UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES

# Implementação de sistema para Governo Eletrônico com interfaces para comunicações móveis.

**Marcelo Augustus Cubas Pereira**
Bacharel em Ciência da Computação pela Universidade Estadual de Londrina

**Orientador: Prof. Dr. Leonardo de Souza Mendes**
PhD em Engenharia Elétrica Syracus University





**Dissertação de Mestrado** apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, Departamento de Comunicações, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Telecomunicações e Telemática**.

Banca Examinadora:

**Prof. Dr. Leonardo de Souza Mendes – DECOM/FEEC/Unicamp**
**Prof. Dr. Maurício Magalhães Ferreira – DCA/FEEC/Unicamp**
**Prof. Dr. Max Henrique Machado Costa – DECOM/FEEC/Unicamp**
**Prof. Dr. Jônatas Manzolli – DM/IA/Unicamp**

Campinas, 05 de julho de 2005.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

P414i

Pereira, Marcelo Augustus Cubas
Implementação de sistema para governo eletrônico com interfaces para comunicações móveis / Marcelo Augustus Cubas Pereira. --Campinas, SP: [s.n.], 2005.

Orientador: Leonardo de Souza Mendes
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Governo eletrônico. 2. Serviços na Web. 3. Sistema de comunicação móvel. 4. Telefonia celular. 5. Java (Linguagem de programação de computador). I. Mendes, Leonardo de Souza. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

RMS-BAE

Titulo em Inglês: System implementation for electronic government with interfaces to mobile communications
Palavras-chave em Inglês: e-government, Web services, Mobile communication systems, Java, Mobile telephone
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Maurício Magalhães Ferreira, Max Henrique Machado Costa, Jônatas Manzolli
Data da defesa: 05/07/2005

## AGRADECIMENTOS

Gostaria de agradecer inicialmente a minha família a força dada durante o período de mestrado no qual estive em Campinas. A minha mãe Dorothy, a minha irmã Luciana e irmãos Alexandre e Júnior. Agradeço também ao cunhado Carlos, sobrinhos Lucas e Bruno assim como tia Diva, Plínio e a minha avó Anita. Agradeço também a meu falecido pai, Antonio Carlos, que me deu condições de ter feito a graduação e que teria dado enorme apoio para que eu fizesse o mestrado, sendo assim agradeço e dedico este trabalho a ele.

Agradeço aos amigos Alan, Gregório, Karime, Raúl, Laura, Walkíria, Wesley, Ricardo Alberti, Marcio Pardo, Cris, Letícia, Fernanda, Mauro, Rodrigo, Ekler, Ana Carolina, Bruna, Celeste, Leonardo Mendes, Cláudio, Maurício, Gean, Meire, Marcelo Tilli, Duarte, Luis, Davison, Ricardo, Henrique.

Agradeço a dupla, Marcio Pardo e Ricardo Alberti a amizade e troca de idéias construtivas e destrutivas durante este período.

Agradeço a Leonardo Mendes, Mauricio, Gean, Meire, Cláudio e Ricardo Alberti a oportunidade de trabalho na empresa IgnisCom assim como a amizade e convivência.

Agradeço a Unicamp pelo complexo esportivo que por todos estes anos utilizei e que tão bem fez a minha saúde. Agradeço, em parte, também pelo Restaurante Universitário que por muitos momentos forneceu boa alimentação, mas que em alguns outros forneceu a terrível e temida carne moída que tanto odiei.

Agradeço também aos professores Maurício, Max e Jonatas por terem aceitado fazer parte da minha banca de avaliação de mestrado.

Ao professor Mário Proença, agradeço o encaminhamento dado durante a graduação que acabou levando a realização deste mestrado na Faculdade de Engenharia Elétrica e Computação da Unicamp.

Por fim o meu muito obrigado vai para o professor Leonardo Mendes por ter me aceitado e orientado durante o mestrado.

## RESUMO

Neste trabalho realizamos uma contribuição ao tópico governo eletrônico. Foi desenvolvido um sistema de controle municipal de dados utilizando o sistema de cadastro de domicílios e moradores de uma cidade, disponibilizado pelo Sistema Único de Saúde do governo federal brasileiro. Este sistema permite a coleta de dados de forma digital através de telefones celulares ou *Personal Digital Assistant* (PDA). Foi criado também um servidor que irá gerenciar os dados coletados no município e a sua transmissão para o servidor federal que irá controlar os registros de todo o país. As funcionalidades disponibilizadas pelo servidor serão visíveis externamente na forma de um serviço *Web*, isto é, a comunicação entre cliente móvel e servidor ocorrerá através da utilização da tecnologia *Web Services*. A plataforma Java foi utilizada em todos os pontos do sistema, pois através da mesma é possível desenvolver aplicativos tanto para dispositivos com baixo poder de processamento, como telefones celulares, quanto para servidores de aplicação.

## ABSTRACT

In this work we did a contribution in Electronic Government (e-Government) topic. Using, as background, the cadastre system of residences and citizens of a city that was created by unique system of health form Brazilian govern, it was developed a system responsible for control the municipal data. It permits gathering data in a digital way through a cell phone or a Personal Digital Assistant (PDA). It was also created a server that is responsible for manage the data gathered on the city and transmit them to a federal server that will control the registers of all country. All the system functionalities will be visible as a web service. To reach this goal the Web Services technology was used. The Java platform is present in all points of the system because through this technology, it is possible to develop applications that can run from devices with low processing power, such as cell phones, to applications servers.

# SUMÁRIO

# ÍNDICE DE FIGURAS

# 1 Introdução

## 1.1 Motivação

Com o surgimento da Internet, a tecnologia da informação tem sido largamente utilizada por setores governamentais de âmbito nacional, regional e municipal em todo o mundo[Marchionini]. A esta utilização deu-se o nome governo eletrônico.

Segundo definições encontradas em [Aoema, Parreiras] governo eletrônico pode ser tido como a utilização da tecnologia da informação por órgãos governamentais no intuito de transformar as suas relações com os seus contribuintes, fornecedores e outros ramos do poder público. Esta utilização visa, entre outras coisas, melhorar o provimento de serviços aos cidadãos, dar maior acesso a informações de gestão geradas pela máquina pública ao contribuinte, assim como melhorar a interação entre os órgãos públicos e a indústria através da otimização dos processos. Outro objetivo seria o fornecimento de meios mais eficientes e rápidos para o acesso a informações para a tomada de decisões levando a um melhor gerenciamento do governo. Governo eletrônico pode ser também dividido em algumas vertentes: Fornecimento de serviços eletrônicos (fornecer serviços de utilidade pública para o contribuinte), Democracia Eletrônica (disponibilizar meios eletrônicos para a participação do cidadão no processo político, por exemplo, urnas eletrônicas) e *e-governance* (suporte digital para a elaboração de políticas públicas e suporte para a tomada de decisões).

Sendo assim podemos enumerar os seguintes requisitos de governo eletrônico:

1. Melhorar o fornecimento de serviços aos cidadãos;
2. Melhorar o acesso dos cidadãos, as informações geradas pelo governo;
3. Otimização de processos;
4. Dar suporte rápido e eficiente ao processo de tomada de decisões;

5. Fornecimento de um meio rápido para a participação do cidadão no governo.

No Brasil, o governo eletrônico está presente nas seguintes formas: A urna eletrônica brasileira é um claro exemplo de democracia digital, na qual o cidadão cumpre o seu dever cívico de votar através de um sistema digital que se mostrou eficaz nas eleições nas quais foi utilizado. Este sistema trouxe melhoras no processo de votação do país, tendo em vista que o seu resultado é conhecido em um tempo muito menor do que o tempo necessário para a contagem de votos utilizando o antigo processo no qual se empregavam cédulas de papel.

O governo criou também um site [Governo Eletrônico] cujo objetivo é divulgar e fornecer serviços disponibilizados de forma eletrônica para os cidadãos.

Sistemas de declaração de imposto de renda, declaração de isentos, consultas on-line a situação de documentos, entre outros serviços também estão disponíveis para a utilização de forma digital.

Além do fornecimento de serviços e da democracia digital, programas de cadastramento de cidadãos em atividades como saúde e educação, entre outros, estão disponíveis. Estes programas são utilizados para o controle de acesso do cidadão a serviços e para a tomada de decisões estratégicas, tais como o direcionamento de medicamentos e ou obras de infra-estrutura para regiões mais necessitadas reveladas pelo cadastro e pelo acompanhamento da utilização do serviço.

Entre os cadastros citados está o cadastro de residências e moradores de um município, chamado CadSus [Datasus], cujo objetivo é o controle de acesso dos pacientes aos serviços de saúde fornecidos pelo Ministério da Saúde através do Sistema Único de Saúde (SUS) Através deste cadastro os dados de uma residência pertencente a um município, assim como os seus atuais moradores, são enviados do sistema local da cidade para o sistema federal de controle de pacientes. O objetivo principal deste envio é a geração de um número único, conhecido como Cartão Nacional de Saúde. Com este número, na forma de um cartão, em mãos, um cidadão poder ter acesso controlado aos serviços federais

de saúde. Este controle de acesso permite um melhor gerenciamento dos serviços de saúde fornecidos pelo país.

O processo de coleta de dados para o cadastramento do SUS ocorre de modo tradicional, isto é, pessoas responsáveis pelo cadastramento são contratadas pelo município para visitarem as residências e realizarem o preenchimento de formulários de papel. Estes formulários serão enviados para uma central de digitação. Uma vez na central, as fichas serão validadas e as que apresentarem erro serão marcadas para correção. As fichas marcadas para correção são enviadas para os responsáveis pelo cadastro, para que os mesmos realizem a nova coleta dos dados junto ao munícipe. Este processo traz algumas desvantagens, tanto para o município quanto para o cidadão. Para o município serão gerados gastos com deslocamento do funcionário, necessidade de novas impressões de fichas de papel, atrasos no repasse de verbas federais devido ao atraso no cadastramento dos cidadãos. Para o município existe o atraso na geração do cartão definitivo de acesso a todos os serviços disponibilizados pelo governo federal no município.

Este cenário fere os requisitos número 1,3 e 4 de governo eletrônico, pois leva a uma demora no fornecimento de serviços de saúde a população e torna o processo de cadastramento lento e custoso.

Tendo em vista as características do processo, tecnologias que oferecem mobilidade são as indicadas para tornar o sistema de cadastramento utilizado na saúde mais de acordo com os requisitos de governo eletrônico do ponto de vista de geração de cadastros. Além de melhora na forma de coleta de dados, existe a necessidade de criação de um sistema de gerenciamento dos dados coletados para que os mesmos sejam vistos tanto por administradores municipais quanto pelos seus equivalentes federais. Cada região terá as suas necessidades reveladas pelo cadastramento.

Atualmente existe um grande crescimento na comercialização e utilização de telefones celulares [Mchugh], sejam eles tradicionais ou inteligentes (*Smart Phones*, aparelhos oferecem funções tanto de telefones celulares quanto de

*Personal Digital Assistants* (PDA)). Com isso, multiplicam-se tanto plataformas de desenvolvimento, quanto aplicativos que focam estes tipos de aparelhos [Pereira]. Outra tecnologia que também cresce, não apenas em numero, mas também em qualidade dos serviços fornecidos, são os servidores de aplicação que permitem o suporte a aplicações coorporativas. Estes servidores fornecem serviços como segurança, serviço de nome, persistência de dados, controle de transação, entre outras. Ao desenvolver uma aplicação, poderão ser utilizados todos os serviços citados, tirando do profissional responsável pelo desenvolvimento da aplicação a necessidade de criar e testa-los.

Estas duas tecnologias base possuem as características necessárias para o desenvolvimento do sistema de geração de cadastros assim como para o sistema de gerenciamento dos dados coletados. Outras duas tecnologias foram empregadas: Java e *Web Services*.

A tecnologia Java foi utilizada tendo em vista que a mesma está presente desde aparelhos móveis até servidores de aplicações, possibilitando o desenvolvimento de aplicativos, com necessidades específicas de hardware, confinado a uma só linguagem básica. Além desta característica, foi considerado também o fato de Java ser suportado por muitos fabricantes de dispositivos móveis, e pela grande quantidade de servidores de aplicações que também dão suporte ao uso da tecnologia Java.

Para tornar o servidor de cadastros local da cidade completamente acessível, um novo modelo de comunicação utilizando a *World Wide Web*(www) como meio de transporte para permitir a publicação e utilização de serviços[Pilioura] foi empregado. Este modelo de comunicação é chamado *Web Services* (WS). Através da utilização deste modelo de comunicação, o sistema servidor ficará acessível tanto para aplicativos sendo executados em dispositivos móveis quando para os aplicativos tradicionais (*desktop*). Além disso, qualquer fornecedor de aplicativos conhecerá a forma de interação com o servidor, pois o mesmo publicará os serviços fornecidos através de um arquivo XML (*eXtensible*

*Markup Language*). A utilização deste arquivo irá permitir a geração de classes que serão empregadas na interação do cliente com o servidor.

## 1.2 Objetivos

Este projeto visa dar os primeiros passos para a criação de uma arquitetura integrada para gestão de governo. Através da análise de um sistema atualmente em uso pelo governo federal, serão levantados requisitos e características necessárias para que o mesmo seja inserido em um contexto mais amplo e unificado de sistemas para governo eletrônico. Características como acessibilidade, agilidade de processo e gerenciamento de informações são focados neste trabalho.

O sistema de cadastro de moradores e residências utilizado pelo governo federal para a coleta de informações de saúde, será analisado e uma arquitetura será proposta e implementada para melhor inserir este sistema no contexto de governo eletrônico.

# 2 Introdução técnica.

## 2.1 Visão geral de Java

Com a evolução dos sistemas computacionais, novas arquiteturas de hardware incompatíveis entre si surgiram. Cada uma delas suportando sistemas operacionais distintos sendo que cada um contendo um ou mais tipos de interfaces gráficas com o usuário[Gosling]. Com o surgimento da Internet, da *World wide Web* e do comércio eletrônico a complexidade do desenvolvimento de aplicações distribuídas tornou-se maior. Desta forma, a utilização de ferramentas de desenvolvimento como C/C++ torna difícil a criação de programas cujo objetivo é ser utilizado em ambientes distribuídos.

Neste cenário surge a tecnologia Java, inicialmente denominada Oak[Java-Language 2, Niemeyer], cujo projeto previa vencer os desafios do desenvolvimento de aplicações em ambientes distribuídos[Java-Language 2, Gosling]. Os alvos principais do projeto foram a entrega segura de aplicativos que consomem a menor quantidade possível de recursos do sistema, poder ser executado em qualquer plataforma de hardware e software assim como ter a possibilidade de ser estendido dinamicamente.

A tecnologia Java surgiu como parte de um projeto de pesquisa que visava o desenvolvimento de uma plataforma de software avançada para ser utilizada em uma grande variedade de dispositivos de rede e de sistemas embutidos[Java-Language, Gosling]. Estes dispositivos são pequenos, confiáveis, portáteis e distribuídos. O objetivo era desenvolver uma plataforma operacional para atuar neste segmento. Como resultado[Gosling] deste projeto surgiu a plataforma Java que provou ser ideal para o desenvolvimento de aplicações seguras, distribuídas e baseadas na rede, em ambientes que vão deste de dispositivos de rede até a *World Wide Web* e *desktop*.

Três são os conceitos presentes em Java[Flanagan]: linguagem de programação, máquina virtual de execução e plataforma.

A linguagem de programação Java tem como característica ser de uso geral, concorrente (*multithread*), baseada em classes e orientada a objetos possuindo uma sintaxe similar a da linguagem C. Ela foi projetada para ser simples o suficiente para ser utilizada por muitos programadores. Os projetistas queriam uma linguagem que fosse poderosa, mas ao mesmo tempo, eles tentaram eliminar a grande quantidade de características complexas que prejudicaram outras linguagens orientadas a objeto como C++. Uma característica importante do projeto é o modelo de gerenciamento de memória do Java, através dele objetos são criados pela simples invocação do comando *new*, e a sua liberação ocorre quando o sistema de coleta de lixo disponibilizado pelo ambiente entra em ação. Esta característica tira do programador a responsabilidade pelo gerenciamento da memória utilizada pelos programas criados, diminuindo a possibilidade de erros que tanto incomodava programadores que utilizavam a linguagem C/C++ por exemplo[Gosling]. Como resultado de um projeto elegante e com características de próxima geração, a linguagem Java se tornou popular entre os desenvolvedores que estavam acostumados a trabalhar com linguagens difíceis e menos poderosas[Flanagan].

Para poder atuar em ambientes heterogêneos, um programa criado na linguagem de programação Java, quando compilado, é convertido para um código de bytes que é um formato intermediário independente de arquitetura, projetado para transportar código de maneira eficiente para múltiplas plataformas de hardware e software.

A máquina virtual Java, também conhecida como interpretador, é o ponto chave da portabilidade oferecida. Sistemas criados em Java somente poderão ser executados em uma determinada plataforma se existir a implementação do interpretador para a mesma. A máquina virtual pode ser codificada diretamente em hardware, porém ela normalmente é fornecida na forma de programa. As plataformas mais utilizadas, *windows*, *linux* e *unix* fornecem interpretadores. Assim como sistemas *desktops*, existem máquinas virtuais para *set-top box* e versões reduzidas estão disponíveis para dispositivos de mão tais como celulares e PDA's.

Apesar de sistemas interpretados não serem considerados de alto desempenho, a máquina virtual Java tem apresentado um bom desempenho e tem sido aperfeiçoada a cada nova versão[Flanagan]. Existe uma variação de máquina virtual denominada compilação *Just-in-Time*(JIT) que consiste na conversão do código de bytes gerado, na compilação de um programa compilado em Java, para a linguagem de máquina da plataforma onde está sendo executado o sistema. A tecnologia *HotSpot* criada pela empresa Sun, empresa responsável pela criação e gerenciamento do Java, é uma boa implementação de uma JIT.

A plataforma Java é o conjunto de classes criado para dar suporte ao desenvolvimento de programas. Classes relacionadas são agrupadas em pacotes sendo que a plataforma Java define pacotes para entrada e saída de dados, utilização da rede, criação de interfaces gráficas e segurança entre muitas outras.

Muitas características e capacidades foram adicionadas ao Java desde a sua versão inicial 1.0. A versão 1.2 foi um marco significante na trajetória da plataforma. Nela a quantidade de classes triplicou em relação a versão anterior conhecida como 1.1. Pacotes para a criação de interfaces gráficas mais sofisticadas e uma poderosa e flexível coleção de API's para trabalhar com conjuntos, listas e mapas de objetos foram adicionados. Tendo em vista muitas novas características incluídas na versão 1.2, a plataforma foi relançada como o nome plataforma Java 2. Atualmente os trabalhos se concentram na versão 1.4., sendo que a versão 5 está na sua fase de trabalho.

Com a plataforma Java é possível a criação de aplicativos em Java sem sacrificar as características avançadas disponíveis para programadores que estejam escrevendo código nativo focado em um determinado sistema operacional.

A medida que o ambiente de desenvolvimento Java cresceu e se expandiu para contemplar numerosas necessidades de aplicações, a quantidade de pacotes disponíveis aumentou. Desta forma, tanto para poder gerar um agrupamento de pacotes, tendo em vista a necessidade geral de comunidades de desenvolvedores, assim como para criar um mecanismo de geração de revisão de

pacotes, a Sun agrupou a plataforma Java2 em três edições[White] como pode ser visto na Figura 1.

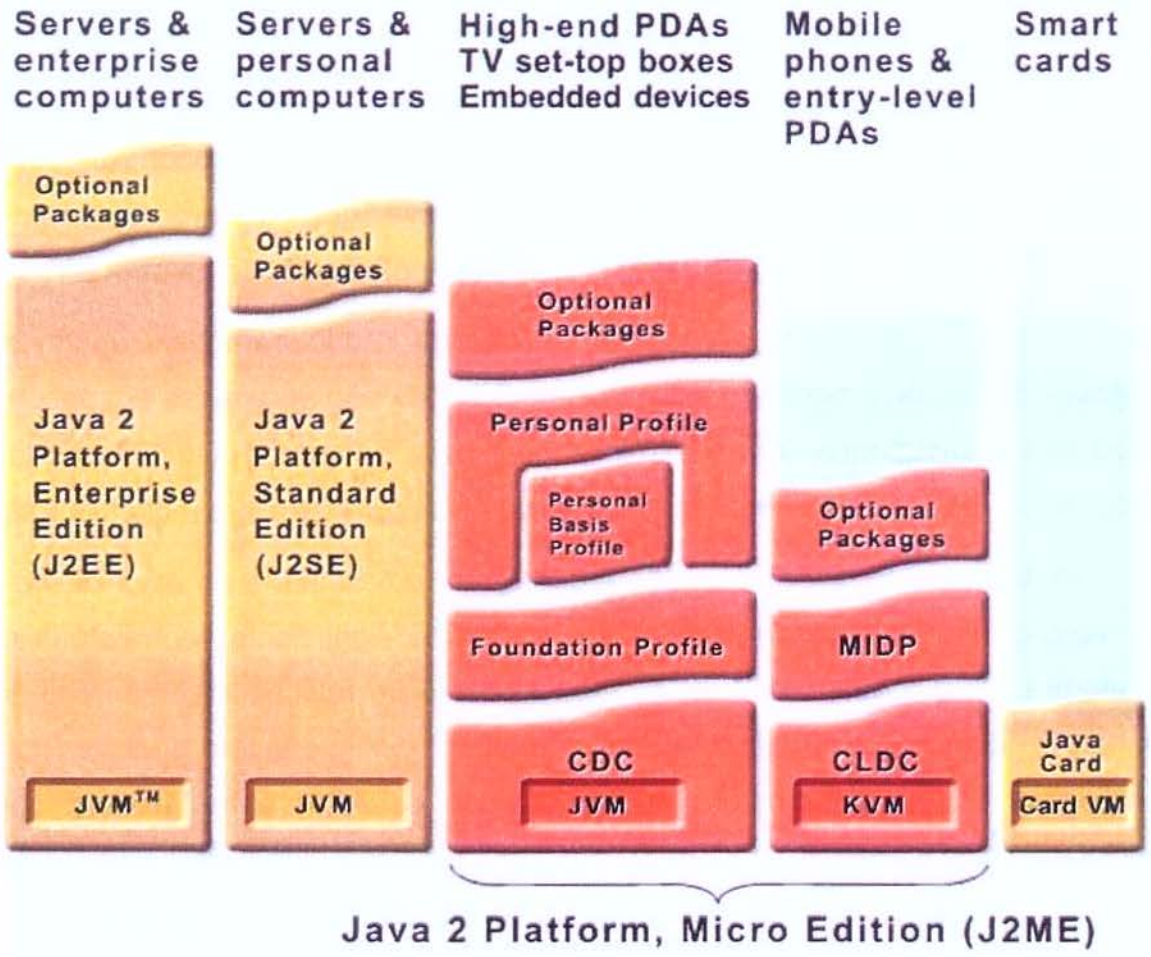


**Figura 1 - Três edições da plataforma Java 2 [Datasheet].**

Cada edição abrange as necessidades de um conjunto específico de aplicações. Em sua base as três edições são muito similares. A sintaxe da linguagem e a arquitetura base de cada edição são praticamente a mesma. No entanto cada edição oferece características únicas aplicáveis ao tamanho do dispositivo para o qual a edição foi construída.

A Edição *Java 2 Standard Edition* (J2SE) representa o ambiente básico Java. Nela estão presentes as classes e API's que constituem o núcleo da plataforma. Este núcleo permite o desenvolvimento de aplicações servidoras e clientes *desktop*, incluindo aplicações que rodam em navegadores WEB. Esta edição serve de base para a plataforma *Java 2 Enterprise Edition* (J2EE).

A Edição J2EE representa um agrupamento de classes e API's Java e tecnologias não Java. Utilizada geralmente para criar aplicações multicamada e potencialmente distribuídas. J2EE é frequentemente descrita como uma camada de software de suporte a comunicação em rede ou tecnologia que atua no lado servidor de uma aplicação. Tecnologias que não são controladas pela empresa Sun e que não estão necessariamente conectadas ao Java, tais como XML e CORBA, estão presentes na edição J2EE.

As características desta edição vêm de encontro às necessidades atuais dos desenvolvedores, que necessitam cada vez mais construir aplicações distribuídas e transacionais utilizando a velocidade, confiabilidade e segurança oferecida pelas tecnologias presentes no lado servidor[Armstrong]. Para suprir as exigências atuais de projeto de aplicações que custem menos, sejam mais rápidas e que ocupem menos recursos possível, imposto pelo mundo do comércio eletrônico e tecnologia da informação, o ambiente J2EE foi projetado para fornecer uma abordagem baseada em componentes para o projeto, desenvolvimento, montagem e implantação de aplicações levando desta forma a redução de custos e maior velocidade de desenvolvimento.

A lógica da aplicação está dividida em componentes de acordo com a sua função, sendo que cada um deles está instalado em máquinas diferentes dependendo da camada da qual ele faz parte dentro do ambiente multicamada proposto pelo ambiente J2EE. Os componentes presentes nesta edição podem ser divididos nas camadas [Armstrong, Monson, Maclaugblin, Bond]:

- Camada Cliente (*Client Tier*): Nesta camada estão presentes os componentes que atuam no lado cliente de uma aplicação. Estes componentes podem ser páginas dinâmicas, programas criados utilizando J2SE assim como *Applets* (aplicativos que são executados em navegadores WEB).
- Camada WEB (*Web Tier*): Componentes que atuam no lado servidor e são formados por *Java Server Pages* (JSP) e *Servlets*. *Servlets* são programas presentes no servidor que processam solicitações

vindas dos clientes. Páginas JSP são documentos baseados em texto que são executados como *Servlets*, porém permitem uma abordagem mais natural para a criação de conteúdo estático.

- Camada de negócio (*Business Tier*): Estão presentes nesta camada componentes responsáveis por conter a lógica de negócio do sistema. Esta camada realiza a interação com a camada de informações do sistema. *Enterprise JavaBeans*(EJB) são os representantes desta camada. Eles podem ser de três tipos: *beans* de sessão (*session beans*), *beans* de entidades(*entity beans*) e *beans* guiados por mensagem(*message driven bean*).
- Camada de informações do sistema (*EIS Tier*): Estão presentes nesta camada sistemas de banco de dados assim como sistemas de informação legados.

A Figura 2 apresenta a arquitetura multicamada disponibilizada pela edição J2EE. Uma aplicação pode não conter todas as camadas disponíveis, porém sistemas mais complexos tendem a utilizar componentes de todas as camadas disponibilizadas pelo J2EE. A Figura 3 mostra a interação entre os componentes das camadas oferecidas. Nela é possível notar que existe a possibilidade de utilização de uma camada de objetos *JavaBeans* intermediária, para o empacotamento de dados a serem trocados entre a camada WEB e a camada de negócio.

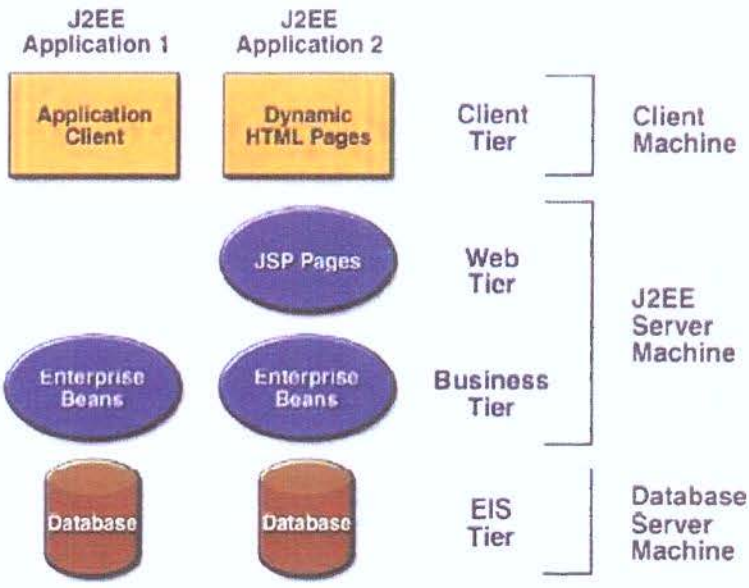


**Figura 2 - Camadas disponibilizadas para o desenvolvimento na edição J2EE [Armstrong].**

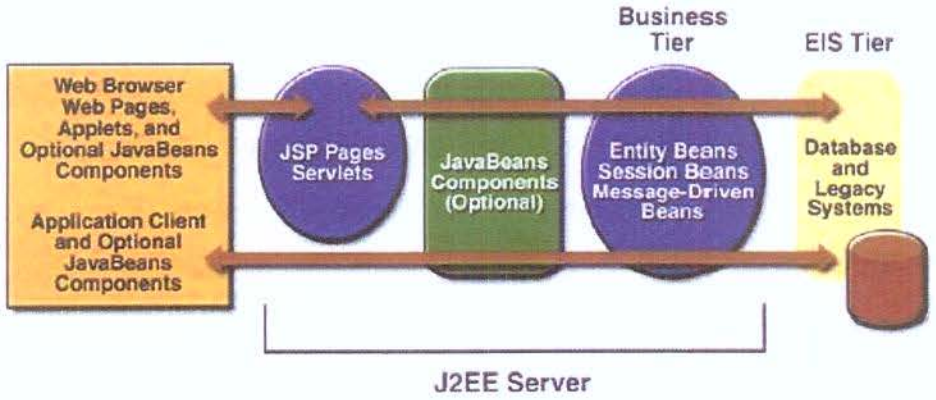


**Figura 3 - Interação entre os componentes das camadas J2EE [Armstrong].**

Por fim está presente a edição *Java 2 Micro Edition*. Esta edição da plataforma Java foca o mundo dos dispositivos com baixa capacidade de processamento tais como celulares, PDA's, *TV set-top boxes*, *pagers*, etc.

