e memória). Por fim, temos a plataforma *Micro Edition* que teve seu projeto apoiado no aumento em larga escala do número de dispositivos móveis. Esta plataforma destina-se a dispositivos com limitações de memória e de poder de processamento. Na Figura 1 podemos observar o relacionamento entre as edições da plataforma Java.

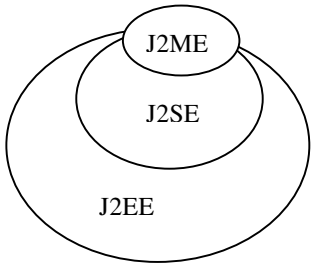

Figura 1. Relação entre as edições da plataforma Java.

Inicialmente os dispositivos móveis não apresentavam a opção de instalar um novo aplicativo, além daqueles que já vinham instalados de fábrica. A introdução da tecnologia J2ME a estes dispositivos permitiu que aplicações fossem instaladas nos mesmos após o processo de fabricação, garantindo assim maior satisfação para o cliente final que realiza um investimento em um aparelho mais versátil, uma vez que inicialmente tinha uma natureza "estática".

Se olharmos ao nosso redor podemos hoje perceber o quanto os equipamentos eletrônicos estão presentes e modificando nossas vidas. Hoje em dia os telefones móveis não só nos comunicam com outra pessoa quando estamos longe de casa ou do trabalho, como também nos permitem acessar a Internet, ler e-mails e executar diversas novas aplicações que vão desde jogos eletrônicos até programas que buscam posições de outros aparelhos no globo terrestre.

Java é uma linguagem de programação fácil de dominar, provê um ambiente seguro e portável. Ela já foi adotada pelos maiores fabricantes de telefones móveis e pelas grandes operadoras de telefonia móvel e já é utilizada por uma comunidade de desenvolvedores estimada em mais de 2 milhões de pessoas [9].

3.1.1 As Configurações

Com o intuito de atender a uma grande variedade de dispositivos que se encaixam dentro do escopo da plataforma J2ME, seu time de desenvolvimento introduziu o conceito de Configuração. Cada configuração define os recursos da linguagem e as bibliotecas que estarão presentes na Máquina Virtual para um determinado conjunto de dispositivos.

A classificação de em qual configuração um determinado aparelho se encaixa é feita através da qualidade de cada uma das seguintes características: quantidade de memória,

qualidade do vídeo, conectividade de rede e poder de processamento. Abaixo veremos as duas configurações definidas juntamente com as características dos dispositivos em cada uma delas:

A Configuração de Dispositivo Conectado (CDC – *Connected Device Configuration*) geralmente é implementada em dispositivos que possuem uma arquitetura de 32 bits, utilizam no mínimo dois *megabytes* de memória disponível e possuem uma máquina virtual completa. Exemplos destes dispositivos são: *set-top boxes*, sistemas de navegação e posicionamento e *smart phones*.

A seguir temos as características típicas dos dispositivos que implementam a CDC:

- 512 *kilobytes* (no mínimo) de memória para executar o Java.
- 256 *kilobytes* (no mínimo) de memória para alocação de memória em tempo de execução.
- Conectividade de rede: largura de banda possivelmente persistente e alta.

A Configuração de Dispositivo Conectado Limitado (CLDC – *Connected Limited Device Configuration*) é implementada em dispositivos que utilizam uma arquitetura de 16 ou 32 *bits*, possuem quantidade de memória geralmente entre 128KB e 512KB, são alimentados através de bateria e utilizam uma conexão de rede sem fio de banda estreita. Estes dispositivos utilizam uma versão reduzida da máquina virtual Java chamada de *Kilobyte Virtual Machine* (KVM). Como exemplos destes dispositivos estão: pagers, Assistente Pessoal Digital (PDAs) e telefones celulares.

A seguir temos as características típicas dos dispositivos que implementam a CLDC:

- 128 kilobytes de memória para executar o Java.
- 32 kilobytes de memória para alocação de memória em tempo de execução.
- Interface com o usuário restrita.
- Normalmente alimentado por bateria.
- Conectividade de rede: normalmente dispositivos sem fio com largura de banda baixa e acesso intermitente.

Na Seção 3.2 veremos em mais detalhes a Configuração CLDC, que será utilizada no estudo de caso deste trabalho.

3.1.2 Os Perfis

Com o avanço da tecnologia os dispositivos terão cada vez mais poder de processamento, mais memória e recursos de tela. Este avanço faz com que haja uma sobreposição entre as categorias de dispositivo cada vez maior. Os perfis surgiram para tratar a variação de recursos existentes e para dar maior flexibilidade à plataforma na medida em que a tecnologia avança.

"Um perfil é uma extensão de uma configuração. Ele fornece as bibliotecas para um desenvolvedor escrever aplicativos para um tipo em particular de dispositivo" [9]. Podemos citar como exemplo o perfil MIDP (*Mobile Information Device Profile*) que, levando em consideração as limitações inerentes aos dispositivos, especifica uma série de APIs para componentes, entrada e tratamento de interface com o usuário, persistência, conexão com rede entre outros. Sete perfis já foram definidos são eles: *Foundation Profile*, *Game profile*, *Mobile information Device Profile*, *PDA Profile*, *Personal Profile*, *Personal Basis Profile* e *RMI Profile*.

A Figura 2 mostra os componentes da plataforma J2ME. Acima do *hardware* do aparelho e do sistema operacional podemos observar uma camada representando a máquina virtual Java, em seguida temos a camada de configurações e por último os perfis que nada mais é do que uma extensão das configurações.

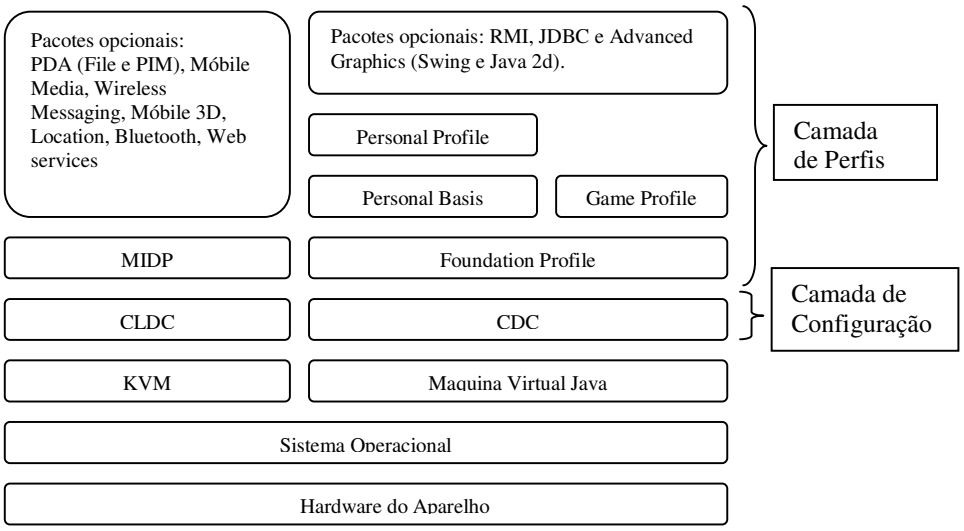

Figura 2. Resumo dos componentes da plataforma J2ME.

3.1.3 As Máquinas Virtuais

O mecanismo por trás de toda a aplicação desenvolvida com a tecnologia Java está na Maquina Virtual Java (JVM – *Java Virtual Machine*). Após compilar todo o código-fonte Java de uma aplicação, a JVM é responsável por interpretar os *bytescodes* gerados após a compilação. É através deste mecanismo que o código Java torna-se portável. A máquina virtual também é responsável por alocar memória para a aplicação e gerenciar linhas de execução.

A configuração CDC utiliza a mesma especificação da máquina virtual usada para a edição J2SE da Java. Já no caso da configuração CLDC o time de desenvolvimento da Sun [10] criou o que é chamado de implementação de referência de uma máquina virtual, conhecida como *Kilobyte Virtual Machine* ou simplesmente KVM.

Algumas exigências que a máquina virtual (JVM) possui para ser executada:

- A máquina virtual propriamente dita exige apenas de 40 a 80 *kilobytes* de memória.
- Apenas 20 a 40 *kilobytes* de memória dinâmica (*heap*).
- Pode ser executada em processadores de 16 *bits*, com freqüência de apenas 25MHz.

Segunda a Sun [10] a maquina virtual KVM e a configuração CLDC se relacionam da seguinte forma: "A CLDC é a especificação para uma 'classe' das máquinas virtuais Java que podem ser executadas nas categorias de dispositivos destinados a CLDC e ao suporte dos perfis".

3.2 Perfil de Dispositivo de Informação Móvel (MIDP)

Esta Seção mostrará quais requisitos são necessários para um dispositivo que pretende implementar o MIDP, uma visão da arquitetura dos aplicativos que executam em telefones móveis. Também veremos o que são os arquivos JAR e JAD e para que servem.

3.2.1 Arquitetura do MIDP

Para conhecer melhor como funciona a arquitetura do perfil MID descreveremos, na Figura 3, todas as camadas envolvidas no perfil.

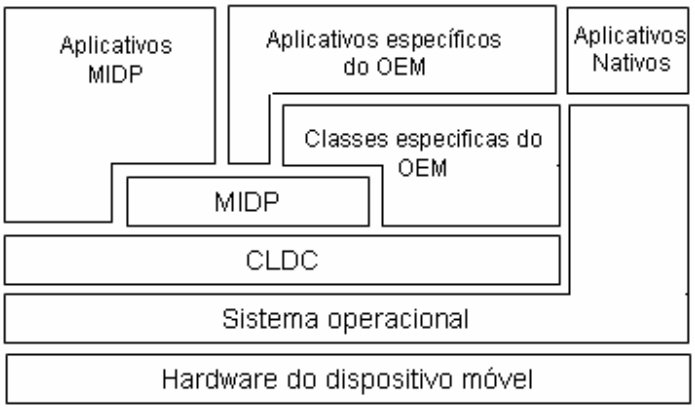

Figura 3. Arquitetura de um dispositivo de informação móvel.

Na camada inferior temos o hardware do dispositivo móvel. Na camada acima é implementado o sistema operacional nativo do aparelho. Os primeiros dispositivos móveis, antes de adotarem a plataforma J2ME, possuíam apenas os aplicativos nativos que executam através do sistema operacional nativo, eles podem ser observados no canto superior direito da Figura 3.

A base do perfil para dispositivos de informação móvel é a Configuração de Dispositivo Conectado Limitado ou CLDC. A cima da CLDC temos a implementação do MIDP. No canto superior esquerdo da Figura 3 temos os aplicativos MIDP que podem utilizar as bibliotecas da CLDC e do MIDP.

Os fabricantes de dispositivos adicionam bibliotecas privadas ao aparelho, como por exemplo: acesso à agenda de contatos, capacidade de realizar e/ou receber chamadas entre outras. Na Figura 3 estas bibliotecas específicas de fabricantes de aparelhos é observada na camada de classes específicas do OEM (*Original Equipment Manufacturer*). Como estas classes são específicas para cada dispositivo, uma aplicação desenvolvida com as mesmas deixa de lado a portabilidade para outros aparelhos. As aplicações desenvolvidas com classes específicas de fabricantes também podem utilizar as bibliotecas especificadas no MIDP.

3.2.2 Arquivos JAR (*Java Archive*) e JAD (*Java Application Descriptor*)

Uma aplicação antes de ser liberada para produção deve ser empacotada. Neste pacote existem vários arquivos, além das classes Java existem as imagens, arquivos de propriedades entre outros. O arquivo responsável por empacotar todos os arquivos é chamado de JAR ou Arquivo Java.

Para manter a organização sabemos que quando criamos um pacote com várias informações devemos descrever em algum lugar um resumo de quais informações estão contidas no pacote. O arquivo JAR também tem um arquivo chamado de manifesto (*manifest.mf*) responsável por descrever o conteúdo do Arquivo Java. A Tabela 1 descreve a lista de atributos que podem existir no manifesto.

Tabela 1. Atributos do arquivo manifesto [9].

Atributo	Descrição	Obrigatório
MIDlet-Name	Nome do conjunto de MIDlets.	X
MIDlet-Version	Número da versão da aplicação.	X
MIDlet-Vendor	Nome de quem desenvolveu a aplicação.	X
MIDlet-<n>	Este atributo deve descrever três informações: o nome da MIDlet, o ícone (opcional) e o nome da classe que o Gerenciador de Aplicativos irá chamar pra carregar essa MIDlet.	X
MicroEdition-Profile	Perfil J2ME utilizado pela aplicação.	X
MicroEdition-Configuration	Configuração J2ME utilizada pela aplicação.	X
MIDlet-Icon	Imagem (PNG) usada pelo Gereciador de Aplicativos para representar a MIDlet.	

Além do arquivo JAR o pacote de uma aplicação deve conter um arquivo JAD, ou Descritor de Aplicação Java, responsável por fornecer informações sobre a aplicação. Parâmetros adicionais podem ser passados, através do JAD, para a MIDlet[1] sem que o arquivo JAR seja alterado. A extensão deste arquivo é *.jad.*

Na Tabela 2 temos a descrição dos atributos para o arquivo JAD, lembrando que o desenvolvedor pode criar seus próprios parâmetros.

Tabela 2. Atributos do arquivo JAD [9].

Atributo	Descrição	Obrigatório
MIDlet-Name	Nome do conjunto de MIDlets.	X
MIDlet-Version	Número da versão da aplicação.	X
MIDlet-Vendor	Nome de quem desenvolveu a aplicação.	X
MIDlet-<n>	Deve descrever o nome da MIDlet, o ícone (opcional) e o nome da classe que o Gerenciador de Aplicativos irá chamar pra carregar essa MIDlet.	X
MIDlet-Jar-URL	URL do arquivo JAR.	X
MIDlet-Jar-Size	Tamanho, em *bytes*, do arquivo JAR.	X
MIDlet-Data-Size	Número mínimo de *bytes* necessário para armazenamento de dados persistentes.	
MIDlet-Description	Descrição da aplicação.	

Uma restrição dada pelo Gerenciador de Aplicativos é que os atributos MIDlet-Name, MIDlet-Version e MIDlet-Vendor devem conter o mesmo valor no manifesto do JAR e no Descritor de Aplicação Java. Para acessar, dentro da MIDlet, atributos declarados no arquivo JAD utilizamos o método *javax.microedition.midlet.MIDlet.getAppProperty(String name)*, onde o parâmetro *name* é o nome do atributo no JAD.

3.3 Fundamentos da Programação com MIDP

Esta Seção mostrará algumas das principais classes da API do Perfil de Dispositivo de Informação Móvel.

3.3.1 A MIDlet

"Uma MIDlet é um aplicativo Java projetado para ser executado em um dispositivo móvel" [9]. Geralmente são aplicações para ser executadas em dispositivos de informação

[1] Aplicação escrita em Java, utilizando o perfil MID, para executar em dispositivos móveis.

móvel, como jogos, calculadoras entre outros. Uma MIDlet deve executar em qualquer dispositivo que implemente o Perfil de Dispositivo de Informação Móvel.

Como todos os programas Java, uma MIDlet é portável, sendo feita para rodar em várias plataformas. A seguir temos alguns requisitos que devem ser seguidos para uma MIDlet executar em um dispositivo móvel:

- A classe central da aplicação deve herdar de *javax.microedition.midlet.MIDlet*
- Uma MIDlet deve ser empacotada em um arquivo JAR (*Java Archive*) nas próximas seções veremos como isto pode ser feito.

O Gerenciador de Aplicativos é a aplicação do dispositivo móvel responsável por instalar, executar e remover MIDlets. Quando um dispositivo móvel está executando uma MIDlet e recebe uma ligação, é responsabilidade do Gerenciador de Aplicativos solicitar a paralisação ou finalização do MIDlet.

Uma aplicação desenvolvida para um dispositivo com o perfil MID pode se encontrar em três estados: em atividade, em pausa e destruído. Uma MIDlet tem uma mudança de estado notificada através do Gerenciador de Aplicativos. Estas notificações ocorrem através da chamada de três importantes métodos: o *startApp()*, o *pauseApp()* e o *destroyApp()*.

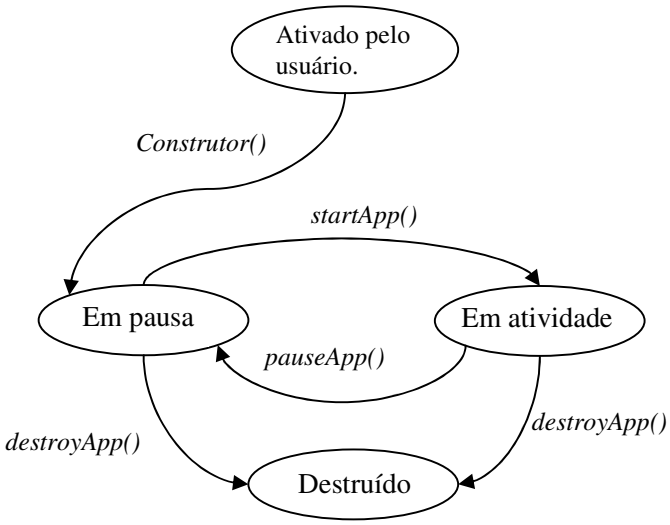

Figura 4. Ciclo de vida de uma MIDlet.

O método *startApp()* é chamado quando a MIDlet está sendo iniciada, após a chamada deste método podemos dizer que a aplicação encontra-se em atividade. Caso algum processo de maior prioridade seja iniciado pelo dispositivo, como o recebimento de uma ligação ou de uma mensagem de texto, o Gerenciador de Aplicativos é responsável por notificar a MIDlet através da chamada do método *pauseApp()*. Neste momento a aplicação J2ME estará no estado de pausa. Após a execução do processo de maior prioridade o Gerenciador de Aplicativos pode reiniciar a MIDlet chamando novamente o *startApp()*. A seguir temos algumas observações importantes que devem ser consideradas no desenvolvimento destes métodos:

- O *startApp()* não pode conter processamentos complexos, caso isto aconteça a MIDlet pode levar um bom tempo para iniciar.

- O método *pauseApp()* deve ser responsável por salvar o estado da aplicação antes da mesma entrar no estado de pausa.

- O método *startApp()* deve controlar o reinício de uma MIDlet, carregando o estado anterior à pausa, se necessário.

Por fim, temos o método *destroyApp()*, chamado através do Gerenciador de Aplicações quando a MIDlet é finalizada. Este método deve liberar os recursos que estavam sendo utilizados pela aplicação. Na Figura 4 temos a representação dos estados envolvidos no ciclo de vida de uma MIDlet.

Podemos observar na Figura 5 um exemplo de implementação de uma MIDlet. Na linha 3 o código define uma classe publica chamada *Exemplo* que por sua vez herda da classe *MIDlet* definida no pacote *javax.microedition.midlet*. Na linha 8 definimos o construtor da MIDlet. Em seguida definimos os métodos *startApp()*, *pauseApp()* e *destroyApp()* nas linhas 14, 21 e 28 respectivamente.

```
1  import javax.microedition.midlet.MIDletStateChangeException;
2
3  public class Exemplo extends javax.microedition.midlet.MIDlet {
4
5      /**
6       * Construtor publico. (opcional)
7       */
8      public Exemplo() {
9      }
10
11     /**
12      * Chamado pelo Gerenciador de aplicativos para iniciar uma MIDlet
13      */
14     protected void startApp() throws MIDletStateChangeException {
15         // TODO Auto-generated method stub
16     }
17
18     /**
19      * Chamado pelo Gerenciador de Aplicativos antes da pausa da MIDlet
20      */
21     protected void pauseApp() {
22         // TODO Auto-generated method stub
23     }
24
25     /**
26      * Chamado pelo Gerenciador de Aplicativos antes da finalização da MIDlet.
27      */
28     protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
29         // TODO Auto-generated method stub
30     }
31 }
```

Figura 5. Exemplo de implementação de uma MIDlet.

Durante o desenvolvimento de aplicações Java é importante conhecer o que as bibliotecas da linguagem podem fornecer, para isto descreveremos, ao longo deste capítulo os métodos de algumas classes envolvidas no Perfil MID. Começaremos mostrando outros métodos da MIDlet os quais ainda não foram descritos nesta seção.

- *final void notifyDestroyed()*: Chamado quando a aplicação deseja ser finalizada.
- *final void notifyPaused()*: Chamado quando a aplicação deseja fazer uma pausa.
- *final void resumeRequest()*: Chamado quando a aplicação esta pedindo para se tornar ativa, ocorre após uma pausa.
- *final String getAppProperty(String key)*: Recupera atributos dos arquivos JAR e/ou JAD.

3.3.2 Os objetos *Display* e *Displayable*

Para cada MIDlet temos um objeto *Display*. Através deste objeto podemos recuperar informações importantes a respeito da tela do dispositivo. O objeto *Display* é o objeto responsável por gerenciar as telas de uma MIDlet, definindo o que é mostrado e quando. Apenas objetos que sejam subclasse de *Displayable* podem ser exibidos no objeto Display. Mostraremos mais adiante quais objetos podem ser mostrados.

O objeto *Display* pode ser recuperado através de uma chamada ao método estático da classe *Display*. A Figura 6 é um exemplo de como poderíamos recuperar este objeto dentro da MIDlet. Na linha 1 temos um *import* necessário para utilizar a classe *Display*. Na linha 9 é definido um atributo global para o objeto *display*. Este atributo recebe um valor no construtor da classe na linha 17, através do método estático *getDisplay* da classe *Display*.

```
1  import javax.microedition.lcdui.Display;
2  import javax.microedition.midlet.MIDletStateChangeException;
3
4  public class Exemplo extends javax.microedition.midlet.MIDlet {
5
6      /**
7       * Referencia única para o objeto Desplay.
8       */
9      private Display display;
10
11     /**
12      * Construtor publico. (opcional)
13      */
14     public Exemplo() {
15         // Chamada ao método estatico passando o próprio MIDlet como
16         // parâmetro.
17         display = Display.getDisplay(this);
18         ...
19     }
20     ...
```

Figura 6. Recuperando o objeto *Display*.

Alguns métodos envolvidos na classe *Display* do pacote *javax.microedition.lcdui* são:

- *staticDisplay getDisplay(MIDlet m)*: responsável por recuperar o objeto *Display* da MIDlet m.

- *Displayable getCurrent()*: recupera o objeto *Displayable* corrente (um *Displayable* é qualquer objeto que pode ser mostrado na tela).

- *void setCurrent(Alert alert, Displayable displayable)*: primeiramente mostra um alerta e em seguida o objeto *displayable* passado como parâmetro.

- *void setCurrent(Displayable displayable)*: apresenta o objeto *displayable* passado como parâmetro.

- *boolean isColor()*: verifica se o dispositivo suporta ou não cor.

- *int numColors()*: indica quantas cores o *diplay* suporta.

Observamos que o objeto *Display* é único para uma MIDlet. Apesar disto, ele pode exibir uma quantidade indefinida de objetos *Displayable*.

Existem duas subclasses de *Displayable*. A subclasse *Screen* define todos os componentes de interface com o usuário de alto nível, isto quer dizer que os componentes já possuem várias funcionalidades implementadas, precisando apenas de uma configuração antes de ser usado. A segunda subclasse é o objeto *Canvas* que é usada para criação de interface com o usuário de baixo nível, neste caso o desenvolvedor se envolve com a pintura de *pixels* na tela. Descreveremos em mais detalhes interfaces com o usuário na Seção 3.5.5.

Em um *Displayable* podemos adicionar ou remover comandos (*Command*), são eles os responsáveis por fornecer um meio de comunicação com o usuário. Abaixo temos a lista de métodos da classe *Displayable*:

- *void addCommand(Command command)*: adiciona o *command* no objeto.

- *void removeCommand(Command command)*: remove o *command* passado como parâmetro do objeto.

- *void setCommandListner(CommandListner l)*: adiciona o *CommandListner* responsável por escutar os eventos dos comandos.

- *boolean isShown()*: Verifica se o objeto está visível na tela.

3.3.3 Tratamento de Eventos

Toda aplicação deve ter uma forma de tratar as ações do usuário. Não poderia ser diferente para uma MIDlet. Esta Seção tem como objetivo mostrar como ocorre o processamento de eventos e as classes que auxiliam no tratamento de eventos. Iremos nos concentrar nos eventos de interface com o usuário de alto nível, pois será usada no estudo de caso descrito no Capítulo 5.

Podemos generalizar o tratamento de eventos como sendo a identificação de um evento ocorrido juntamente com uma ação a ser tomada baseada no evento. Por exemplo, quando o usuário clica no botão "menu", o evento é o clique do botão "menu" e a ação correspondente deve ser mostrar a tela de menu.

Para deixar mais claro, vamos dividir o processamento de eventos em três etapas. Na primeira etapa, de mais baixo nível, temos o *hardware* identificando qual botão foi pressionado. Em uma segunda etapa o *software* do dispositivo deve ser notificado da ocorrência do evento, o Gerenciador de Aplicativos recebe uma instrução com a informação de qual botão foi pressionado. A última etapa ocorre quando o Gerenciador de Aplicativos notifica a MIDlet de qual tecla foi pressionada. Nesta etapa ocorre a ação implementada pelos desenvolvedores para o tratamento do evento iniciado pelo usuário.

A MIDlet deve configurar o receptor de eventos; isto ocorre através da implementação de uma interface receptora especificada no MIDP, *CommandListner*. O desenvolvedor deve decidir qual classe será responsável pelo processamento dos eventos fazendo com que a classe implemente a interface *CommandListener* e, conseqüentemente, o método *commandAction()*, método chamado pelo Gerenciador de Aplicativos para notificação dos eventos.

Antes de detalharmos como ocorre o tratamento de eventos através do método *commandAction()* descreveremos a classe *Command*.

Command

O objeto *Command* pode ser definido como um botão o qual o usuário seleciona, é neste objeto que estão as informações sobre o evento. Para processar um evento você deve inicialmente seguir os seguintes passos:

1. Criar um objeto *Command* contendo informações sobre um evento.
2. Adicionar o *Command* criado a uma subclasse de *Displayable*. Como mencionado anteriormente iremos focar nas interfaces de alto nível, ou seja, subclasses de *Screen*.
3. Adicionar ao *Displayable* um receptor, responsável pelo tratamento dos eventos.

Ao receber um evento, o Gerenciador de Aplicativos notifica o receptor, responsável por tratar eventos, chamando o método *commandAction(Command c, Displayable d)*. Este método recebe como parâmetro o comando selecionado pelo usuário e a tela (o *Displayable*) corrente no momento da seleção. Na Figura 7 podemos observar algumas linhas de código que descrevem como poderíamos tratar um evento para finalizar a

aplicação, como um botão "Sair" selecionado pelo usuário. Na linha 15 definimos um atributo *tela,* do tipo *Form*, que representa a tela da aplicação. A linha 20 define um botão que será adicionado à tela, é através deste botão que o usuário irá finalizar a aplicação. As linhas 23 e 24 iniciam a tela e o botão respectivamente. A tela é iniciada com o nome *Exemplo de Tela* já o botão terá o nome *Sair*. Na linha 25 adicionamos o botão *botaoSair* à *tela*. Agora devemos configurar o *CommandListener* (linha 26) da aplicação, deve ser a classe que implementa o método *commandAction* (linha 29). No caso da Figura 7 é a própria classe e referenciamos através do *this* (linha 26). Por último temos a implementação do *commandAction* (linha 29). Se a tela corrente for igual a *tela* e se o botão selecionado pelo usuário for igual a *botaoSair* o sistema chamará os métodos das linhas 32 e 33 para finalizar a aplicação.

```
12      /**
13       * Tela da aplicação
14       */
15      private Form tela;
16
17      /**
18       * Botão para sair da aplicação
19       */
20      private Command botaoSair;
21      ...
22      ...
23      tela = new Form("Exemplo de Tela"); // Criando a tela
24      botaoSair = new Command("Sair", Command.EXIT, 1); // Criando o botão
25      tela.addCommand(botaoSair); // Adiciona o botão à tela
26      tela.setCommandListener(this); // Configura o escutador.
27      ...
28      ...
29      public void commandAction(Command c, Displayable d) {
30          if (d == tela) {
31              if (c == botaoSair) {
32                  destroyApp(true);
33                  notifyDestroyed();
34              }
35          }
36      }
```

Figura 7. Exemplo de código para tratamento de eventos.

Na criação de um novo botão são necessários três parâmetros: o rótulo, o tipo do comando e sua prioridade. O rótulo é o texto que representa o *Command* na tela. A Figura 8 apresenta um *Command* com o rótulo "Sair". O tipo mapeia um objeto *Command* diretamente a um botão relevante em um dispositivo, podemos observar na Tabela 3 alguns tipos comandos definidos pela API. Por último temos a prioridade, definida pelo desenvolvedor, quanto maior o número menor é a prioridade do *Command*. A prioridade é útil para o Gerenciador de Aplicativos para ordenar os botões na tela do dispositivo.

```
botaoSair = new Command("Sair", Command.EXIT, 1);
                        |          |          |
                      rótulo      tipo     prioridade
```

Figura 8. Exemplo de apresentação de um *Command* no simulador.

Tabela 3. Tipos de *Command* definidos pela plataforma.

Tipo	**Descrição**
BACK	Para voltar à tela anterior.
CANCEL	Para pedido de cancelamento de algum processamento.
EXIT	Para sair da aplicação.
HELP	Quando o usuário necessita de informações de ajuda.
OK	Para uma confirmação do usuário.

Tanto a prioridade como o tipo de um botão não serão necessariamente respeitados, quem define o mapeamento real e a prioridade atribuída é o fabricante do dispositivo.

3.3.4 Interface com o usuário

Para interface com o usuário a plataforma J2ME permite que seja utilizado qualquer objeto que seja uma subclasse de *Displayable*. As duas classes principais que definem as interfaces de alto e baixo nível são: *Screen* e *Canvas* respectivamente. Este trabalho utilizará os objetos de interface com o usuário de alto nível. Na Figura 9 podemos observar o diagrama de classes para interface com o usuário de alto nível.

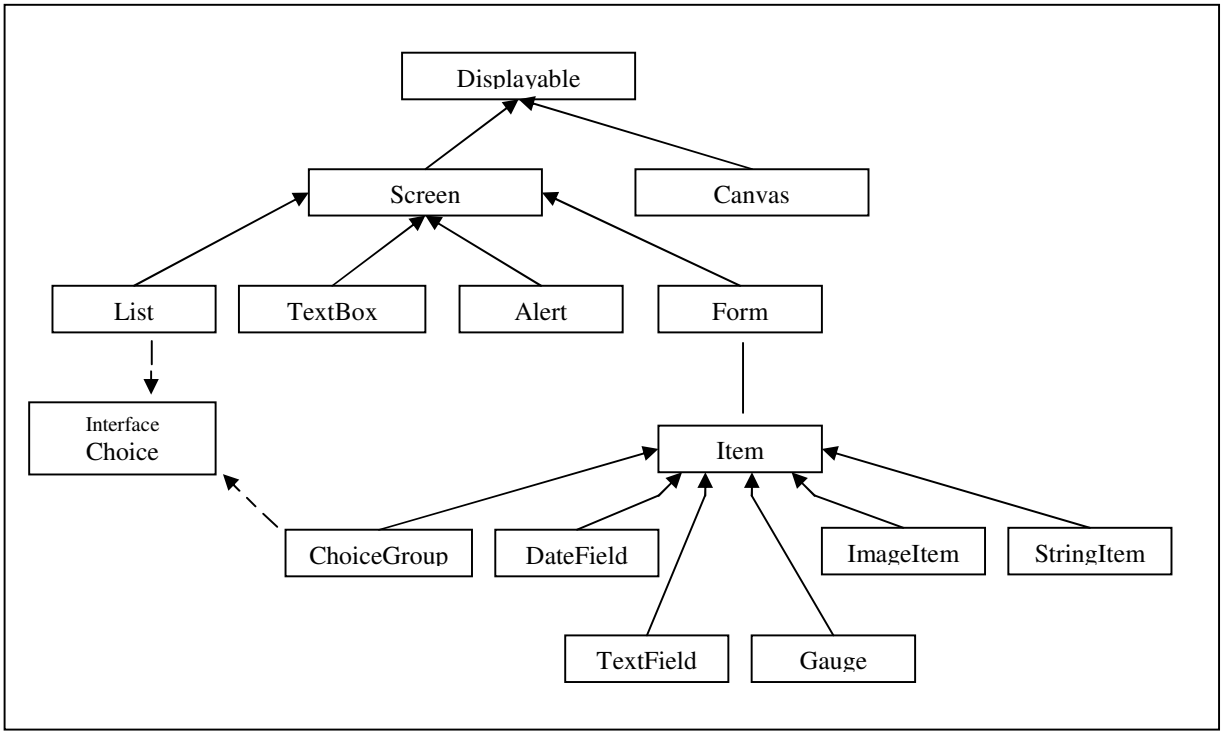

Figura 9. Diagrama de classes para interface com o usuário de alto nível.

Agora veremos uma breve descrição de cada uma das classes do diagrama de classes para interface com o usuário de alto nível.

- *Screen*: esta classe não pode ser visualizada. Esta é a classe pai de todos os componentes que têm uma aparência e um comportamento na tela;

- *Form*: este objeto pode ser considerado um contêiner, para quem conhece a plataforma Java para computadores pessoais, sendo possível adicionar vários componentes de tela. Estes componentes devem ser subclasses de Item. A implementação do *Form*, como a maioria dos componentes de alto nível, já possui uma barra de rolagem implementada;

- *Item*: é um componente que pode ser adicionado em um *Form*. Os componentes *ChoiceGroup, DateField, Gauge, ImageItem, StringItem* e *TextField* são subclasses de *Item* e podem ser adicionado a um formulário, descreveremos cada um deles;

- *DateField*: este componente é usado para o usuário selecionar datas. Na Figura 10 podemos observar exemplo deste componente;

Figura 10. Exemplo visual do componente *DateField*.
Utilizado para atualizar data e hora.

- *Gauge*: quase toda aplicação utiliza indicadores de progresso para uma determinada atividade, um exemplo comum é quando estamos baixando um arquivo da Internet ou durante o processo de instalação de alguma aplicação. O *Gauge* deve ser utilizado quando se precisa de uma barra indicadora de progresso;

- *StringItem*: é um componente estático na visão do usuário. O desenvolvedor define um rótulo e um texto. Um exemplo seria: "Nome: Marina" onde o "Nome" é o rótulo e "Marina" é o texto do componente.

- *TextField*: é basicamente um rótulo mais uma caixa de texto onde o usuário pode entrar com informações textuais ou numéricas dependendo do filtro que já vem implementado no componente. O desenvolvedor pode definir um filtro para restrição como, por exemplo, receber apenas entradas numéricas. Alguns filtros pré-definidos no *TextField* são: para endereço de e-mail, para URL's, valores numéricos, telefones e senhas. A Tabela 4 descreve os filtros para o *TextField*. Eles podem ser: *ANY* para qualquer caractere; *EMAILADDR* para caracteres válidos em e-mails; *NUMERIC* permitindo apenas números; *PHONENUMBER* para caracteres válidos em números de telefones e *URL* para caracteres válidos em URL's.

Tabela 4. Filtros com restrições definidos para o *TextField*.

Filtro	Descrição
ANY	Permiti a entrada de qualquer caractere.
EMAILADDR	Permite apenas caracteres válidos para e-mail.
NUMERIC	Permite apenas entrada de digitos numericos.
PHONENUMBER	Só perrmite caracteres que são válidos para números de telefones.
URL	Apenas caracteres que são válidos em uma URL.

- *ChoiceGroup*: representa um grupo de opções para serem selecionadas pelo usuário, pode ser múltiplo ou exclusivo. Um *ChoiceGroup* múltiplo permite que o usuário selecione mais de uma opção, já o exclusivo permite a escolha de apenas uma opção. Esta classe implementa a interface *Choice*, a qual define métodos relacionados à manipulação de vários tipos de seleções predefinidas;

- *ImageItem*: permite que o desenvolvedor defina o posicionamento de uma imagem, por exemplo, centralizada horizontalmente, à esquerda ou à direita. Este objeto é responsável por encapsular uma imagem, originada da classe *javax.microedtion.lcdui.Image* para que seja apresentada em um formulário;

- *TextBox*: responsável por receber entrada de texto do usuário, pode conter várias linhas e permite filtrar a entrada do usuário para somente texto ou somente números. Como outros componentes de interface de alto nível o *TextBox* adiciona uma barra de rolagem caso a quantidade de caracteres exceda o tamanho da tela. Este componente é uma subclasse de Screen, podendo ser apresentado diretamente na tela;

- *List*: utilizada para apresentar uma lista de opções na tela. Como o *ChoiceGroup* pode ser múltiplo ou exclusivo. Este componente não possui dependência com outros componentes e pode ser apresentado diretamente na tela;

- *Alert*: é uma caixa de diálogo que pode apresentar um texto e/ou um objeto *Image*. Geralmente utilizado para mostrar uma mensagem de aviso ou erro para o usuário. Esta mensagem apresentada pode ser intermitente, onde um comando do usuário é esperado para sair da tela, ou temporária, com um tempo predefinido para ser apresentada.

3.4 Resumo

Neste Capítulo apresentamos uma visão geral da plataforma J2ME, discutindo as Configurações e os Perfis definidos para a plataforma. Mostramos o porquê de utilizarmos a tecnologia J2ME.

Estudamos detalhes da Configuração de Dispositivo Conectado Limitado (CLDC) e do Perfil de Dispositivo de Informação Móvel (MIDP), os quais serão utilizados em nosso estudo de caso.

Por fim, mostramos os fundamentos da programação com MIDP, apresentando as principais classes necessárias para criarmos uma aplicação simples com o perfil MID. Também foi abordado como o ocorre o tratamento de eventos em uma aplicação e os objetos de interface, de alto nível, com o usuário.

Capítulo 4

Engenharia de Software Experimental

A experimentação está entre as atividades a ser desenvolvidas nas áreas de pesquisas. Com os experimentos podemos realizar críticas a uma determinada teoria e assim sugerir melhorias. Engenharia de Software Experimental é um subdomínio da Engenharia de Software e seu foco são os experimentos em sistemas (produtos, processos e recursos) [11]. A partir da Engenharia de Software Experimental podemos rever e criticar teorias e cada vez mais trazer melhorias para o processo de desenvolvimento de software. Apesar da importância, a área de engenharia de software experimental ainda não é o foco para muitos engenheiros de sistemas [30].

Este capítulo tem o objetivo de introduzir os principais conceitos da Engenharia de Software Experimental, baseados no Relatório Técnico escrito por Guilherme Travassos [11]. Iremos utilizar muitos conceitos deste capitulo quando mostrarmos o plano de experimento no Capítulo 5, referente ao estudo de caso.

4.1 Objetivos da Experimentação

A execução de experimentos em Engenharia de Software tem como objetivos caracterizar, avaliar, prever, controlar e melhorar produtos, modelos, teorias entre outros. Dependendo do objetivo em questão o nível de importância e esforços do experimento podem aumentar ou diminuir, por exemplo: um experimento com o objetivo de caracterizar é mais simples e requer menos esforço do que um experimento com um objetivo de avaliar. Em um experimento com o objetivo de caracterizar um produto ou processo teremos que responder perguntas do tipo: "o que está acontecendo?". Entretanto quando estamos realizando um experimento para avaliar um processo ou produto temos que responder perguntas do tipo: "quão bom é o produto?".

4.2 Descrição geral da experimentação

Veremos nesta seção grande parte do vocabulário utilizado na Engenharia de Software experimental, alguns princípios de organização do experimento, como podemos realizar a medição do experimento e a validade dos mesmos.

4.2.1 Vocabulário da experimentação

Entre os principais elementos de um experimento podemos citar: as variáveis, os objetos, os participantes, o contexto do experimento, hipóteses e o tipo de projeto do experimento. A seguir veremos cada um destes elementos.

Em um experimento existem dois tipos de variáveis: dependentes e independentes. As entradas para um processo de experimentação são chamadas variáveis independentes ou fatores; eles afetam o resultado de um experimento. As saídas de um experimento representam as variáveis dependentes.

A Figura 11 apresenta um relacionamento dos conceitos descritos acima.