A arquitetura desta edição define configurações, perfis e pacotes opcionais como elementos para a construção de um ambiente de execução Java completo que satisfaz as necessidades de grande quantidade de dispositivos alvos da plataforma. Cada combinação leva em conta as capacidades de memória,

processamento e disponibilidade de entrada e saída presente na categoria de dispositivos focado.

Uma configuração é formada por uma máquina virtual e por um conjunto mínimo de classes. Em conjunto elas fornecem as funcionalidades básicas para um grupo de dispositivos que compartilham características em comum. Atualmente existem duas configurações disponíveis [Piroumian, White, Datasheet]:

- *Conected Limited Device Configuration*(CLDC): Utilizada por dispositivos que não estão todo o tempo conectados a rede, tais como celulares, pagers e PDA's de baixo poder de processamento.
- *Conected Device Configuration*(CDC): Projetada para dispositivos que possuem mais memória, processadores mais rápidos e grande largura de banda para comunicação em rede, tais como *TV set-top boxes,* sistemas de navegação de veículos e PDA's de alto poder de processamento. Possui uma máquina virtual mais completa e faz uso de um conjunto mais completo de classes da edição J2SE.

Um perfil fornece um ambiente de execução completo focado em uma categoria específica de dispositivos. Em conjunto com uma configuração, os perfis definem o modelo de ciclo de vida de uma aplicação, a interface do sistema com o usuário e o acesso a propriedades específicas de um dispositivo. Os perfis básicos definidos são:

- *Mobile Information Device Profile* (MIDP): Projetado para telefones celulares e PDA's de baixo poder de processamento. Fornece as funcionalidades centrais necessárias por uma aplicação móvel, incluindo interface com o usuário, conectividade em rede, armazenamento local de dados e gerenciamento da aplicação. Este perfil é utilizado em conjunto com a configuração CLDC.
- *Foundation Profile* (FP): Este perfil é o mais baixo utilizado pela configuração CDC. A medida que novas necessidades são exigidas

pela aplicação, tendo em vista o tipo de dispositivo utilizado, outros perfis são adicionados. Fornece implementação para a utilização da rede de comunicação em dispositivos que utilizam CDC e não disponibiliza interface gráfica como o usuário do sistema.

- *Personal Profile* (PP): Perfil utilizado pela configuração CDC empregado em dispositivos que necessitam de suporte completo a interfaceamento gráfico com o usuário do sistema assim como com Applets.
- *Personal Basis Profile* (PBP): Representa um subconjunto do perfil PP, fornecendo um ambiente de aplicação para dispositivos conectados a rede que dispõem de um nível básico de apresentação gráfica ou necessita de um kit de apresentação gráfica para aplicações específicas.

A plataforma J2ME pode ser estendida pela combinação de vários pacotes opcionais com CLDC, CDC e seus perfis correspondentes. Pacotes opcionais oferecem API's padronizadas para a utilização de tecnologias emergentes tais como *BlueTooth* , *Web Services,* entre outras.

## 2.2 Visão geral de *Web Services*

Quando a tecnologia XML (*eXtensible Markup Language*) surgiu, a mesma foi tida como a base de um novo tipo de tecnologia que iria permitir a interoperabilidade entre aplicações de negócio[Chappell]. A linguagem XML fornece uma maneira genérica de representar dados tipados e estruturados. Com o passar do tempo esta tecnologia evoluiu e atualmente está incorporada em todos os tipos de aplicações. XML atualmente faz parte de sistemas operacionais, protocolos de rede, linguagens de programação, bases de dados, aplicações servidoras, servidores *Web* entre outras.

Nesta evolução, a tecnologia XML foi incorporada em muitos protocolos de rede tendo como objetivo fornecer uma maneira padrão para que duas partes de software estabeleçam comunicação. Os protocolos *Simple Object Access Protocol*

(SOAP) e *XML-Remote Procedure Call* (XML-RPC) surgiram como fruto desta evolução provocada pela linguagem XML. O instantâneo sucesso do protocolo SOAP criou a grande possibilidade de interoperabilidade em um nível que nunca tinha sido alcançado antes. Desta forma SOAP se tornou o protocolo base da revolução *Web Service (*WS)[Chappell].

A promessa principal dos WS é permitir a um ambiente distribuído, que qualquer quantidade de aplicações, ou componentes, possam inter-operar sem dificuldade de uma maneira independente tanto da plataforma utilizada quando da linguagem adotada.

*Web Services* estão baseados no conceito de arquitetura orientada a serviços. Esta arquitetura representa a última evolução da computação distribuída, que habilita componentes de software, incluindo funções de aplicações, objetos e processos de diferentes sistemas, serem expostos como serviços[Nagappan]. *Web Services* são aplicações de negócio auto contidas e modulares, que expõem sua lógica de negócio como se fossem serviços fornecidos pela Internet através de interfaces e utilizando protocolos padrão Internet, tais como HTTP e SMTP, com o propósito de fornecer maneiras de encontrar, se vincular e invocar estes serviços.

Como motivação para a utilização de WS pode ser citado:

- WS pode ser invocado através de mecanismos de chamada remota de procedimento baseados em XML através de *firewalls*.
- WS fornece solução independente de plataforma e de linguagem de programação baseada em mensagens XML.
- WS permite interoperabilidade entre aplicações heterogêneas.
- WS facilita a integração de aplicações utilizando uma infra-estrutura leve sem afetar a escalabilidade.

Cinco são os padrões e tecnologias principais para a construção de WS:

- XML: Utiliza a codificação *Unicode* e está estruturada com dados auto-descritivos que podem ser armazenados como simples documentos de texto para representar dados complexos os tornando legíveis. Atualmente XML é o padrão de fato para estruturar dados, conteúdo e formato de dados para documentos eletrônicos.

- *Simple Object Access Protocol* (SOAP): Protocolo que fornece uma estrutura de empacotamento para o transporte de XML, documentos sobre uma grande variedade de tecnologias padrão Internet, incluindo SMTP, HTTP e FTP. Define também padrões para codificação e ligação para a codificação de invocações não XML de métodos em XML para transporte. SOAP fornece também uma estrutura simples para a realização de chamadas remotas de procedimentos: troca de documentos.

- *Web Services Definition Language* (WSDL): É uma tecnologia XML que descreve de maneira padrão a interface de um WS. WSDL padroniza a forma segundo a qual um WS apresenta seus parâmetros de entrada e saída de uma invocação de método, a estrutura de uma função, a natureza de uma invocação e o protocolo de ligação do serviço.

- *Universal Description, Discovery, and Integration* (UDDI): Fornece um registro de alcance mundial para a divulgação do serviço fornecido por um WS, sua localização assim como questões de integração. Utilizado para a descoberta de WS disponíveis através da busca dos mesmos por nome, identificador e categoria.

- electronic business Extensible Markup Language (ebXML): Conjunto modular de especificações utilizadas para a padronização global do XML facilitando interações comerciais entre organizações independente do seu tamanho. Estas especificações fornecem para o comércio, métodos padrão para a troca de mensagens de negócio baseadas em XML, conduz relacionamentos de negócio, realiza a

comunicação de dados de uma maneira comum e define e registra processos de negócio.

## 2.3 Visão Geral do cadastro do Sistema Único de Saúde

A solução proposta nesta dissertação tem como base as regras propostas pelo sistema de cadastro de domicílios e moradores de um município, criado pelo ministério da saúde do Brasil para a formação de uma base nacional de cadastros do Sistema Único de Saúde (SUS), conhecido como CADSUS. Este sistema fornece um conjunto mínimo de informações a serem coletadas por agentes de saúde. Outro módulo disponibilizado é a validação dos dados coletados, habilitando os mesmos a serem transmitidos para o servidor central do sistema localizado no distrito federal.

Através dos dados enviados para o ponto central, são gerados cartões contendo um número que será único e válido para todo o território nacional, permitindo ao seu portador utilizar os serviços de saúde disponibilizados pela federação.

Os municípios são responsáveis por coletar os dados dos seus cidadãos para permitir que os mesmo utilizem os serviços fornecidos pela federação. O governo federal fornece sistemas para gerenciar a coleta e envio dos dados, porém muitos municípios estão buscando soluções próprias para que tais cadastros possam ser utilizados para gerenciar outros serviços disponibilizados pela cidade. Este fato está ocorrendo tendo em vista que a base de dados gerada pelos aplicativos fornecidos pelo ministério da saúde é fechada, impedindo a sua utilização por outros sistemas.

A comunicação entre os sistemas local e central ocorre através da geração local de arquivos texto que contém todas as informações coletadas de um domicílio e de seus moradores, que em conjunto formam o que chamaremos lar. Cada arquivo pode conter um ou mais lares sendo que os mesmos irão formar um lote de transmissão.

Para garantir que os dados, assim como a estrutura do arquivo de comunicações, estão de acordo com o que pede a norma estabelecida para a criação dos arquivos de transmissão de dados, o ministério da saúde distribui um aplicativo chamado Crítica. A função deste aplicativo é validar os dados a ele submetidos assim como gerenciar lares a serem enviados. Quando apresentarem erros, os arquivos deverão ser corrigidos e submetidos novamente ao programa Crítica.

Após o envio dos lares, o município ficará esperando que o mesmo seja processado pela Caixa Econômica Federal, assim como pelo SUS. O resultado do processamento será um arquivo texto contendo informações sobre o usuário processado assim como o resultado deste processamento. Caso esteja correto, será retornado o número do cartão SUS do cidadão. Caso contrário, o erro encontrado será informado para que seja corrigido.

## 2.4 Visão geral do sistema gerado

O sistema desenvolvido está dividido em três partes fundamentais:

- Aplicativo móvel de coleta, edição e transmissão de informações.
- Sistema de envio de dados via *Web Services* do aplicativo coletor para o servidor.
- Plataforma servidora cujo objetivo é receber os dados transmitidos assim como processar e armazenar os mesmos em uma base de dados local. Também é funcionalidade da plataforma servidora visualizar tais registros armazenados tendo como função corrigir os mesmos, caso necessário, assim como gerar os arquivos com os dados de lares a serem enviados para o ponto centralizador do sistema do SUS.

A Figura 4 apresenta a arquitetura criada.

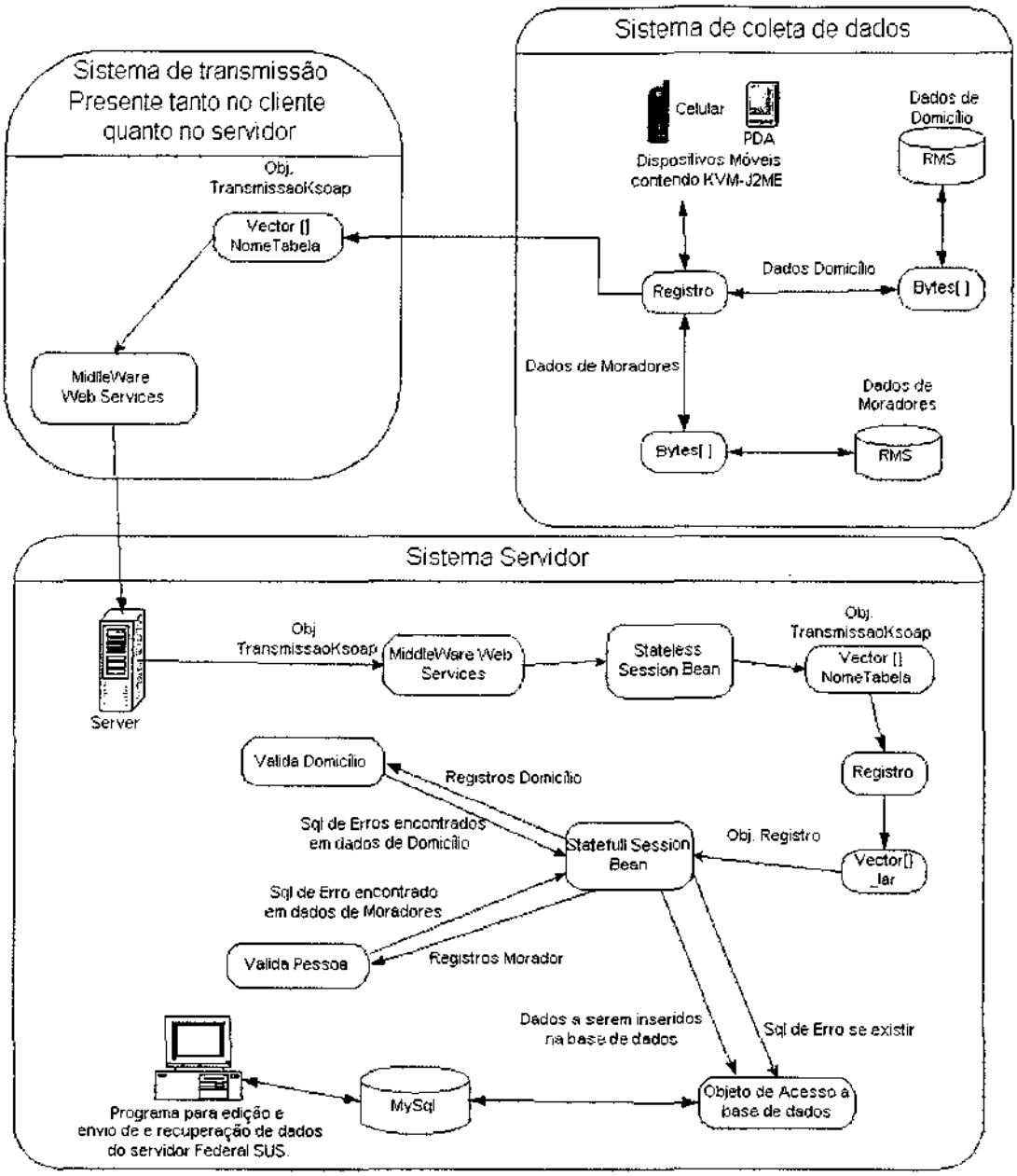


**Figura 4 – Visão geral do sistema**

O módulo de coleta de dados é formado por um programa criado através da tecnologia J2ME. Sendo assim o mesmo está apto a ser instalado em uma grande variedade de celulares e PDA's (*Personal Digital Assistant*) sem que haja a necessidade de grandes modificações para a sua implantação. Os registros inseridos contêm todas as informações requeridas pelo sistema CADSUS. Estes registros ficam armazenados em uma base de dados local do dispositivo sendo permitida a sua edição e complementação antes da transmissão para o servidor.

A transmissão dos registros contidos no aparelho móvel será realizada através da utilização do *middleware Web Services* que utiliza os protocolos padrão Internet para realizar a chamada remota de procedimentos. Por estar sendo empregado em dispositivos que apresentam baixa quantidade de recursos, existe a necessidade de se utilizar uma versão mais compacta deste *middleware* de chamada de procedimentos, denominada *Wireless Web Services*. Foi empregada a API *KSOAP* [Ksoap] para esta tarefa tendo em vista as características reduzidas apresentadas pela mesma, cujo foco principal é o seu uso em ambientes com limitada quantidade de recursos computacionais.

O último módulo do sistema é a plataforma servidora que tem como função principal receber os registros enviados pelo cliente móvel e processar estes registros através da validação dos mesmos segundo leiaute fornecido no sistema CADSUS. Após a validação dos registros poderá ser gerado um registro de erro que será armazenado na base de dados local do servidor. Caso não existam registros iguais na base do servidor, os dados enviados serão armazenados e ficarão disponíveis para consulta através de um aplicativo *desktop* cujo objetivo será a correção em caso de erro apresentado no registro transmitido.

A tecnologia EJB (*Enterprise Java Beans*), presente na API J2EE englobada na plataforma Java, foi utilizada para gerar o código de negócio do servidor. O ponto de entrada do sistema é formado por um *Bean* de sessão sem estado (*Stateless session bean*), que expõe os seus métodos através de *Web Services*. O componente central é formado por um *Bean* de sessão com estado (*Statefull session bean*), cuja função é enviar os registros recebidos para serem validados por objetos auxiliares, que irão informar se os dados apresentam erros ou se estão de acordo com o leiaute usado pelo sistema CADSUS. Após esta validação os dados serão enviados para a base de dados ficando disponíveis para consultas futuras.

Depois de armazenados, os lares que não apresentarem erros serão empacotados em um arquivo de texto e transmitidos para o sistema federal de

coleta de registros da saúde. Os registros errados poderão ser corrigidos e na seqüência transmitidos.

O capítulo 2 irá apresentar a tecnologia J2ME utilizada para a criação do aplicativo de automação de cadastramento, assim como a estrutura de classes e da base de dados local criada para a coleta de registros de domicílios e seus moradores.

O Capítulo 3 irá abordar *Web Services* e as classes criadas para a utilização do mesmo no sistema de transmissão de dados do dispositivo móvel de coleta de registros para o servidor de dados.

No capitulo 4 será apresentada a tecnologia J2EE empregada para a criação do servidor de dados do sistema. Serão apresentadas as classes, assim como a sua função, que foram criadas para que fosse disponibilizado o sistema centralizador de registros de uma cidade. Outra característica apresentada é o sistema de edição, recuperação e transmissão de dados para o sistema do Ministério da Saúde.

Os resultados obtidos estão demonstrados no capítulo 5 sendo que a conclusão e possíveis trabalhos futuros serão apresentados no capítulo 6.

# 3 Sistema de Coleta de dados

O sistema de coleta de dados visa à obtenção e edição de registros de cidadãos e residências de forma móvel através da utilização de dispositivos como telefones celulares e PDA's. A plataforma Java disponibiliza o ambiente de desenvolvimento Java *Micro Edition* conhecido como J2ME. Através dele é possível à criação de aplicativos que podem ser utilizados tanto em telefones celulares quando em PDA's sem que haja necessidade de grandes modificações no código fonte gerado. Isto é possível tendo em vista que existem máquinas virtuais codificadas para estes dois tipos de aparelhos eletrônicos que seguem especificações, tornando as mesmas compatíveis com o padrão estabelecido.

Para permitir a coleta, edição e armazenamento dos registros de moradores e domicílios foram criados subsistemas responsáveis pelo gerenciamento destas funcionalidades apresentadas pelo sistema. Por ser uma linguagem orientada a objetos, o desenvolvimento foi baseado em técnicas que visam à concepção da solução tendo como base a interação entre unidades fundamentais que possuem partes estáticas (variáveis) e partes dinâmicas (funções), cujo objetivo é alterar as características da parte estática do componente.

Nas seções que seguem será apresentada à plataforma J2ME assim como uma visão geral do aplicativo gerado e os subsistemas de coleta, armazenamento, busca e edição dos dados de moradores e domicílios de um município, efetuadas pelo aplicativo móvel.

## 3.1 Plataforma J2ME

Atualmente existem duas versões tanto da configuração CLDC quando do perfil MIDP. As configurações CLDC 1.0[JSR-30] e a CLDC 1.1[JSR-139] tem como principal diferença a inserção da manipulação de números em ponto flutuante[JSR-139]. Qualquer aplicativo desenvolvido para a versão 1.0 poderá ser utilizado sem problemas na nova versão 1.1 e terão agora a possibilidade de

manipulação direta de números em ponto flutuante. Esta configuração foi projetada para ser utilizada em dispositivos com[JSR-30]:

- 160 kB a 512 kB total de memória disponível para a plataforma Java.
- Processadores de 16 ou 32 bits.
- Baixo consumo de energia, normalmente operando como a energia de uma bateria.
- Conectividade com algum tipo de rede, normalmente com uma conexão sem fio e intermitente e com baixa largura de banda (banda menor ou igual a 9600 bits por segundo).

Dispositivos como telefones celulares, alguns *pagers*, PDA's entre outros, são suportados por esta especificação.

Os perfis MIDP 1.0 [JSR-37] e MIDP 2.0 [JSR-118] são utilizados em conjunto com a configuração CLDC. Os grupos responsáveis pelo desenvolvimento destes perfis concordaram em limitar o conjunto de API's especificadas, atuando somente em áreas funcionais que foram consideradas absolutamente necessárias para que fosse alcançada grande portabilidade e implantação como sucesso. Dentre as áreas abordadas estão o ciclo de vida de uma aplicação, modelo de assinatura da aplicação, segurança de transações fim a fim (https), registro, rede de comunicação, armazenamento persistente, som, temporizadores e interface com o usuário. Na sua revisão 2.0, estão incluídas novas características tais como interface gráfica com o usuário final melhorada, funcionalidades de jogos e multimídia, maior conectividade, segurança fim a fim.

A configuração CLDC e o perfil MIDP foram utilizados para a codificação do sistema de coleta de dados tendo em vista os dispositivos alvos do aplicativo, isto é, celulares e PDA's. Por não apresentar nenhuma característica presente apenas nas versões 2 tanto da configuração quanto do perfil a ser utilizado, o sistema de coleta em dispositivos móveis irá rodar tanto na combinação CLDC1.0 e MIDP1.0 quanto na sua revisão CLDC1.1 e MIDP2.0.

O perfil MIDP é dividido em dois níveis: API de alto nível e API de baixo nível. Na sua parte alto nível são fornecidos componentes gráficos predefinidos que podem ser adicionados e utilizados em um visor de apresentação. Com esta API, o dispositivo, e não a aplicação gerencia o leiaute, rolagem, navegação e características visuais tais como cor, forma, fonte e pintura dos elementos de tela. Juntamente com componentes de tela de alto nível, MIDP fornece manipuladores de evento de alto nível.

A API de baixo nível fornecida pelo perfil MIDP permite que a aplicação tenha maior controle sobre os elementos a serem apresentados na tela. Esta característica foi projetada tendo em vista o desenvolvimento de aplicações que necessitam de um controle maior sobre a colocação de elementos gráficos na tela, como é o case de jogos.

### *3.1.1 Elementos de Interface Gráfica J2ME*

MIDP define um conjunto de componentes responsáveis pela interface como o usuário do sistema. A Figura 5 apresenta as classe e pacotes disponibilizadas para este fim. As classes marcadas como abstratas deverão ser estendidas por outras classes para que possam ser instanciadas. A classe **Displayable** define a natureza fundamental de todo componente que pode ser apresentado na tela, e a classe **Screen** define a abstração fundamental para a interface com o usuário do MIDP - a tela. A classe **Canvas**, utilizada para a criação de desenhos na tela, e **Graphics**, que representa o contexto gráfico nativo do dispositivo, fazem parte da API de baixo nível MIDP. Todas as classes que têm como objetivo apresentar objetos de interação com o usuário devem estender a classe básica **Midlet**. Os elementos de interação com o usuário são formados pelas classes **ChoiceGroup** (grupo de itens selecionáveis), **DataField** (Apresenta data e tempo), **Gauge** (Apresentação visual de um processamento em andamento), **ImageItem** (A representação de uma imagem), **StringItem** (A representação em tela de uma String) e **TextField** (Campo para a entrada de dados no sistema por parte do usuário).

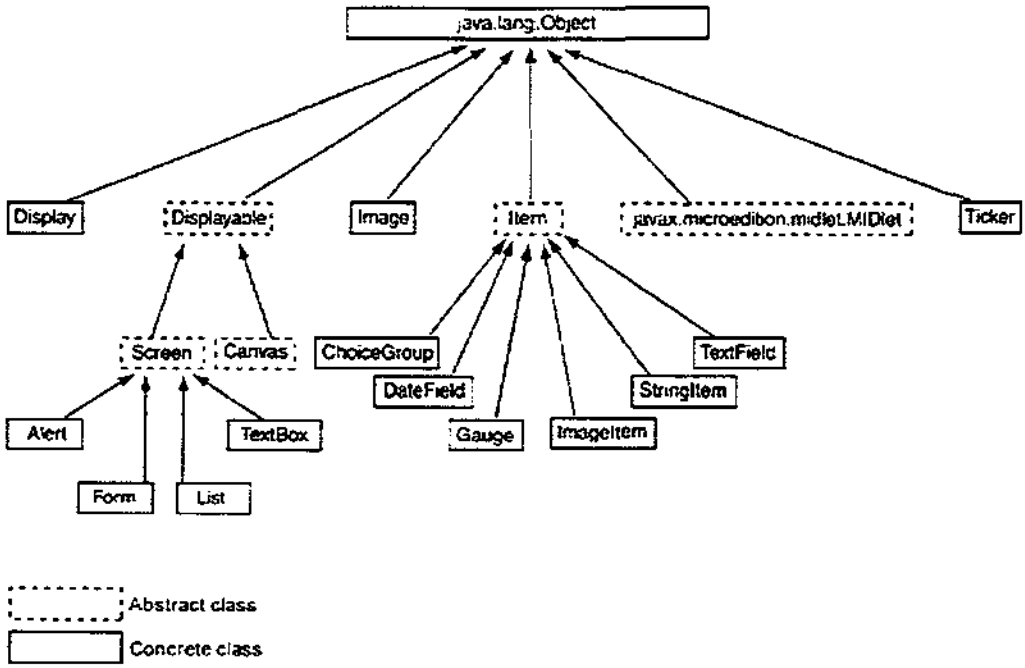


**Figura 5 - Componentes de interface com o usuário do perfil MIDP [Piroumiani] .**

### *3.1.2 Sistema de armazenamento definitivo de dados*

Outra característica importante presente no perfil MIDP é o seu sistema de armazenamento definitivo de dados chamado *Record Management System* (RMS). Esta API permite que dados sejam transportados de forma permanente em dispositivos móveis. Tendo em vista a natureza limitada, em termos de recursos de processamento e armazenamento, dos aparelhos alvos do perfil MIDP, este sistema foi codificado da forma mais simples possível sendo realizado a gravação na forma de vetores de bytes.

Devido a simplicidade da base de dados RMS e a ausência de uma linguagem de consulta, manipulação de conjunto de resultados e assim por diante, presentes em base de dados de aplicativos focados em plataformas que tem mais recursos, a API RMS necessita de somente uma classe principal, **RecordStore**, para armazenar e recuperar dados na plataforma MIDP.

Os dados são armazenados na forma de vetores de bytes que em conjunto formam a base de dados do dispositivo móvel. Esta base de dados está associada

a um **Midlet,** sendo que ao ser excluído do sistema, a sua base associada também será removida.

Cada registro terá um único identificador do tipo inteiro denominado **RecordId.** A medida que são inseridos novos registros este identificador será aumentado em uma unidade em relação ao valor anterior. A localização de registros é feita através do seu **RecordId**.

A API RMS é formada pelas classes e interfaces:

- **RecordStore** é a classe que implementa a base de dados RMS no dispositivo, sendo que ela fornece um método estático (método invocado através da classe e não de uma instancia da mesma) responsável por criar um RMS, e métodos de instancia para a adição, remoção e atualização de registros na base.
- **RecordComparator** Interface responsável por fornecer meios para que seja realizada a comparação entre dois registros presentes na base de dados. Como resultado de uma busca um registro pode ser igual, maior ou menor que o outro. Por ser uma interface, a mesma deve ser implementada por um objeto que deverá implementar o método **compare** que recebe como parâmetro dois vetores de byte que deverão ser comparados através de lógica estabelecida pelo desenvolvedor do sistema. Este método poderá retornar 0 caso os registros sejam equivalentes, 1 se o primeiro vier depois do segundo em ordem de busca ou ordenação e -1 caso o primeiro preceda o segundo em ordem de busca ou ordenação.
- **RecordFilter** interface utilizada para a coleta de registros específicos na base de dados. Por ser uma interface, um objeto deve implementar o método **match** definido. Este método recebe um vetor de bytes e verifica se o mesmo está de acordo ou não com o critério de filtragem fornecido pelo desenvolvedor.

- **RecordEnumeration** interface empregada para a realização de um agrupamento (enumeração) de registros obedecendo algum critério de ordenação. Um comparador de registros e ou um filtro podem ser utilizados em conjunto com uma enumeração para fazer caminhar pelos vários registros presentes na base de dados para atuar somente em certo conjunto de registros filtrados. Através do método *enumerateRecords* da classe **RecordStore**, uma enumeração é obtida. Este método recebe como parâmetro um objeto que implementa a interface **RecordFilter** e outro que implementa a interface **RecordComparator**.

## 3.2 Visão Geral do Aplicativo de Cadastramento Móvel

O aplicativo de cadastramento móvel concebido para ser utilizado em aparelhos cuja principal característica é a mobilidade, visa a coleta de dados de moradores e domicílios de uma cidade como especificado pelo Ministério da Saúde do Brasil. Nele estão disponibilizadas as funcionalidades de cadastrar domicílio, cadastrar morador, buscar domicílio, buscar morador e transmitir dados para o servidor do sistema. A Figura 6 apresenta o menu inicial do sistema de coleta móvel de dados.