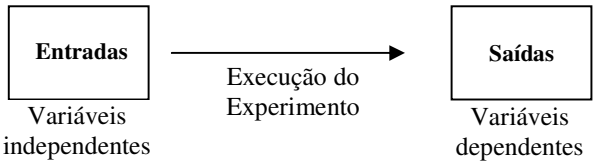

Figura 11. Conceitos de variáveis de um sistema.

O objeto de um experimento é o alvo do processo; entradas serão aplicadas ao conjunto de objetos do experimento gerando assim a saída ou resultado para serem analisados.

Os participantes são os indivíduos selecionados numa população os quais irão conduzir o experimento. É importante que a quantidade de participantes seja suficiente para representar a população. Quanto maior é a variedade de uma determinada população maior deve ser a quantidade de participantes envolvidos no experimento.

O ambiente e as condições em que o experimento está sendo executado definem o contexto do experimento. Caso o experimento seja executado em um contexto específico, os resultados serão válidos apenas para aquele determinado contexto e não para o domínio da Engenharia de Software como um todo.

São as hipóteses que geralmente formulam um experimento. A hipótese nula é a principal das hipóteses, o objetivo dela é dizer que não existe nenhum relacionamento estatístico entre causa e efeito. O experimento tem como principal objetivo rejeitar a hipótese nula, aceitando alguma ou algumas das hipóteses alternativas. Em nosso estudo de caso veremos um exemplo das hipóteses nula e alternativas.

O projeto do experimento determina a maneira como um experimento será conduzido. A maneira como as variáveis independentes serão aplicadas aos objetos é definida no projeto do experimento. A quantidade e a seqüência dos testes experimentais definem o projeto do experimento.

4.2.2 Medição

Um estudo experimental tem como parte central a medição dos experimentos. O mapeamento do mundo experimental para o mundo formal é quem define a medição. O objetivo principal é caracterizar os resultados dos experimentos. O julgamento de um experimento não será feito em cima das entidades reais, mas sim a partir dos números ou símbolos atribuídos a estas entidades. O número ou símbolo atribuído a uma determinada entidade chama-se medida.

As medidas podem ser classificadas como: objetiva ou subjetiva e direta ou indireta. Quando a medida depende apenas do objeto em questão ela é dita objetiva. Uma medida objetiva pode ser recuperada diversas vezes e o valor sempre será o mesmo. Já medida subjetiva pode tomar diferentes valores quando o objeto é medido várias vezes, isto acontece pois a medida subjetiva depende do objeto e ao mesmo tempo da perspectiva na qual o valor foi tomado. A medida que não envolve a medição de outros atributos é chamada medida direta. Quando a medida deriva das medições de outros atributos dizemos que a medida é indireta.

Muitas vezes a medição na Engenharia de Software utiliza as próprias métricas do software, por exemplo, as métricas do processo medem as características do processo do desenvolvimento de software. As métricas de recurso são usadas pra medir os objetos tais como equipe, hardware entre outros.

Na Engenharia de Software o principal objetivo da medição é aumentar o entendimento do processo e do produto, definir com antecedência atividades corretivas e identificar possíveis pontos de melhorias.

4.2.3 Validade dos Experimentos

Um assunto muito importante a respeito dos experimentos é quão válido são os resultados do mesmo. Os resultados devem ser válidos para uma determinada população e contexto. Podemos citar quatro tipos de validade dos resultados em um experimento: validade de conclusão, validade interna, validade de construção e a validade externa.

A validade de conclusão está relacionada com a habilidade de chegar a uma conclusão correta a respeito dos relacionamentos entre as variáveis independentes ou entradas do experimento e o resultado do experimento.

Quando a relação entre as variáveis independentes e o resultado é casual dizemos que o experimento possui uma validade interna. Durante a avaliação desta validade devemos ter uma maior atenção aos participantes do experimento, ou seja, quais foram os participantes selecionados da população. Outro ponto importante que se deve ter atenção é como foram aplicadas as entradas ao experimento. O risco contido neste tipo de validade está ligado aos participantes.

O relacionamento entre a teoria e a observação está ligado à validade de construção, é necessário saber se as entradas refletem bem a causa e se as saídas refletem bem o efeito. Nesta avaliação devem ser considerados os aspectos relevantes ao projeto e os fatores humanos. Problemas com esta avaliação podem ocorrer por causa dos participantes ou de quem esta realizando o experimento.

A validade externa especifica um conjunto de condições para limitar a generalização dos resultados de um experimento. No processo de avaliação desta validade as condições do ambiente devem ser consideradas. Podem ocorrer problemas devido ao uso incorreto da instrumentação ou o experimento pode ser realizado em um tempo especial que venha a afetar os resultados.

Geralmente a ordem da importância dos tipos de validade, em experimentos na área de Engenharia de Software é: interna, externa, de construção e conclusão.

4.3 Tipos de Experimentos

Como a experimentação é uma abordagem relativamente nova na área de Engenharia de Software existem muitas classificações para os experimentos. Geralmente a literatura apresenta três principais tipos:

1. *Survey*
2. Estudo de Caso
3. Experimento

Nesta seção descreveremos cada um destes tipos. O pesquisador deve escolher a estratégia que melhor se enquadre ao seu estudo. Veremos cada um dos tipos em detalhes.

O *Survey*

Pode ser definido como um estudo sobre determinado assunto para, por exemplo, descrever uma tecnologia e descobrir suas vantagens e desvantagens. Este tipo de experimento deve ser realizado quando algumas técnicas ou ferramentas já tiverem sido utilizadas.

Dentre os principais objetivos de um *Survey* podemos citar:

- Descrever as características do objeto em estudo.

- Explicar a razão de se utilizar o objeto em questão, mostrando suas vantagens e desvantagens.
- Explorar profundamente com o objetivo de encontrar novidades.

Uma forma de coletar informações no início de um *Survey*, é através de questionários. Este tipo de abordagem tem a habilidade de mostrar um grande número de variáveis para serem analisadas.

Estudo de Caso

Utilizado quando se pretende monitorar projetos, atividades e atribuições. Os estudos de casos têm o objetivo de investigar um atributo específico. Geralmente é a melhor abordagem quando as questões centrais são o "como" e o "porque".

O Estudo de Caso possui um nível de controle menor que o *Survey*, mas ao contrário deste último possui o controle sobre a medição das variáveis. O mais difícil em um Estudo de Caso é diferenciar um efeito causado por outro fator que por sua vez foi proveniente de um terceiro fator, este problema é conhecido como fatores de confusão ou *confounding factors*.

O Experimento

Realizado geralmente em laboratório, possui o maior nível de controle dentre as outras abordagens. O objetivo de um experimento é medir o efeito causado pela manipulação de algumas variáveis. Existem duas formas de executar um experimento. A primeira é sob condições de laboratório (*in-vitro*), a segunda forma é sob condições normais (*in-vivo*).

Os experimentos possuem um custo maior para ser executado e deve ser utilizado para confirmar teorias, conhecimento convencional, validar medidas, entre outros. Uma grande vantagem do experimento é a possibilidade de repetição do mesmo.

A Tabela 5 mostra uma comparação das estratégias experimentais.

Tabela 5. Comparação das estratégias empíricas [11].

Fator	***Survey***	**Estudo de Caso**	**Experimento**
O controle da execução	Nenhum	Nenhum	Tem
O controle da medição	Nenhum	Tem	Tem
O controle da investigação	Baixo	Médio	Alto
Facilidade da repetição	Alta	Baixa	Alta
Custo	Baixo	Médio	Alto

4.4 Processo de experimentação

Esta seção descreverá os passos envolvidos no processo de experimentação. Serão apresentadas metodologias de experimentação e por fim as fases para conduzir um processo experimental.

4.4.1 Metodologias de experimentação

O experimento geralmente tem o objetivo de formular ou verificar uma teoria. Várias metodologias foram desenvolvidas para auxiliar a atingir este objetivo. Alguns itens que são descritos em uma metodologia: as fases do processo de experimentação, as ferramentas de empacotamento dos experimentos entre outras.

Um ótimo exemplo de metodologia da experimentação é o QIP (*Quality Improvement Paradigm* – Paradigma de Melhoria e Qualidade) [12]. Esta metodologia tem como objetivo a melhoria contínua do processo de software.

Um conceito importante utilizado pelo QIP é o da Fábrica da Experiência, uma unidade organizacional, formada por um conjunto de ferramentas responsáveis por armazenar informações experimentais e empacotar experimentos para serem analisados e reutilizados por outras pessoas. A Figura 12 descreve a Fábrica de Experiência, do lado esquerdo temos a organização do projeto de experimento, nesta etapa caracterizamos o projeto e seus objetivos e a escolhemos um processo. Após o plano de projeto estabelecido passamos para a etapa de execução do processo, onde os experimentos são executados. A Fábrica de Experiência é responsável por dar suporte ao plano experimental, por analisar os resultados dos experimentos e por empacotar os mesmos em bases de experiências.

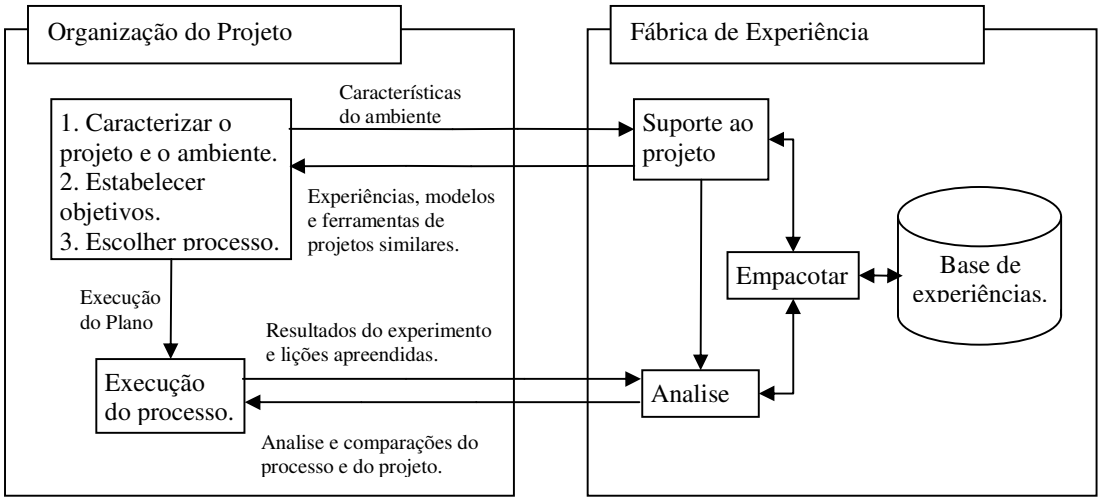

Figura 12. Fábrica de Experiência [12].

Outra metodologia de experimentação é a GQM ou *Goal/Question/Metric* (Objetivos/ Questões/Métricas) [13]. Os principais objetivos desta abordagem é

compreender, controlar e melhorar. Para todos os objetivos se deve levar em consideração o custo, o risco, o tempo e a qualidade.

A GQM é formada por quatro fases:

- Fase do planejamento: elaboração do plano de projeto. Envolve a seleção do projeto de experimento, definição, caracterização e planejamento do mesmo;
- Fase da definição: nesta fase os objetivos, as questões, as métricas e as hipóteses devem ser estabelecidos;
- Fase da coleta de dados: a coleta de dados é feita a partir da execução dos experimentos. Resulta em um conjunto de dados que devem ser interpretados;
- Fase da interpretação: os dados são analisados levando em consideração as métricas, questões e objetivos definidos.

A fase de definição utiliza uma abordagem de cima para baixo. Primeiro os objetivos são especificados, depois ocorre a elaboração das questões e em seguida a descrição das Métricas. Já a fase de interpretação utiliza uma abordagem de baixo pra cima. Primeiramente recupera o conjunto de dados, para com eles gerarem as respostas às questões elaborados na definição e por fim uma apresentação dos objetivos alcançados. A Figura 13 apresenta um diagrama da metodologia GQM.

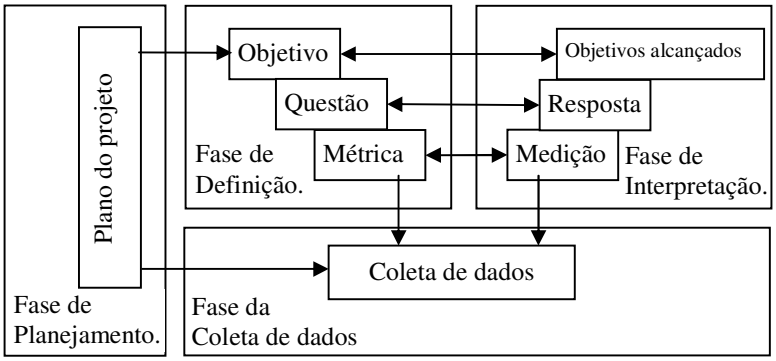

Figura 13. Diagrama da metodologia GQM [13].

4.4.2 Fases do experimento

Agora serão apresentadas as cinco fases que geralmente estão presentes em um experimento: definição, planejamento, execução, análise e interpretação e apresentação e empacotamento.

A primeira fase é a definição, especifica os objetivos, o objeto de estudo, o foco da qualidade, o ponto de vista e o contexto. Esta fase pode ser estruturada da seguinte forma:

- Analisar (Objeto do Estudo): Descreve o que será investigado ao longo do experimento. Ex.: o trabalho tem como objeto de estudo o uso de padrões de projetos [8] em aplicações desenvolvidas com a plataforma J2ME [9].

- Com o propósito de (Objetivo): Define os objetivos do experimento, como por exemplo, caracterizar, avaliar, predizer, controlar, ou melhorar. Ex.: o trabalho tem como objetivo a melhoria do processo de desenvolvimento de aplicações J2ME.

- Com respeito a (Foco da qualidade): Apresenta os aspectos de qualidade que estão sendo estudados, como por exemplo eficiência, confiabilidade, produtividade entre outros. Ex.: no contexto deste trabalho os seguintes aspectos de qualidade serão investigados: desempenho, capacidade de manutenção e confiabilidade do produto.

- Do ponto de vista (Perspectiva): Especifica o ponto de vista pelo quais os resultados serão avaliados. Ex.: os resultados dos experimentos deste trabalho serão avaliados do ponto de vista do desenvolvedor e do usuário final.

- No contexto de (Contexto): Descreve em qual ambiente o experimento está sendo executado. Um exemplo de contexto será descrito no próximo capítulo.

O planejamento é a segunda fase em um processo experimental. Esta fase é responsável pela especificação das hipóteses, definição das variáveis do experimento, seleção dos participantes, definição do projeto do experimento, consideração da validade do experimento entre outros.

Com o planejamento concluído pode-se iniciar a fase de execução do experimento. Os dados provenientes da execução do experimento devem ser coletados, sem causar efeito no processo de execução.

Os dados coletados na fase de execução agora devem ser analisados e interpretados, esta é a quarta fase em um processo de experimentação. Esta fase tem como objetivos analisar a possível rejeição da hipótese nula e a verificação das outras hipóteses.

A última fase é da apresentação e empacotamento do experimento. Nesta fase os resultados devem ser apresentados e o experimento empacotado possibilitando assim a repetição do mesmo por outros pesquisadores.

A Figura 14 apresenta um fluxograma das fases envolvidas em um processo experimental.

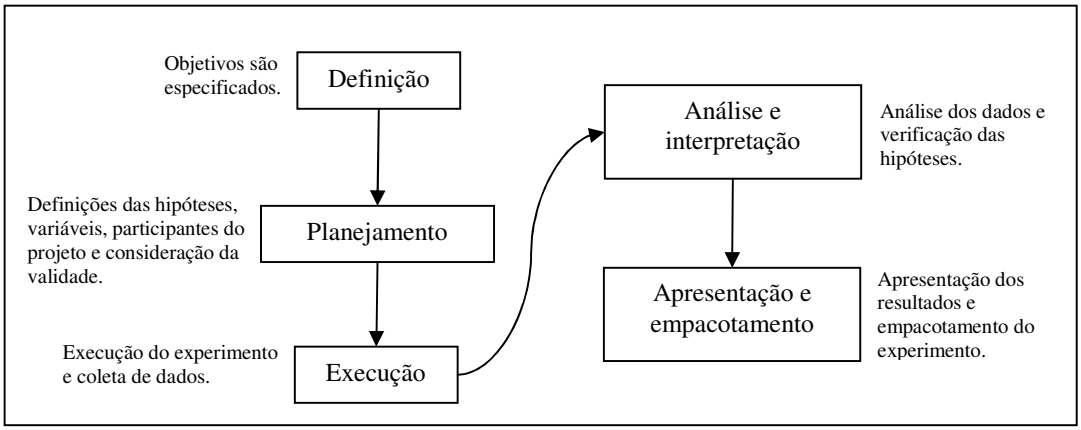

Figura 14. As cinco fases de um processo experimental.

4.5 Resumo

Este capítulo mostrou os conceitos básicos envolvidos em processos experimentais na área de Engenharia de Software. Vimos que a execução de experimentos pode ter como objetivos a caracterização, avaliação, previsão, controle e melhoria a respeito de um determinado objeto.

Estudamos três estratégias de organização e execução dos estudos experimentais: *Survey,* Estudo de Caso e Experimento. Verificamos a importância da medição e descrevemos os quatro tipos de validade para um estudo experimental: validade de conclusão, interna, externa e de construção.

Também foram abordadas metodologias para organização dos experimentos, descrevemos o Paradigma da Melhoria da Qualidade (QIP) [12] e a abordagem *Goal/Question/Metric* (GQM) [13]. Por último apresentamos cinco fases que podem constituir um processo experimental: definição, planejamento, execução, análise / interpretação e apresentação / empacotamento.

A utilização de processos experimentais na área de Engenharia de Software contribuirá com uma grande parcela para o desenvolvimento desta área. A experiência pratica é uma questão primordial para tornar a teoria um fundamento real.

Capítulo 5

Estudo Experimental

Com o objetivo de entender melhor quais as vantagens e desvantagens do uso de padrões de projetos em aplicações J2ME, faremos vários experimentos em uma aplicação J2ME. Inicialmente esta aplicação não possui padrões de projeto implementados. Ao longo do estudo, introduzimos padrões de projeto.

A escolha da aplicação foi uma das dificuldades deste trabalho, pois muitas das aplicações analisadas não condiziam com a realidade. Não seria interessante realizar os experimentos em uma "aplicação de brinquedo", pois poderíamos ter os resultados dos experimentos distorcidos. A aplicação escolhida foi desenvolvida por Bruna Bunzen [15] em seu trabalho de graduação apresentado no Departamento de Sistemas Computacionais (DSC) da Universidade de Pernambuco.

As motivações que levaram a escolha desta aplicação foram as seguintes:

- Primeiramente, a aplicação envolve a solução de um problema real. Muitas "aplicações de brinquedo" foram analisadas, mas nenhuma delas representava uma aplicação real.

- A aplicação é de interesse do Departamento de Sistemas Computacionais, desta forma este trabalho deixará uma contribuição para o mesmo.

A aplicação é responsável por obter informações sobre docentes e discentes do NUPEC (Núcleo de Pesquisa em Engenharia da Computação) da Escola Politécnica da Universidade de Pernambuco. A aplicação envolve dois módulos: um cliente, desenvolvido em J2ME, responsável por fazer as requisições; e outro servidor, desenvolvido utilizando a plataforma J2EE, responsável por responder às requisições. Estas consultas envolvem informações, disponibilizadas em uma base de dados, sobre projetos e pessoas do NUPEC.

Como o foco deste trabalho é a utilização de padrões de projeto em aplicações J2ME, os experimentos serão realizados apenas envolvendo a aplicação cliente. A aplicação servidora é responsável apenas por responder às requisições realizadas pelo cliente.

O estudo experimental foi realizado através de um plano de experimento, descrito neste capítulo, e envolveu a implementação de padrões de projetos selecionados na aplicação juntamente com a recuperação de métricas estáticas e dinâmicas, à medida que os padrões de projetos foram aplicados.

A métrica dinâmica analisada foi o desempenho da aplicação à medida que os padrões de projeto foram implementados. As métricas estáticas escolhidas foram: complexidade ciclomática definida por McCabe [17], número total de linhas de código da aplicação, número total de classes e número médio de linhas de código por método. As métricas estáticas: complexidade ciclomática e número médio de linhas de código por método; foram selecionadas com o intuito de medir a complexidade do código. Se estas métricas aumentam podemos dizer que a complexidade do código aumentou, caso contrario dizemos que o código da aplicação ficou mais legível. As outras duas métricas estáticas selecionadas: número total de linhas de código e número de classes; informam se o tamanho da aplicação aumenta ou diminui com a implementação de padrões de projetos. Esperamos, com estas variáveis, mostrar vantagens e desvantagens do uso dos padrões selecionados.

Nas próximas seções descreveremos o plano de experimento.

5.1 Definição dos Objetivos

Objetivo Global

Mostrar se é viável a utilização de alguns padrões de projetos no desenvolvimento de uma aplicação J2ME. Para isto será utilizada a aplicação J2ME desenvolvida por Bruna Bunzen em seu trabalho de graduação [15].

Objetivo da Medição

Inicialmente serão realizados alguns experimentos com a aplicação sem padrões de projetos implementados. Os resultados destes experimentos iniciais serão coletados e servirão de base para caracterizar:

1. O que acontece com a aplicação quando implementarmos os padrões de projetos selecionados?
    a. Será que a aplicação ganha ou perde desempenho?
    b. O número total de linhas de código aumenta ou diminui?
    c. O número total de classes do projeto aumenta ou diminui?
    d. O número médio de linhas de código por método aumenta ou diminui?
    e. A média das complexidades ciclomáticas aumenta ou diminui?

De maneira geral este trabalho irá analisar o desempenho da aplicação e as métricas estáticas à medida que os padrões de projetos forem implementados.

Objetivo do estudo

- **Analisar** o uso de alguns padrões de projetos no desenvolvimento de uma aplicação J2ME.
- **Com o propósito de** verificar a viabilidade do uso de alguns padrões de projetos, em aplicações J2ME.
- **Com respeito à** implementação dos padrões de projetos: *Singleton e Factory Method*.
- **Do ponto de vista** do desenvolvedor da aplicação e do usuário do sistema.
- **No contexto de** uma aplicação desenvolvida em um Projeto de Final de Curso (Engenharia da Computação, DSC, EPP).

Questões

- Q1: A implementação do padrão de projeto na aplicação diminui o desempenho dela?
  Métrica: A média do desempenho de algumas funcionalidades da aplicação.
- Q2: A implementação do padrão de projeto à aplicação aumenta o desempenho da aplicação?
  Métrica: A média do desempenho de algumas funcionalidades da aplicação.

5.2 Planejamento

Esta seção definirá as hipóteses para o experimento, também descreverá a instrumentação, o contexto no qual os experimentos estarão inseridos, a seleção dos padrões de projetos a serem implementados, as variáveis dos experimentos e a validade do mesmo.

Definição das Hipóteses

- **Hipótese nula (H0)**: O desempenho da aplicação sem padrões de projetos implementados é maior do que com padrões de projetos.
  Dsp – Desempenho sem padrões de projetos implementados
  Dcp – Desempenho com padrões de projetos implementados
  H0: Dsp > Dcp

- **Hipótese alternativa (H1)**: O desempenho da aplicação sem padrões de projetos implementados é menor do que com padrões de projetos.
  Dsp – Desempenho sem padrões de projetos implementados
  Dcp – Desempenho com padrões de projetos implementados
  H1: Dsp < Dcp

- **Hipótese alternativa (H2)**: O desempenho da aplicação sem padrões de projetos implementados é igual do que com padrões de projetos.
  Dsp – Desempenho sem padrões de projetos implementados
  Dcp – Desempenho com padrões de projetos implementados
  H2: Dsp = Dcp

Descrição da instrumentação

A medida em que os padrões de projetos forem implementados, oferecer as seguintes escolhas apresentadas na Tabela 6 para os resultados dos experimentos.

Tabela 6. Descrição da instrumentação do plano de experimento.

Desempenho na inicia-lização da Aplicação (DI)	Desempenho na 1º requisição de informações ao servidor (DR1)	Desempenho na 2º requisição de informações ao servidor (DR1)
1. Mantém-se o mesmo. 2. Aumenta 3. Diminui.	1. Mantém-se o mesmo. 2. Aumenta 3. Diminui.	1. Mantém-se o mesmo. 2. Aumenta 3. Diminui.

O resultado dos experimentos deve ser classificado em uma das métricas descrita na Tabela 7. Por exemplo, se após um experimento for verificado que o desempenho na inicialização da aplicação se manteve o mesmo e o desempenho durante 1º e 2º requisição de informações para o servidor aumentem, o resultado deste experimento será classificado na métrica 5, significando que o desempenho geral da aplicação aumentou e consequentemente a questão 2 (Q2) é verdadeira. Outro possível resultado seria: o desempenho aumentar na durante a inicialização, o DR1 diminuir e o DR2 se manter o mesmo. Este cenário nos leva à Linha 16 na Tabela 7, em sua descrição temos que o desempenho é diferente. Isto significa que não podemos concluir nem que aumentou nem que diminuiu. Neste caso não temos nenhuma questão relacionada com esta descrição, por isto na Linha 16, coluna “Questões” colocamos N/A, significando Não Aplicável.

Tabela 7. Métricas para classificação dos resultados dos experimentos.

Nº	DI	DR1	DR2	Descrição	Questões
1	1	1	1	Desempenho se mantem o mesmo.	N/A
2	1	1	2	Desempenho geral aumenta.	Q2
3	1	1	3	Desempenho geral diminui.	Q1
4	1	2	1	Desempenho geral aumenta.	Q2
5	1	2	2	Desempenho geral aumenta.	Q2
6	1	2	3	Desempenho é diferente.	N/A
7	1	3	1	Desempenho geral diminui.	Q1
8	1	3	2	Desempenho é diferente.	N/A
9	1	3	3	Desempenho geral diminui.	Q1
10	2	1	1	Desempenho geral aumenta.	Q2
11	2	1	2	Desempenho geral aumenta.	Q2
12	2	1	3	Desempenho é diferente.	N/A
13	2	2	1	Desempenho geral aumenta.	Q2
14	2	2	2	Desempenho geral aumenta.	Q2
15	2	2	3	Desempenho geral aumenta.	Q2
16	2	3	1	Desempenho é diferente.	N/A
17	2	3	2	Desempenho geral aumenta.	Q2
18	2	3	3	Desempenho geral diminui.	Q1
19	3	1	1	Desempenho geral diminui.	Q1
20	3	1	2	Desempenho é diferente.	N/A
21	3	1	3	Desempenho geral diminui.	Q1
22	3	2	1	Desempenho é diferente.	N/A
23	3	2	2	Desempenho geral aumenta.	Q2
24	3	2	3	Desempenho geral diminui.	Q1
25	3	3	1	Desempenho geral diminui.	Q1
26	3	3	2	Desempenho geral diminui.	Q1
27	3	3	3	Desempenho geral diminui.	Q1

Seleção do Contexto

O contexto de um experimento é caracterizado por quatro dimensões: o processo, os participantes, a realidade e a generalidade. Vejamos cada uma delas.

- O processo: on-line ou off-line. Em nosso estudo o processo é on-line, pois as variáveis são recuperadas à medida que os padrões de projetos são implementados, e não em um determinado instante e congelados.

- Os participantes: são os padrões de projetos selecionados a serem implementados.

- A realidade: pode ser real ou modelada. Em nosso caso o estudo é modelado, pois as medidas de desempenho são recuperadas executando a aplicação em um simulador, e não em um aparelho real.

- Generalidade: é a última dimensão e pode ser específica ou geral, como os resultados são recuperados e analisados apenas em uma aplicação, então, nosso contexto é específico.

Seleção dos Padrões de Projetos a ser implementados

Inicialmente um questionário foi elaborado (ver Apêndice A) com a intenção de saber quais dos padrões de projetos estudados são mais utilizados e quais trazem mais benefícios para as aplicações desenvolvidas em um domínio específico. O questionário foi distribuído para engenheiros de algumas empresas do Porto Digital [16].

A pesquisa foi realizada com 11 engenheiros que desenvolveram ou desenvolvem aplicações em J2ME. Após uma análise das respostas, verificamos que 100% das pessoas entrevistadas acham válido o uso de padrões de projetos no desenvolvimento de aplicações J2ME. Destas, 81,81% já utilizaram padrões de projetos no desenvolvimento de aplicações J2ME e apenas 18,19% das pessoas não utilizaram padrões de projetos em suas aplicações. Abaixo podemos ver um gráfico com os resultados às perguntas 1 e 3 do questionário, na vertical temos a quantidade de pessoas numa escala de 0 a 1.

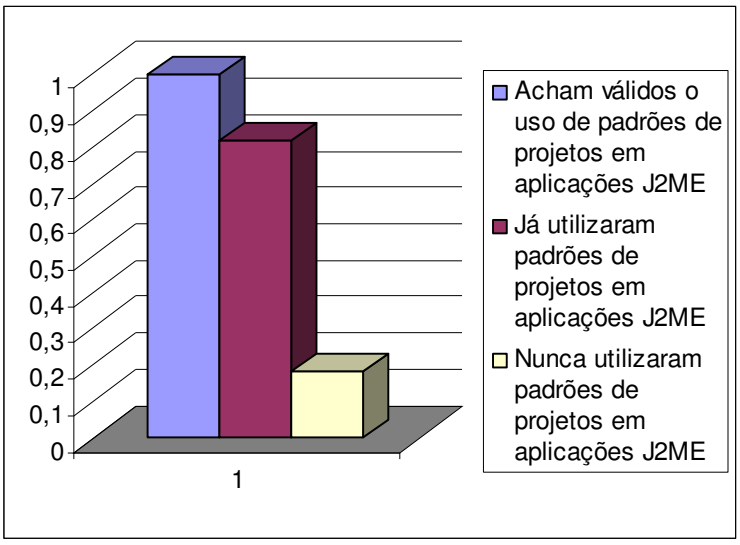

Figura 15. Gráfico com o resumo das respostas às perguntas 1 e 3 do questionário (ver Apêndice A), numa escala de 0 a 1.

Também extraímos da pesquisa os motivos pelos quais os entrevistados acham interessante o uso de padrões de projetos em aplicações J2ME. Dentre os motivos temos:

- Diminui os custos de manutenção;
- Aumenta a legibilidade do código;
- Aumento da produtividade, consequentemente reduzindo o tempo de desenvolvimento;
- Aumento da qualidade do produto;

- Contribui para a construção de códigos reusáveis;
- Diminuição do acoplamento do código;
- Permite o desenvolvimento baseado em componentes.

Na Figura 16 temos um gráfico com um resumo dos motivos mais citados pelos quais os desenvolvedores acham viável o uso dos padrões de projetos em aplicações J2ME:

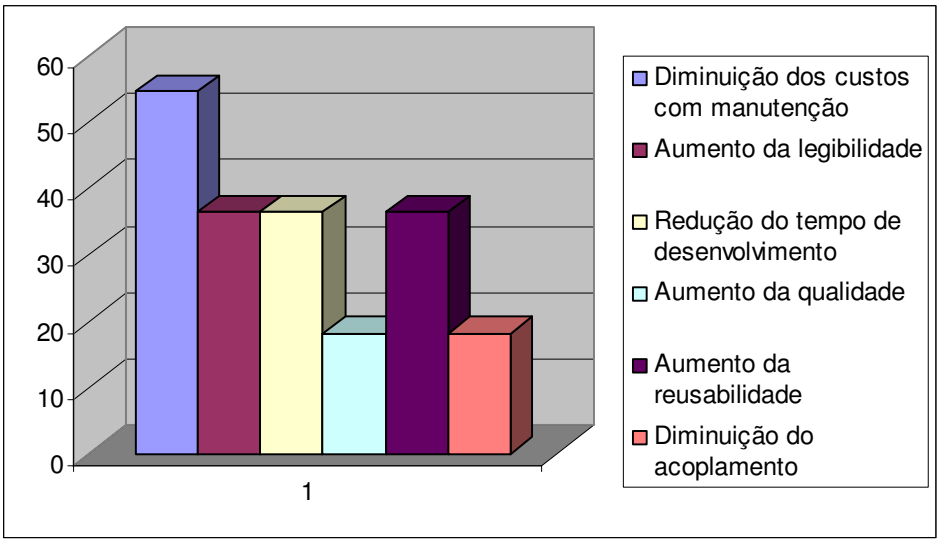

Figura 16. Porcentagem das motivações mais citadas, pelos desenvolvedores, que justifique a utilização de padrões de projetos em aplicações J2ME.

A pesquisa mostrou que os padrões de projetos mais utilizados são: *Singleton, Facade, Factory, Delegator, Closer Setter e Decorator* [8]. Na Figura 17 temos o resultado de quais padrões são mais utilizados, pelos desenvolvedores que participaram da pesquisa.

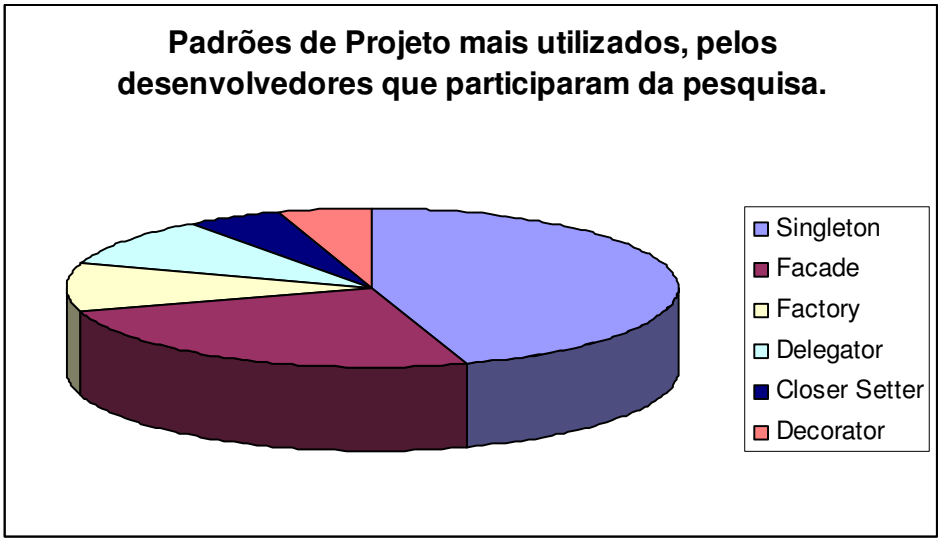

Figura 17. Padrões de projeto mais usados, pelos desenvolvedores que participaram da pesquisa.

Podemos observar através da Figura 17 que o padrão de projeto *Singleton* [8] foi o mais citado pelos desenvolvedores, uma vez que é um padrão de fácil implementação e entendimento e quase invariavelmente traz grandes benefícios à aplicação desenvolvida. Isto pode ser observado com o resultado (Figura 18) da última pergunta do questionário (ver Apêndice A) a qual pede para o desenvolvedor descrever quais padrões de projetos utilizados causou maiores benefícios.

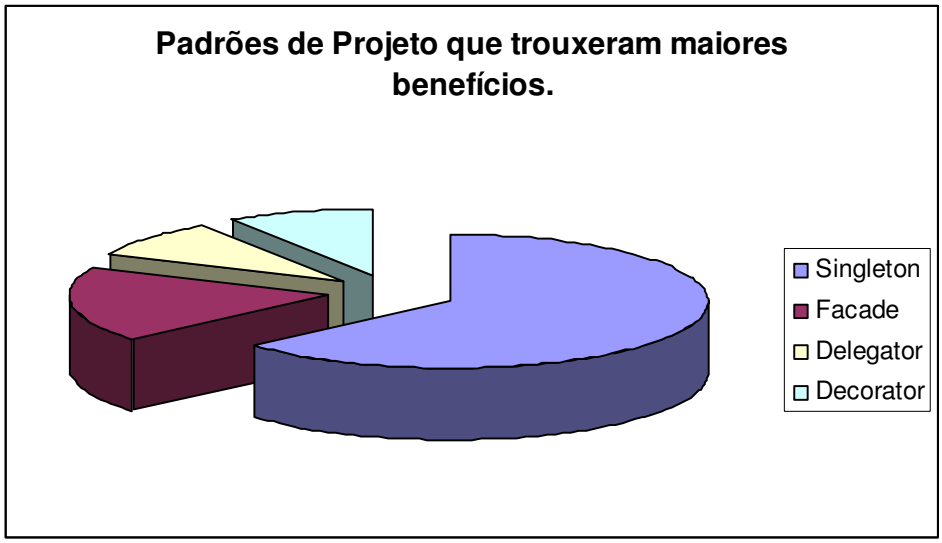

Figura 18. Padrões de projetos que trouxeram mais benefícios, descritos pelos desenvolvedores que participaram da pesquisa.

Com base nos resultados da pesquisa, em entrevistas realizadas com alguns desenvolvedores e no estudo da aplicação escolhida, os seguintes padrões de projetos foram selecionados para serem implementados durante os experimentos: *Singleton* e *Factory Method* [8]. O *Factory Method* foi selecionado para diminuir a complexidade do MIDlet da aplicação.

As variáveis do estudo

Como vimos no Capítulo 4, em um experimento existem variáveis independentes e variáveis dependentes. As entradas para um processo de experimentação são chamadas variáveis independentes e afetam o resultado de um experimento. As saídas de um experimento representam as variáveis dependentes, é o resultado do experimento. Abaixo estão as variáveis envolvidas neste processo de experimentação.

- Variável independente: a implementação dos padrões de projetos selecionados.
- Variáveis dependentes:
    - o desempenho da aplicação, que pode receber os seguintes valores:
        - igual, quando as medidas de desempenho têm o valor DI, DR1, DR2 = {1,1,1} (métrica 1 da Tabela 7).

  - maior, quando as medidas de desempenho têm o valor DI, DR1, DR2 = {1,1,2}, {1,2,1}, {1,2,2}, {2,1,1}, {2,1,2}, {2,2,1}, {2,2,2}, {2,2,3}, {2,3,2}, {3,2,2} (métricas 2, 4, 5, 10, 11, 13,14, 15, 17, 23 da Tabela 7).
  - menor, quando as medidas de desempenho têm o valor DI, DR1, DR2 = {1,1,3}, {1,3,1}, {1,3,3}, {2,3,3}, {3,1,1}, {3,1,3}, {3,2,3}, {3,3,1}, {3,3,2}, {3,3,3} (métricas 3, 7, 9, 18, 19, 21, 24, 25, 26, 27 da Tabela 7).
  - diferente, quando as medidas de desempenho têm o valor DI, DR1, DR2 = {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1} (métricas 6, 8, 12, 16, 20, 22 da Tabela 7).
- as métricas de qualidade interna que serão analisadas à medida que os padrões forem implementados: número total de linhas de código da aplicação, número de classes, número médio de linhas de código por método e complexidade ciclomática definida por McCabe [17].

Validade

Todo experimento deve ter uma forma de validação e no Capítulo 4 vimos algumas formas de determinar a validade. Para nosso processo de experimentação vimos a necessidade de utilizar a validade de conclusão, que será realizada através da verificação das hipóteses, a qual deverá ser feita por meio de uma simples conferência dos resultados das variáveis dependentes. Como neste estudo experimental os participantes são os padrões de projeto, as outras validades (interna, de construção e externa) não foram utilizadas, uma vez que levam em consideração fatores humanos, de ambiente ou tempo.

Definição do ambiente para os experimentos

Serão utilizadas as seguintes ferramentas para a realização dos experimentos:

- J2SE *Software Development Kit (SDK)*: Pacote de desenvolvimento para a plataforma J2SE [22]. Versão: 1.4.2.

- Eclipse: uma ferramenta de desenvolvimento integrado para criação de sistemas [19]Versão: 3.1.2.

- EclipseME: é uma ferramenta que pode ser adicionada ao Eclipse, para ajudar no desenvolvimento de aplicações J2ME [20]. Versão: eclipseme.feature_1.5.0_site.

- Apache Tomcat: pode ser considerado um servidor de aplicações, é o contêiner de *servlet* utilizado nas referências oficiais das implementações das tecnologias: *Java Servlet* e *JavaServer Pages* [21]. Versão Tomcat 5.5.

- tomcatPluginV3: uma ferramenta adicional para o Eclipse que realiza a integração entre o Tomcat e o Eclipse.