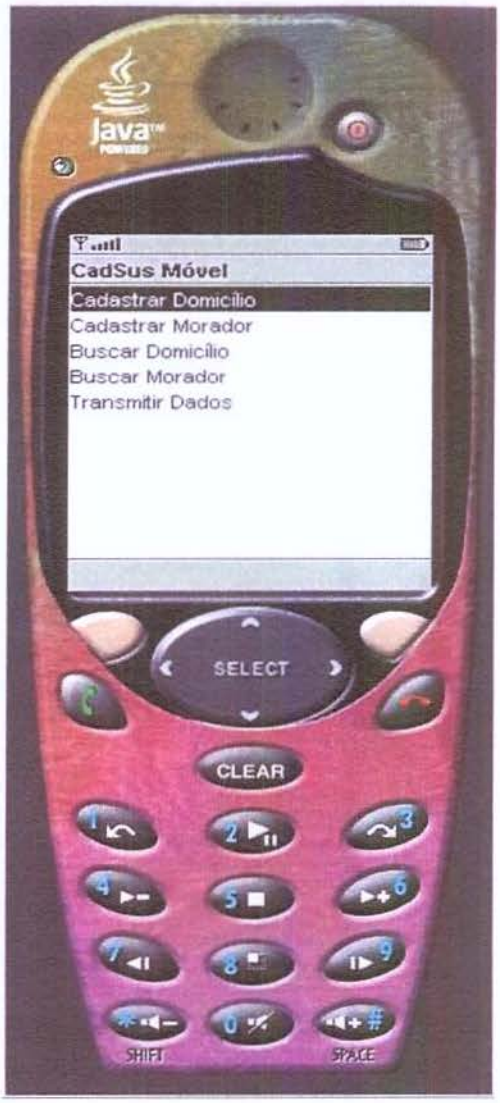


**Figura 6 - Tela inicial do aplicativo móvel de coleta de dados do CadSus.**

## 3.3 Subsistema de coleta

Para o gerenciamento da coleta de registros foram criadas estruturas de dados que visam o armazenamento temporário da informação sendo digitada, assim como uma hierarquia de classes para o fornecimento de interfaces gráficas visando a entrada de dados no sistema.

### *3.3.1 Estrutura de dados*

As classes DEFINICAOREGISTRO, REGISTRO, REGISTRODOMICILIO E REGISTROMORADOR, apresentadas na Figura 7, Figura 8, Figura 9 e Figura 10

respectivamente, formam as estruturas de definição e armazenamento de dados do sistema. Estas estruturas são utilizadas para armazenar os dados enquanto as interfaces gráficas estão sendo preenchidas e posteriormente permitir que os dados coletados sejam armazenados na base de dados local.

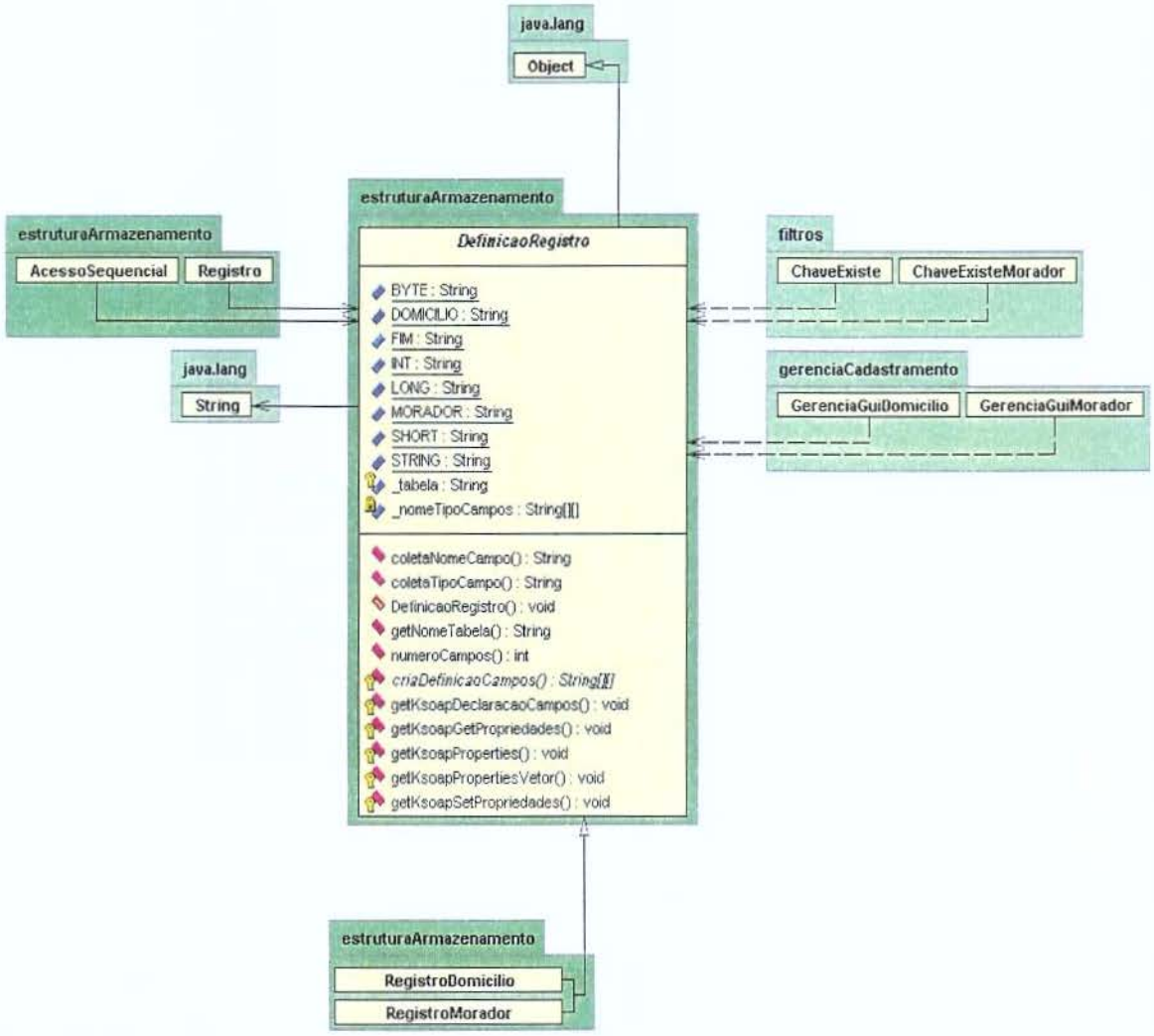


**Figura 7 - Classe Definição Registro**

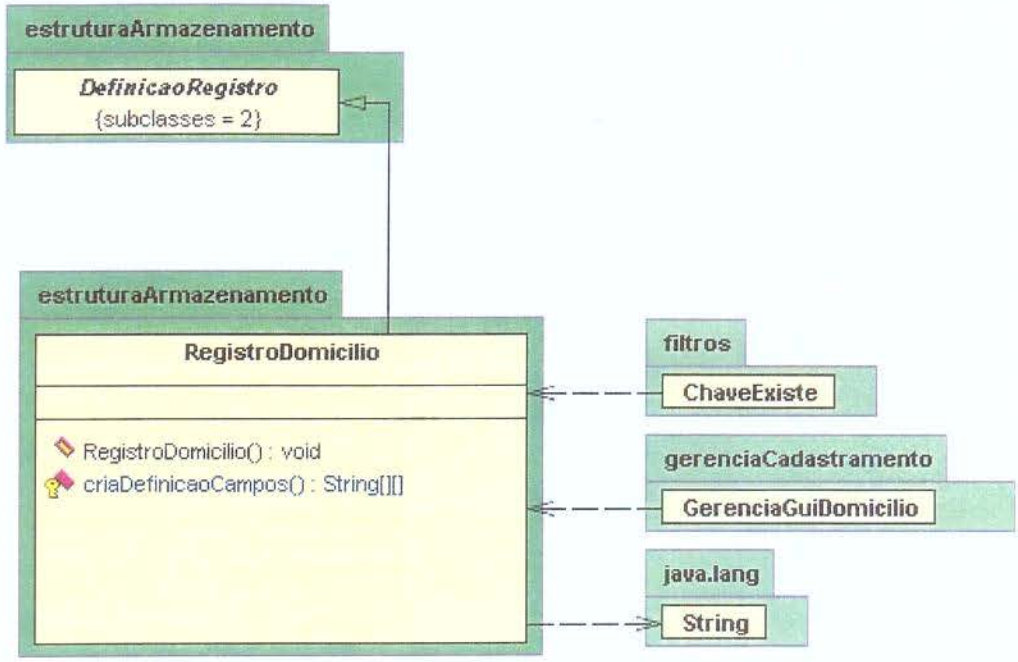


**Figura 8 - Classe RegistroDomicílio**

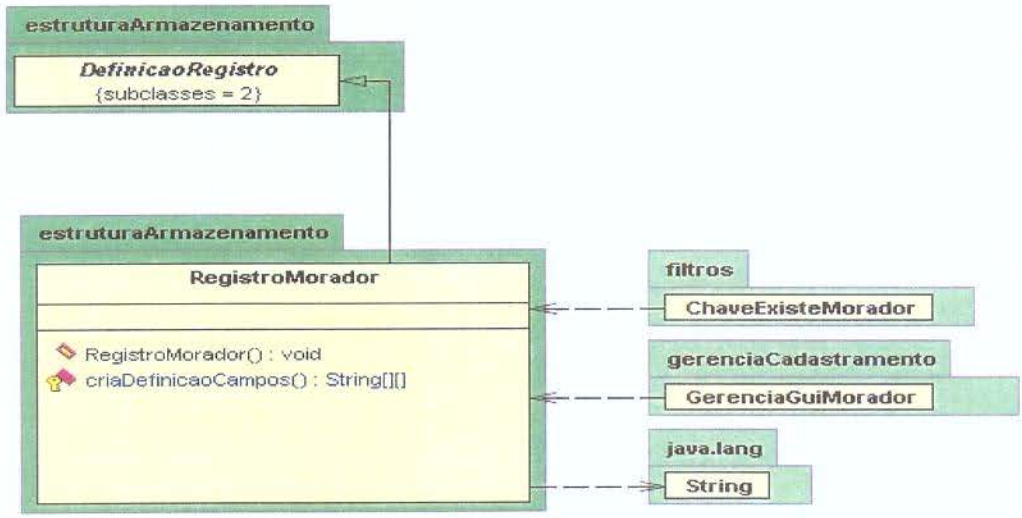


**Figura 9 - Classe Registro Morador**

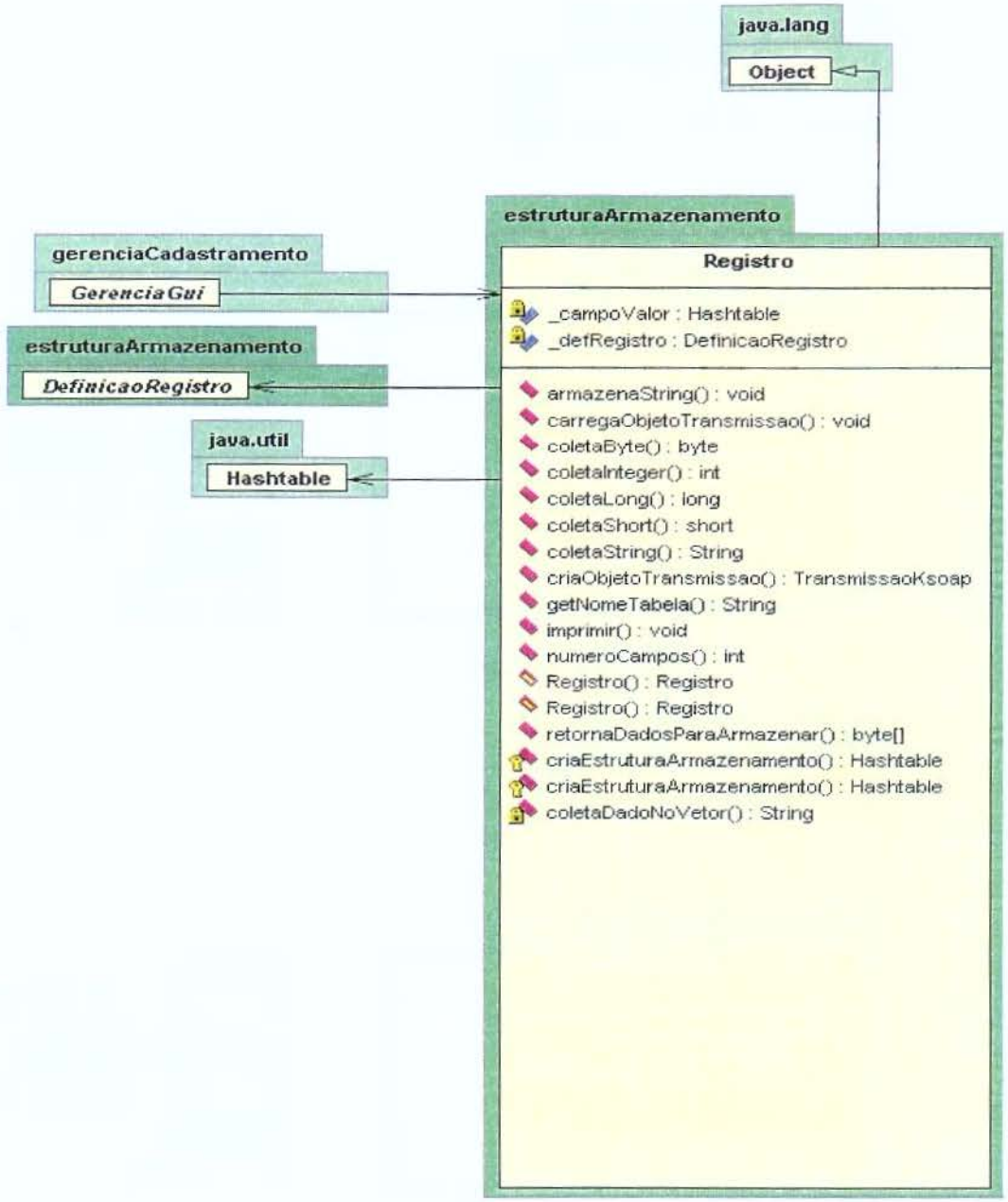


**Figura 10 - Classe Registro**

A classe DEFINICAOREGISTRO servirá de base para a definição das estruturas de armazenamento empregadas. Tal classe contem a variável **_nomeTipoCampo**, a qual é utilizada para informar o nome e o tipo de cada campo a ser coletado via interfaces gráficas do sistema. Este atributo irá atuar como uma definição de tabela em uma base de dados de forma equivalente ao comando SQL *Create table* (tem como objetivo criar uma tabela tendo para isso a descrição dos campos a serem gerados). Outro atributo desta classe é **_tabela**

que informará em qual tabela da base de dados gerenciada pelo módulo servidor do sistema estes campos devem ser armazenados. Este atributo é utilizado para a transmissão dos registros sendo explicado na seção referente ao módulo servidor do sistema.

Através da função abstrata **criaDefinicaoRegistros**, subclasses estarão definindo as estruturas de armazenamento a serem manipuladas.

Esta classe define também funcionalidades de gerenciamento tais como a coleta do nome ou do tipo de um campo dado um determinado índice, coleta do número de campos presentes na estrutura e nome da tabela da base de dados presente no lado servidor do sistema.

No construtor da classe DEFINICAOREGISTRO existe uma chamada a função abstrata **criaDefinicaoRegistros** para que a estrutura **_nomeTipoCampo** seja criada pela subclasse. A chamada é a seguinte: **_nomeTipoCampos=criaDefinicaoCampos();**

As classes REGISTRODOMICILIO e REGISTROMORADOR são subclasses de DEFINICAOREGISTRO, como pode ser visto na Figura 7 localizada na página 30, e tem como objetivo codificar a função abstrata **criaDefinicaoRegistro**, gerando desta forma a definição da estrutura de armazenamento desejada. REGISTRODOMICILIO irá gerar um vetor de STRINGS contendo o nome e o tipo de dado dos campos especificados pelo sistema CadSus além de um identificador (**id**) de tipo inteiro utilizado para o armazenamento dos registros gerados a partir destes dados na base de dados local, como será explicado na seção de armazenamento local de dados. Outro fator observado na função é a atribuição do nome da tabela para o atributo **_tabela** definido na superclasse. O trecho de código presente na Tabela 1, apresenta a codificação da função **criaDefinicaoCampos** que irá preencher os atributos da superclasse DEFINICAOREGISTROS a partir da classe REGISTRODOMICILIO.

```
protected String[][] criaDefinicaoCampos() {
        //informa o tipo da tabela
```

```
_tabela=DOMICILIO;
//criando a estrutura da tabela.
String[][] campos={
        {"id",INT},
        {"n_matricula",STRING},
        {"data_preenchimento",STRING},
        {"uso_municipal",STRING},
        {"cobertura",BYTE},
        {"distrito",BYTE},
        {"areaCS",BYTE},
        {"areaequipe",SHORT},
        {"microarea",BYTE},
        {"setor_censitario",INT},
        {"familia",SHORT},
        {"cod_domicilio",STRING},
        {"n_ficha",INT},
        {"n_pessoas",SHORT},
        {"nome_logradouro",STRING},
        {"numero",STRING},
        {"complemento",STRING},
        {"tipo_logradouro",STRING},
        {"bairro",STRING},
        {"cep",INT},
        {"ddd",SHORT},
        {"telefone",INT},
        {"ponto_referencia",STRING},
        {"tipo_domicilio",BYTE},
```

```
                                {"n_comodos",BYTE},
                                {"tratamento_agua",BYTE},
                                {"abastecimento_agua",BYTE},
                                {"energia_eletrica",BYTE},
                                {"esgotamento_sanitario",BYTE},
                                {"destino_lixo",BYTE}
                        };
                        return campos;
        }
```

**Tabela 1 - Definição dos dados de um domicílio.**

A classe REGISTROMORADOR possui objetivos semelhantes na sua definição tendo, porém a codificação apresentada na Tabela 2.

```
protected String[][] criaDefinicaoCampos() {
                //informa o tipo da tabela a ser criada.
                _tabela=MORADOR;
                //estruta da tabela.
                String[][] campos={
                                {"id",INT},
                                {"n_matricula",STRING},
                                {"data_preenchimento",STRING},
                                {"n_dousuario",BYTE},
                                {"situacao",STRING},
                                {"nome",STRING},
                                {"sexo",STRING},
                                {"cor",BYTE},
                                {"data_nascimento",STRING},
                                {"n_prontuario",STRING},
```

```
{"municipio_nascimento",STRING},
{"uf_nascimento",STRING},
{"cod_ibge_nascimento",INT},
{"nacionalidade",BYTE},
{"pais_origem",INT},
{"data_naturalizacao",STRING},
{"n_portaria",LONG},
{"data_entrada",STRING},
{"escolaridade",BYTE},
{"situacao_conjugal",BYTE},
{"frequenta_escola",BYTE},
{"cbor",STRING},
{"pis_pasep",LONG},
{"cpf",LONG},
{"cartao_nacional",LONG},
{"tipo_certidao",BYTE},
{"nome_cartorio",STRING},
{"n_livro",STRING},
{"n_folha",STRING},
{"n_termo",STRING},
{"data_emissao_certidao",STRING},
{"n_identidade",LONG},
{"identidade_complemento",STRING},
{"identidade_uf",STRING},
{"identidade_orgao",BYTE},
{"identidade_data_emissao",STRING},
{"n_ctps",INT},
```

```
                    {"ctps_nserie",INT},
                    {"ctps_uf",STRING},
                    {"ctps_data_emissao",STRING},
                    {"titulo_eleitor",LONG},
                    {"titulo_eleitor_zona",INT},
                    {"titulo_eleitor_secao",INT},
                    {"nome_pai",STRING},
                    {"nome_mae",STRING},
                    {"distrito",BYTE},
                    {"areaCS",BYTE},
                    {"areaequipe",SHORT},
                    {"microarea",BYTE},
                    {"familia",SHORT}
            };
            return campos;
    }
```

**Tabela 2 - Definição dos dados de um morador.**

É possível notar, na Tabela 2, que o nome da tabela será **morador** e que os campos criados são os especificados pelo Ministério da Saúde para o registro de um morador de uma determinada residência.

As classes apresentadas até agora servem para a criação e manipulação das estruturas de dados fundamentais do sistema. A classe REGISTRO presente no diagrama da Figura 10, situada na página 32, utiliza esta estrutura para realizar a ligação entre a forma de armazenamento dos dados e a entrada dos mesmos no sistema via interfaces gráficas com o usuário final.

Um objeto do tipo registro contem dois atributos, **_defRegistros,** que é um objeto da classe DEFINICAOREGISTRO, o qual representa a estrutura de definição dos dados e **_campoValor** que será uma tabela *Hash* encarregada de

armazenar os dados tendo como chave de busca e armazenamento os campos presentes na estrutura de definição representada pelo atributo **_defRegistros**.

Métodos adicionais são fornecidos para que seja possível o armazenamento e resgate dos dados. Para inserir um valor na tabela, é necessário o fornecimento da chave de ligação, isto é, o nome do campo e o valor do mesmo. A recuperação dos dados ocorre pelo fornecimento do nome do campo.

Existem dois construtores para um objeto do tipo registro, um deles prevê que somente a estrutura de definição dos dados foi informada, neste caso, a estrutura de armazenamento será criada contendo o valor vazio para cada campo inserido. Este cenário ocorre quando se está iniciando a coleta de dados de domicílios ou moradores. O construtor número dois recebe não apenas a estrutura de definição, mas também um vetor de bytes que contem o valor dos campos de um determinado registro. Este caso ocorre quando está se realizando a edição de dados previamente armazenados na base de dados do dispositivo móvel utilizado.

### *3.3.2 Interface com o usuário*

Um conjunto de classes foi criado para permitir a inserção de dados no sistema por parte de um usuário final. Por ter como alvos dispositivos móveis com limitado espaço de apresentação visual de informações, muitas telas foram geradas para permitir a total coleta dos dados. Sendo assim, o padrão de desenvolvimento que prevê a construção de fábricas de objetos foi utilizado.

A Figura 11 apresenta o diagrama de classes criado para a geração do módulo de interação do sistema com o usuário. Nele pode ser constatada a presença de uma superclasse chamada FORMULARIOSCOLETA.

Esta classe contem o atributo **_relacionamento** que é uma tabela Hash responsável por fazer a ligação entre cada campo da interface gráfica com um campo na estrutura de armazenamento. A função abstrata **constroiRelacionamento** permite que cada tela definida a partir de

FORMULARIOSCOLETA seja a responsável por construir todos os mapeamentos entre elementos de interface com um campo na base de dados.

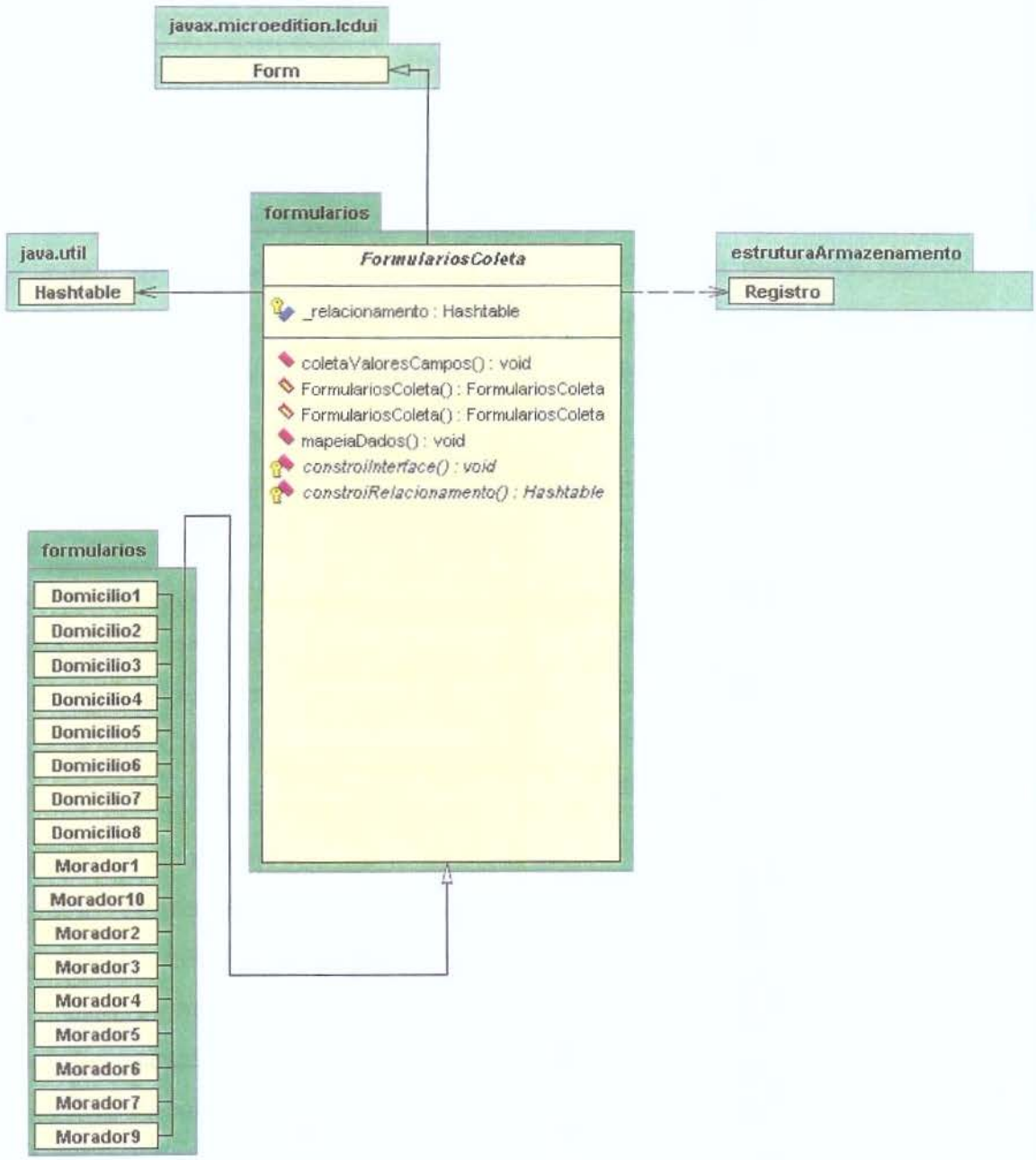


**Figura 11 - Superclasse FormulariosColeta**

Outra funcionalidade abstrata é a construção da interface de apresentação. Tendo em vista que cada interface tem elementos constituintes diferentes, as

classes filhas irão construir a sua interface através da geração de código para a função abstrata **constroiInterface.**

A chamada destes métodos abstratos irá ocorrer no construtor da superclasse permitindo desta forma a personalização necessária das classes filhas. Como parâmetro para o construtor da interface é informada a estrutura de armazenamento, representada por um objeto do tipo registro, que servirá para o mapeamento dos dados presentes na estrutura de armazenamento temporária para a tela que está sendo apresentada em um determinado instante. Os campos a serem utilizados em uma determinada tela são coletados do objeto do tipo registro tendo como base os campos presentes no atributo **_relacionamento** que representa o subconjunto de dados sendo coletados por cada tela.

Para completar o quadro de funcionalidades da classe FORMULARIOSCOLETA, base de todas as interfaces gráficas, estão o mapeamento de dados de um objeto do tipo registro, passado como parâmetro para a tela, para os objetos de apresentação visual de informações, isto é, campos de edição, campos de seleção entre outros apresentados na seção plataforma J2ME. Outra funcionalidade disponível é o mapeamento inverso, isto é, dados presentes na tela são gravados no objeto registro também com o auxílio da estrutura de definição de relacionamentos criada.

Duas fábricas de objetos, FABRICADOMICILIO, apresentada na Figura 12 e FABRICAMORADOR, apresentada na Figura 13, foram criadas tendo como objetivo tornar transparente para o desenvolvedor à criação de classes que representam telas de coleta de dados de domicílios e moradores. Futuras modificações na forma de criação das classes podem ser realizadas sem que haja a necessidade da modificação de cada chamada de instanciação de objetos de tela, mas sim a modificação apenas da classe fábrica, mantendo desta forma a interface de chamada no resto do programa.