- *Sun Java Wireless Toolkit*: é um conjunto de ferramentas para desenvolvimento de aplicações para dispositivos móveis baseados na Configuração de Dispositivo Conectado Limitado (CLDC) e no Perfil de Dispositivo de Informação Móvel (MIDP) definidos na plataforma J2ME [23]. Versão 2.3.

- Motorola iDEN SDK para plataforma J2ME™: esta ferramenta suporta a construção e emulação de aplicações J2ME para a plataforma iDEN [24]. Esta ferramenta foi utilizada para automatizar os testes, pois não foi possível faze-lo utilizando o conjunto de ferramentas do Wireless Toolkit. Versão utilizada: i605 1.2.0.

- *Metrics*: ferramenta que pode ser adicionada ao Eclipse para a recuperação de métricas de um determinado código, como por exemplo, complexidade ciclomática, número de linhas de código de cada método, número de classes entre outros [18]. Versão 1.3.6.

5.3 Operação e Resultados

Inicialmente foram executados experimentos com a aplicação sem padrões de projetos, para formar a base de informações que será utilizada para comparar com as outras bases, à medida que os padrões de projetos forem implementados. Para cada implementação são feitas 50, 100, 500 e 1000 execuções, estas quantidades foram selecionadas com o intuito mostrar a variação desempenho de acordo com o a quantidade de experimentos executados. Foi necessário alterar a aplicação selecionada, de forma que pudéssemos recuperar o desempenho do sistema em três pontos: na inicialização da aplicação, em uma primeira requisição de informações ao servidor e numa segunda requisição de informações ao servidor. A primeira e a segunda requisição com o servidor diferem pela quantidade de informações requisitadas, a primeira requisita mais informações que a segunda. A seguir descreveremos as alterações de código necessárias.

Foi definida uma variável global *currentTime*, para o *MIDlet* da aplicação (*TccMIDlet*), responsável por contabilizar o tempo, em milisegundos, de cada funcionalidade:

```
public static long currentTime = System.currentTimeMillis();
```

Podemos observar que a variável é iniciada com o retorno da função *currentTimeMillis(),* responsável por recuperar o tempo atual do sistema em milisegundos. Esta variável será contabilizada através de impressões no console do simulador. O primeiro ponto em que a variável é contabilizada é no método *startApp()* como pode ser visto na Figura 19.

```
220     /*
221      * (non-Javadoc)
222      *
223      * @see javax.microedition.midlet.MIDlet#startApp()
224      */
225     protected void startApp() throws MIDletStateChangeException {
226         midlet = this;
227         alert = new Alert("Monografia");
228         this.theDisplay.setCurrent(alert, menuPrincipalList);
229         System.out.print(((System.currentTimeMillis() - currentTime)) + ";");
230         simulandoRequisicao1();
231     }
```

Figura 19. Método *startApp()* do *MIDlet,* com a aplicação sem padrões de projetos.

Na Linha 229 da Figura 19 está sendo registrado o tempo que a aplicação levou para ser iniciada. Logo em seguida, na Linha 230 iniciamos o método responsável por fazer a primeira requisição ao servidor. Na Figura 20 podemos ver o código do método *simulandoRequisicao1()*. Este método cria uma conexão passando três argumentos (*"0", "1", prof*); o primeiro informa o método de conexão que será utilizado; o segundo informa para o servidor qual consulta será realizada; no nosso caso o argumento "1" significa consulta relativa a um professor e o terceiro argumento informa o nome de usuário do professor relativo à consulta. A Figura 20 mostra, na Linha 458, a variável *currentTime* sendo atualizada com o tempo atual do sistema. Este é o início do segundo ponto que terá o desempenho analisado.

```
452     /**
453      * Método para simular a requisição 1 realizada por um usuário. Para
454      * automatizar os testes.
455      *
456      */
457     private void simulandoRequisicao1() {
458         currentTime = System.currentTimeMillis();
459         String prof = "FBLM";
460         Conexao conexao;
461         conexao = new Conexao("0", "1", prof);
462     }
```

Figura 20. Método *simulandoRequisicao1* do *MIDlet*, responsável por simular a primeira requisição. Com a aplicação sem padrões de projetos.

Após estabelecer conexão com o servidor e receber os dados com a resposta, a conexão chama o método do *MIDlet setResponse,* que funciona como um escutador. Este método é responsável por criar a tela que mostrará as respostas da requisição ao usuário. A Figura 21 apresenta as linhas de código do método *setResponse*.

```
417    public void setResponse(String resp) {
418        resultadoProjForm = new Form("Resultado da Pesquisa");
419        StringItem item = null;
420        item = new StringItem("", resp);
421        resultadoProjForm.append(item);
422        resultadoProjForm.addCommand(voltarCommand);
423        resultadoProjForm.addCommand(exitCommand);
424        resultadoProjForm.setCommandListener(this);
425        theDisplay.setCurrent(resultadoProjForm);
426        if (finalizar) {
427            System.out.println((System.currentTimeMillis() - currentTime));
428            destroyApp(true);
429        } else {
430            System.out.print((System.currentTimeMillis() - currentTime) + ";");
431            simulandoRequisicao2();
432        }
433    }
```

Figura 21. Método *setResponse* do *MIDlet*, responsável por receber a resposta à requisição.

O método *setResponse* apresentado na figura acima, recebe a resposta do servidor como parâmetro, monta a tela com a resposta que será mostrada ao usuário (Linha 418 a 424). Na Linha 425 a aplicação configura a tela com a resposta para ser mostrada no visor do aparelho. Por fim, o método verifica se é o momento ou não de destruir a aplicação, esta decisão é realizada através da variável global *finalizar* que foi adicionada ao *MIDlet*:

public boolean finalizar = false;

Esta variável inicia com o valor falso, ou seja, na primeira vez que o método *setResponse* (ver Figura 21) é chamado, o bloco de código dentro do *else* (linhas 430 e 431 da Figura 21) é executado. Desta forma a duração para a primeira requisição ao servidor é contabilizada (Linha 430) e o método *simulandoRequisicao2()*, para iniciar a segunda requisição com o servidor, é chamado.

O método apresentado *simulandoRequisicao2()* (Figura 22) cria uma nova conexão com o servidor. Desta vez para requisitar informações referentes a horários de um determinado professor (parâmetro “3” na Linha 472 da Figura 22) e reinicia o valor da variável *currentTime* com o tempo atual do sistema em milisegundos.

```
464    /**
465     * Método para simular a requisição 2 realizada por um usuário. Para
466     * automatizar os testes.
467     *
468     */
469    private void simulandoRequisicao2() {
470        finalizar = true;
471        currentTime = System.currentTimeMillis();
472        String login = "CABM";
473        Conexao conexao;
474        conexao = new Conexao("0", "3", login);
475    }
```

Figura 22. Método *simulandoRequisicao2* do *MIDlet*, responsável por simular a segunda requisição. Com a aplicação sem padrões de projetos.

Observa-se neste método que é atribuído o valor *true* para a variável *finalizar*. Desta forma quando o método *setResponse* (Figura 21) for chamado para mostrar os resultados referente à segunda requisição, o sistema registrará a duração da segunda requisição e chamará o método *destroyApp* do *MIDlet*.

```
247    protected void destroyApp(boolean arg0) {
248        try {
249            // Agenda o MIDlet para iniciar após
250            // 2000 milisegundos
251            scheduleMIDlet(2000);
252        } catch (ConnectionNotFoundException e) {
253            e.printStackTrace();
254        } catch (SecurityException e) {
255            e.printStackTrace();
256        } catch (ClassNotFoundException e) {
257            e.printStackTrace();
258        }
259        notifyDestroyed();
260    }
```

Figura 23. Método *destroyApp* do *MIDlet*.

O método *destroyApp*, apresentado na Figura 23, além de notificar o Gerenciador de Aplicativos que o *MIDlet* foi destruído (Linha 259), é responsável por agendar o *MIDlet* para ser iniciado após 2000 milisegundos (Linha 251).

```
432     /**
433      * Agenda o MIDlet's para iniciar após um tempo em milisegundos.
434      *
435      * @param deltatime
436      *             Tempo em milisegundos para inicializar o MIDlet.
437      */
438     private void scheduleMIDlet(long deltatime) throws ClassNotFoundException,
439             ConnectionNotFoundException, SecurityException {
440
441         String midletName = this.getClass().getName();
442         // Recupera o tempo atual chamando Date.getTime()
443         Date alarm = new Date();
444         long t = PushRegistry.registerAlarm(midletName, alarm.getTime() + deltatime);
445     }
```

Figura 24. Método *scheduleMIDlet,*responsável por agendar a próxima incialização do MIDlet.

Para agendar a próxima execução do *MIDlet* foi utilizada a classe *PushRegistry*, como pode ser observado na Figura 24. O nome do *MIDlet* (Linha 441) e o tempo atual (linha 443) são recuperados, por último é realizada uma chamada ao método *registerAlarm* (linha 444), passando o nome do *MIDlet* e o tempo atual mais um tempo delta (*deltatime* em milisegundos, passado como argumento na Linha 438). Desta forma os experimentos são feitos de forma automática.

Agora que foram mostradas as alterações necessárias, no código, para executarmos os experimentos, serão realizadas as primeiras execuções para a criação da base de informações sem padrões de projetos implementados.

Com o modulo servidor da aplicação rodando através do Tomcat [21]integrado no Eclipse [19], iniciamos a aplicação cliente através do Motorola iDEN SDK [24] para o modelo de parelho i605 como mostrado na Figura 25 e Figura 26.

Figura 25. Aparelho i605 com a aplicação cliente, através do SDK da Motorola [24].

Figura 26. Aparelho i605 executando a aplicação cliente, através do SDK da Motorola [24].

Após alguns segundos de execução pode ser visto no console do SDK da Motorola as saídas da aplicação (ver Figura 27), cada uma delas com três valores separados por ponto

e vírgula. O primeiro valor é a duração para iniciar a aplicação, o segundo é o tempo para a primeira requisição com o servidor e o terceiro é a duração para a segunda requisição de informações para o servidor.

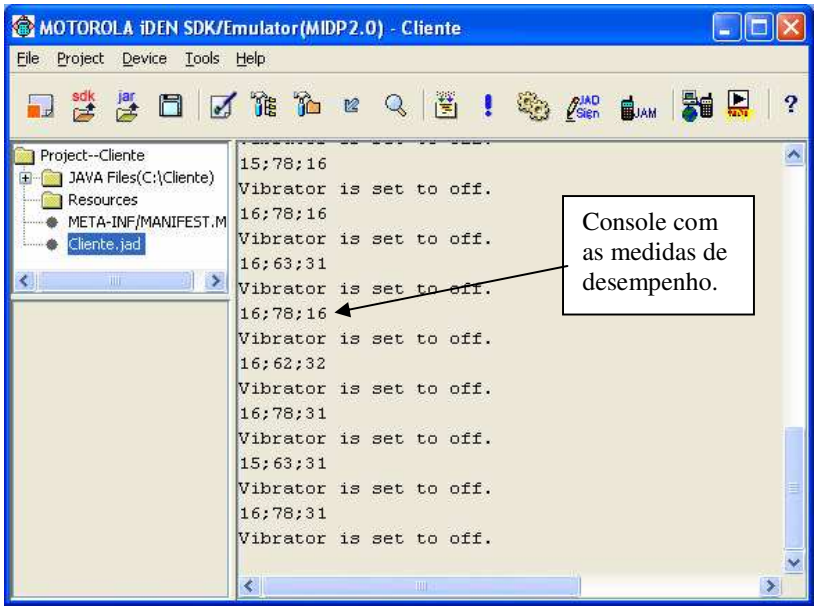

Figura 27. SDK da Motorola [24] com as medidas de desempenho impressas no console.

Após a execução dos experimentos para a aplicação sem os padrões de projetos implementados, chegamos aos resultados apresentados na Tabela 8 e Tabela 9, representando a base inicial que será usada para comparar com os resultados das outras implementações.

A Tabela 8 mostra os resultados de 50 execuções. A coluna “Execução” mostra o número da execução, a coluna “DI” (Desempenho na Inicialização) apresenta os valores de desempenho para iniciar a aplicação, a coluna “DR1” (Desempenho na Requisição 1) apresenta o desempenho da aplicação na primeira requisição de informações para o servidor. Por fim, a coluna “DR2” (Desempenha na Requisição 2) mostra a duração para a segunda requisição com o servidor. No final da Tabela 8 está a média e o desvio padrão para cada coluna das 50 execuções.

Tabela 8. Resultados de 50 execuções, com a aplicação sem padrões de projetos implementado.

Execução (50)	DI	DR1	DR2
1	16	109	47
2	0	110	31
3	15	110	31
4	16	109	32
5	16	62	32
6	15	79	31
7	16	62	31
8	15	110	15
9	15	79	15
10	15	78	16
....	....	....	....
41	16	78	15
42	16	125	31
43	16	78	15
44	15	79	15
45	15	63	31
46	16	62	32
47	16	62	32
48	16	62	31
49	16	78	15
50	16	109	16
Média	**14,4**	**83,48**	**24,62**
Desvio Padrão	**4,3141**	**20,2133**	**8,5114**

Ao invés de apresentar as tabelas com 50, 100, 500 ou 1000 linhas, será mostrada apenas uma tabela com o resumo das 4 tabelas. A Tabela 9 contém este resumo com a média e o desvio padrão para cada medida de desempenho. Para cada rodada de experimentos será mostrado um gráfico com os resultados das três colunas.

Tabela 9. Resumo dos resultados das 50, 100, 500 e 1000 execuções, com a aplicação sem padrões de projeto implementado.

Qtd. Execuções		**DI**	**DR1**	**DR2**
50	**Média**	**14,4**	**83,48**	**24,62**
	Desvio Padrão	**4,3141**	**20,2133**	**8,5114**
100	**Média**	**15,69**	**77,11**	**22,4**
	Desvio Padrão	**0,4648**	**19,0670**	**7,7224**
500	**Média**	**6,448**	**69,77**	**22,314**
	Desvio Padrão	**7,7168**	**30,8777**	**9,1749**
1000	**Média**	**7,293**	**73,298**	**24,2**
	Desvio Padrão	**10,8029**	**31,6013**	**13,3463**

A Figura 28 apresenta o gráfico referente às 50 primeiras execuções.

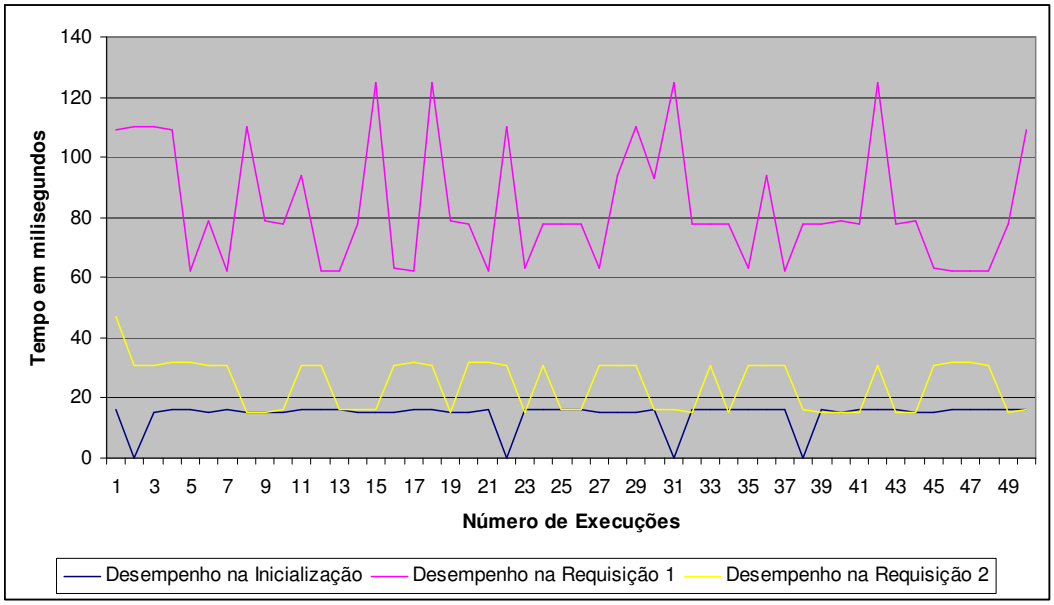

Figura 28. Gráfico com o resultado do experimento para 50 execuções.
Com a aplicação sem padrões de projetos.

Na Figura 29 apresentamos o gráfico com os resultados do experimento para 100 execuções:

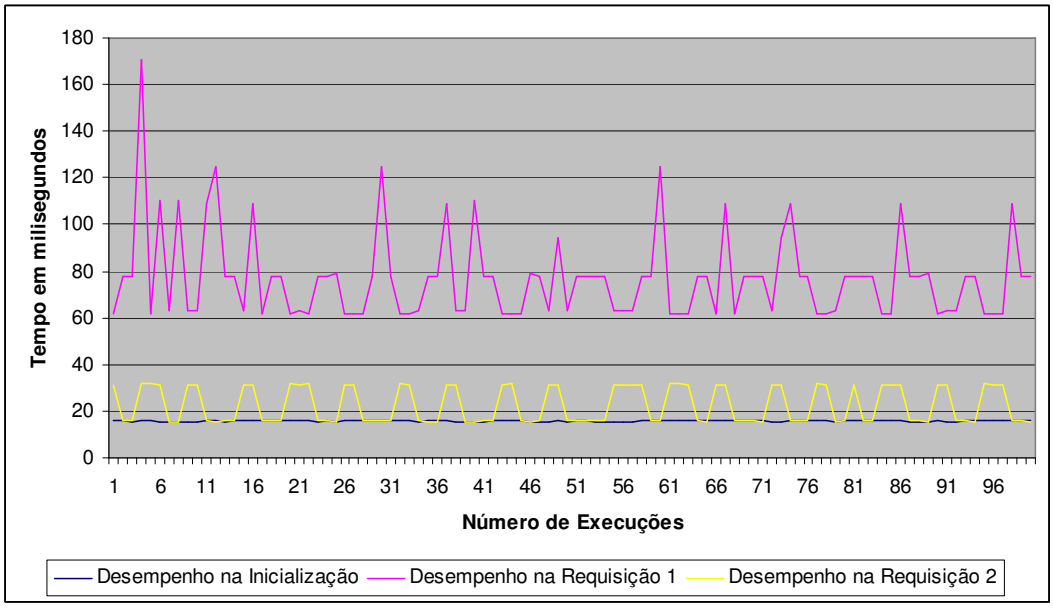

Figura 29. Gráfico com o resultado do experimento para 100 execuções.
Com a aplicação sem padrões de projetos.

A representação gráfica dos resultados da rodada com 500 execuções é encontrada na Figura 30.

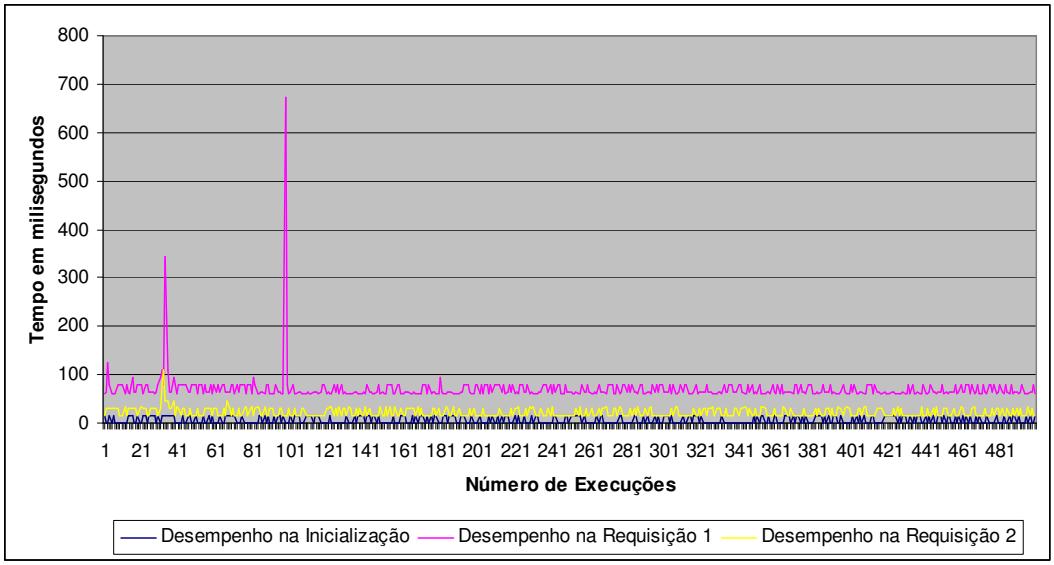

Figura 30. Gráfico com o resultado do experimento para 500 execuções.
Com a aplicação sem padrões de projetos.

Na Figura 31 temos os resultados do experimento para 1000 execuções.

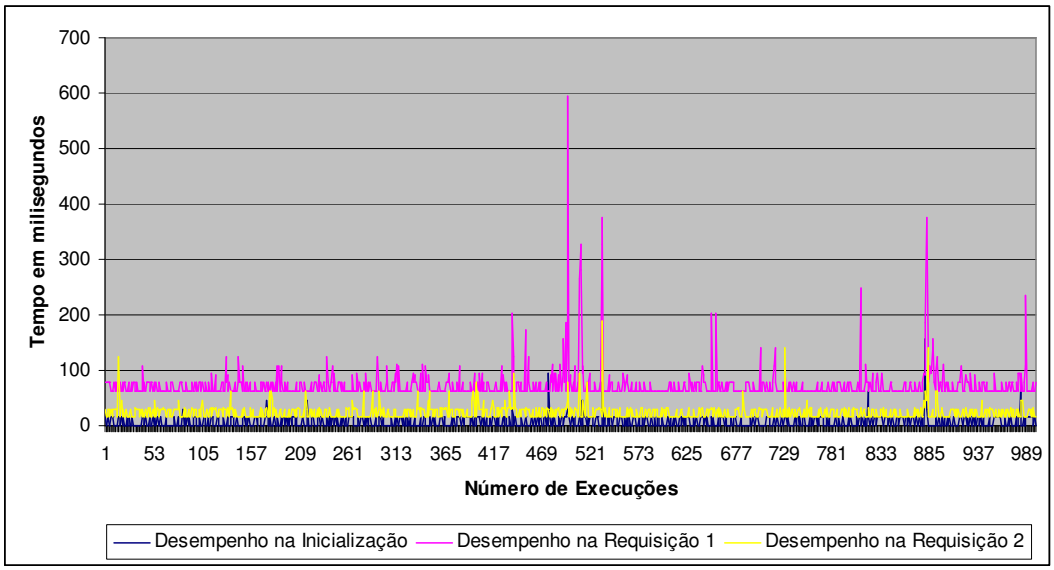

Figura 31. Gráfico com o resultado do experimento para 1000 execuções.
Com a aplicação sem padrões de projetos.

Após a execução dos experimentos com a implementação sem padrões de projetos foi utilizada a ferramenta Metrics [18] para recuperar as métricas estáticas. Os valores obtidos encontram-se na Tabela 10.

Tabela 10. Métricas estáticas recuperadas da implementação sem padrões de projeto.

Métrica	Valores
Número total de linhas de código da aplicação	522
Número total de classes da aplicação	5
Número médio de linhas de código por método	7,857
Média da Complexidade Ciclomática	2,286

Experimentos com padrão de projeto *Singleton* [8]

Foi observado que todas as vezes que o cliente quer realizar uma requisição para o servidor uma nova conexão é criada. Isto causa um aumento do uso de memória em tempo de execução, podendo degradar o desempenho da aplicação. Para resolver este problema será implementado o padrão de projeto *Singleton* na classe *Conexao*.

```
private static Conexao conexao;

private Conexao() {
}

public static synchronized Conexao getInstance() {
    if (conexao == null) {
        conexao = new Conexao();
    }
    return conexao;
}
```

```
//construtor vazio
public Conexao() {
    this.start();
}

//inicializa os parametros
public Conexao(String metodoConexao,
    String formatoMensagem, String dataEscolhida) {
    this.metodoConexao = metodoConexao;
    this.formatoMensagem = formatoMensagem;
    this.dataEscolhida = dataEscolhida;
    this.start();
}
```

Figura 32. Principais diferenças entre o código com o padrão de projeto *Singleton* (lado esquerdo) e sem o padrão (lado direito).

A Figura 32 mostra as principais diferenças entre o código sem o padrão (lado direito da figura) e o código com o padrão (lado esquerdo da figura). Foi criada uma variável global *conexao* na classe *Conexao* com o visibilidade *private*, ou seja, poderá ser visualizada apenas dentro da classe *Conexao*. O construtor também teve a visibilidade colocada para *private*, o que impede de outras classes criarem novas instancias do mesmo. Como os parâmetros para configurar a *Conexao* eram passados pelo construtor, foram criados métodos *set* para os atributos *metodoConexao, formatoMensagem* e *dataEscolhida*. Para ter acesso a uma instância da classe *Conexao*, as classes poderão chamar o método público *getInstance* que verifica se a única instancia de *Conexao* já não foi criada e se necessário cria a mesma. Desta forma, existirá apenas uma instância da classe *Conexao* para toda a aplicação e quando for necessário utilizá-la basta configurar seus parâmetros com os métodos *set* e iniciar a conexão. Esta mudança só foi possível, pois, em toda a aplicação apenas uma conexão é utilizada por vez. Caso a aplicação utilizasse mais de uma conexão ao mesmo tempo, não poderíamos aplicar o padrão de projeto *Singleton*.

Após a implementação do padrão de projeto *Singleton*, realizamos os experimentos para 50, 100, 500 e 1000 execuções e obtivemos os resultados descritos na Tabela 11 e nos gráficos abaixo.

Tabela 11. Resumo dos resultados das 50, 100, 500 e 1000 execuções, com o padrão de projeto *Singleton* implementado.

Qtd. Execuções		DI	DR1	DR2
50	**Média**	**15,78**	**83,38**	**26**
	Desvio Padrão	**5,4969**	**22,7478**	**11,5352**
100	**Média**	**12,58**	**85,6**	**25,34**
	Desvio Padrão	**7,2504**	**27,7652**	**15,3038**
500	**Média**	**14,068**	**61,77**	**23,274**
	Desvio Padrão	**6,5348**	**10,1223**	**14,2505**
1000	**Média**	**15,513**	**66,486**	**25,391**
	Desvio Padrão	**13,8019**	**31,0435**	**21,1495**

Na Figura 33 temos os resultados dos experimentos para 50 execuções, com o padrão de projeto *Singleton* implementado.

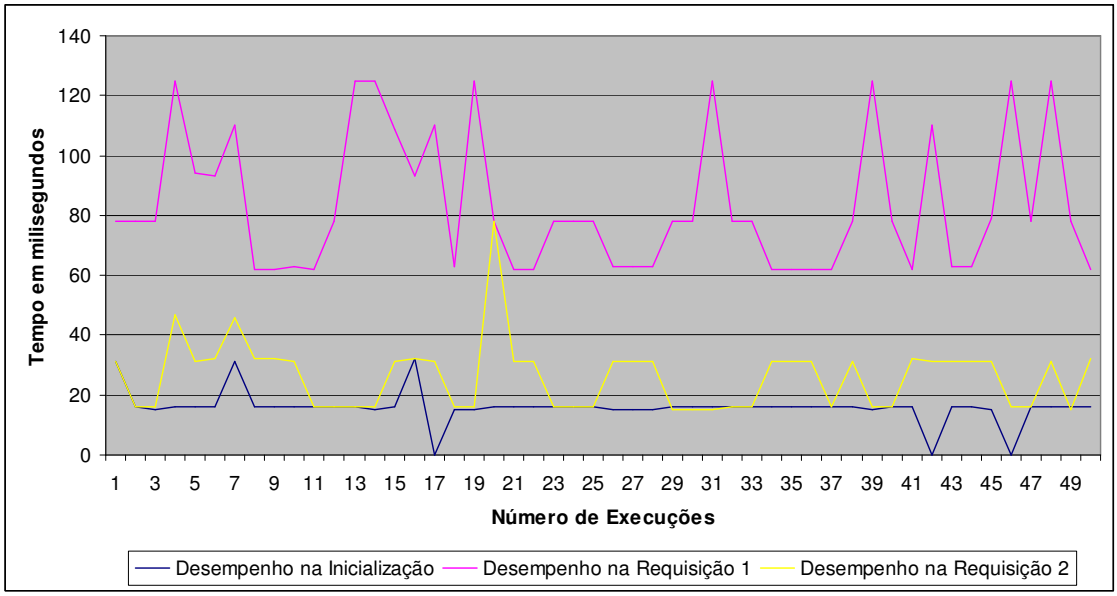

Figura 33. Gráfico com o resultado do experimento para 50 execuções. Com o padrão de projeto *Singleton* implementado.

Os resultados dos experimentos para 100 execuções, com o padrão de projeto *Singleton* implementado estão representados graficamente na Figura 34.

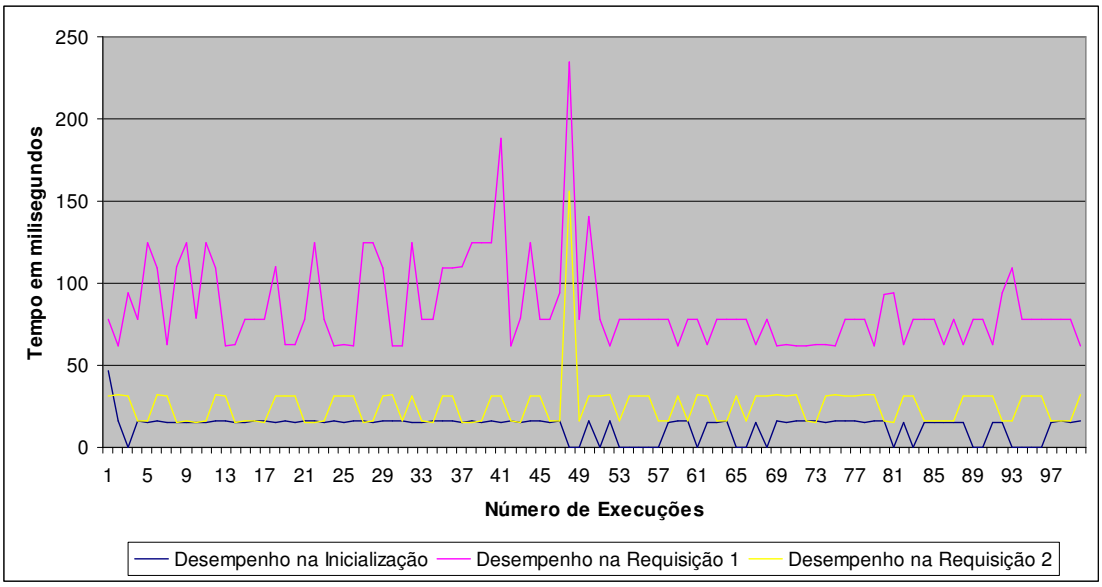

Figura 34. Gráfico com o resultado do experimento para 100 execuções.
Com o padrão de projeto *Singleton* implementado.

As 500 execuções, com o padrão de projeto *Singleton* implementado, estão representadas na Figura 35.

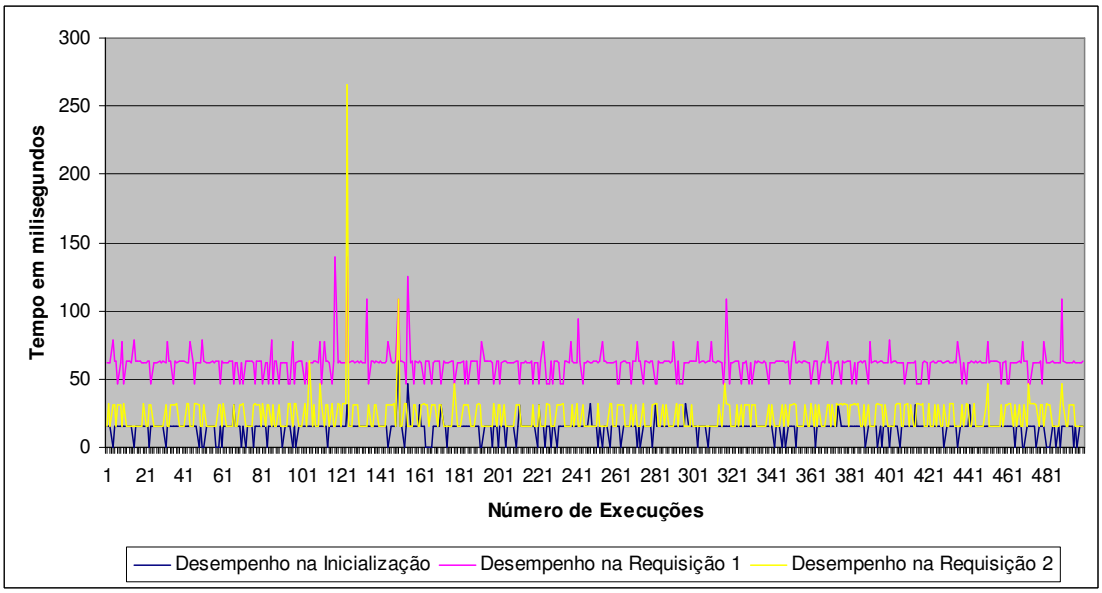

Figura 35. Gráfico com o resultado do experimento para 500 execuções.
Com o padrão de projeto *Singleton* implementado.

Na Figura 36 temos os resultados dos experimentos para 1000 execuções.

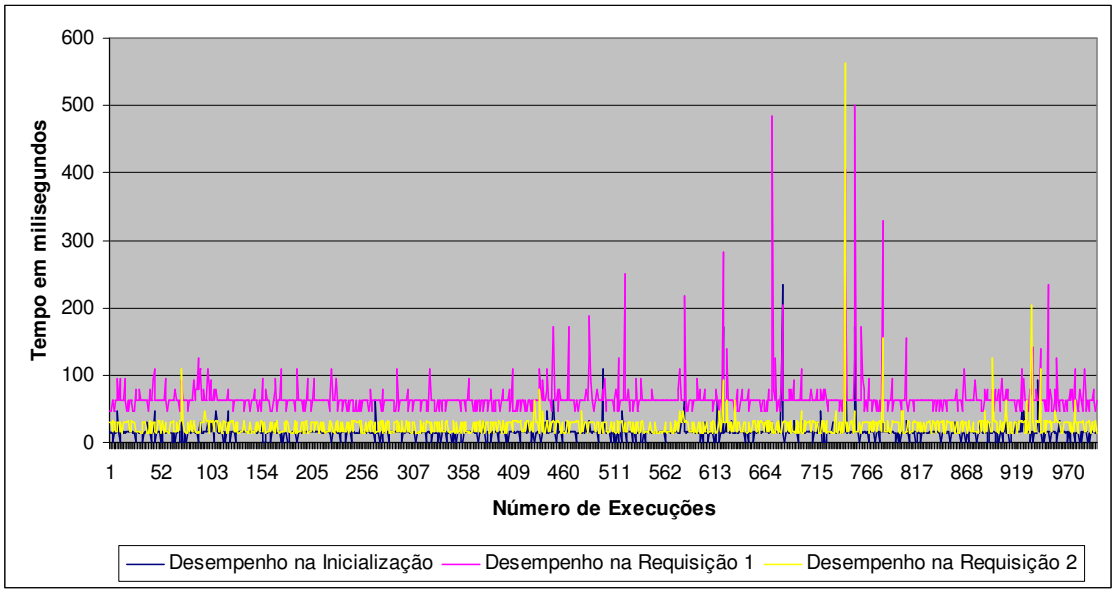

Figura 36. Gráfico com o resultado do experimento para 1000 execuções. Com o padrão de projeto *Singleton* implementado.

Após a execução dos experimentos com a implementação do padrão de projeto *Singleton* recuperamos as métricas estáticas apresentadas na Tabela 12.

Tabela 12. Métricas estáticas recuperadas da implementação com o padrão de projeto *Singleton*.

Métrica	Valores
Número total de linhas de código da aplicação	603
Número total de classes da aplicação	6
Número médio de linhas de código por método	8,37
Média da Complexidade Ciclomática	2,217

Experimentos com padrão de projeto *Factory Method* [8]

Após a execução dos experimentos com o padrão de projeto *Singleton*, recuperamos a aplicação inicial sem padrões de projetos, para colocar o padrão *Factory Method*. Criamos um gerenciador de telas, para toda a aplicação, que ficará responsável por criar todas as telas e, caso a *MIDlet* precise de alguma delas, basta chamar o método de criação de tela passando como parâmetro uma constante referente a tela desejada.

Como o objetivo é ter apenas um gerenciador de telas para toda a aplicação implementamos a classe *ControladorDeTelas* com o padrão *Singleton* [8]. A Figura 37 mostra o método principal da classe responsável por fabricar as telas. A Figura 37 mostra na Linha 109 o parâmetro do método *telaSelecionada*, responsável por determinar qual tela deve ser fabricada. Como todas as telas herdam da classe *Displayable*, o retorno do método é um *Displayable*. As telas são criadas na medida em que são requisitadas e isto diminui a

carga inicial das telas. Quando uma tela é criada, a mesma é colocada em uma *Hashtable telas*, da classe *ControladorDeTelas*, desta forma o método *fabricarTela* precisa apenas recuperar a tela desejada passando o parâmetro *telaSelecionada*.

```
109    public Displayable fabricarTela(int telaSelecionada) {
110        Displayable nextDisplayable = null;
111        switch (telaSelecionada) {
112            case Constantes.TELA_MENU_ALERTA:
113                criarAlerta();
114                break;
115            case Constantes.TELA_MENU_PRINCIPAL:
116                criarMenuPrincipal();
117                break;
118            case Constantes.TELA_MENU_PROJETOS:
119                criarMenuProjetos();
120                break;
121            case Constantes.TELA_INFORMACOES:
122                criarTelaInfo();
123                break;
124            case Constantes.TELA_HORARIOS:
125                criarTelaHorarios();
126                break;
127            case Constantes.TELA_PROFESSORES:
128                criarTelaProfessores();
129                break;
130            case Constantes.TELA_RESULTADO_PROJ:
131                break;
132            case Constantes.TELA_SELEC_DATA:
133                criarTelaSelecData();
134                break;
135            case Constantes.TELA_ALUNOS:
136                criarTelaAlunos();
137                break;
138        }
139        nextDisplayable = (Displayable) telas.get(new Integer(
140                telaSelecionada).toString());
141        return nextDisplayable;
142    }
```

Figura 37. Método responsável por fabricar as telas da aplicação, recebe como parâmetro um identificador para a tela desejada.

Os métodos de criação de telas (linha 116, da Figura 37, por exemplo) seguem o padrão do código descrito na Figura 38.

```
152    private void criarMenuPrincipal() {
153        // menu principal
154        if (menuPrincipalList == null) {
155            menuPrincipalList = new List("Menu Principal", List.IMPLICIT,
156                    Textos.ITENS_MENU_PRINCIPAL, null);
157            menuPrincipalList.addCommand(Comandos.EXIT_COMMAND);
158            menuPrincipalList.addCommand(Comandos.SELEC_COMMAND);
159            menuPrincipalList.setCommandListener(commandListner);
160            telas.put(Constantes.TELA_MENU_PRINCIPAL_KEY,menuPrincipalList);
161        }
162    }
```

Figura. 38 Exemplo de método de criação de tela, (*criarMenuPrincipal*).

A aplicação verifica se a tela já não foi criada (linha 154 da Figura 38). Caso ainda não tenha sido criada, a aplicação instancia a tela e coloca a mesma na *Hashtable* de telas.

Para a *MIDlet* a diferença será na hora de utilizar a tela, ao invés de utilizar a tela diretamente, pois todas as telas estão no próprio *MIDlet* (implementação sem padrões de projetos) será necessário fazer uma chamada ao método *fabricarTela* (ver Figura 37). Isto porque, com o padrão de projeto, as telas não fazem parte mais do *MIDlet*, e sim da classe *ControladorDeTelas*. Podemos observar um pequena alteração necessária para a aplicação J2ME selecionada. De acordo com o padrão proposto por Gamma et al [8], a classe *ControladorDeTelas* deveria implementar um determinado tipo de fábrica, isto para que diferentes fábricas do mesmo tipo fossem criadas, como em nosso caso o objetivo é ter apenas uma fábrica de telas, não há necessidade de criar este tipo abstrato de fábrica.