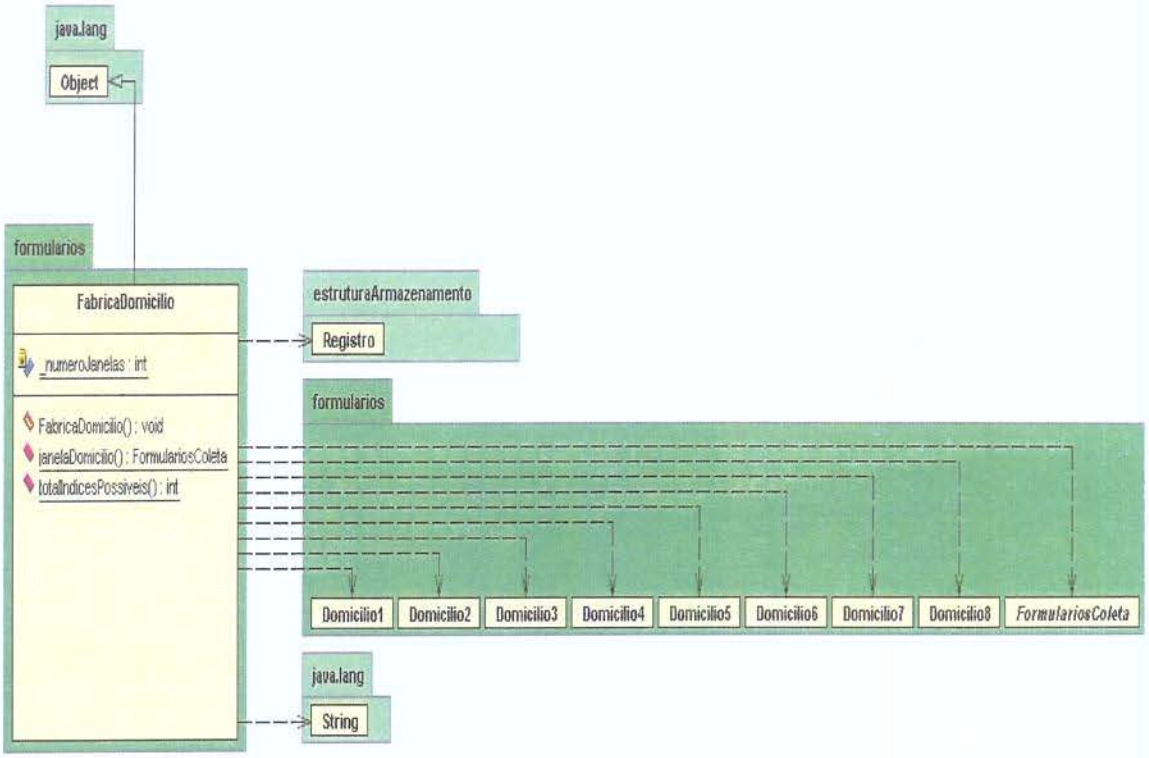


**Figura 12 - Classe FabricaDomicilio**

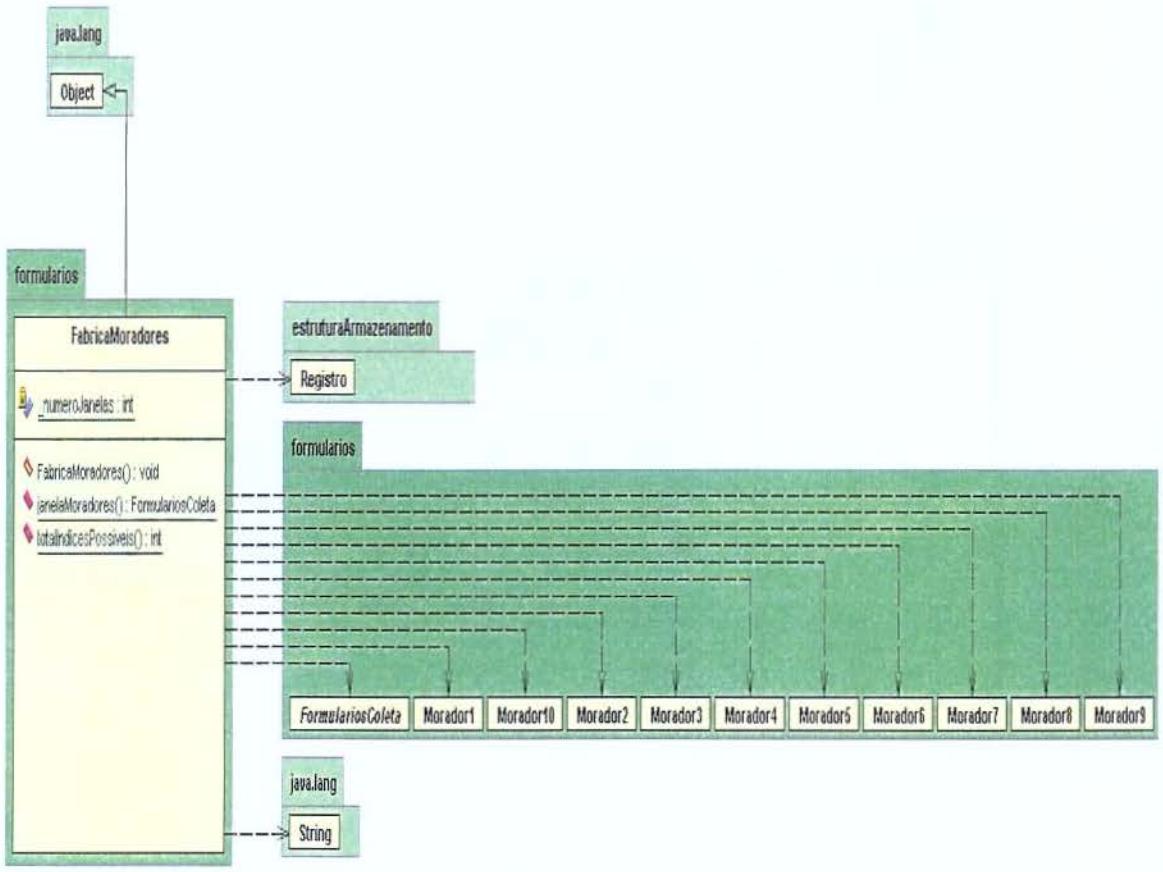


**Figura 13 - Classe FabricaMorador**

As classes FABRICADOMICILIO e FABRICAMORADOR disponibilizam de forma estática, isto é, sem que haja a necessidade de instanciação do objeto fábrica, as funcionalidades **totalIndicesPossiveis** e **janelaMoradores**, para a fábrica de moradores, ou **janelaDomicilio**, para a fábrica de domicílios. A primeira funcionalidade permite que seja feito um laço de repetição no qual todas as interfaces são apresentadas de forma seqüencial. A segunda prevê a criação da interface dado um determinado índice da mesma. Como ponto de entrada para o segundo método está um objeto do tipo registro. Este parâmetro será utilizado como memória de trabalho do sistema de coleta de dados, isto é, a cada nova tela criada e destruída, os dados nela digitados serão armazenados para que novas apresentações desta tela possam ser feitas sem que os dados sejam perdidos.

Uma vez terminada a coleta, é possível armazenar os dados na base local tendo como referência o objeto do tipo registro utilizado como memória de trabalho.

### 3.3.3 Gerenciamento da coleta

Por ter como alvo dispositivo móveis com baixa quantidade de recursos computacionais, assim como mostradores pequenos, um grande conjunto de interfaces para a coleta de dados foi gerado, fazendo com que fosse necessária a criação de classes para o gerenciamento desta etapa. A Figura 14, Figura 15 e Figura 16 apresentam os seus diagrama de classe.

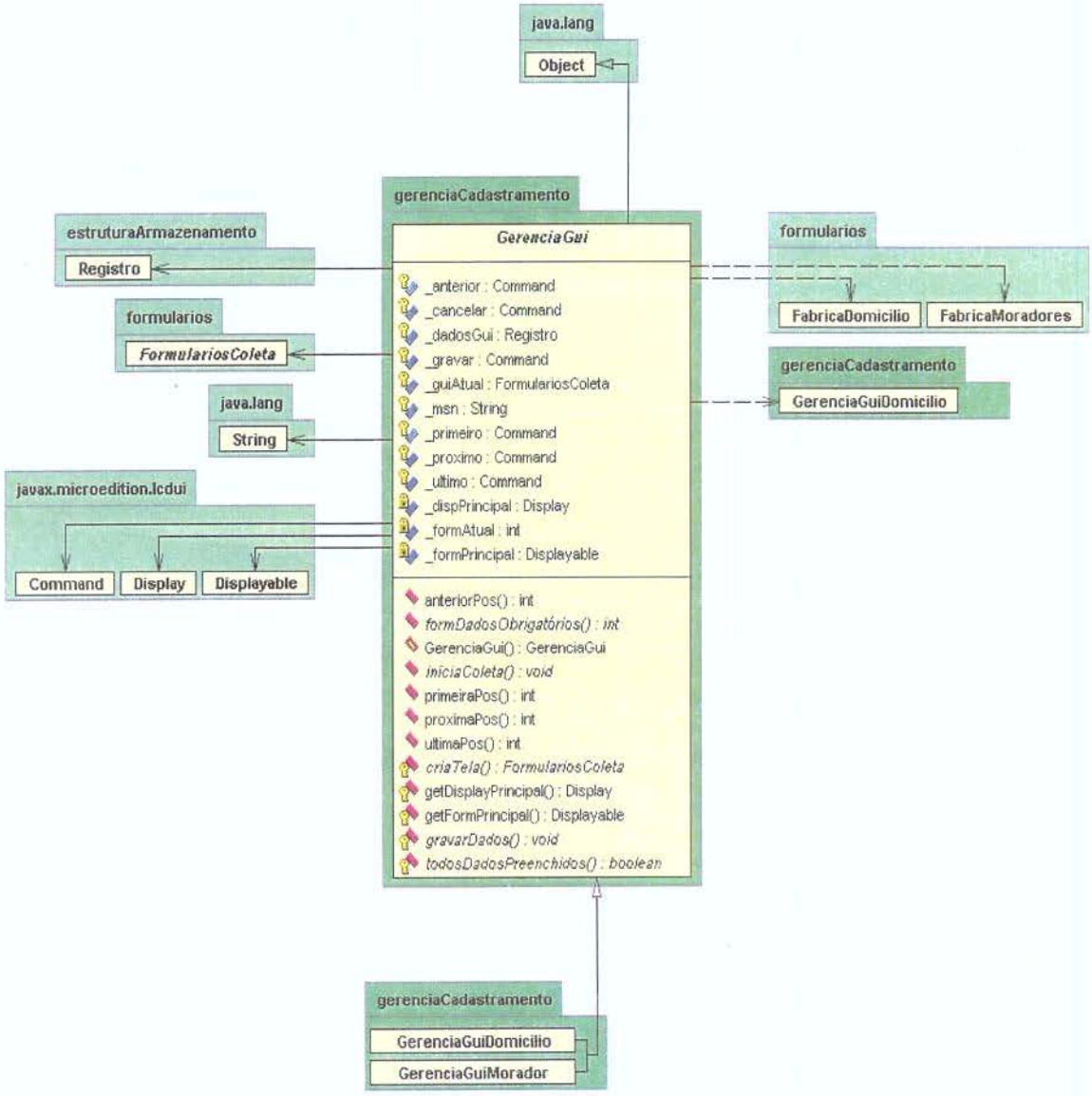


**Figura 14 - SuperClasse GerenciaGui**

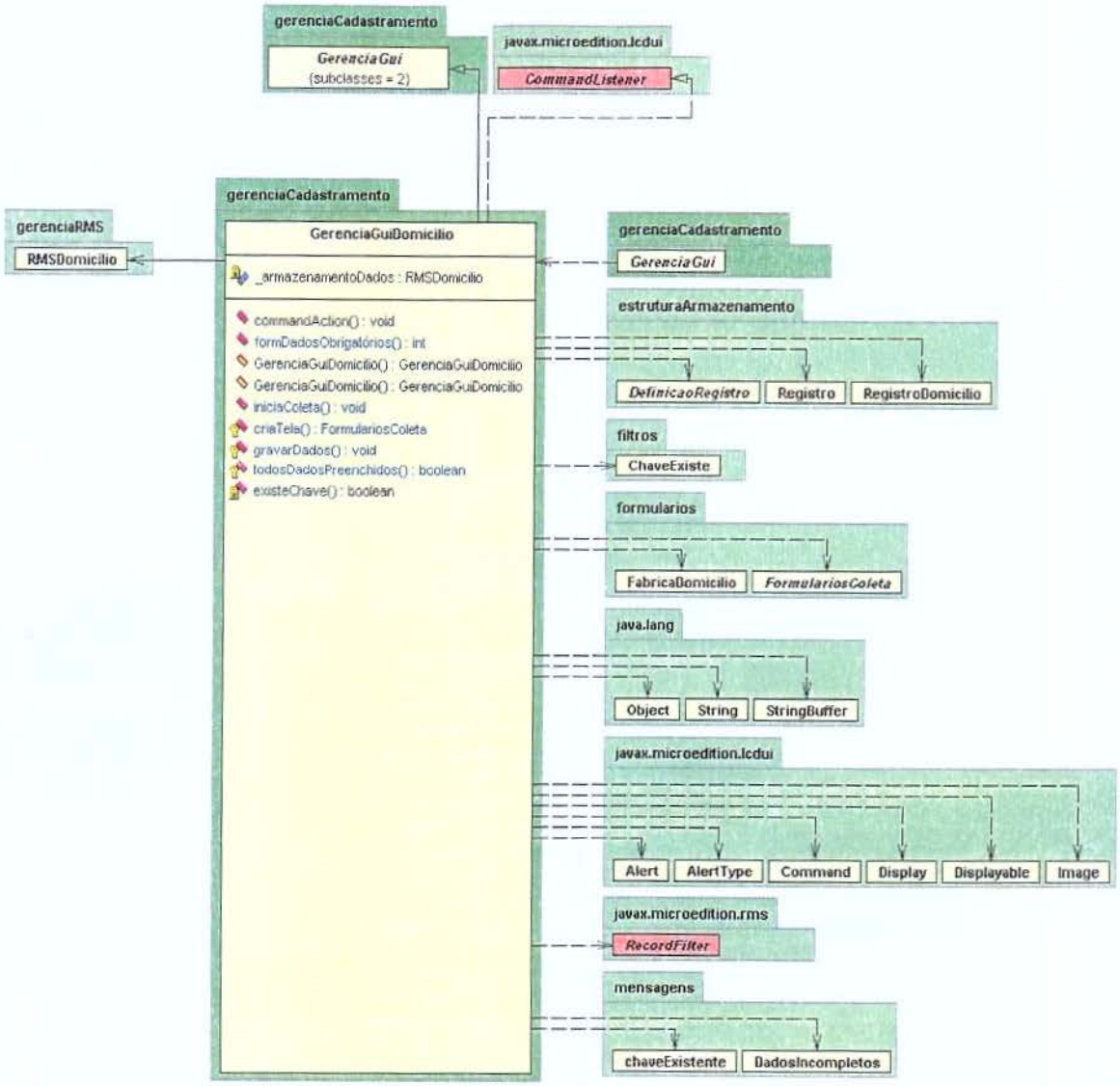


**Figura 15- Classe GerenciaGuiDomicilio**

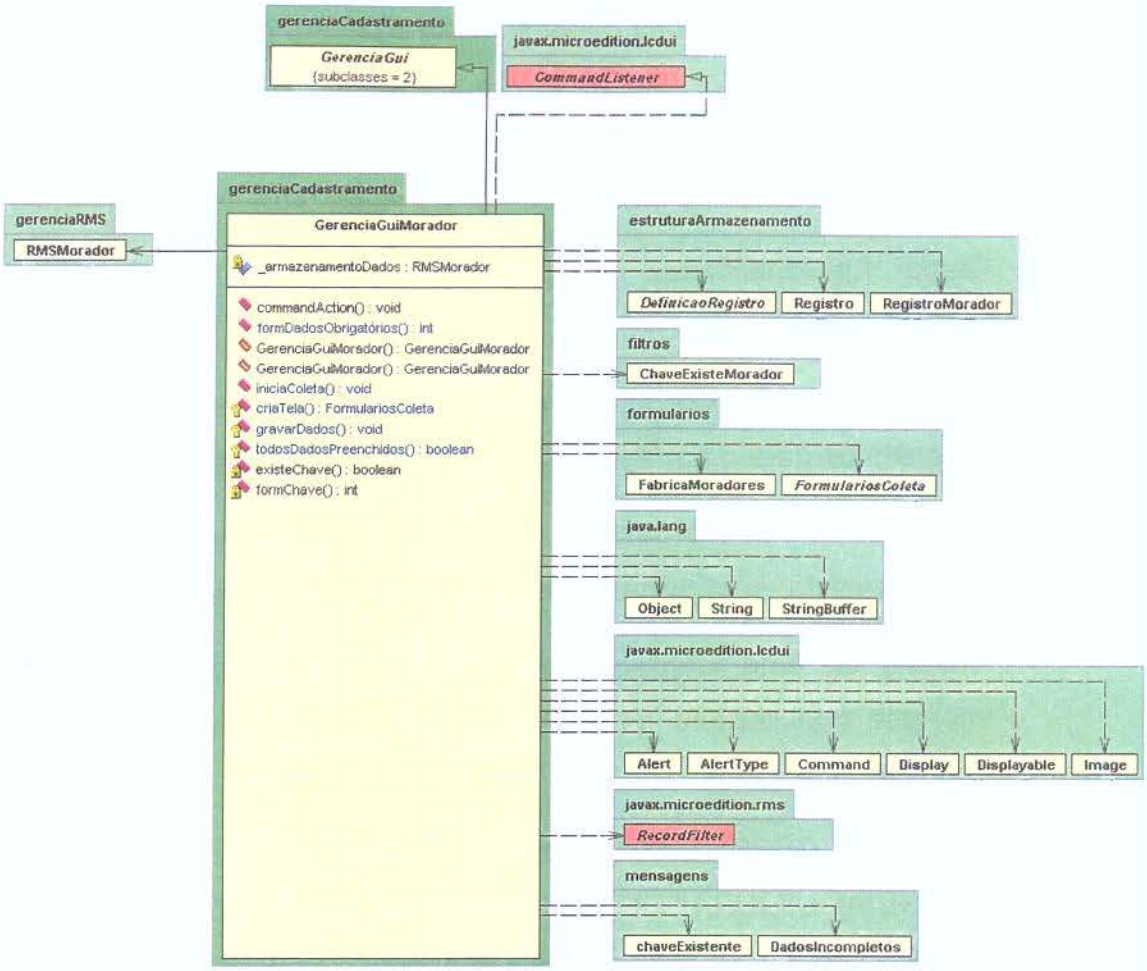


**Figura 16 - Classe GerenciaGuiMorador**

A classe GERENCIAGUI, apresentada na Figura 14 localizada na página 43, representa a superclasse abstrata do sistema de gerenciamento de coleta. Nela estão presentes três atributos principais:

- **_formAtual** representa o índice do formulário atualmente sendo apresentado no dispositivo, o mesmo é utilizado em conjunto com a fábrica de objetos para apresentar uma interface.
- **_guiAtual** contem uma referência para a interface atualmente sendo apresentada.
- **_dadosGui**, um objeto do tipo registro, que atua como memória de trabalho para a coleta das informações digitadas nas telas de domicílios ou moradores.

Entre as funcionalidades comuns estão a navegação pelas telas do registro que está sendo coletado, isto é, funcionalidades que permitem a ida para a próxima tela dentro do conjunto de telas permitidas pela fábrica de objetos de tela, ir para a tela inicial, ir para a tela final ou ainda voltar uma tela na digitação.

Como funcionalidades abstratas a serem definidas pelas classes descendentes estão:

- Verificação do preenchimento de todos os campos considerados obrigatórios para o armazenamento dos dados na base de dados local.
- Gravação do registro gerado em uma base local tendo em vista que cada tipo de registro será armazenado em uma base diferente.
- Criação de uma interface dado um determinado índice e o índice da interface que contem os dados obrigatórios verificados no momento de gravação dos dados.

As classes GERENCIAGUIDOMICILIO, apresentada na Figura 15 página 44 e GERENCIAGUIMORADOR, apresentada na Figura 16 página 45, herdam a classe GERENCIAGUI. As mesmas são responsáveis por definir o comportamento personalizado através da codificação das funcionalidades abstratas definidas na superclasse. Como característica local, estas classes contam com um atributo que é responsável por realizar a conexão deste objeto de gerenciamento de interface gráfica com a base de armazenamento permanente.

A Figura 17 apresenta algumas das interfaces gráficas responsáveis pela coleta de dados de um domicilio. Na última figura do conjunto está presente o menu disponível durante a coleta dos dados. Nele é possível navegar entre as interfaces que fazem parte do conjunto de coleta de um domicílio assim como a possibilidade de armazenar os dados digitados até o momento.


**Figura 17 - Intefaces gráficas de coleta de dados de domicílio**

A Figura 18 apresenta algumas interfaces de coleta de dados de um morador, assim como o menu disponível visto na última figura do grupo.


**Figura 18 - Interfaces gráficas de coleta de dados de moradores**

Para que seja possível a inserção de um registro de morador existe a necessidade anterior de que um domicílio previamente cadastrado seja encontrado através de uma busca na base de dados local. Este processo de busca será abordado na seção subsistema de busca e edição.

## 3.4 Subsistema de Armazenamento

Após ter sido realizada a coleta dos dados, os mesmos se encontram disponíveis no objeto do tipo registro. Este objeto possui código necessário para que seja realizado o armazenamento do seu conteúdo em um sistema de persistência de dados local do dispositivo chamado RMS explicado na seção plataforma J2ME.

O objeto do tipo registro possui um método que é responsável por retornar um vetor de bytes contendo todos os dados a serem armazenados na base RMS. O primeiro dado é um identificador de registro representado por um número inteiro incrementado de maneira seqüencial e que representa a chave de busca dentro da base. Na seqüência serão colocados todos os outros campos obedecendo à estrutura criada através do objeto do tipo DEFINICAOREGISTRO.

Uma vez resgatado um registro da base RMS, o mesmo é carregado para a estrutura de trabalho do objeto do tipo registro através da consulta da estrutura de definição de campos. Esta consulta permite que se saiba qual a posição e o tamanho dos campos dentro do vetor de bytes uma vez que a gravação dos mesmos ocorreu seguindo a mesma estrutura.

Para gerenciar o armazenamento dos dados gerados no sistema, uma hierarquia de classes foi criada.

A classe abstrata GERENCIARMS, apresentada na Figura 19, representa a superclasse dos objetos destinados a gerenciar a interação entre o aplicativo e a base de dados. Nela está presente o atributo **_base** que representa a base local propriamente dita onde serão armazenados e resgatados os registros de domicílios e moradores no dispositivo móvel.

Como funcionalidades esta classe apresenta o armazenamento de novos registros, a edição de registros já presentes na base, o retorno do número do próximo registro a ser armazenado assim como uma função para realizar a filtragem dos dados previamente inseridos.

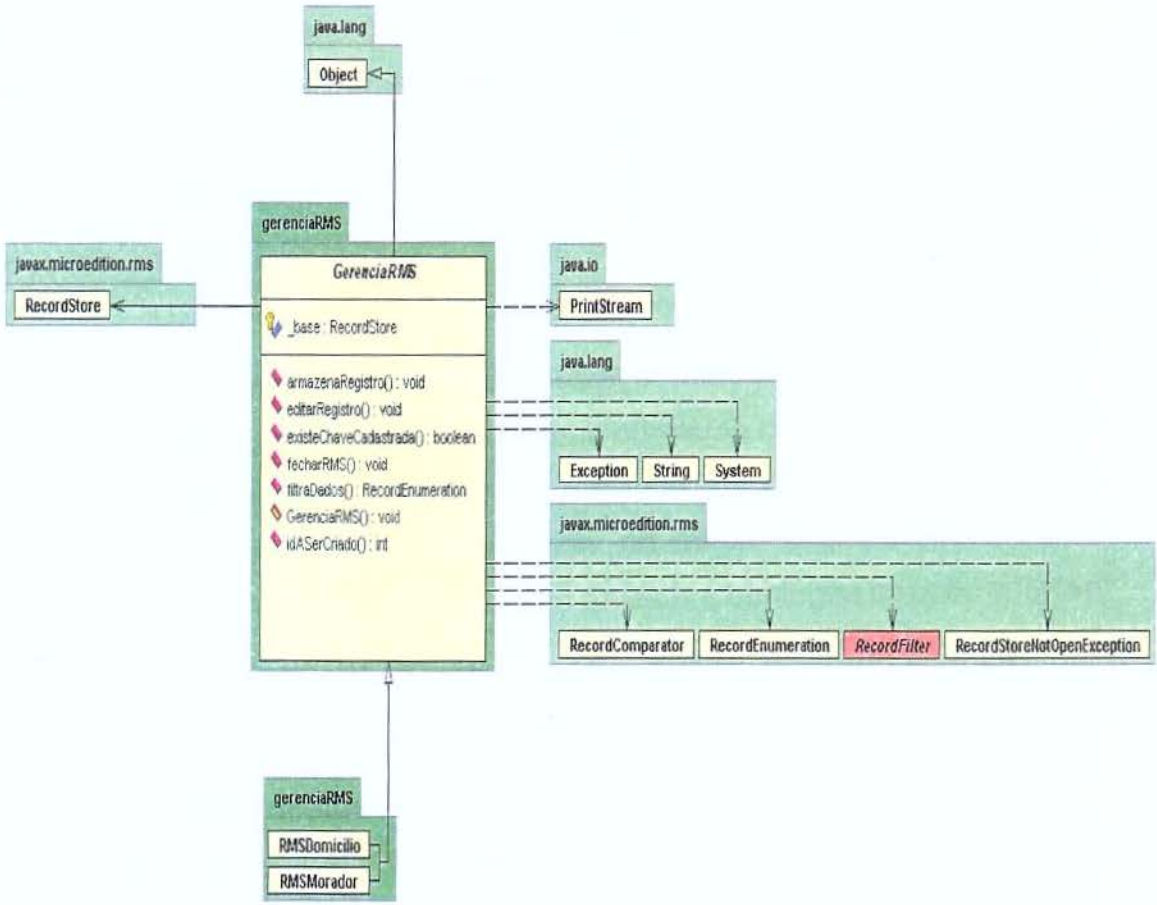


**Figura 19 - Classe GerenciaRMS e as subclasses RMSDomicilio e RMSMorador.**

Para o caso de um domicilio, a funcionalidade **existeChaveCadastrada** foi criada para que antes que ocorra a inserção dos dados na base seja verificada a existência ou não de uma chave geográfica, formada por número do distrito, área de centro de saúde, área de equipe, micro-área e família, previamente cadastrada.

As classes RMSMORADOR e RMSDOMICILIO herdam a classe GERENCIARMS tendo como principal objetivo criar um construtor no qual o atributo **_base** será atribuído. A classe que gerencia os moradores irá abrir ou criar um RMS para moradores e a que gerencia os domicílios fará o mesmo com um RMS para domicílios.

## 3.5 Subsistema de Busca e Edição

O subsistema de busca de dados é formado por dois conjuntos de classes que tem por objetivo final levar os dados de um domicílio ou morador para as telas de cadastramento dos mesmos, para que eles possam sofrer ou não a edição do seu conteúdo. Este conjunto de gerenciadores é apresentado na Figura 20.

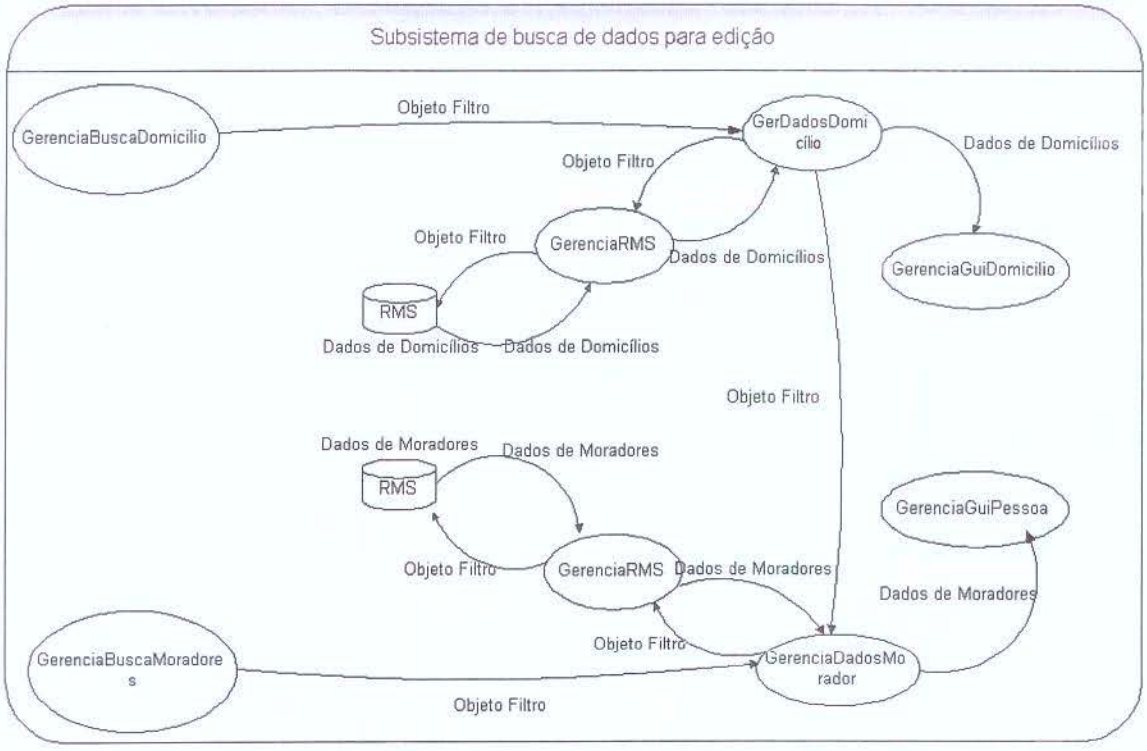


**Figura 20 - Conjunto de classes gerenciadoras de busca de dados.**

As classes **GerenciaBuscaDomicilio** e **GerenciaBuscaMoradores** representam as telas nas quais um subconjunto de dados será solicitado para que a partir deles seja gerado um objeto de filtro contendo os dados informados pelo usuário. O próximo passo será executado pelas classes **GerDadosDomicilio** e **GerenciaDadosMorador**, através do recebimento do objeto filtro gerado pela classe de busca. Todos os registros presentes na base que satisfazem a condição de filtro serão apresentados em uma tela específica para que o usuário selecione qual deseja visualizar ou editar o seu conteúdo.