Após a implementação do padrão de projeto *Factory Method*, realizamos os experimentos para 50, 100, 500 e 1000 execuções e obtivemos os resultados descritos na Tabela 13 e nos gráficos abaixo.

Tabela 13. Resumo dos resultados das 50, 100, 500 e 1000 execuções, com o padrão de projeto *Factory Method* implementado.

Qtd. Execuções		DI	DR1	DR2
50	**Média**	**21,9**	**58,12**	**24,7**
	Desvio Padrão	**8,4521**	**13,3899**	**9,6515**
100	**Média**	**24,21**	**61,96**	**26,06**
	Desvio Padrão	**8,2134**	**21,5185**	**10,7551**
500	**Média**	**21,206**	**94,116**	**24,08**
	Desvio Padrão	**10,0786**	**48,1503**	**17,0327**
1000	**Média**	**22,782**	**62,187**	**23,992**
	Desvio Padrão	**10,5393**	**34,8345**	**17,7194**

A Figura 39 apresenta os resultados dos experimentos para 50 execuções.

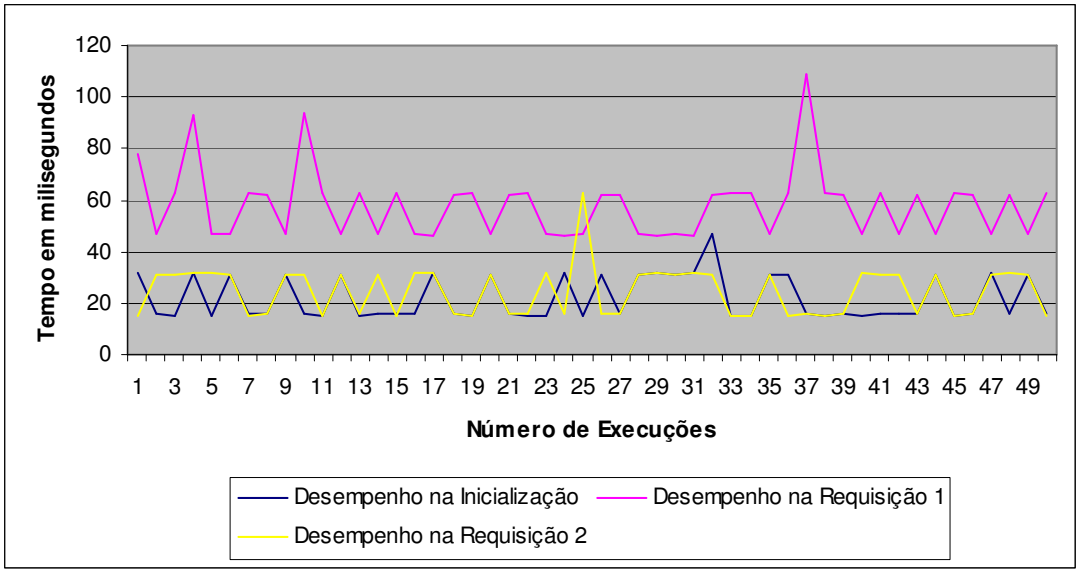

Figura 39. Gráfico com o resultado do experimento para 50 execuções.
Com o padrão de projeto *Factory Method* implementado.

Os resultados dos experimentos para 100 execuções, com o padrão de projeto *Factory Method* implementado, estão representados na Figura 40.

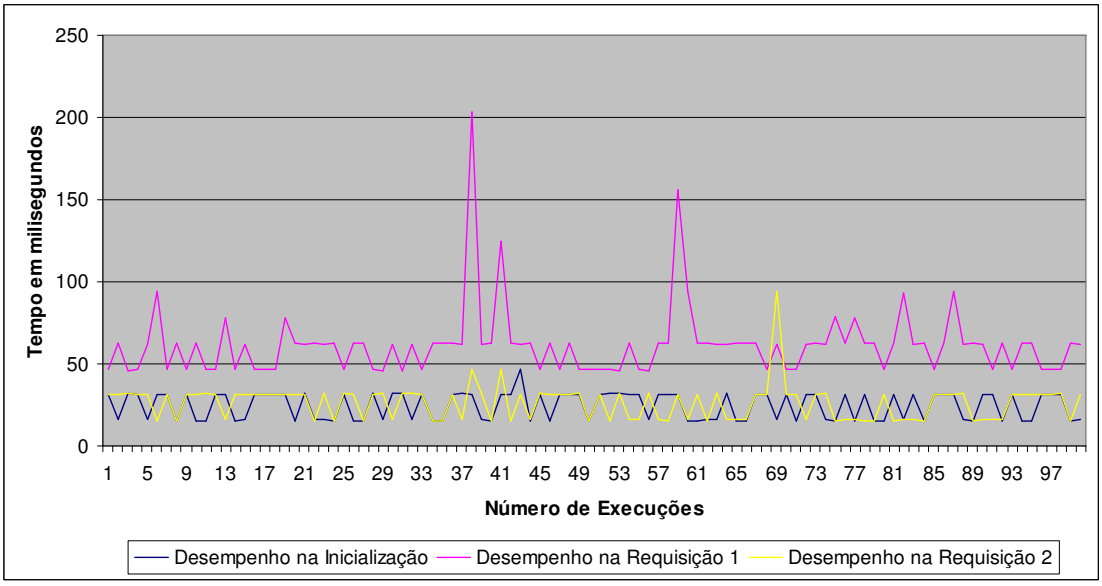

Figura 40. Gráfico com o resultado do experimento para 100 execuções.
Com o padrão de projeto *Factory Method* implementado.

Na Figura 41 temos os resultados dos experimentos para 500 execuções, com o padrão de projeto *Factory Method* implementado.

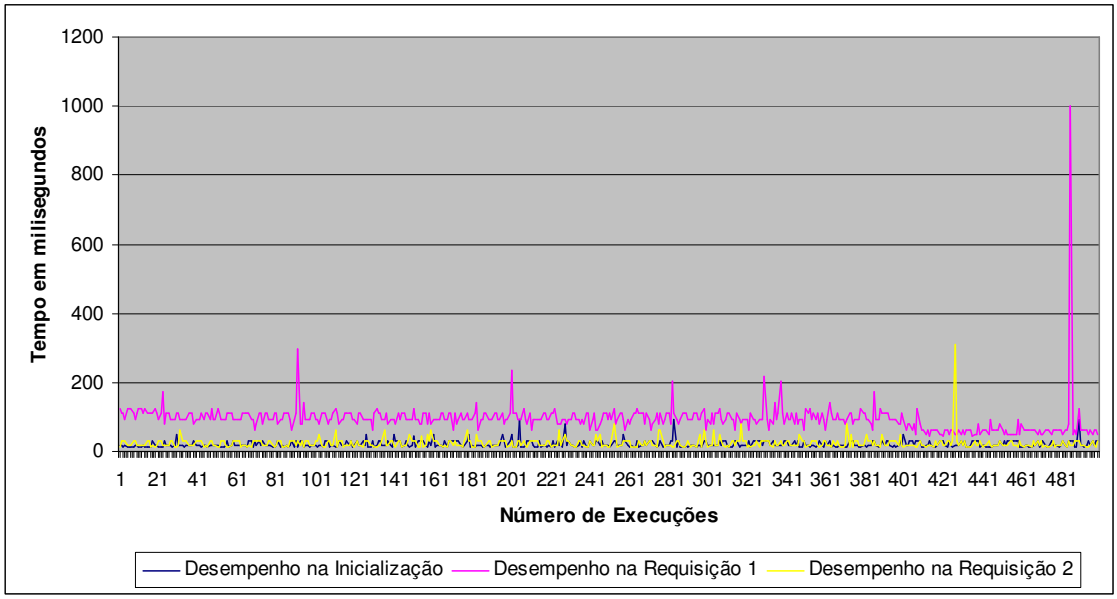

Figura 41. Gráfico com o resultado do experimento para 500 execuções.
Com o padrão de projeto *Factory Method* implementado.

Os resultados dos experimentos para 1000 execuções, com o padrão de projeto *Factory Method* implementado estão na Figura 42.

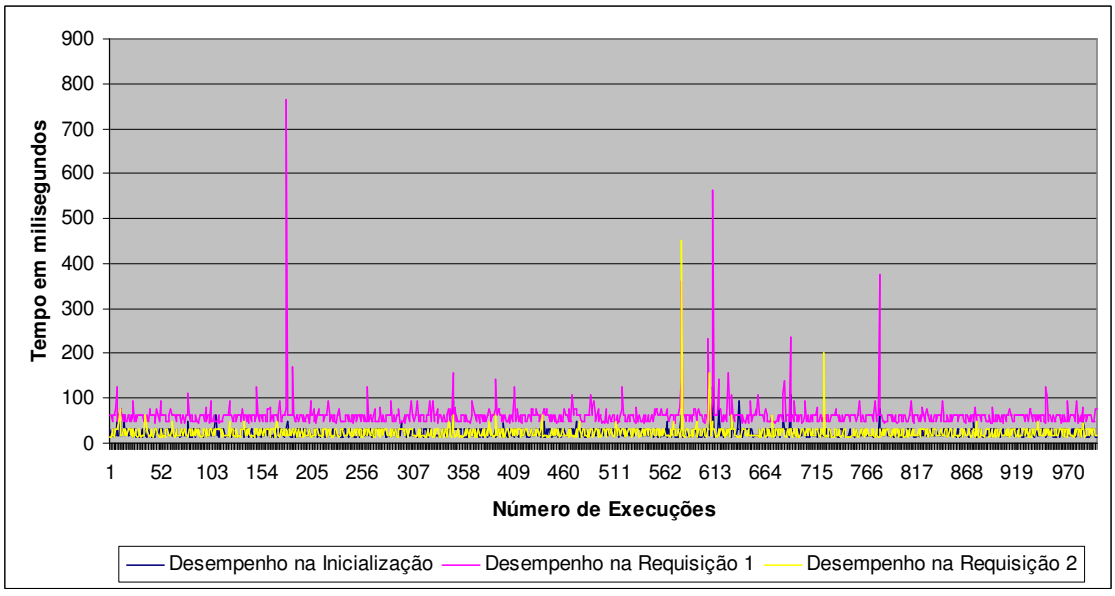

Figura 42. Gráfico com o resultado do experimento para 1000 execuções.
Com o padrão de projeto *Factory Method* implementado.

Após a execução dos experimentos com a implementação do padrão de projeto *Factory Method* recuperamos as métricas estáticas apresentadas na Tabela 14.

Tabela 14. Métricas estáticas recuperadas da implementação com o padrão de projeto *Factory Method*.

Métrica	Valores
Número total de linhas de código da aplicação	727
Número total de classes da aplicação	10
Número médio de linhas de código por método	7,508
Média da Complexidade Ciclomática	2,254

Experimentos com os padrões de projeto *Singleton* e *Factory Method* [8]

Para finalizar os experimentos, criaremos uma implementação com ambos os padrões de projetos descritos acima e executaremos os testes para esta aplicação para 50, 100, 500 e 1000 rodadas. Desta forma teremos quatro implementações: a primeira sem padrões de projetos, uma segunda apenas com o *Singleton,* a terceira com o *Factory Method* e agora uma quarta com os dois padrões de projeto implementados.

Após a criação desta quarta aplicação, realizamos os experimentos para 50, 100, 500 e 1000 execuções e obtivemos os resultados descritos na Tabela 15 e nos gráficos abaixo.

Tabela 15. Resumo dos resultados das 50, 100, 500 e 1000 execuções, com os padrões de projetos *Singleton e Factory Method* implementados.

Qtd. Execuções		DI	DR1	DR2
50	**Média**	**33,4**	**66,94**	**24,72**
	Desvio Padrão	**13,7722**	**14,1066**	**9,4286**
100	**Média**	**31,31**	**56,15**	**22,63**
	Desvio Padrão	**0,4648**	**10,4023**	**7,7311**
500	**Média**	**33,408**	**64,07**	**24,072**
	Desvio Padrão	**22,6563**	**46,9778**	**9,5646**
1000	**Média**	**30,119**	**45,362**	**22,736**
	Desvio Padrão	**11,1211**	**13,1966**	**9,3001**

Os resultados dos experimentos para 50 execuções, com os dois padrões de projetos implementados está apresentado na Figura 43.

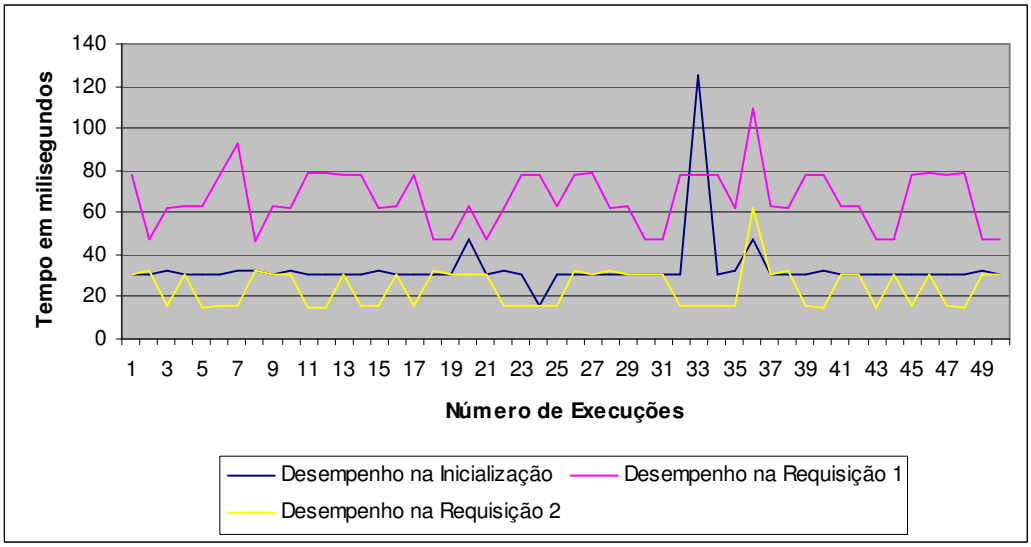

Figura 43. Gráfico com o resultado do experimento para 50 execuções. Com os padrões de projetos *Singleton e Factory Method* implementados.

Na Figura 44 temos os resultados dos experimentos para 100 execuções. Comparando os gráficos da Figura 43 e Figura 44, com as Figuras 28 e 29 podemos observar claramente um aumento no tempo durante a inicialização e uma diminuição no tempo durante a segunda requisição.

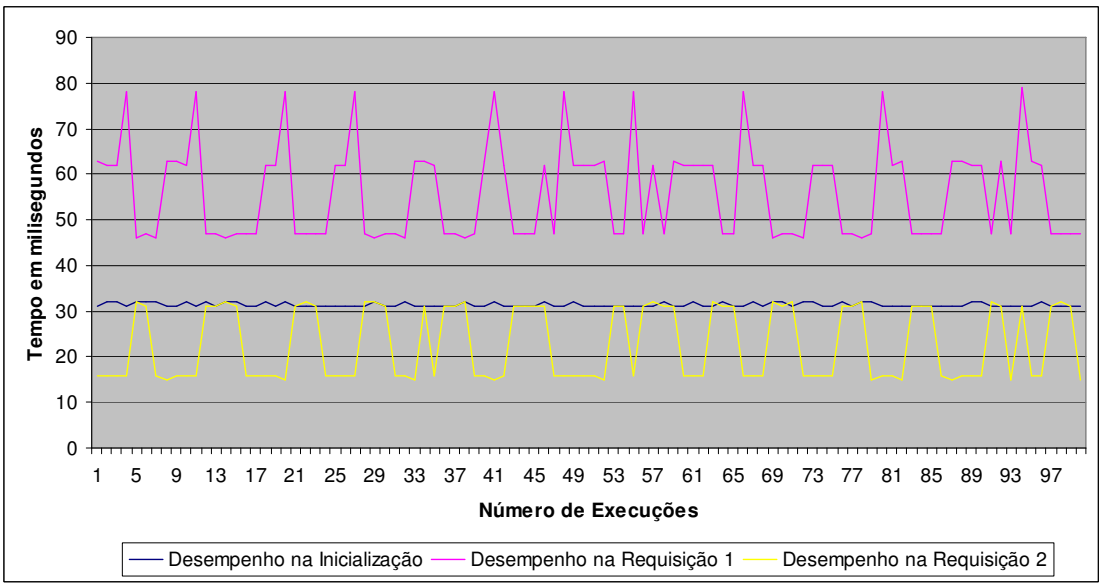

Figura 44. Gráfico com o resultado do experimento para 100 execuções. Com os padrões de projetos *Singleton e Factory Method* implementados.

A Figura 45 apresenta o gráfico referente aos resultados dos experimentos para 500 execuções.

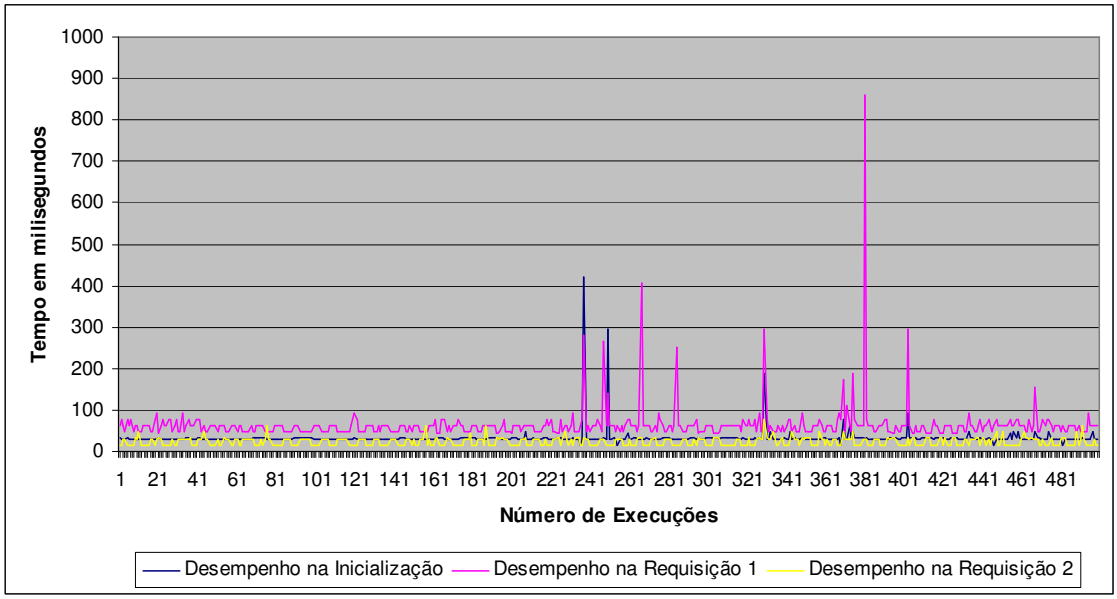

Figura 45. Gráfico com o resultado do experimento para 500 execuções. Com os padrões de projetos *Singleton e Factory Method* implementados.

Na Figura 46 temos o gráfico referente aos resultados dos experimentos para 1000 execuções, com os dois padrões de projetos implementados:

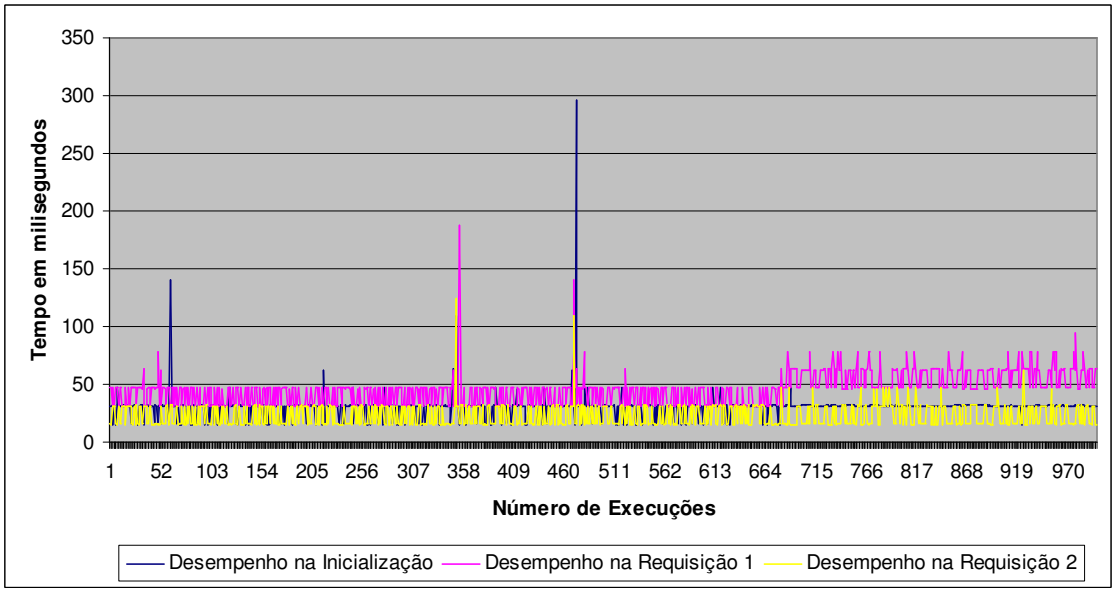

Figura 46. Gráfico com o resultado do experimento para 1000 execuções. Com os padrões de projetos *Singleton e Factory Method* implementados.

Após a execução dos experimentos com a implementação dos padrões de projetos *Singleton* e *Factory Method* recuperamos as métricas estáticas da Tabela 16.

Tabela 16. Métricas estáticas recuperadas da implementação com os padrões de projeto *Singleton* e *Factory Method*.

Métrica	**Valores**
Número total de linhas de código da aplicação	801
Número total de classes da aplicação	10
Número médio de linhas de código por método	7,841
Média da Complexidade Ciclomática	2,206

5.4 Análise e interpretação dos resultados

Na seção anterior mostramos os resultados dos experimentos realizados para as quatro implementações. Agora iremos analisar e interpretar os resultados recuperados.

Primeiramente podemos visualizar nos gráficos que para cada implementação testada, os resultados com 50, 100, 500 e 1000 execuções segue um padrão. Em alguns instantes há picos na coordenada do tempo em milisegundos, estes picos podem acontecer por conta de alguma irregularidade não detectado durante os experimentos, mas aparentemente não causarão distorções nas análises a seguir, uma vez que a grande maioria dos resultados segue um padrão.

Para uma melhor análise dos dados será criada uma tabela com a média de desempenho dos experimentos com 50, 100, 500 e 1000 para cada implementação e a cada funcionalidade. Anteriormente foi mostrado que funcionalidades estão sendo analisadas: a DI ou Desempenho na Inicialização, a DR1 ou Desempenho na Requisição 1 e a DR2 que corresponde ao Desempenho na Requisição 2. Com respeito às implementações temos nas colunas da Tabela 17 as seguintes siglas:

1. Dsp: Desempenho da aplicação sem padrões de projetos implementados.
2. Dcs: Desempenho da aplicação com o padrão de projeto *Singleton* implementado.
3. Dcf: Desempenho da aplicação com o padrão de projeto *Factory Method* implementado.
4. Dcsf: Desempenho da aplicação com os padrões de projeto *Singleton* e *Factory Method* implementados.

Tabela 17. Média de desempenho, em milisegundos, de cada funcionalidade analisada para cada implementação.

	Dsp	Dcs	Dcf	Dcsf
DI	**10,9577**	14,4852	22,5245	32,0592
DR1	75,9145	74,309	69,0957	**58,1305**
DR2	**23,3835**	25,0012	24,708	23,5395

Podemos observar que o melhor desempenho médio para a inicialização foi o da implementação sem padrão de projeto (10,9577 milisegundos) já o melhor desempenho médio durante a primeira requisição de informações ao servidor foi de 58,1305 milisegundos na implementação com os dois padrões de projeto. Por último, temos na linha DR2 da Tabela 17 o desempenho médio de 23,3835 milisegundos, da implementação sem padrões de projetos como sendo o menor para a segunda requisição de informações com o servidor.

Com base nas métricas definidas na Seção 5.2 deste capítulo, temos a seguinte classificação com a média de desempenho para cada implementação com padrão de projeto:

- Dcs: DI passa de 10,9577 para 14,4852, ou seja, o desempenho na inicialização diminui (3); DR1 passa de 75,9145 para 74,309 o que indica que o desempenho para este ponto aumentou (2); por fim o desempenho na requisição 2 passa de 23,3835 para 25,0012 significando mais uma diminuição do desempenho (3). Esta análise nos leva a classificar a aplicação como sendo DI, DR1, DR2 = {3,2,3}, ou seja, métrica 24 (o desempenho geral da aplicação diminuiu).
- Dcf: DI passa de 10,9577 para 22,5245, ou seja, o desempenho na inicialização diminui (3); DR1 passa de 75,9145 para 69,0957 o que indica que o desempenho para a primeira requisição aumentou (2); por fim o desempenho na requisição 2 passa de 23,3835 para 24,708 significando mais uma diminuição do desempenho (3). Esta análise nos leva a classificar a aplicação como sendo também DI, DR1, DR2 = {3,2,3}, ou seja, métrica 24 (o desempenho geral da aplicação diminuiu).
- Dcsf: observando a Tabela 17 podemos ver que a aplicação também se encaixa na métrica 24 (desempenho geral da aplicação diminui). Observando a linha DR2 e coluna Dcsf da tabela, percebe-se que por menos de 1 milisegundo a aplicação não se enquadra à métrica 23 na qual o desempenho geral da aplicação aumenta.

Antes de analisar as variáveis estáticas dependentes, podemos finalizar a análise do desempenho mostrando que, de maneira geral, a introdução de padrões de projetos à aplicação não degradou a mesma como aparenta com a classificação das métricas acima. Podemos visualizar melhor isto no gráfico apresentado na Figura 47, criado a partir dos valores da Tabela 17.

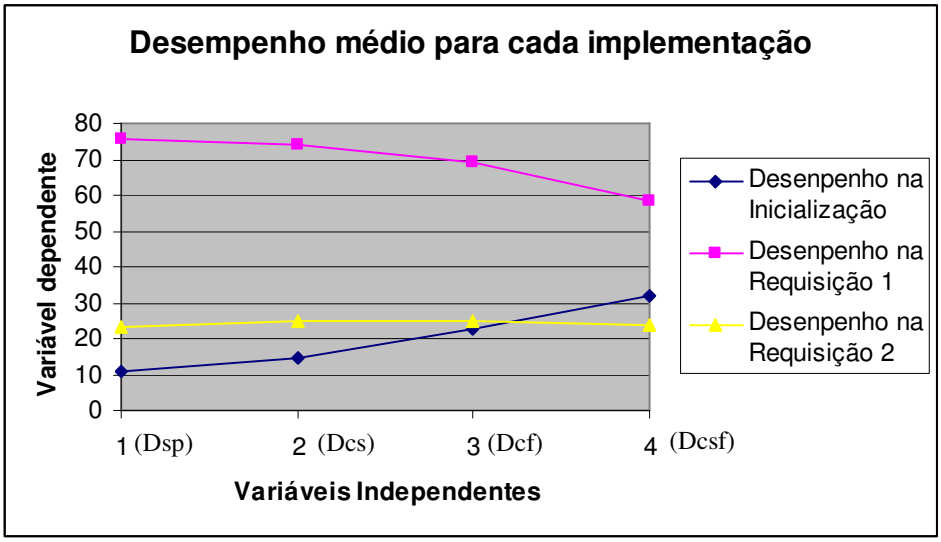

Figura 47. Gráfico com média de desempenho, em milisegundos, de cada funcionalidade analisada para cada implementação.

A Figura 47 apresenta nas coordenada das variáveis independentes cada implementação como sendo Dsp, Dcs, Dcf e Dcsf, respectivamente. Observa-se, portanto, que tivemos uma perda no desempenho durante a inicialização da aplicação, um ganho de desempenho na primeira requisição de informações com o servidor e por último uma quase estabilidade no desempenho durante a segunda requisição de informações com o servidor.

A Tabela 18 apresenta um resumo das métricas estáticas recuperadas em cada implementação.

Tabela 18. Métricas estáticas recuperadas para cada implementação.

	Dsp	**Dcs**	**Dcf**	**Dcsf**
Número total de linhas de código.	522	603	727	801
Número total de classes.	5	6	10	10
Número médio de linhas de código por método.	7,857	8,37	7,508	7,841
Média da Complexidade Ciclomática.	2,286	2,217	2,254	2,206

Podemos observar que as implementações dos padrões de projetos selecionados não causaram muitos impactos no número médio de linhas de código por método e nem na média da Complexidade Ciclomática. Apesar da pequena diminuição nestas métricas, que significa termos uma aplicação com maior legibilidade facilitando assim na manutenção da mesma, os valores não foram significativos. As mudanças mais significativas foram no número total de linhas de código e no número de classes da aplicação, isto não quer dizer apenas que o tamanho da aplicação aumentou, o aumento destas métricas é resultado de um desacoplamento na aplicação.

Fazendo uma análise no código apresentado pode-se perceber que as principais mudanças realizadas pelos padrões selecionados foram:

- O desacoplamento das telas da classe *MIDlet* para uma classe que cuida especificamente de telas, o *ControladorDeTelas*;

- E a criação de uma instância única da classe *Conexao* para toda a aplicação, diminuindo assim o consumo em memória em tempo de execução, utilizada pela aplicação.

Verificação das Hipóteses

Hipótese nula (H0): Para verificar a hipótese nula devemos responder a questão Q2 definida na Seção 5.1. Foi mostrado nesta seção que todas as implementações dos padrões de projeto selecionados se classificaram na métrica DI, DR1, DR2 = {3,2,3} (métrica 24 da Tabela 4.). Desta forma podemos fazer a conclusão que houve uma diminuição do desempenho geral da aplicação, levando em consideração os pontos analisados (DI, DR1 e DR2).

5.5 Diretrizes para o uso de padrões de projetos

Após o estudo, foi possível observar que os passos para o uso de padrões de projeto em aplicações J2ME não são diferentes dos que existem para outras aplicações. Gamma et al [8] apresenta diretrizes para aplicar um padrão de projeto.

Vimos neste capítulo que ajustes podem ser necessários durante a fase de implementação do padrão de projeto, mas isto depende de cada aplicação específica. Com a experiência cada engenheiro pode desenvolver sua própria maneira de trabalhar com padrões de projeto. Para as aplicações J2ME deve-se ter uma atenção especial ao desempenho da aplicação durante a implementação do padrão de projeto, uma vez que existem aparelhos que possuem capacidades muito limitadas de memória.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Contribuições

A principal contribuição deste trabalho foi mostrar vantagens e desvantagens do uso de padrões de projeto em aplicações J2ME. Podemos ver, através da pesquisa apresentada no Capítulo 5, que engenheiros, de algumas empresas do Porto Digital [16], já utilizam padrões de projetos em suas aplicações e muitos são os benefícios envolvidos. Alguns benefícios do uso de padrões de projetos são: a diminuição dos custos com manutenção, o aumento legibilidade do código, a redução do tempo de desenvolvimento, aumento da qualidade interna do produto, entre outros.

A pesquisa também mostrou, no contexto de algumas empresas do Porto Digital [16], padrões de projetos que estão sendo mais utilizados: *Singleton, Facade, Factory, Delegator e Decorator* [8].

Selecionamos dois padrões de projetos do catalogo de padrões para a plataforma J2EE [25] (*Singleton* e *Factory Method*) para serem implementados em uma aplicações desenvolvida por uma aluna do DSC [14], em seu trabalho de graduação.

O Capítulo 5 apresentou algumas conseqüências da implementação dos padrões de projeto selecionados. Foram criadas quatro implementações: uma sem os padrões de projeto selecionados, outra com apenas o padrão *Singleton,* uma terceira com o *Factory Method* implementado e a última com os dois padrões de projetos selecionados. Selecionamos três funcionalidades, da aplicação, para terem suas medidas de desempenho analisadas em cada uma das implementações. Para cada implementação foram realizados 50, 100, 500 e 1000 execuções. Foi possível observar a queda do desempenho em algumas funcionalidades e o ganho em outras.

Métricas estáticas, envolvendo LOC e complexidade ciclomática [17], também foram analisadas a cada implementação. Foi observado que as principais mudanças geradas

pelos padrões selecionados foram o desacoplamento das telas dentro do *MIDlet* e a diminuição do número de instâncias da classe *Conexao* utilizada pela aplicação.

Ao implementarmos um padrão de projeto, em uma aplicações J2ME, verificamos a necessidade de alguns ajustes ao padrão, como foi o caso do *Factory Method*. Isto requer uma análise minuciosa do problema envolvido na aplicação antes de implementar o padrão.

Vimos que padrões de projeto aumentam o tamanho da aplicação. Quando estamos trabalhando com aparelhos de capacidade limitada, o uso de alguns padrões torna-se inviável. Com o crescimento tecnológico a tendência é que o poder de processamento e a memória dos dispositivos móveis aumentem, permitindo que cada vez mais aplicações estruturadas sejam criadas com o uso de padrões de projetos, sem impactar o desempenho.

A descrição de um padrão indica quando ele pode ser aplicado, mas só a experiência pode prover um maior entendimento para decidir quando um padrão pode melhorar a arquitetura ou solucionar um determinado problema de cada sistema em particular.

Concluímos lembrando que um dos grandes gastos de empresas que trabalham com tecnologia da informação refere-se à manutenção de sistemas desenvolvidos. Pesquisas mostram que este gasto pode atingir um percentual de até 60% [17] do custo de desenvolvimento de um projeto. Este trabalho mostrou que o uso de padrões de projetos pode diminuir este percentual gasto com manutenção. Isto porque muitos dos padrões aumentam a legibilidade do código e o que é melhor, como descrito no Capítulo 2, os padrões de projetos refletem as experiências dos melhores programadores, descrevendo as melhores soluções para problemas recorrentes. Conseqüentemente a utilização de tais soluções tem como conseqüência a diminuição da quantidade de trabalho e também o custo na fase de manutenção de um sistema.

6.2 Trabalhos Futuros

Para trabalhos futuros sugerimos o uso de outros padrões de projetos para a realização dos experimentos com a aplicação selecionada.

Uma camada adicional, responsável por controle, poderia ser criada para tirar esta complexidade do *MIDlet* da aplicação. Esta complexidade está no método *commandAction*, responsável por controlar as ações do usuário na aplicação. Ao invés de ser apenas um método, com grande quantidade de linhas de código e alta complexidade ciclomática [17], podem ser criadas classes responsáveis por fazer o controle de diferentes funcionalidades da aplicação. Com isto poderíamos utilizar o padrão de projeto *Facade* [8], para que o acesso a estas classes de controle ocorra através de uma fachada.

Outra possibilidade de trabalho futuro é a realização dos experimentos em outras aplicações J2ME. Desta forma aumentaríamos a amostra dos experimentos produzindo teorias que sejam válidas para um contexto maior.

Durante o estudo dos padrões de projetos não encontramos catálogos de padrões de projetos para serem utilizados em aplicações J2ME. Diferentemente dos padrões para a plataforma J2EE [25], a qual já possui uma grande quantidade de catálogos de padrões publicados na comunidade. Esta falta de catálogos para J2ME não é de se estranhar visto que é uma tecnologia recente. Um trabalho futuro de bastante impacto é a análise detalhada dos padrões de projetos existentes, para a criação de um catálogo de padrões de projetos para aplicações J2ME.

Bibliografia

[1] ALEXANDER, C., ISHIKAWA S., SILVERSTAIN M., et al. *A Pattern Language*: *Towns, Buildings, Construction*. Oxford University Press, 1977.
[2] GABRIEL R. Artigo: *A Timeless Way of Hacking*.
[3] FOWLER M. *Analysis Patterns: Reusable Object Models*. Addison Wesley, 1997.
[4] BOOCH G. et al. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.
[5] FOWLER M. e SCOTT K. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison Wesley, 1999.
[6] UML Resource center at Rational: http://www.rational.com/uml/index.jsp, acessado em Janeiro de 2006.
[7] TOGETHERJ de Together Software: http://www.togethersoft.com, acessado em Janeiro de 2006.
[8] GAMMA, E. et al. (GOF). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman, Inc., 1995.
[9] MUCHOW, J. *Core J2ME – Technology & MIDP*, Pearson Makron Books, 2004.
[10] SUN Corporation. Java http://java.sun.com/, acessado em janeiro de 2006.
[11] TRAVASSOS G., GUROV D. e AMARAL E. Introdução à Engenharia de Software Experimental. Relatório Técnico ES-590/02, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2002.
[12] BASILI, V., CALDEIRA, G. e ROMBACH, H. *Experience Factory*, *Encyclopedia of Software Engineering,* ed. J.J. Marciniak, Vol. I, Wiley, 1994.
[13] SOLINGEN, R. e BERGHTOUT, E. *The Goal/Question/Metric Method: a Practical Guide for Quality Improvement of Software Development*, McGraw-Hill Companies, UK, 1999.
[14] Departamento de Sistemas Computacionais: http://www.dsc.upe.br/, acessado em maio de 2006.
[15] BUNZEN, B. Desenvolvimento de Aplicativos para Dispositivos Móveis na plataforma J2ME, Trabalho de Conclusão de Curso, Departamento de Sistemas Computacionais, Universidade de Pernambuco, 2005.
[16] PORTO DIGITAL, http://www.portodigital.org/, acessado em maio de 2006.
[17] PRESSMAN, R. *Software Engineering: A Practitioner's Approach*, McGraw-Hill Companies, 2004.
[18] METRICS, http://metrics.sourceforge.net/, acessado em maio de 2006.
[19] ECLIPSE, http://www.eclipse.org/, acessado em maio de 2006.
[20] ECLIPSEME, http://eclipseme.org/, acessado em maio de 2006.
[21] APACHE TOMCAT, http://tomcat.apache.org/, acessado em maio de 2006.
[22] J2SE *Software Development Kit*, (SDK) http://java.sun.com/j2se/1.4.2/, acessado em maio de 2006.

[23] SUN Java *Wireless Toolkit*, http://java.sun.com/products/sjwtoolkit/, acessado em maio de 2006.

[24] MOTOROLA iDEN SDK, http://developer.motorola.com/?path=1.2.6.25, acessado em maio de 2006.

[25] JAVA 2, *Enterprise Edition* (J2EE), http://java.sun.com/j2ee/index.jsp, acessado em maio de 2006.

[26] ZELOSGROUP, http://www.zelosgroup.com/, acessado em setembro de 2005.

[27] YUAN, M. *Enterprise J2ME - Developing Mobile Java Applications*, Prentice Hall, 2004.

[28] JAVA Community Process, http://www.jcp.org/en/home/index, acessado em setembro de 2005.

[29] ALUR, D., CRUPI, J. e MALKS, D. Core J2EE Patterns - As melhores práticas e estratégias de design, Editora CAMPUS, 2001

[30] SOARES, S. *An Aspect-Oriented Implementation Method*. PhD thesis, Centro de Informática (CIn) — Universidade Federal de Pernambuco (UFPE), 2004.

[31] SMALLTALK, http://www.smalltalk.org/, acessado em junho de 2006.

Apêndice A

Questionário sobre a utilização de padrões de projetos* no desenvolvimento de aplicações J2ME

- Entender padrões de projetos como "*Design Pattern*" descrito no livro *Design Patterns: Elements of Reusable Object-Oriented Software* publicado em 1994 (Gamma et al), não se limitar aos padrões de projetos descritos no livro.

Seção 1 – Identificação e Perfil
Os dados a seguir não são obrigatórios.

Nome: _____ Data: ___/___/_____
Curso: ( ) Ciência da Computação ( ) Engenharia da Computação
( ) Sistemas de Informação ( ) Processamento de Dados
( ) Outros _____
Nível: ( ) Curso Técnico ( ) Graduação
( ) Mestrado ( ) Doutorado ( ) Especialização

Seção 2 - Perguntas

1. Você acha válido o uso de padrões de projetos* no desenvolvimento de aplicações J2ME?

( ) Sim.
( ) Não.

2. Justifique a pergunta anterior (número 1).

3. Você já utilizou padrões de projetos* no desenvolvimento de aplicações J2ME?

( ) Sim.
( ) Não.

As perguntas a seguir são apenas para as pessoas que responderam "Sim" à pergunta anterior.

4. Quais padrões de projetos* você já utilizou no desenvolvimento de uma aplicação J2ME?

5. Dos padrões utilizados quais causaram maiores benefícios à aplicação desenvolvida?