A classe **GerDadosDomicilio** atua também como um gerenciados de busca, isto é, gera um objeto filtro, quando for requisitada a funcionalidade de

buscar os moradores de um domicílio. Por fim um objeto do tipo registro é gerado e enviado para a classe responsável por gerenciar a coleta e edição dos dados. Para dados de domicílio, o objeto do tipo registro é passado para a classe **GerenciaGuiDomicilio** e os dados de moradores para **GerenciaGuiMorador**.

As figuras Figura 21, Figura 22 e Figura 23, apresentam os diagramas das classes responsáveis pelo gerenciamento da geração de filtros para a busca de registros existentes na base de dados.

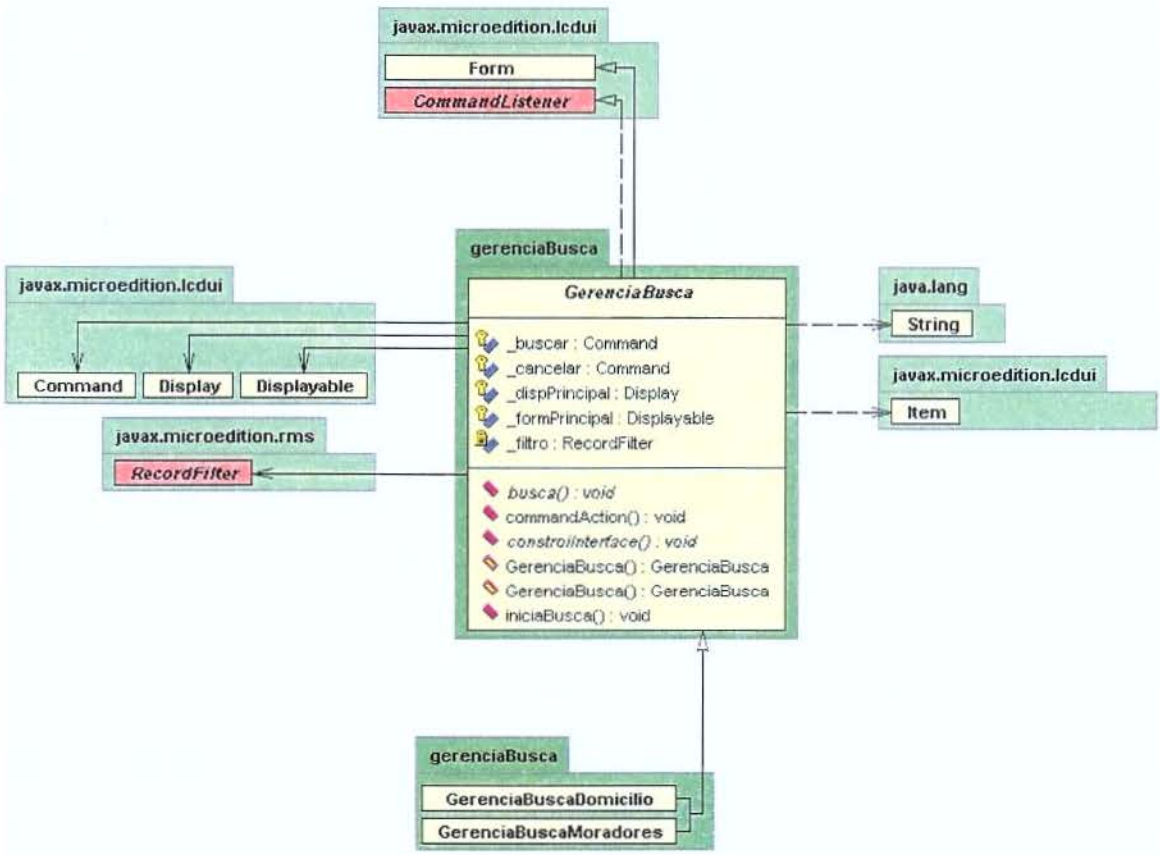


**Figura 21 - SuperClasse GerenciaBusca**

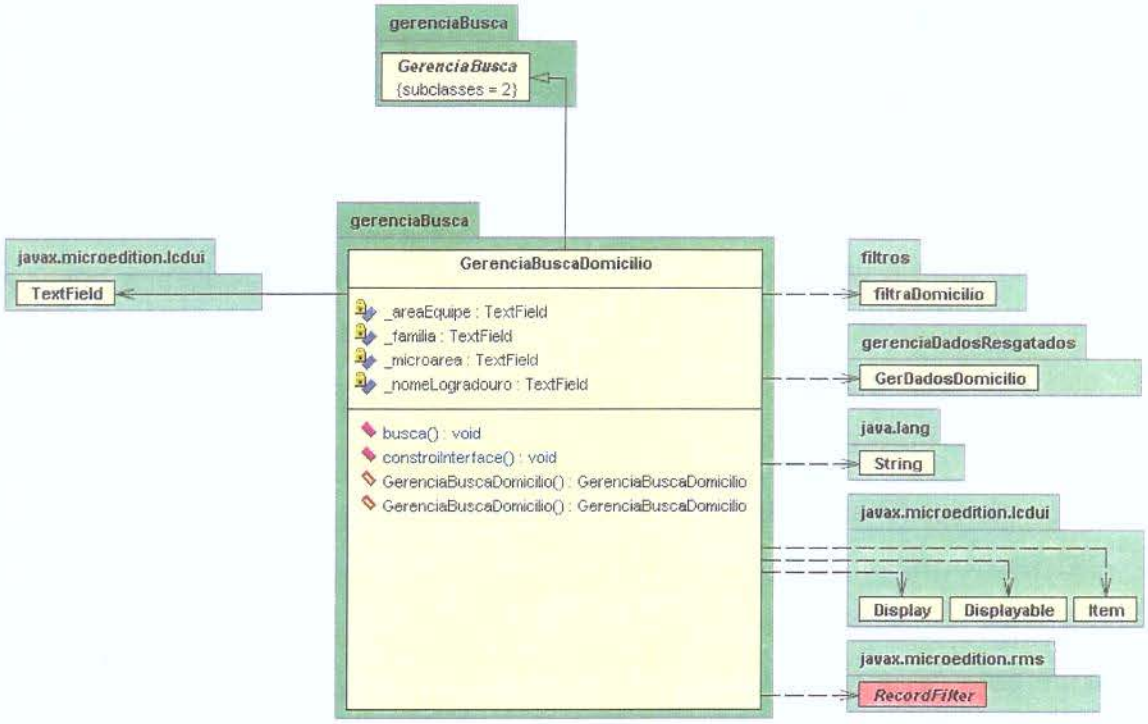


**Figura 22 - Classe GerenciaBuscaDomicilio**

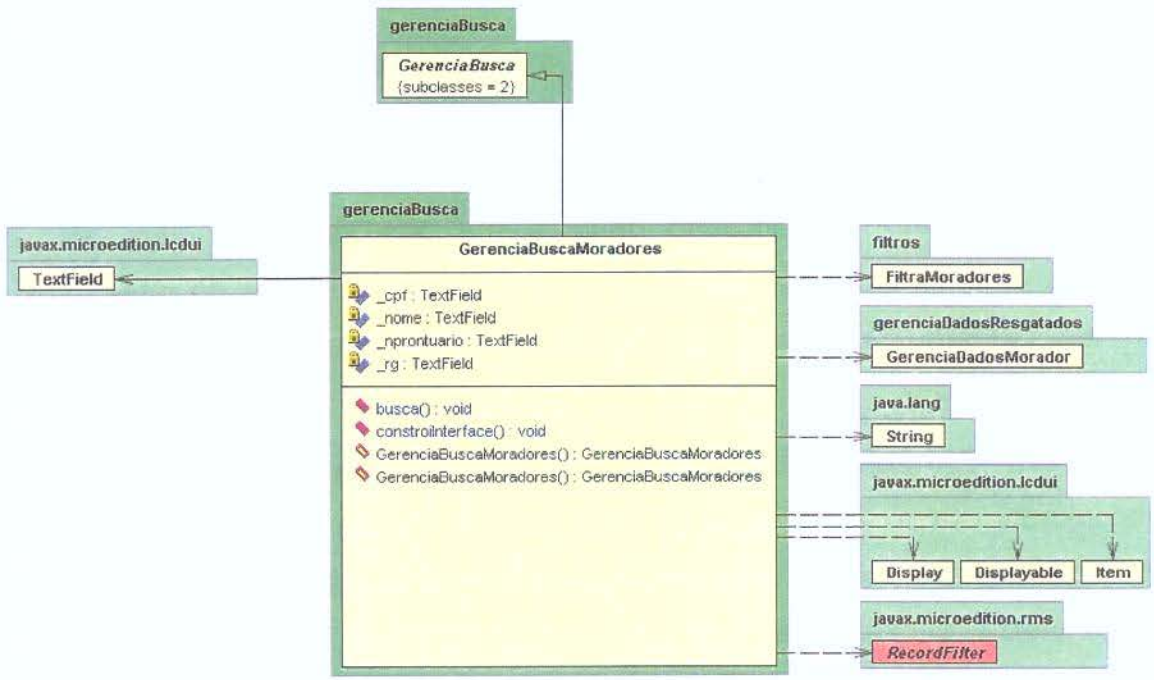


**Figura 23 - Classe GerenciaBuscaMoradores**

A superclasse abstrata GERENCIABUSCA, apresentada na Figura 21, tem como objetivo criar a estrutura básica da interface de geração de filtros. Comandos, processadores de comando de tela assim como o formulário básico

estão definidos nesta classe. O atributo **_filtro** irá apontar para o filtro gerado e que será utilizado para disparar o segundo conjunto de classes responsáveis por efetuar o filtro e apresentar os resultados.

Os métodos responsáveis por: construir a interface gráfica a ser apresentada para o usuário, chamar a classe responsável por realizar a busca no RMS e o responsável por apresentar o resultado, são definidos abstrato para que as classes descendentes moldem de forma personalizada como tais funcionalidades serão realizadas.

As classes filhas GERENCIABUSCADOMICILIO, apresentada na Figura 22, e GERENCIABUSCAMORADORES, apresentada na Figura 23, ficam responsáveis por apresentar-se no visor do dispositivo contendo campos a serem utilizados para gerar um objeto de filtro. No método reescrito **busca** o objeto filtro gerado é passado para a classe de gerenciamento de dados resgatados correspondentes.

Estão definidos dois tipos de filtros, o primeiro deles lida com a filtragem de dados vindos de um RMS que está armazenando domicílios. Todos os filtros deste tipo herdam a classe padrão J2ME RECORDFILTER e implementam a interface **FiltroTipoDomicilio**, identificando desta forma que este filtro atua em um domicilio. O segundo tipo de filtro está relacionado com dados de moradores e também herdam da classe padrão assim como implementam a interface **FiltroTipoMorador.**

Cada filtro contém um conjunto de atributos que irão definir juntos um subconjunto chave de campos a serem buscados na base de dados para identificar registros compatíveis. Os filtros apresentam também o método **matches** cujo objetivo é, em sendo realizado uma iteração na base, informar quais registros estão de acordo com os dados pedidos na filtragem e quais não estão.

Os vetores de byte, retornados nas filtragens realizadas na base de dados local, são formados por dados de domicílios ou moradores. Desta forma existe a necessidade de que seja possível dado um nome de campo, o seu valor seja extraído deste vetor de bytes. Tendo esta informação será possível aplicar o filtro

criado no registro coletado avaliando desta forma a sua adequação ou não ao mesmo.

A classe ACESSOSEQUENCIAL foi criada para realizar o acesso de forma seqüencial ao vetor de bytes retornado da base de dados local. A mesma utiliza as estruturas de definição de registros de domicílios e moradores para saber exatamente em que posição do vetor de bytes obtido se encontra um determinado campo que faz parte do filtro.

A Figura 24(A) apresenta a atuação da classe GERENCIABUSCADOMICILIO e a Figura 24(B) a classe GERENCIABUSCAMORADOR.


**Figura 24 - Busca de registros.**

As figuras: Figura 25, Figura 26, Figura 27, contêm o diagrama de classes do segundo conjunto de objetos utilizados pelo subsistema de busca e edição de registros. Estas classes são responsáveis por realizar a filtragem de registros assim como apresentar estes registros para que o usuário o selecione o que levará a sua apresentação nas telas de edição.

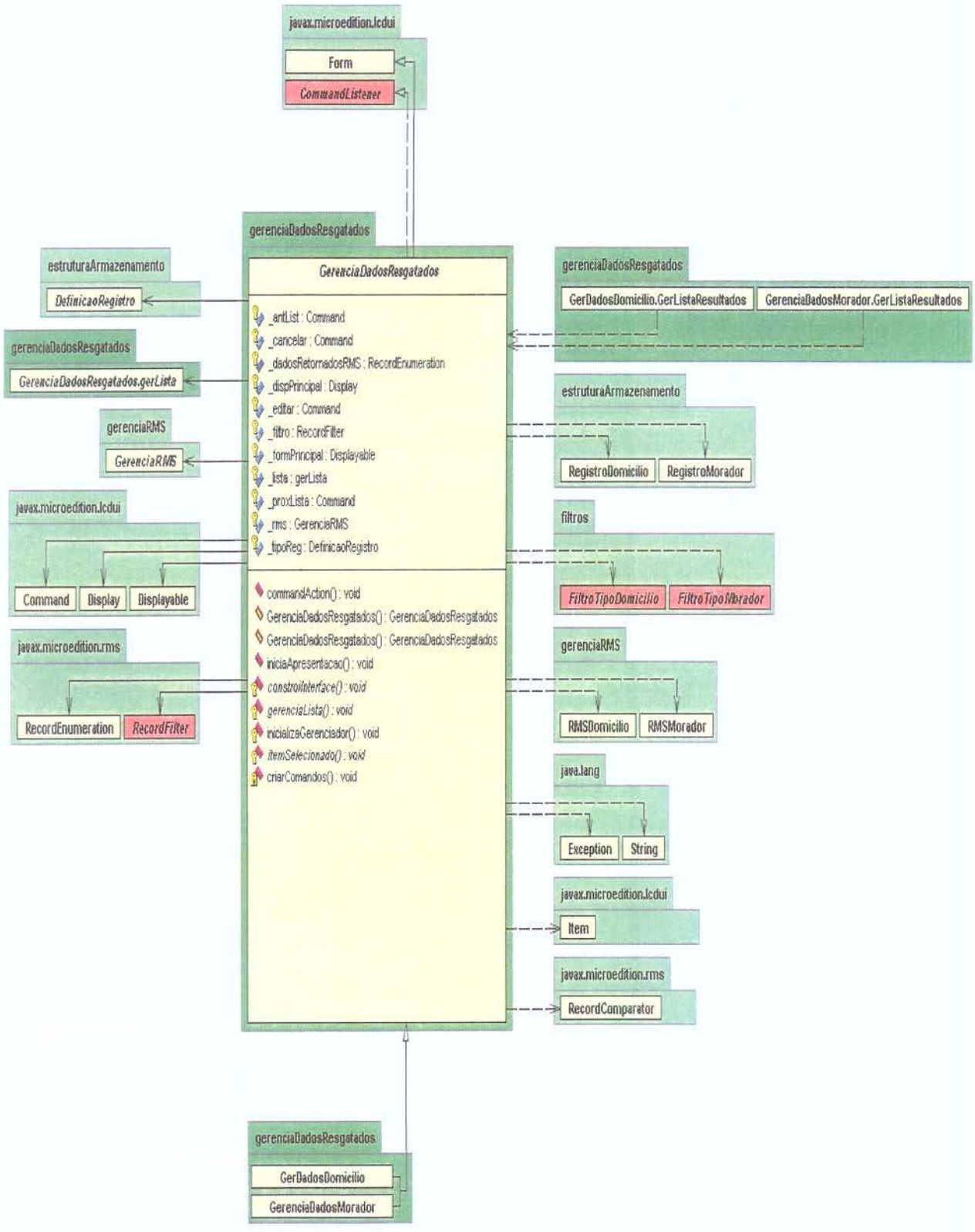


**Figura 25 - Superclasse GerenciaDadosResgatados**

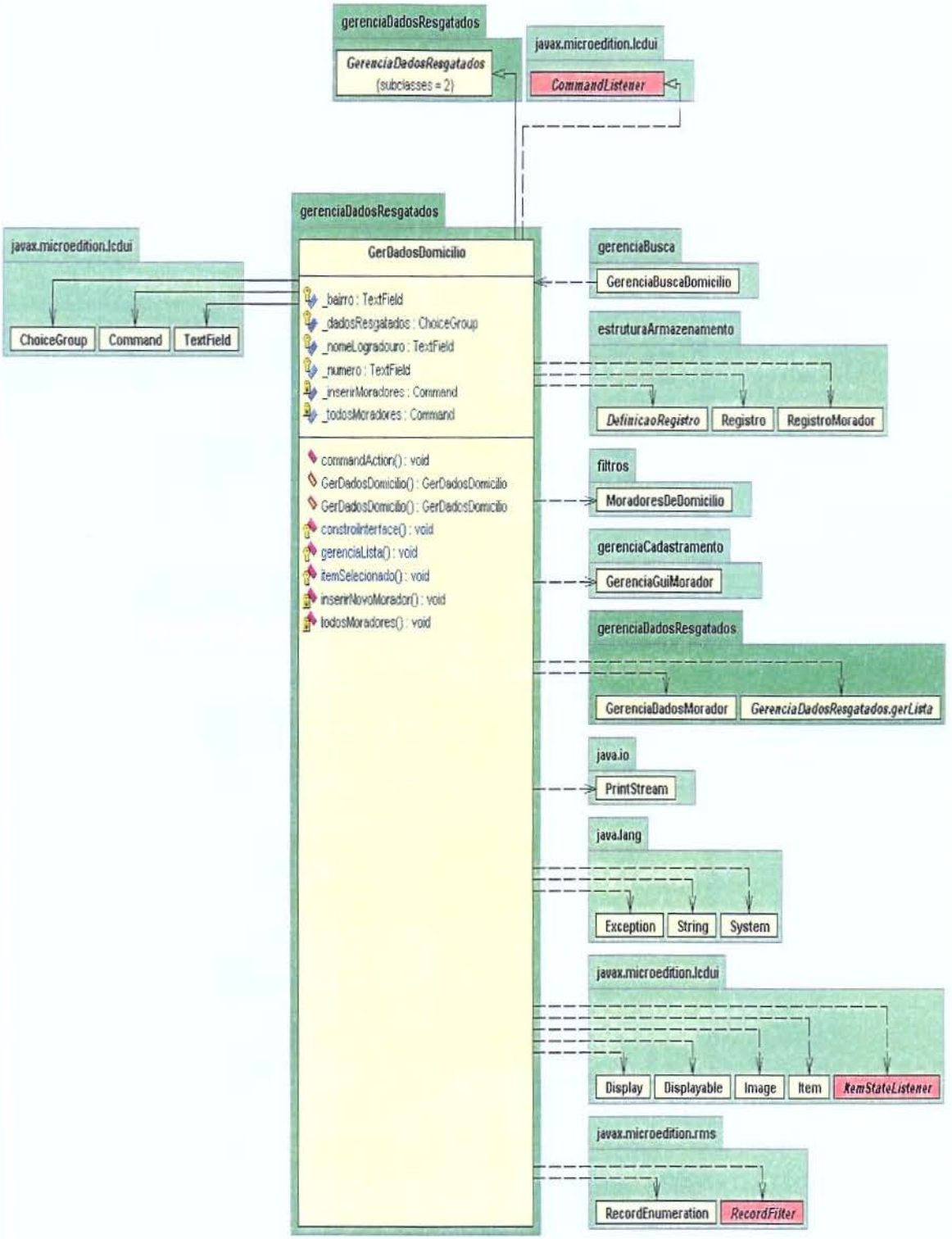


**Figura 26 - Classe GerDadosDomicilio**

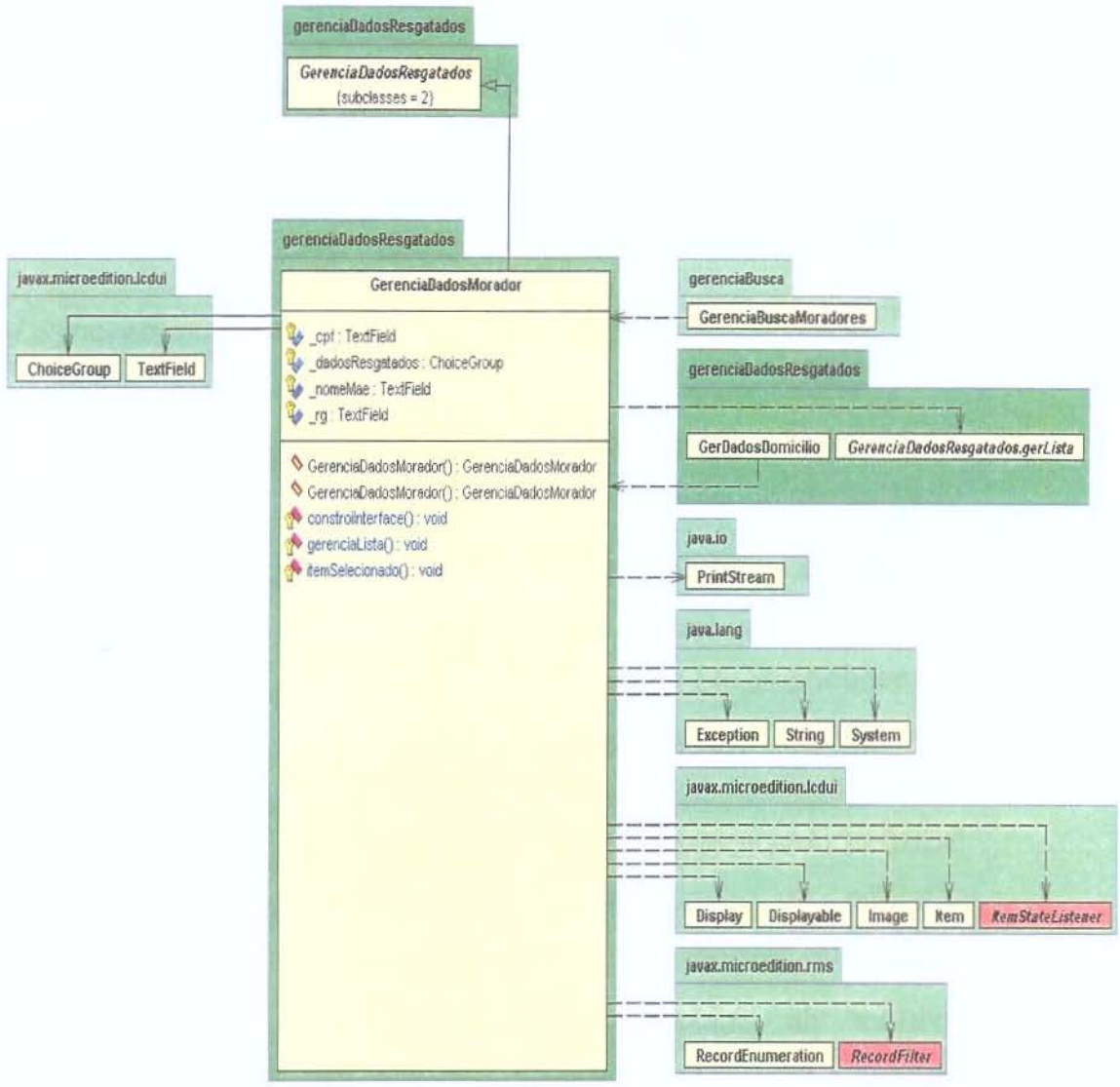


**Figura 27 - Classe GerenciaDadosMorador**

A superclasse GERENCIADADOSRESGATADOS, Figura 25 página 57, contém como atributos principais o filtro a ser utilizado para selecionar dados, um objeto RECORDENUMERATION presente na especificação J2ME para abrigar o resultado de uma filtragem e a definição do registro que está sendo utilizado, isto é, a definição ou de um domicilio ou de um morador. Além destes atributos está presente também um objeto da classe GERENCIARMS que será responsável por interagir com a base de dados local, coletando dela os registros que estão de acordo com os dados presentes no filtro utilizado.

Uma classe abstrata interna, chamada GERLISTA, está definida, sendo que a sua função é gerenciar a lista de resultados fornecidos por uma busca. Ela pré-

define que o conjunto de registros apresentados no formato de lista será de 3 elementos tendo em vista o tamanho limitado do display, podendo este número ser configurado para apresentar mais elementos tendo em vista a sua utilização em dispositivos que contem maior área de apresentação visual de dados. Esta classe gerencia também:

- Movimentação dos registros obtidos no resultado dentro da janela de 3 possíveis.
- Apresentação de detalhes do registro selecionado para melhor visualização do mesmo
- Retorno de um objeto do tipo registro contendo os dados do item selecionado na tela e dispara o gerenciador de cadastramento correspondente ao tipo de dados sendo apresentado no momento. Isto significa que o objeto do tipo registro criado a partir das informações selecionados na lista serão enviados para a classe GERENCIAGUIDOMICILIO, se eles representarem cadastro de domicilio ou para GERENCIAGUIMORADOR, caso contenha cadastro de pessoas.

No construtor da superclasse o método **inicializaGerenciador** é responsável por dado o filtro passado, descobrir se o mesmo está filtrando dados de um domicílio ou dados de um morador através da verificação de qual interface o filtro implementou. Se foi implementada a interface **FiltroTipoMorador** então a classe estará gerenciando dados de um morador, logo um objeto que gerencia o RMS de domicílio deverá ser criado e atribuído para a variável **_rms.** A filtragem será realizada e por último, para o atributo tipo de registro, será atribuído um objeto REGISTROMORADOR. Um procedimento semelhante será realizado caso seja verificado um filtro do tipo domicilio.

As classes GERENCIADADOSMORADOR e GERDADOSDOMICILIO são descendentes da classe GERENCIADADOSRESGATADOS. Elas têm como objetivo personalizar as características de construir a sua interface gráfica para a apresentação dos dados filtrados da base, gerar a lista inicial de registros

coletados assim como personalizar a classe interna de gerenciamento da lista de resultado.

A classe interna de gerenciamento de resultados irá realizar a apresentação da chave principal de identificação de cadastro na lista e retornar os detalhes do registro selecionado para que o usuário possa ter uma maior segurança quanto a escolha de um determinado registro para edição como apresentado na Figura 28. Por fim, também é responsável por realizar a chamada do objeto de gerenciamento de dados cadastrados para que sejam apresentados os dados presentes no registro.

A última imagem presente na Figura 28 exibe o menu da operações possíveis após ter sido realizada a localização de registros de domicílios. Nele é possível constatar que além da navegação na lista de resultados apresentados e a edição de um dos itens da lista, é possível a inserção de um morador assim como a visualização de todos os moradores que estão presentes em um determinado domicilio. A Figura 20, página 51 apresentou a passagem de um filtro da tela de gerenciamento de dados de domicílio para a tela de gerenciamento de dados de moradores. Desta forma, a tela de gerenciamento de dados de domicílios atua também como uma tela de gerenciamento de dados de busca, uma vez que ela permite a visualização de todos os moradores de um domicilio previamente cadastrado.

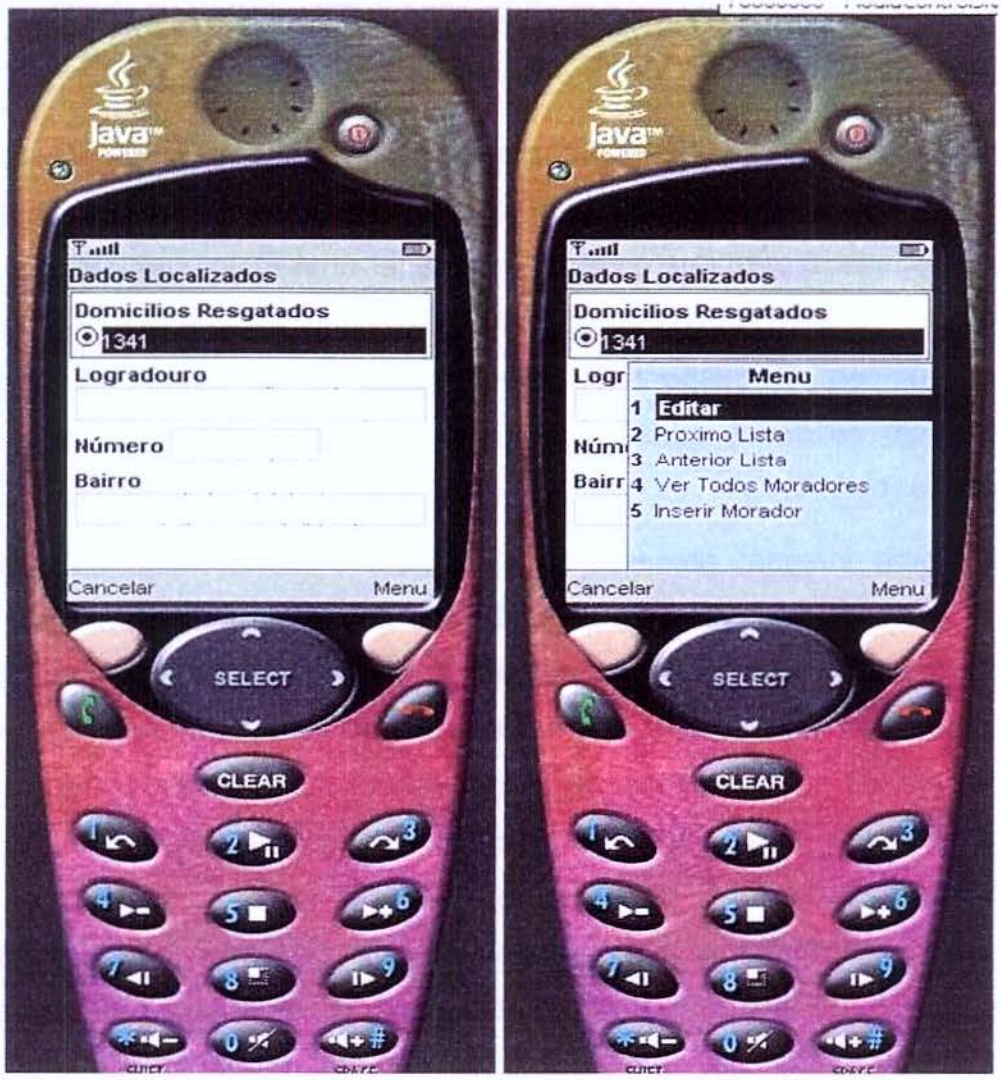


**Figura 28 - Gerenciamento de dados de domicílios resgatados.**

Todos os moradores a serem inseridos no sistema devem estar vinculados a um domicilio. Desta forma ao ser solicitada a inserção de um morador no sistema a classe de gerenciamento de busca de domicílios será chamada para que seja feita a escolha do domicilio de destino do morador. Por se tratar da chamada da classe de gerenciamento de busca de um domicílio, as funcionalidades oferecidas por ela ficam completamente disponíveis como apresentado na Figura 28, página 62. A Figura 29, página 63 exemplifica um morador cadastrado na base de dados. Nela pode ser notada a diferença na personalização feita nas superclasses de gerenciamento de dados resgatados de um morador em relação à de um domicílio.

As funcionalidades disponibilizadas apresentadas na última imagem da Figura 29, página 63, mostra que são possíveis a navegação pelos registros selecionados na filtragem e que estão disponíveis na lista de resultados, assim como a edição do registro selecionado.

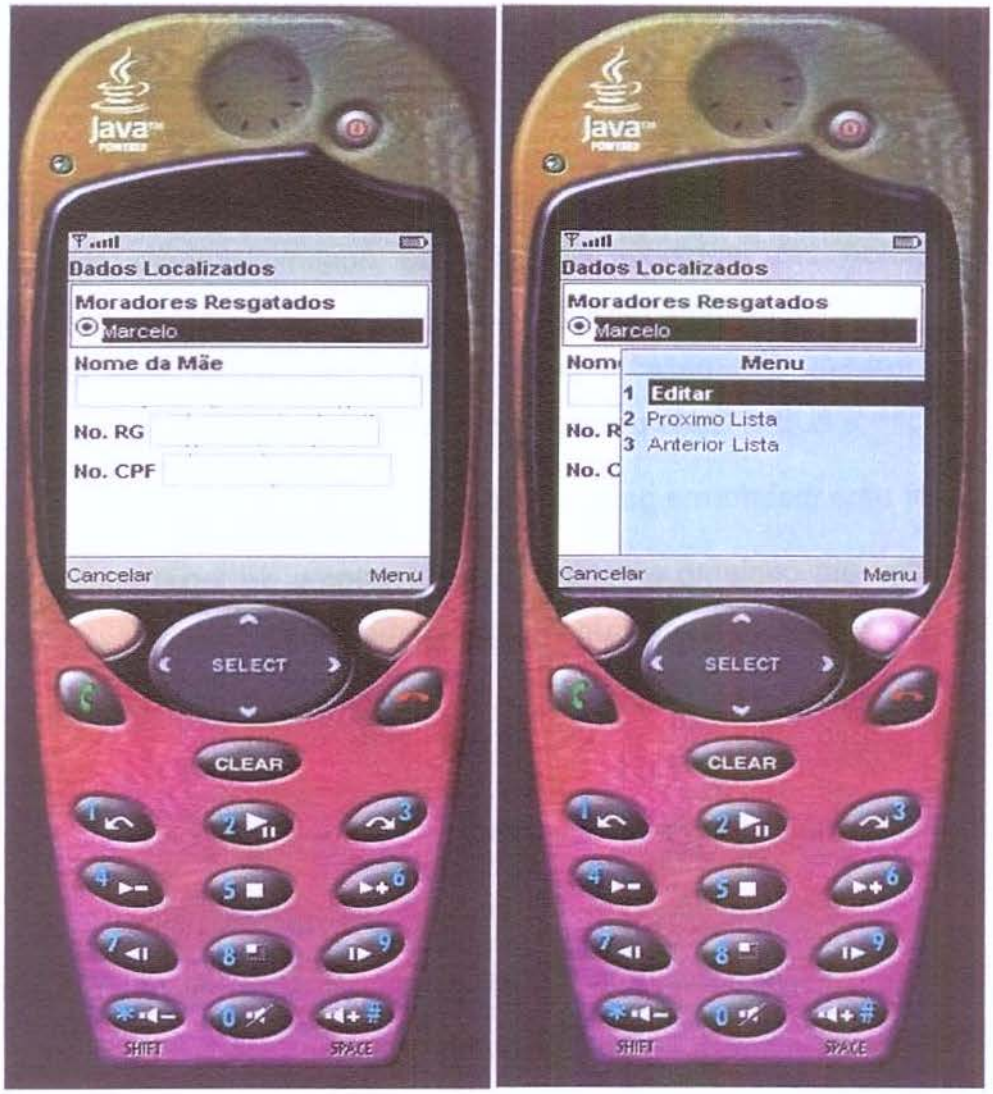


**Figura 29 - Busca de um morador.**

## 3.6 Tecnologias utilizadas

Duas foram as tecnologias principais utilizadas para a criação do cliente J2ME para o sistema móvel de cadastramento de moradores e residências de um município: Eclipse[Eclipse] como o ambiente integrado de desenvolvimento e o *plug-in* EclipseME[EclipseME].

### *3.6.1 Ambiente Eclipse*

O ambiente eclipse é uma ferramenta de desenvolvimento que visa ser universal, sendo um ambiente integrado de desenvolvimento aberto que é extensível. Ele foi projetado para ser possível a sua utilização na criação de aplicações diversas que passam por sites *Web*, aplicativos Java embarcados, programas C++ e EJB's. Suas principais metas de projeto foram:

- Prover uma plataforma para o desenvolvimento de ferramentas
- Suportar um conjunto irrestrito de fornecedores de ferramentas, incluindo desenvolvedores de software independentes.
- Suportar a manipulação de tipos de dados arbitrários, tais como HTML, Java, C, XML, entre outros.
- Facilitar a integração de ferramentas.
- Rodar em uma grande variedade de sistemas operacionais.
- Suportar o desenvolvimento de aplicações baseadas ou não em interfaces gráficas.

A Figura 30 apresenta a arquitetura da plataforma Eclipse. Nela é possível notar que existe um núcleo do sistema (*platform RunTime*) que fornece a base para a plataforma como um todo. Todas as outras funcionalidades são fornecidas através de pequenas unidades funcionais conhecidas como *plug-ins*. Geralmente uma ferramenta pequena é escrita como um único *plug-in*, já aplicações mais complexas são escritas na forma de vários *plug-ins*. Tirando o núcleo do sistema, todas as funcionalidades disponibilizadas são *plug-ins*. Outros componentes presentes na plataforma Eclipse e que estão presentes na Figura 30 são:

- *WorkSpaces*: Consiste de um ou mais projetos, no qual cada projeto é mapeado para um diretório especificado por um usuário.
- SWT: Conjunto de pequenos programas que definem a função dos símbolos gráficos e biblioteca gráfica integrada com o sistema de janelas nativo, porém com uma API independente de sistema operacional.
- JFace: Um conjunto de ferramentas gráficas implementadas com SWT que simplifica tarefas comuns de programação de interfaces com o usuário do sistema.
- *WorkBench*: Ao contrário do SWT e JFace, que são kits de ferramentas para o desenvolvimento de interfaces gráficas, fornece a interface gráfica da plataforma eclipse e fornece a estrutura na qual ferramentas interagem com um usuário.
- *Team*: A plataforma eclipse permite que um projeto pertencente a um *WorkSpace* seja colocado em um sistema de controle de versões e controle de configurações tendo um repositório associado.
- *Help*: Permite que ferramentas definam e criem documentação para um ou mais livros *online* que serão consultados posteriormente.

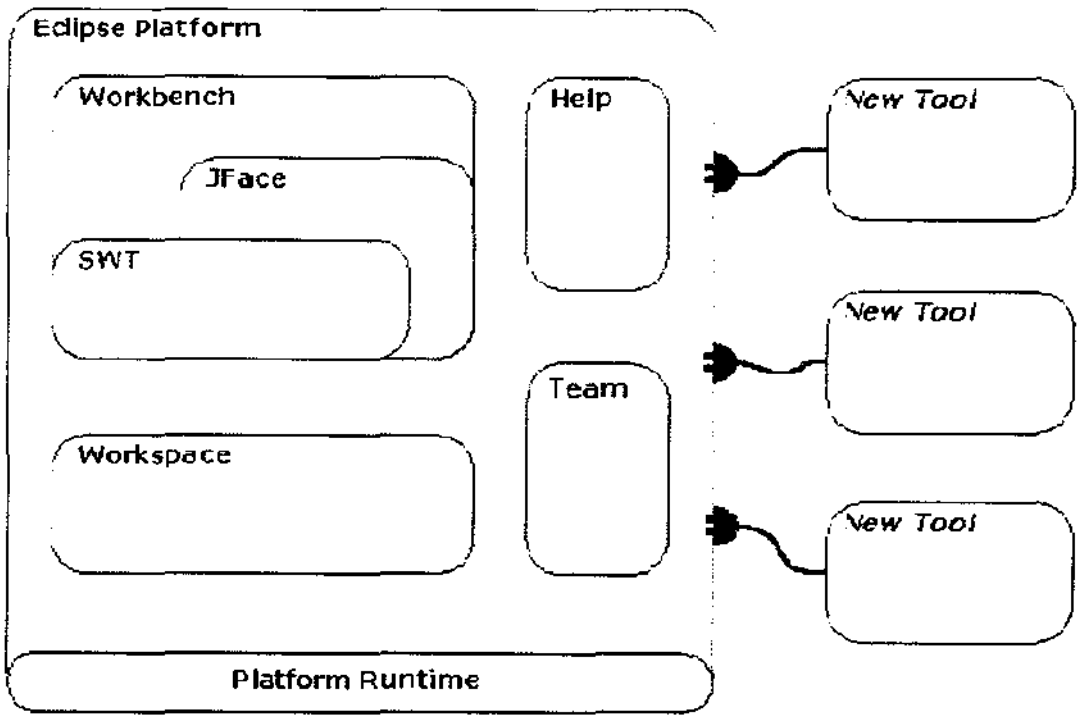


**Figura 30 - Arquitetura da plataforma Eclipse[Eclipse].**

Este ambiente integrado de desenvolvimento foi utilizado para o desenvolvimento do sistema de coleta e transmissão de dados como um todo, isto é, tanto a parte cliente quanto a parte servidora do sistema. Cada parte do sistema irá utilizar um *plug-in* específico para dar suporte a tecnologia utilizada.

### *3.6.2 Plug-in EclipseME*

Este *plug-in* foi concebido para auxiliar no desenvolvimento de aplicativos J2ME. Através dele é feita a integração entre os kits *Wireless* fornecidos pela Sun, para dar suporte a criação de programas que visam as características da plataforma J2ME, e o ambiente integrado Eclipse.

Um kit *Wireless* fornece o conjunto de API's necessárias para que um programa atue na plataforma J2ME. Estas API's se referem principalmente a configuração CLDC e ao perfil MIDP.

Este *plug-in* foi empregado para a criação do aplicativo cliente do sistema de coleta. Através dele, MIDLETS foram criados para permitir a coleta de dados

junto ao usuário. Outra funcionalidade utilizada foi a geração de empacotamento de versão assim como ambiente de lançamento e teste do aplicativo criado.

# 4 Sistema de transmissão

## *4.1 Web Services*

Como citado na seção Visão geral de *Web Services* (WS), o protocolo SOAP se tornou a base para a revolução proporcionada pelos WS. Este protocolo de comunicação se tornou o padrão de fato para a criação e invocação de aplicações expostas pela rede. A representação dos dados é feita de forma textual através da utilização da linguagem XML.

SOAP pode utilizar, como seu transportador, uma grande variedade de protocolos padrão Internet, tais como HTTP e SMTP, e fornece convenções para representar modelos de comunicação como chamada remota de procedimentos e mensagens baseadas em documentos.

### *4.1.1 SOAP*

Segundo[Soap] SOAP versão 1.2 é um protocolo leve que tem como objetivo permitir a troca de dados estruturados em um ambiente distribuído e descentralizado. Ele utiliza XML para definir um quadro de trabalho de mensagens extensíveis fornecendo uma mensagem que pode ser trocada sobre uma grande variedade de protocolos. O quadro de trabalho foi projetado independente de qualquer modelo de programação e outras semânticas específicas de implementação.

A especificação define uma mensagem SOAP em quatro partes:

- Envelope SOAP: Representa o item de informação mais externo de uma mensagem.
- Cabeçalho SOAP: Conjunto de zero ou mais blocos de cabeçalho, sendo que cada um pode focar um destino da mensagem SOAP.
- Blocos de Cabeçalho SOAP: Elemento de informação utilizado para delimitar dados que logicamente constituem uma única unidade computacional dentro de um cabeçalho SOAP.

- Corpo SOAP: Conjunto de zero ou mais informações destinadas a um receptor SOAP.
- Falta SOAP: Elementos de informação responsáveis por gerar dados de erros ocorridos em um nó SOAP.

A Figura 31 apresenta a estrutura de uma mensagem SOAP.

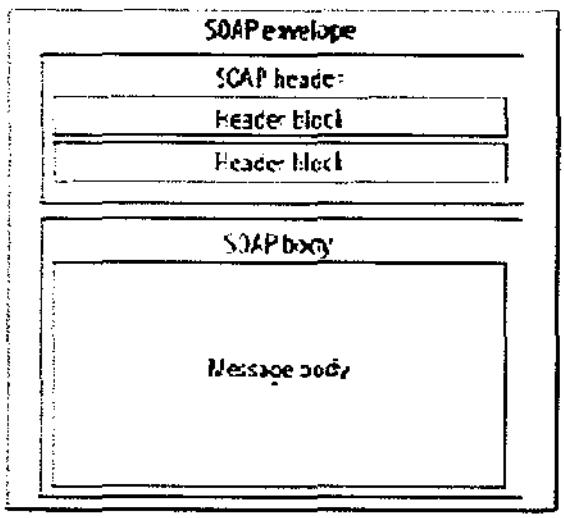


**Figura 31 - Estrutura de uma mensagem SOAP [Snell].**

### *4.1.2 Envelope SOAP*

Um envelope SOAP representa o container primário da estrutura de uma mensagem SOAP. A sua presença é obrigatória, sendo o mesmo representado como o elemento raiz da mensagem.

Todo envelope deve conter exatamente um elemento corpo. Se um envelope contiver um cabeçalho, ele deve conter não mais do que um, e deve apresentar como primeiro filho do elemento envelope antes do elemento corpo como apresentado na Figura 31.

O conteúdo do elemento corpo representa a mensagem em si. Ele é formado por conjuntos de elementos XML bem formados que representam informações sendo trocadas.

Cada elemento presente no cabeçalho da mensagem representa um bloco de cabeçalho. O objetivo destes blocos é comunicar informações contextuais relevantes para o processamento da mensagem SOAP.

A Figura 32 apresenta um exemplo de mensagem SOAP. Nela pode ser observada a presença dos elementos envelope, cabeçalho e corpo.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               soap:encodingStyle="http://schemas.xmlsoap.org/soap/
                                   encoding/">
  <soap:Header>
     <h:from xmlns:h="http://www.wrox.com/Header">SoapGuy@wrox.com</h:from>
  </soap:Header>
  <soap:Body>
     <w:GetSecretIdentity xmlns:w="http://www.wrox.com/heroes/">
        <w:codename>XSLT-Man</w:codename>
     </w:GetSecretIdentity>
  </soap:Body>
</soap:Envelope>
```

**Figura 32 - Exemplo de mensagem SOAP.**

### *4.1.3 Comunicação SOAP*

Para permitir a comunicação entre dois nós SOAP, este protocolo suporta os seguintes tipos de comunicação:

- SOAP RPC: Prevê uma comunicação síncrona, baseada em chamadas remotas de procedimentos, entre dois nós SOAP, no qual um nó envia e recebe mensagens utilizando métodos de solicitação e resposta assim como trocam valores de retorno e parâmetros.
- Mensagens SOAP: Define uma comunicação baseada em documentos na qual nós SOAP enviam e recebem documentos baseados em XML utilizando mensagens síncronas ou assíncronas.

### *4.1.4 Ligação SOAP com protocolos de comunicação*

A especificação SOAP não define nenhum protocolo de comunicação obrigatório para ser utilizado como meio de transporte. A ligação SOAP define os requisitos para o envio e recebimento de mensagens utilizando um protocolo entre os nós SOAP. Esta ligação especifica também regras de semântica e sintaxe para

o processamento de mensagens recebidas e a serem enviadas e um conjunto suportado de padrões de troca de mensagens.

Os protocolos HTTP e SMTP são os mais utilizados como camada de comunicação para o SOAP, porém outros protocolos podem também ser utilizados, entre eles estão o FTP, POP3, BEEP, JMS, *middlewares* customizados orientados a mensagens assim como protocolos proprietários que utilizam *sockets* TCP/IP.

### *4.1.5 Web Services Description Language*

Além do suporte para comunicação, fornecido pelo SOAP, existe a necessidade de fornecer uma maneira de descrever as interfaces disponibilizadas pelos serviços expostos de tal forma que possíveis usuários possam estudar esta interface e determinar se o serviço suporta ou não o comportamento necessário. A linguagem *Web Services Description Language* (WSDL) fornece esta descrição, sendo que ela define quatro partes importantes de informações sobre um serviço:

- Informações descrevendo todas as funções públicas disponibilizadas.
- Informações sobre os tipos de dados para solicitações assim como retorno para estas funções.
- Informações sobre ligação com um protocolo a ser utilizado para a invocação das funcionalidades disponibilizadas por um serviço.
- Informações de endereço para a localização de um serviço específico.

WSDL é capaz de descrever WS que são implementados utilizando qualquer linguagem e implantadas em qualquer plataforma. Isto contribui para a interoperabilidade na arquitetura WS.

A definição de um documento WSDL consiste de sete estruturas:

- *Definitions*: Representa um conjunto de definições que especificam um nome para o WS e também declara os *namepaces* (conjunto de

tipos de elementos XML assim como nomes de atributos) utilizados pelo resto do documento. Esta estrutura aparece com a raiz do documento WSDL.

- *Types*: Define todos os tipos de dados que serão utilizados para descrever as mensagens que são trocadas entre um WS e o usuário do serviço.
- *Message*: Representa a definição lógica dos dados que serão transmitidos entre o WS e o seu usuário.
- *PortType*: Este elemento define as operações suportadas pelo serviço exposto, através da combinação de várias mensagens de solicitação e resposta definidas por elementos *Message*.
- *Binding*: Especifica o protocolo e formato de dado utilizado para representar as operações e mensagens definidas por um *portType*.
- *Port*: Informa um endereço para ligação com o WS.
- *Service*: Agrega um conjunto relacionado de elementos *port*, que especifica de forma única informações de ligação de um WS.

## 4.2 Subsistema de transmissão

O sistema de transmissão de registros de domicílios e moradores cadastrados no aparelho móvel é utilizado tendo em vista o envio de dados para o servidor de processamento do sistema empregando a tecnologia *Web Services* para dispositivos com poucos recursos de processamento. A API KSOAP (explicada na seção *wireless web service* – KSOAP) foi utilizada neste cenário, pois a mesma foi concebida tendo como alvos dispositivos que utilizam a máquina virtual Java KVM.

O objeto do tipo registro apresentado na seção que descreve o sistema de coleta de dados foi utilizado como base para a exportação de dados.

Uma vez selecionada a funcionalidade de transmissão de dados presente na tela principal do aplicativo móvel, como pode ser visto na Figura 6, página 29,

uma iteração será realizada na base de dados para que cada registro nela presente seja enviado para um objeto do tipo registro que o representará. O passo seguinte consiste na geração do objeto **TransmissãoKsoap**, tendo como base o objeto do tipo registro criado, que conterá o tipo do dado sendo transmitido, isto é, se os dados são de um domicilio de uma pessoa ou de fim de domicílio, além de um vetor contendo os dados a serem enviados.

Ao ser recebido no lado do servidor, o objeto **TransmissãoKsoap** é carregado para um objeto do tipo registro, permitindo assim que o mesmo seja processado pelos componentes presentes no lado servidor.

### *4.2.1 Gerenciamento da transmissão de registros*

O diagrama presente na Figura 33, apresenta a classe **GerenciaTransmissao** que é responsável por gerenciar o envio de todos os registros de domicílios e moradores presente na base de dados local em um determinado momento. Esta classe irá estabelecer contato via web services com o servidor para a transmissão de dados assim como abrir conexão com as bases de dados locais.

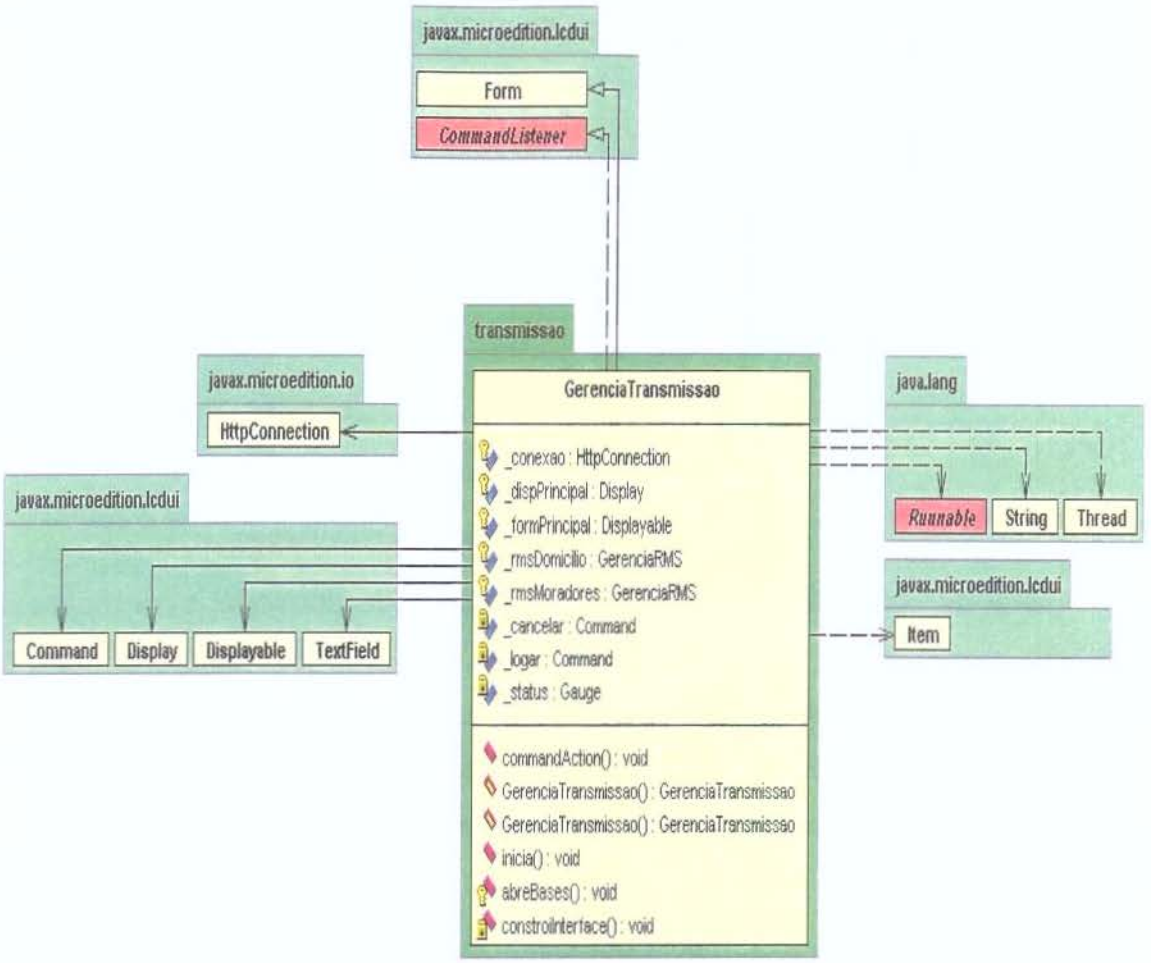


**Figura 33 - Classe GerenciaTransmissão**

O método **abreBases** irá realizar a abertura das bases RMS que serão gerenciadas via os atributos **_rmsDomicilio**, aponta para a base de registros de domicílios coletados, ou **_rmsMorador** que irá apontar para a base de registros de moradores. Esta ação é executada no construtor da classe de gerenciamento de transmissão de registros.

Para coordenar o envia dos registros locais, a classe interna **WSTransmissao**, apresentada na Figura 34, foi criada. Através dela, os apontadores para as bases RMS locais serão utilizados para uma iteração na qual cada domicílio e seus moradores sejam transmitidos. Esta classe interna possui o método **getTodosTransmissao** que irá buscar todos os registros de domicílio. Cada registro coletado será armazenado em um objeto do tipo registro, sendo que pela invocação do método **criaObjetoTransmissão** da classe registro,um objeto

**TransmissaoKsoap** será gerado contendo os dados de um domicílio, neste caso, a ser enviado pela rede.

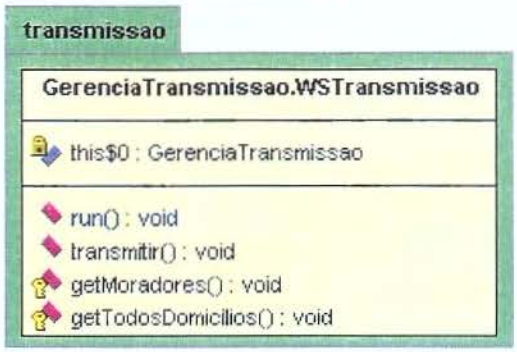


**Figura 34 - Classe Interna WSTransmissao**

Uma vez coletado e transmitido um registro de domicilio, será realizada a filtragem, presente no método **getMoradores**, da base de moradores para que sejam coletados todos os moradores existes no domicilio em questão. Cada registro coletado irá gerar um objeto do tipo registro sendo o mesmo empregado para a geração de um objeto **TransmissaoKsoap** a ser propagado pela rede na direção do servidor.

Um domicílio em conjunto com seus moradores forma o conceito de um lar, desta forma foi criada uma sinalização que indique para o servidor que está formado um lar e que o mesmo já pode processar todos os registros já enviados. Esta sinalização consiste no envio de um objeto **TransmissaoKsoap** contendo como nome da tabela enviada a palavra FIM.

O método **transmitir** da classe interna **WSTransmissao** é empregado para enviar o objeto **TransmissaoKsoap** obtido para o servidor utilizando a api KSOAP. Um objeto **HttpTransport** irá ser o transportador dos dados para o lado servidor.

### *4.2.2 Objeto de transmissão*

O sistema de transmissão utiliza o objeto **TransmissaoKsoap**, presente na Figura 35, para encapsular os dados dos registros a serem transmitidos via KSOAP. Este objeto irá conter todas as informações necessárias para que um registro possa ser reconstituído completamente no lado servidor.

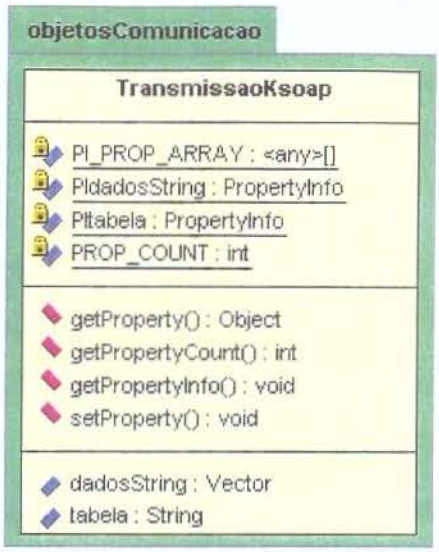


**Figura 35 - Classe TransmissãoKsoap**

Um objeto **TransmissaoKsoap** é formado por um vetor que irá conter todos os dados presentes em um objeto do tipo registro, sendo que a inserção de dados no mesmo irá obedecer a ordem de campos presente no objeto de definição de campos presente no objeto do tipo registro origem. Este mesmo objeto de definição será utilizado pelo objeto do tipo registro presente no lado do servidor para a remontagem do registro transmitido.

Outro atributo presente no objeto de transmissão é o nome da tabela cujos dados estão sendo enviados, isto é, um atributo que informa se os dados sendo transmitidos formam um domicílio, um morador ou ainda se o objeto representa o fim de um lar. Os possíveis valores para este atributo são: MORADOR para um registro de pessoa residente, DOMICÍLIO para os dados de uma residência e FIM para informar ao servidor que um lar está formado e que o mesmo deve ser processado.

Além destes atributos está presente a definição de propriedades que representam o vetor de dados e o nome da tabela. Estas propriedades são utilizadas pela API KSOAP para gerenciar a transmissão de dados via web services. Cada atributo a ser transmitido deverá conter uma propriedade que o represente. Para a String tabela a ser transmitida foi criada a propriedade **PItabela** e para o vetor a ser enviado a propriedade **PIdadosString** como pode ser visto na

Figura 35, página 77. Além dessas propriedades fundamentais foram criados um vetor contendo o nome de cada propriedade gerada assim como a variável **Prop_Count** que informa o total de propriedades presentes no objeto de transmissão.

Métodos para a atribuição e coleta de valores para os atributos vetor de dados e nome da tabela estão definidos e permitem que o objeto do tipo registro possa gerenciar o envio e recuperação de dados de e para o objeto de transmissão empregado.

Outro ponto importante do objeto **TransmissaoKsoap** é o fato dele implementar a interface **KVMSerializable**, o que o torna apto a ser serializado, via KSOAP, isto é, os seus atributos serão coletados e enviados via rede para outro local onde serão atribuídos a um objeto **TransmissaoKsoap** lá criado.

### *4.2.3 Recriando o objeto Registro no lado servidor*

Uma vez criado o objeto de transmissão no lado do cliente móvel, o mesmo será enviado para o servidor via *Web Services* e deverá ser remontado.

A utilização da definição de campos, descrita na seção que apresenta o sistema de coleta de dados, é empregada para permitir que os valores dos campos presentes em um objeto do tipo registro sejam mapeados para um objeto **TransmissaoKsoap** e vice-versa. Um procedimento semelhante ocorre no momento da gravação dos dados na base RMS. O atributo **_defRegistro** que é um objeto do tipo **DefinicaoRegistro**, contém a definição da estrutura do registro sendo encapsulado.

Os métodos **criaObjetoTransmissao** e **carregaObjetoTransmissao** presentes no objeto do tipo registro, são responsáveis por realizar o mapeamento dos dados do objeto do tipo registro para um objeto **TransmissaoKsoap** e o processo inverso respectivamente.

## 4.3 Wireless Web Service API – KSOAP

O pacote KSOAP [Ksoap, Ksoap2] foi projetado tendo em vista a utilização do protocolo SOAP por aplicativos que tem como alvo a plataforma J2ME para dispositivos com baixo poder de processamento como celulares e PDA's. Ele é magro, fácil de utilizar e bem documentado[Mchugh].

A classe **SoapObject** permite que aplicativos construam chamadas a procedimentos através do protocolo para *Web Services* chamado SOAP. A utilização dos métodos **getProperty** e **setProperty** permitem a criação e obtenção de resultados destas chamadas.

A interface **KVMSerializable** é disponibilizada para que desenvolvedores possam continuar utilizando seus próprios objetos que armazenam dados. Utilizar esta interface se faz importante dentro de um ambiente de negócio de larga escala onde o modelo de dados precisa ser numeroso e definido previamente.

Este pacote é disponibilizado na forma de um arquivo comprimido disponibilizado pela plataforma Java conhecido como JAR. Para que a comunicação entre a parte cliente e servidora do sistema ocorra, este pacote deve estar presente nos dois lados.

Este pacote é empregado no sistema cliente móvel, para que o mesmo possa consumir serviços publicados, via *Web Services*, por um servidor.

# 5 Sistema Servidor

## 5.1 Plataforma J2EE

Como foi explicada na seção Visão Geral da Plataforma Java, a plataforma J2EE é formada por API's que permitem a criação de aplicativos distribuídos multicamadas. Nesta seção será enfocada a camada de negócio tendo em vista que os componentes presentes na mesma foram utilizados para criar o lado servidor do aplicativo de coleta móvel de dados.

A camada de negócios de uma aplicação J2EE é formada por *Enterprise Java Beans* (EJB). EJB é uma arquitetura de componentes do lado servidor que simplifica o processo de criação de classes de empreendimento de aplicações formadas por componentes distribuídos[Roman]. É possível desenvolver componentes no lado servidor rápido e facilmente em Java através da utilização de uma infra-estrutura distribuída prescrita fornecida pela indústria conhecida como servidor de aplicações ou Container. Através deste container são fornecidos serviços comuns tais como rede, escalonamento de recursos, serviço de nomes, controle de transação, ciclo de vida do componente, segurança entre outros.

Para padronizar a interação entre o servidor de aplicações e os componentes, existe um acordo, na forma de um conjunto de interfaces, entre o Container e o componente. Este acordo, conhecido como EJB, permite que qualquer componente seja executado em qualquer servidor de aplicações.

EJB's não são elementos de interface com o usuário, mas sim componentes que estão no servidor realizando todo o processamento pesado do sistema, podendo os mesmos ser responsáveis pela lógica de negócio, lógica de conexão com a base de dados utilizada no sistema ou ainda acesso a outros sistemas.

A tecnologia EJB está baseada em duas outras:

- *Java Remote Method Invocation* sobre o protocolo *Internet Inter-ORB* (RMI-IIOP): Protocolo utilizado para a realização da comunicação em rede dos

componentes do sistema. RMI-IIOP permite que componentes escritos em Java realizem a comunicação. Através deste protocolo, componentes construídos no sistema CORBA poderão interagir com os componentes EJB.

- *Java Naming and Directory Interface*(JNDI): API responsável por fornecer uma interface padrão para localizar usuários, máquinas, redes, objetos e serviços. Pode ser utilizada, por exemplo, para localizar uma impressora na rede. JNDI é uma maneira padrão de buscar coisas na rede. Em J2EE, JNDI é utilizado para obter referências para a API de transação Java (JTA), conectar o sistema a fábricas de recursos tais como JDBC(*Java DataBase Connectivity*) e JMS (*Java Message System*) e para que EJB busque outros EJB's.

O componente e sua interface devem estar de acordo com as especificações. A mesma manda que alguns métodos obrigatórios sejam expostos permitindo que o Container gerencie o *bean* uniformemente independente do Container que esteja sendo utilizado.

### *5.1.1 Tipos de EJB's*

Na especificação EJB 2.0 [DEMICHIEL] estão definidos três tipos de EJB's:

- *Beans* de sessão (*Session Beans* - SB): Modelam processos de negócio. Representam ações no sistema.
- *Beans* de Entidade (*Entity Beans* - EB): Modelam os dados de negócio. Representam objetos de dados. Fazem *cache* de informações da base.
- *Beans* guiados por mensagens (*Message Driven Beans* - MDB): São ações assim como os *beans* de sessão. A diferença é que somente é possível invocar este tipo de *bean* através do envio de mensagens.

SB são componentes de vida curta sendo que a sua duração depende de uma sessão do cliente. O Container é responsável por gerenciar o ciclo de vida do

SB. Outra característica deste tipo de bean é a sua natureza não persistente, isto é, ele é destruído juntamente como elementos presentes na memória assim que uma sessão com um cliente termina.

Existem dois tipos de SB, sendo que cada um é apropriado para um tipo de interação como o usuário: *Stateless Session Bean*(SlessSB) e *Statefull Session Bean*(SfullSB).

- SfullSB mantem o estado produzido pelas requisições de um cliente . Se o estado de um SfullSB é modificado durante uma solicitação, esta modificação estará visível para o mesmo cliente na sua próxima solicitação.
- SlessSB são utilizados para negócios que não guardam estado entre chamadas. A operação é realizada em uma única solicitação. Após cada chamada o Container decide se destrói ou recria o *Bean*. Para que este tipo de *Bean* seja útil, o cliente deve passar todos os dados que serão necessários como parâmetro para o método, ou o *Bean* pode obter os dados que precisa da base de dados. Todas as instancias de um SlessSB são equivalentes para um cliente uma vez que eles não mantêm o estado de uma conversação.

*Beans* de entidade são objetos persistentes que são armazenados em um sistema de armazenamento definitivo, como por exemplo, uma base de dados. Dois são os tipos de *Beans* de entidade: *Bean Managed Persistence* (BMP) e *Container Managed Persistence* (CMP):

- BMP: Este tipo de componente prevê que o desenvolvedor forneça todo e qualquer código de acesso a base de dados. O programador fica responsável por fornecer uma implementação para que uma instância de um BMP seja mapeada de e para o sistema de armazenamento permanente. Este mapeamento geralmente ocorre através da utilização de API de base de dados tais como JDBC ou SQL/J.

- CMP: Neste tipo de componente, o Container ficará responsável por manipular todos os acessos requisitados pelo *Bean*. A codificação deste componente não inclui nenhuma chamada de acesso a base de dados do sistema. Como resultado o código deste *Bean* não está amarrado a nenhum mecanismo de persistência de dados. Por causa desta flexibilidade, a implantação deste elemento em outro servidor de aplicações não exigirá modificações ou re-compilação. Para que as chamadas de acesso a base de dados sejam feitas, existe a necessidade de que seja informado para o Container quais são os relacionamentos e os campos presentes na base de dados e que serão gerenciados pelo CMP criado.

O último tipo de *Bean* especificado em [Demichiel] é o *Bean* guiado por mensagens. Eles são similares aos SB no que ser refere ao fato de ambos representarem ações a serem executadas. O que os diferencia é a forma de ativação de cada um. Enquanto que em SB a invocação se dá pela chamada direta de um método por parte de um cliente, em um *Bean* guiado por mensagem o envio de uma mensagem irá gerar a ativação do *Bean*. Estes componentes podem realizar chamadas a outros *Beans* uma vez ativados.

Tendo este cenário, a Figura 36 apresenta a interação possível entre clientes e a camada de negócios representada pelos componentes EJB. Nela pode ser notado que a camada de apresentação de dados interage diretamente com os *Beans* que representam as ações do sistema como os *Beans* de sessão e os *Beans* guiados por mensagem.

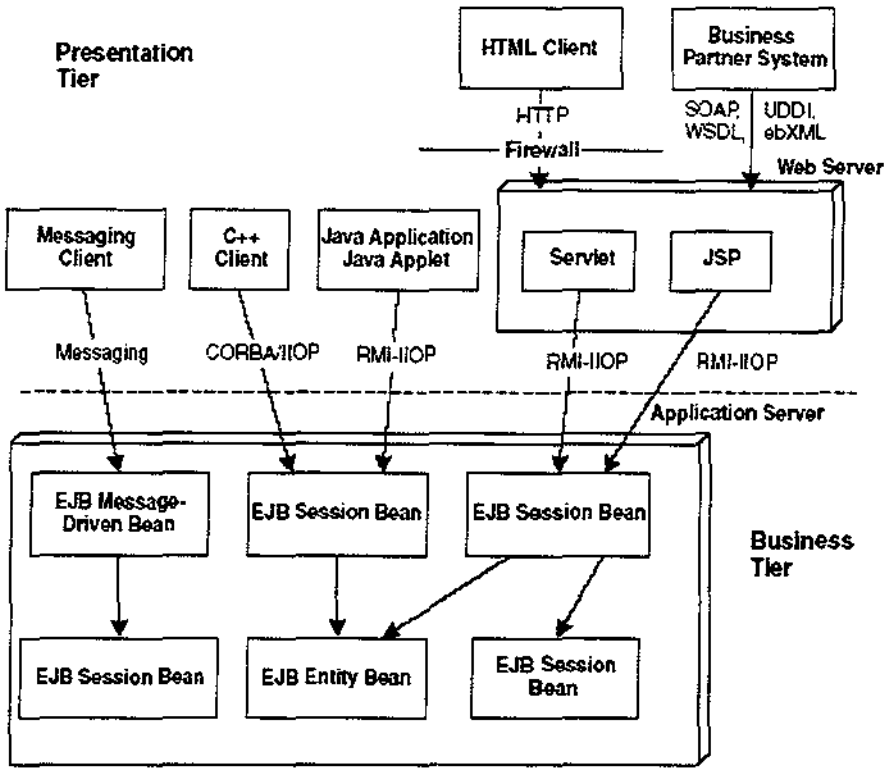


**Figura 36 - Clientes interagindo com a camada de EJB. [Roman]**

### *5.1.2 Interação com o Container EJB*

Solicitações não são encaminhadas diretamente para os *Beans*, mas sim são interceptadas pelo Container que fornece serviços e depois encaminha tal solicitação. Este conceito de interceptação evita que o usuário necessite produzir e manipular código de *middleware*.

O Container EJB fornece o gerenciamento de transações, segurança, gerenciamento de recursos e ciclo de vida de componentes, persistência entre outros. Desta forma o Container atua como um nível intermediário entre código cliente e código de componentes *Bean*. Este nível se apresenta como um único objeto em rede chamado Objeto EJB (OEJB). Uma interface fornecida pelo produtor do EJB é utilizada para que o OEJB conheça os métodos fornecidos pelo EJB, podendo, desta forma, o OEJB interceptar as chamadas dos clientes. Esta interface é chamada Interface remota (IR), sendo que a mesma segue regras previstas na especificação[Demichiel]. A Figura 37 apresenta o processo de chamada de métodos através do uso de um OEJB.

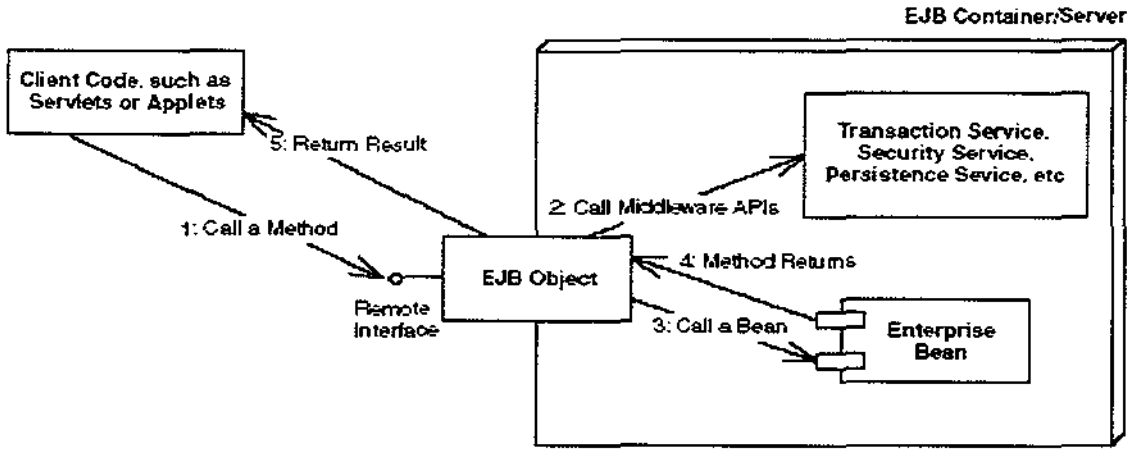


**Figura 37 - Interação de um cliente com o objeto EJB [Roman] .**

Para obter uma referência para um OEJB, um cliente deve perguntar a uma fábrica de OEJB. Esta fábrica instancia e destrói estes objetos. Na especificação, esta fábrica é chamada de Objeto *Home*(OH) que são proprietárias e específicas de cada container.

A especificação define uma interface chamada interface *Home* que permite ao desenvolvedor especificar parâmetros para a criação, destruição e localização de objetos EJB. A Figura 38 apresenta o processo de criação de um objeto EJB por parte do Container tendo o mesmo recebido uma solicitação de um cliente.

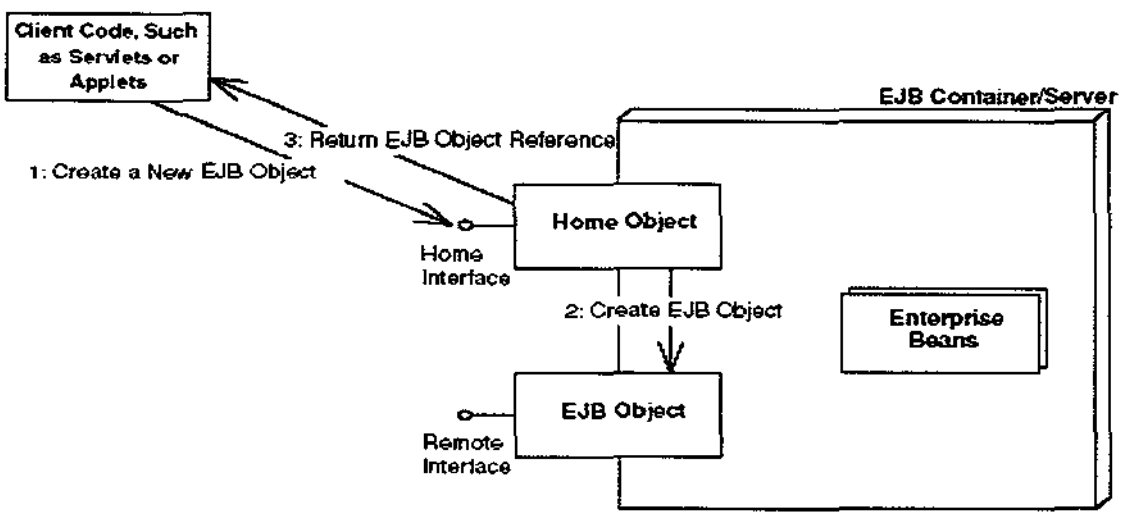


**Figura 38 - Interação com o objeto Home. [Roman]**

Desta forma ao ser implantado, um sistema que utiliza EJB deverá fornecer a interface remota e *Home* para os componentes expostos de tal forma que o container possa permitir que clientes acessem tal aplicação.

## 5.2 Sistema servidor

A parte servidora do projeto é responsável por disponibilizar um ponto de entrada via *Web Services* para que os dados de domicílios e moradores sejam recebidos dos dispositivos móveis. Uma vez recebidos, os registros serão validados segundo regras pregadas pelo ministério da saúde e armazenados na base de dados do servidor. Por fim um aplicativo será utilizado para concertar os registros inválidos assim como montar lotes de lares em arquivos texto para que os mesmos sejam enviados para o servidor federal, lá processado, e a partir daí coletados arquivos com o resultado deste processamento.

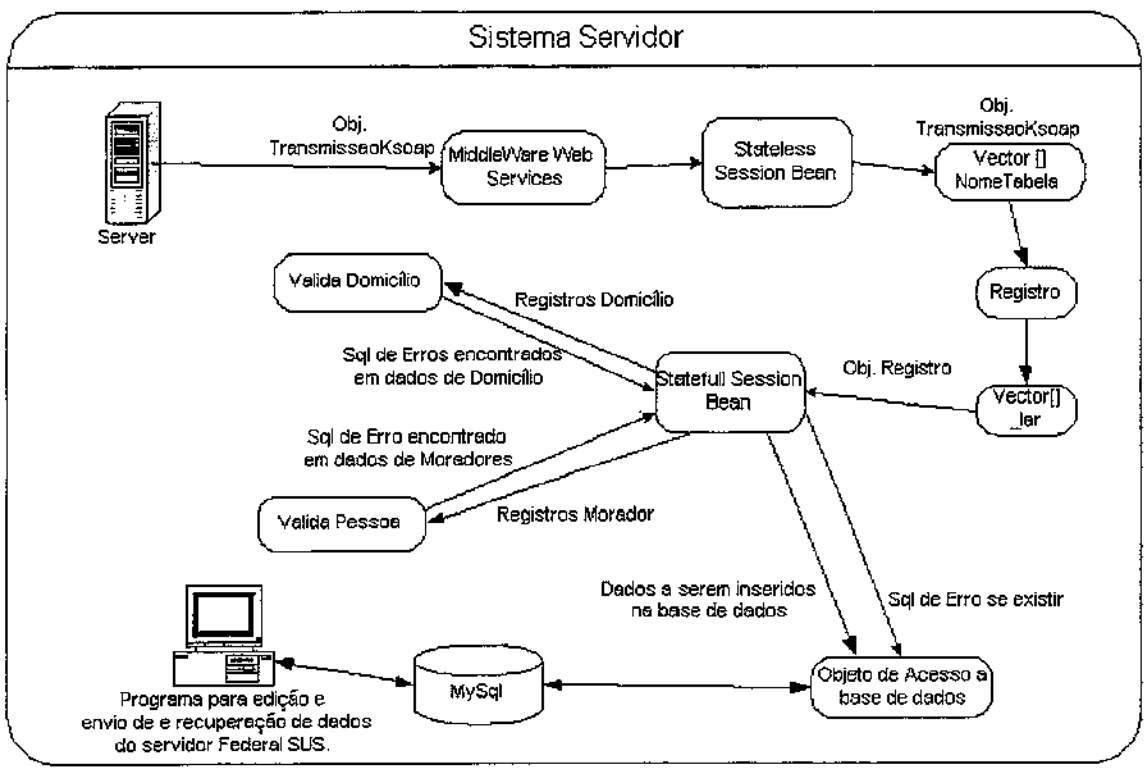


**Figura 39 - Sistema servidor.**

O módulo servidor é formado por dois EJB's principais como mostrado na Figura 39. Um stateless session bean atua como ponto de entrada coletando os objetos **TransmissaoKsoap** enviados pelo cliente. Um objeto do tipo registro é

montado a partir do objeto recebido pela rede e a partir dele ocorre a invocação de uma operação específica presente no EJB de processamento.

O componente EJB de processamento é do tipo statefull session bean, através dele os registros recebidos serão enviados para um objeto de validação de dados que segue as normas estabelecidas pelo ministério da saúde. Existem dois tipos de tais objetos, um para validar dados de domicílio e outro para validar dados de moradores. Caso seja constatada a presença de algum erro no registro recebido, será gerado um registro na base de dados informando quais campos não estão de acordo com o exigido.

Um objeto de comunicação com a base foi criado, seguindo o padrão de desenvolvimento conhecido como DAO (Data Access Object), para que toda interação com a base de dados seja realizada através deste componente do sistema, separando desta forma código de negócio de código de acesso a base de dados.

Por fim existe um aplicativo desenvolvido com a tecnologia java J2SE cuja função é abrir os cadastros enviados para o servidor, visualizar a presença de erros no mesmo, corrigir e realizar por fim, com os registros corretos segundo o padrão estabelecido, a formação de lotes de registros a serem transmitidos para o sistema centralizador presente no distrito federal.

### *5.2.1 Ponto de entrada do sistema servidor*

O ponto de entrada do sistema servidor é formado por um EJB do tipo stateless session bean chamado CadsusAccessBean como visto na Figura 39, página 87. Este tipo de bean foi escolhido tendo em vista que na versão 1.4 da especificação J2EE, somente este tipo de componente pode disponibilizar uma interface Web Services, contendo métodos a serem invocados remotamente. Somente um método está presente neste EJB de entrada do sistema, porém o mesmo irá disparar os componentes necessários do sistema para que o total processamento dos dados enviados seja realizado.

Para que o sistema de *Web Services* utilizado no lado servidor possa reconhecer os dados transmitidos, existe a necessidade de mapeamento dos mesmos junto a classe **ClassMap**. As classes **TransmissaoKsoap** e **vector** serão registradas através da utilização do método **addMapping** sendo para isto passada uma referência para a definição da classe. Este mapeamento é realizado no construtor do objeto EJB criado para atender as chamadas dos clientes. Alem deste mapeamento será realizada também a criação do objeto de processamento para atender a sessão do cliente que deseja transmitir os dados em um determinado momento.

Objetos do tipo **TransmissaoKsoap** são enviados pelos clientes, como foi explicado na seção sobre o sistema de transmissão, contendo um vetor com os dados assim como o tipo dos dados enviados, isto é, se os dados representam um morador ou um domicílio.

Tendo este objeto, será feita a chamada a um método do EJB de processamento correspondente ao tipo dos dados transmitidos, isto é, tomando o nome do tipo dos dados transmitidos, domicílio ou morador, será realizada a chamada de um método cujo objetivo será processar estes dados tendo em vista as regras propagadas pelo ministério da saúde. Objetos de validação de conteúdo serão invocados, assim como objetos de armazenamento de registros na base de dados local do servidor, para que os dados estejam disponibilizados para que as medidas seguintes sejam tomadas.

### *5.2.2 EJB processador de registros*

Um EJB statefull session bean foi criado tendo como objetivo coordenar todo e qualquer processamento necessário de um registro recebido a partir do cliente remoto. Existem dois conjuntos de objetos auxiliares: objetos de validação e objeto de comunicação com a base de dados local. Os objetos de validação verificam a presença de todos os dados pedidos pelo ministério da saúde gerando registros de erro na base de dados informando quais campos devem ser corrigidos. O objeto de comunicação com a base de dados concentra todo e

qualquer método de interação com a base de dados mantida para o sistema servidor.

Na Figura 39, página 87, pode ser verificado através do fluxo de informações, que o componente EJB de processamento representado pela classe **CadSusActionBean** recebe um objeto do tipo **transmissaoKsoap** vindo do objeto **CadSusAccessBean**. Este objeto ao ser recebido será utilizado para a geração de um objeto do tipo registro no lado do servidor. Esta classe registro irá utilizar o nome da tabela de dados informada no objeto de transmissão para remontar a estrutura de dados utilizada para a manipulação de dados de trabalho no sistema.

A Figura 40 apresenta o diagrama de classe do EJB de processamento de registros. Nesta figura estão presentes os atributos principais **_lar**, **_validaDomicilio** e **_validaMorador**.

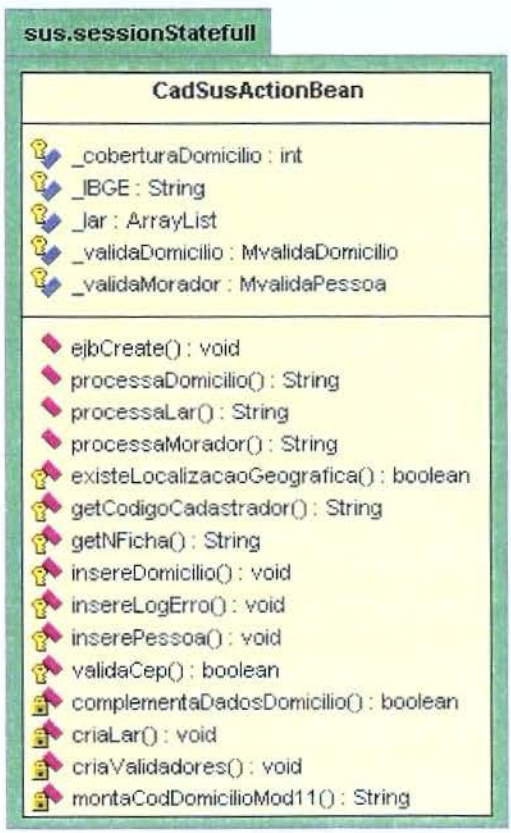


**Figura 40 - diagrama de classes do objeto CadSusActionBean.**

O atributo **_lar** representa um acumulador de objetos do tipo registros que em conjunto formam uma família transmitida pelo cliente. Um objeto **transmissaoKsoap** contendo como nome da tabela transmitida a palavra FIM irá significar que os dados de um domicílio assim como os de todos os seus moradores coletados já foram transmitidos sendo que na seqüência deve ser disparado o seu processamento.

Os atributos **_validaDomicilio** e **_validaMorador** representam os objetos cuja finalidade é validar os dados de domicílios e Moradores, respectivamente, recebidos pelo servidor. Registros que apresentarem erro terão o nome dos campos errados gravados na base de dados.

Os métodos **processaDomicilio** e **processaMorador** são os pontos de entrada do objeto de processamento, eles irão coordenar todas as ações necessárias para que os dados sejam armazenados na base e que os registros de erros sejam gerados adequadamente.

O processamento de domicílios, feito pelo método **processaDomicílio**, prevê a re-geração do objeto do tipo registro assim como a sua inserção na estrutura de armazenamento **_lar**.

Os moradores enviados são processados pelo método **processaMorador**, nele será verificado o tipo de dado enviado. Caso não seja constatado o fim de um lar, o objeto de dados enviado será utilizado para re-gerar um objeto do tipo registro e o mesmo será inserido na estrutura **_lar**. Uma vez recebido o sinal de fim de lar, representado pelo sinal FIM passado no atributo que representa o nome da tabela enviada presente no objeto **transmissaoKsoap**, o método **processaLar** será invocado levando ao processamento dos registros transmitidos.

### *5.2.3 Processamento de Lares*

O conjunto de um domicílio mais os seus moradores irá representar um lar formado. O processamento deste lar irá ocorrer através da verificação da pré-existência na base de dados do servidor da localização geográfica de um domicílio a ser inserido. Outro passo consiste na complementação dos dados transmitidos

caso os mesmo estejam ausentes. Estes dados representam informações geradas através do processamento do valor existente em outros campos. O campo código do domicílio, por exemplo, representa um dado deste tipo que será calculado no lado servidor principalmente para cliente com baixo poder de processamento como é o caso de dispositivos móveis.

O próximo passo para o processamento de um lar é a submissão dos registros coletados a seu respectivo objeto de validação de conteúdo. Para o caso de domicílios o objeto **ValidaDomicílio** será invocado, para um registro do tipo morador o objeto **ValidaMorador** será utilizado.

O passo final é representado pela interação do EJB de processamento de registros com o objeto de interação com a base de dados representado na Figura 39, página 87, pelo objeto chamado **AcessaBaseDAOImpl**. Esta interação prevê a geração tanto do registro com os dados enviados quanto um registro com os erros encontrados pelos objetos de validação de conteúdo.

### *5.2.4 Classe de validação de dados de Domicílios*

A classe **MValidaDomicilio,** apresentada na Figura 41, terá como entrada um objeto do tipo registro contendo todos os campos pedidos pelo ministério da saúde. A partir deles, as normas exigidas pelo SUS serão utilizadas para verificar o correto preenchimento do formulário de dados de domicílios.


**Figura 41 - Diagrama de classe MValidaDomicilio**

Os atributos **camposValidar**, **ValoresGravar** e **_dadosDomicilio** representam os três principais atributos da classe de validação de domicílios. O atributo **camposValidar** representa um vetor multidimensional no qual estão gravados o nome do campo a ser validado, o número do campo no formulário de preenchimento e por fim a sua exigência segundo o tipo de programa de cobertura ao qual o domicílio se encontra.

O nome do campo presente no vetor multidimensional é o mesmo presente na estrutura de definição de campos do objeto do tipo registro. Desta forma é feita a ligação entre os campos a serem validados e o valor dos campos preenchidos que se encontra no objeto do tipo registro. O número do campo é utilizado para a geração de relatórios de erros encontrados em fichas enviadas. Este número será o mesmo número utilizado na ficha de coleta de dados por parte do agente coletor de dados. Esta ligação visual facilitará a localização do erro por parte do agente coletor. Por fim está previsto a obrigatoriedade de um determinado campo segundo o programa de cobertura ao qual o domicílio está exposto. Os seus possíveis valores são PSF e TODOS. Caso a cobertura do domicílio seja PSF, PACS e Similares PSF, os campos marcados com PSF no vetor multidimensional terão o seu valor obrigatório no registro coletado. Caso a forma de cobertura do domicílio seja OUTROS então os campos marcados com PSF serão opcionais. Os campos marcados com TODOS terão a sua obrigatoriedade de preenchimento verificada independente do tipo de cobertura experimentada pelo domicílio.

O atributo **camposGravar** representa um armazenador no qual o número dos campos que apresentaram erros estão presentes para que ao final seja gerado um registro na base de dados informando que a ficha transmitida contem erros e que os campos com os números presentes neste registro devem ser corrigidos para que os dados possam ser enviados para o distrito federal. Caso este atributo esteja vazio ao final da validação representa que nenhum erro foi encontrado e que o registro está pronto para transmissão.

Por fim o atributo **_dadosDomicilio** contem os dados sendo validados contidos em um objeto do tipo registro que permite entre outras coisas a coleta do

valor dos dados nele armazenados através do fornecimento do nome de um campo. Esta funcionalidade permite que todos os campos presentes no vetor multidimensional contendo a definição do que será validado sejam coletados e validados e que o número do campo esteja presente no registro de erro para facilitar a localização e correção do mesmo.

Os métodos **valida**, **validarOutros** e **validarPSF** representam as principais ações tomadas pelo objeto que valida domicílios. A ação **valida** realiza a coordenação do processo de validação, através dela os campos presentes no vetor multidimensional serão coletados no objeto do tipo registro e enviados para verificação da sua adequação as normas exigidas pelo SUS. Dependendo do tipo de cobertura usufruída pelo domicílio será disparada a sua validação adequada. Caso o domicílio seja coberto pelos programas PSF, PACS ou Similares a PSF então a ação **validarPSF** será disparada caso contrário o domicílio será coberto pelo programa OUTROS e a ação **validarOutros** será disparada. Como resultado final destas ações, o atributo **valoresGravar** irá conter o número dos campos que apresentaram erro ou estará vazio informando que o registro está correto.

O retorno da ação **validar** será um comando SQL *insert* contendo o registro a ser inserido na base de dados informando quais campos, através do seu número, apresentaram erro e que devem ser concertados ou nada caso não exista erro.

### *5.2.5 Classe de validação de dados de Moradores*

Um objeto da classe **MValidaPessoa** apresentada na Figura 42 será utilizado para a validação dos dados de um morador enviado.


**Figura 42 - Diagrama de classes MValidaPessoa.**

Esta classe apresenta os mesmos atributos da classe de validação de um domicílio, isto é, os atributos **camposValidar**, **valoresGravar** e **_dadosPessoa** então presentes.

O vetor multidimensional **camposValidar** contem os atributos nome do campo a ser validado e o número deste campo na apresentação visual do formulário de coleta de dados.

O atributo **_dadosPessoa** é um objeto do tipo registro contendo todos os dados de um morador a ser validado e o atributo **valoresGravar** representa um armazenador que irá conter o número dos campos que apresentaram erro segundo a estrutura pedida pelo ministério da saúde.

Como ações principais estão valida, validaConjuntoTipoCertidao, validaConjuntoIdentidade, validaConjuntoCTPS, validaConjuntoTituloEleitor e validaConjuntoNacionalidadeMoradia.

A ação **valida** irá coordenar a coleta de dados vindos do objeto **_dadosPessoa,** que é um objeto do tipo registro, e envio dos mesmos para uma validação básica que contempla a presença ou não do dado no registro coletado. Após a passagem na rotina básica de validação, as rotinas de validação referencial são executadas, tais rotinas prevêem que um conjunto de dados seja validado em conjunto, pois a presença ou ausência de um deles pode levar a obrigatoriedade ou não de outro campo que faz parte do grupo.

As ações cujo nome começa com valida representam ações de validação referencial. Como exemplo pode ser citado **validaConjuntoIdentidade,** cujo objetivo é validar todos os campos que dizem respeito ao conjunto de dados que representam a identidade de um cidadão. Caso um dos campos presentes no conjunto esteja preenchido todos os outros deverão estar preenchidos também para que o conjunto seja considerado correto.

Assim como ocorre com a classe que realiza a validação de dados de domicílios, esta classe irá oferecer como retorno da invocação da sua função gerente chamada **valida**, um comando SQL *insert* que conterá os campos que apresentaram erro. Caso nenhum erro tenha sido encontrado nada será retornado.

### *5.2.6 Classe para acesso a base de dados*

O sistema servidor utiliza um padrão de desenvolvimento no qual todo acesso a ser realizado a base de dados ocorrerá através de um objeto conhecido por DAO(*Data Access Object*). Como visto na Figura 39, página 87, a classe **AcessaBaseDaoImpl** representa o DAO empregado.

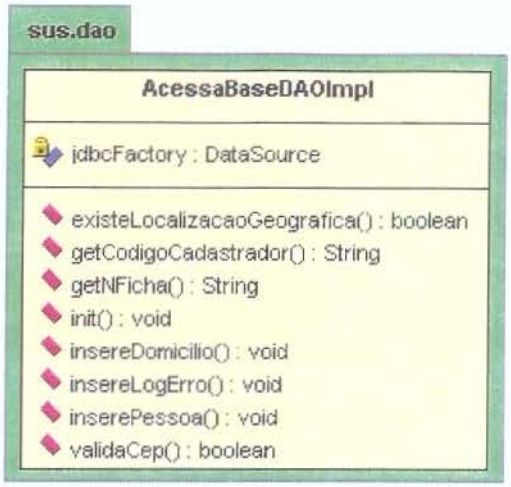


**Figura 43 - Diagrama de classes AcessaBaseDaoImpl**

O diagrama da Figura 43, apresenta a classe de acesso a base de dados utilizada no sistema.

O atributo **jdbcFactory** representa uma origem de dados. Esta origem de dados será coletada do contêiner J2EE utilizado tendo em vista o seu nome. Uma vez coletado o recurso, o mesmo será utilizado para abrir conexões com a base de dados todas as vezes que for necessária uma interação.

Todos os métodos presentes na classe representam ações sendo executadas na base. Ações como validar CEP(**validaCep**), inserir o registro de erro de dados enviados(**insereLogErro**), inserir os dados de domicílio coletados(**insereDomicilio**), inserir dados de moradores(**inserePessoa**), verificar a preexistência de uma localização geográfica o que implica em domicílio já cadastrado(**existeLocalizacaoGeografica**), coleta do código do agente de cadastramento(**getCodigoCadastrador**) e coleta do número da ficha a ser criada(**getNFicha**).

Através deste padrão de desenvolvimento o código de acesso a base de dados fica centralizado permitindo a fácil localização e modificação do mesmo assim como permite que seja feita a troca da base de dados utilizada no sistema sem que o código de negócio precise ser modificado uma vez que a interface fornecidas pelo DAO continuará a mesma sendo que o seu código fonte será modificado para que as novas particularidades da nova base sejam assimiladas.

### *5.2.7 Visualizar e editar os registros recebidos*

Uma vez inseridos na base de dados local, os registros enviados pelos clientes móveis ficarão disponíveis para que sejam realizadas as correções necessárias nos registros que apresentaram erros assim como o envio dos mesmos para o sistema central do cartão SUS localizado no distrito federal. Este trabalho é realizado por um programa baseado na tecnologia JAVA J2SE criado para este fim.

A busca ocorre através de chaves que identificam um registro. Para buscar um domicílio é utilizado como chave o código do domicílio ou uma parte da localização geográfica do mesmo formada pela área da equipe, micro área e família como pode ser visto na Figura 44.

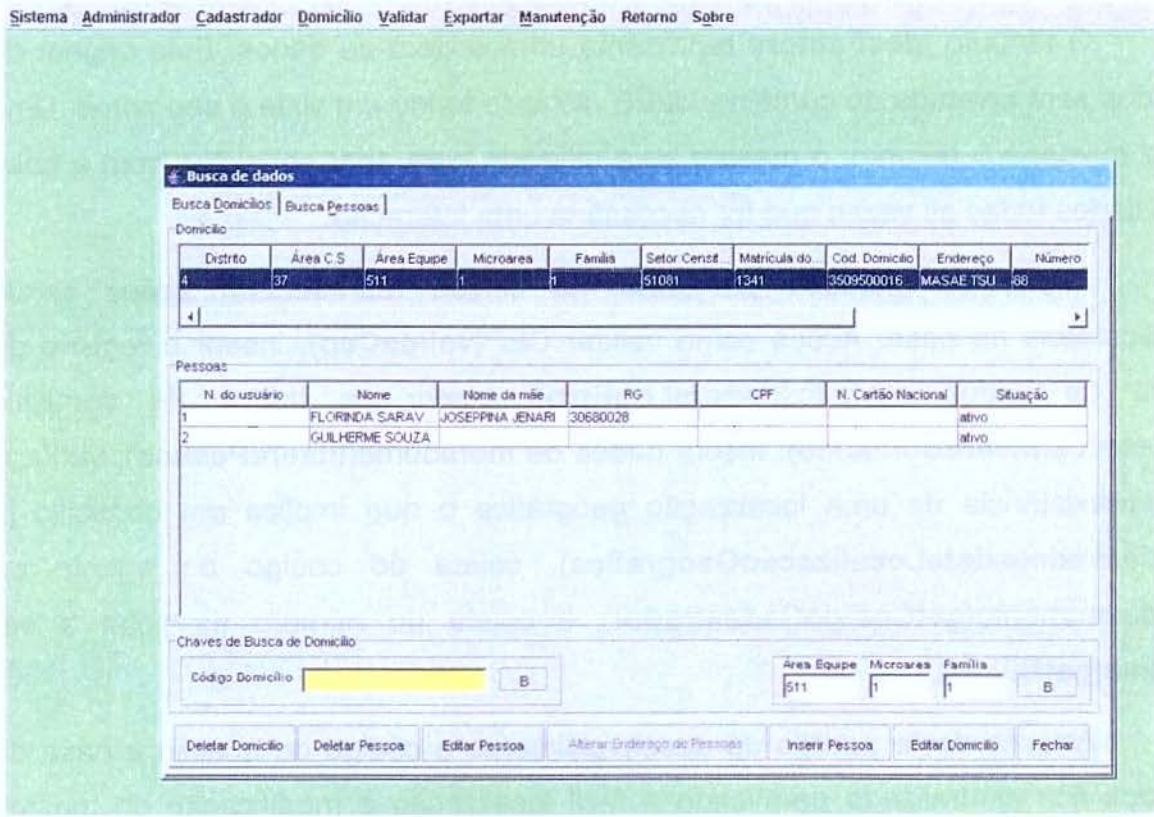


**Figura 44 - Busca de Domicílio**

Como resultados são apresentados os dados de domicílio que uma vez selecionados apresentam os moradores pertencentes a aquela residência. A busca de pessoas ocorre de forma semelhante utilizando chaves específicas.

Ao ser pedida a edição de dados, na tela de edição irão ficar destacados com vermelhos os campos que apresentaram erros segundo os testes de validação aos quais o registro foi submetido. A Figura 45 apresenta a edição de dados de um morador enviado contendo erros.

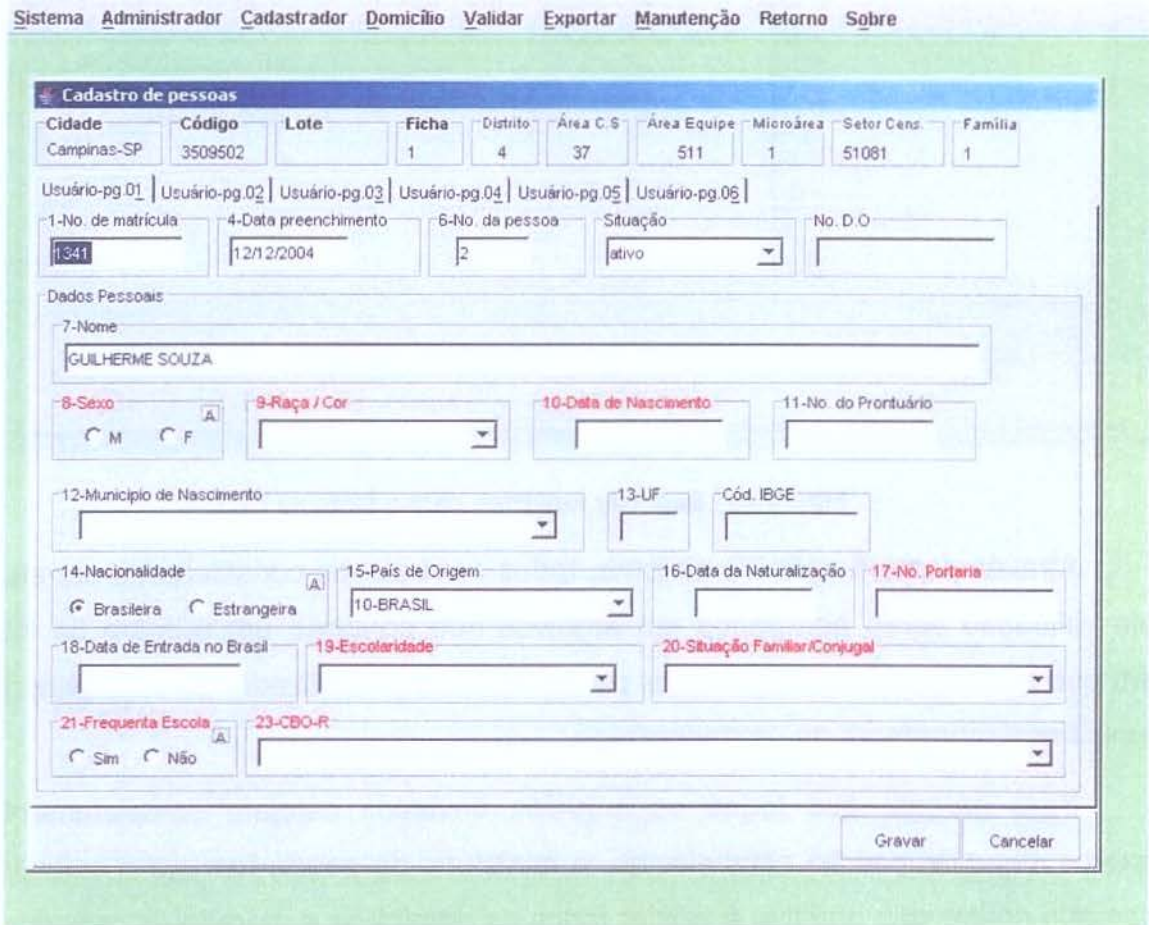


**Figura 45 - Editando um registro com erro.**

Tendo estas informações destacadas é feita a correção das mesmas e na seqüência uma nova validação será realizada para que seja confirmada a sua adequação as normas.

Como passos seguintes, serão gerados lotes nos quais domicílios e moradores serão enviados para o sistema do SUS. A Figura 46 apresenta a exportação de dados para o sistema SUS.

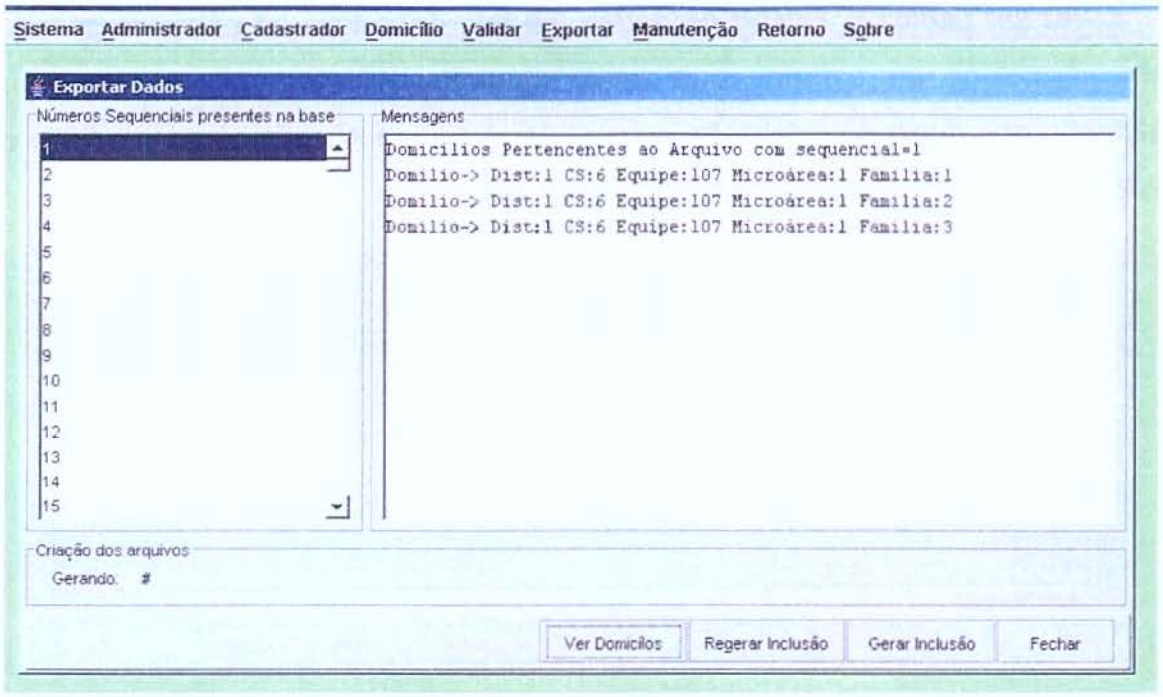


**Figura 46 - Exportar registros para o sistema SUS**

Através desta função do sistema, todos os registros considerados corretos pela validação serão agrupados em arquivos que conterão um número de lote para que os mesmos sejam enviados para o sistema centralizador de cadastros de domicílios e moradores de um município.

Para garantir que todos os arquivos enviados estejam completamente corretos segundo padrão estabelecido, o ministério da saúde fornece o software chamado crítica cujo objetivo é validar todos os domicílios e moradores presentes em um arquivo de transmissão e gerenciar a transmissão destes arquivos para a base de dados central do SUS. Tendo em vista esta regra, os arquivos para transmissão gerados pelo sistema serão submetidos ao programa de critica para que a partir dele seja feita a revalidação e transmissão dos dados assim como a coleta do resultado do processamento efetuado pelo sistema centralizador. O resultado do processamento ocorrerá através do envio de arquivos de retorno que serão coletados pelo programa de crítica. Os dados retornados serão submetidos ao sistema para que o resultado do processamento dos registros enviados seja disponibilizado. A Figura 47 contem a funcionalidade de processar os arquivos de retorno.

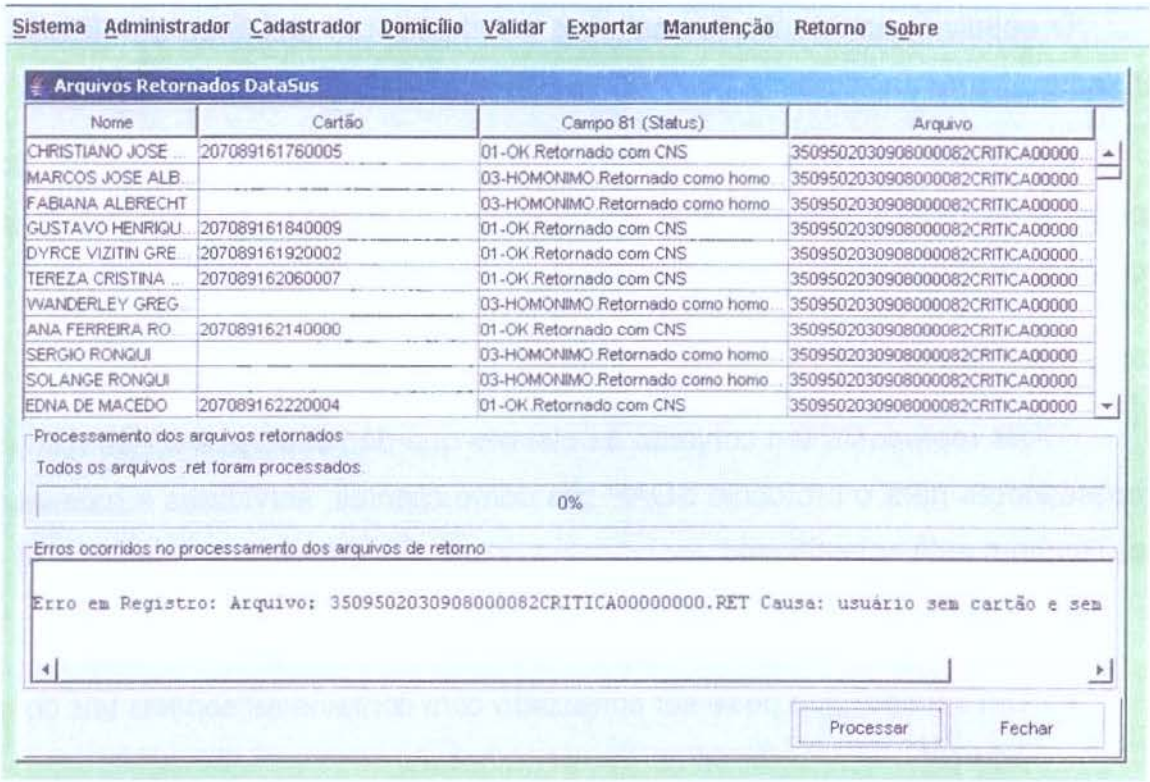


**Figura 47 - processamento dos arquivos de retorno**

O arquivo de retorno irá conter o nome do morador processado, o resultado do processamento informado através do campo 81. Registros enviados e que não apresentaram nenhum problema no seu processamento irão conter como retorno o número do cartão nacional de saúde que será utilizado pelo morador para acesso aos serviços públicos oferecido pelo governo federal.

## 5.3 Tecnologias utilizadas no lado Servidor

No lado servidor do sistema de coleta móvel de dados de saúde de um município, foram utilizadas as tecnologias: JBOSS [Jboss], AXIS [Axis], KSOAP [ksoap, ksoap2], MySQL [Mysql], Plugin Eclipse LomBoz [Lomboz].

### *5.3.1 JBOSS*

JBOSS é um servidor de aplicações ou container J2EE de código aberto e de acordo com os padrões. Ele foi totalmente implementado utilizando a liguagem Java.

O objetivo primário deste container é fornecer um ambiente de suporte a EJB's.

Container utilizado no sistema para dar suporte a execução dos componentes EJB's criados para disponibilizar o recebimento dos dados de moradores e domicílios em um município.

### *5.3.2 Axis*

Axis representa um conjunto de classes que dão suporte a construção de processadores para o protocolo SOAP tais como clientes, servidores e gateways, etc. Também está especificado:

- Servidor único.
- Um servidor que pode ser conectado com containeres servlets tais como Tomcat.
- Grande suporte a *Web Service Description Language* (WSDL)
- Ferramenta de emissão para a geração de classes Java a partir de um arquivo WSDL.
- Uma ferramenta para monitorar pacotes TCP/IP

Empregado para disponibilizar a chamada de função para o envio de dados de domicílios e moradores através de *Web Services*. AXIS disponibilizará a descrição do serviço através da linguagem WSDL. Aplicativos que rodam em plataformas computacionais com maior poder de processamento irão utilizar esta descrição, para a geração de classe que serão utilizadas para a invocação de métodos presentes em componentes localizados no servidor.

Clientes móveis não empregam este pacote tendo em vista que o mesmo foi projetado para ser utilizado em plataformas com maior poder de processamento. Para utilizar *Web Services*, os dispositivos móveis empregam a API KSOAP.

### 5.3.3 KSOAP

Pacote responsável por disponibilizar *Web Services* através da utilização do protocolo SOAP.

Chamadas de procedimentos são realizadas através da utilização da classe base **SOAPEnvelop**.

Utilizado para dar suporte a chamada remota de procedimento, por parte do dispositivo móvel, empregando o protocolo SOAP. Este pacote está presente também no cliente móvel.

Esta API não irá disponibilizar a descrição do serviço sendo divulgado pelo servidor na forma de WSDL. Esta descrição está disponível apenas para clientes como maior poder de processamento.

### 5.3.4 MySQL

MySQL representa a base de dados do sistema. Ela fornece um servidor SQL(*Strutured Query Language*) robusto, multi-usuário e multi-Thread.

Este sistema pode ser utilizado como produto de software livre ou código aberto obedecendo aos termos da licença GNU *General Public License*, ou pode ser comprada a sua versão comercial.

Empregado para disponibilizar todo o armazenamento de dados necessário pelo sistema. Tabelas para o armazenamento de dados de moradores e domicílios foram criadas assim como tabelas para o armazenamento de dados secundários.

### 5.3.5 Plug-in Eclipse Lomboz

Lomboz é um *plug-in* eclipse utilizado para o desenvolvimento de aplicações na plataforma J2EE, tais como JSP, Servlets e EJB's.

Suas principais características são:

- Assistente para a criação e montagem de módulos J2EE.
- Checagem de sintaxe de páginas JSP.

- Suporte a todos os servidores de aplicações Java como lançamento flexível e capacidades de debug.
- Assistentes para a criação de EJB's.
- Assistentes para a criação de *Web Services* baseados em AXIS.

Este *plug-in* foi utilizado na criação do sistema, para dar suporte a geração dos EJB's de processamento de envio de registros de domicílios e moradores por parte de um dispositivo móvel. Outro ponto de destaque disponibilizado por este componente foi a geração de descritores de implantação dos EJB's no servidor de aplicações utilizado, isto é, o container JBOSS. Este *plug-in* permite também o lançamento do servidor de dentro da plataforma eclipse, facilitando a depuração do sistema.

# 6 Resultados Obtidos

Como resultado deste projeto, será apresentado o sistema em operação, incluindo o cadastramento de usuários do Sistema Único de Saúde, a sua transmissão para o servidor que centraliza os dados pertencentes a um município assim como o próprio servidor em ação, recebendo e editando os registros enviados.

A utilização do sistema começa com a escolha da funcionalidade "cadastrar domicílio" no menu do aplicativo presente nos dispositivos móveis de coleta de dados, tal como apresentado na Figura 48.


**Figura 48 - Menu principal do sistema de automação da força de cadastramento de dados.**

Na seqüência será pedida a inserção de dados referentes ao domicílio como apresentado na Figura 49. Nela pode ser observada a inserção de um domicílio cujo nome do logradouro é "Rua Samuel Moura n.333 bairro centro"


**Figura 49 - Telas para cadastro de um domicílio.**

Após o preenchimento dos dados estará disponível as ações apresentadas no menu presente na Figura 50. Neste menu consta a navegação entre as telas de cadastro fornecidas assim como a opção de armazenar os dados digitados na base de dados local do dispositivo. Para este caso será realizada a inserção através da seleção do item número 5 do menu.


**Figura 50 - Ações possíveis a serem realizadas durante o cadastro de um domicílio.**

Uma vez realizada a inserção de um domicílio, a próxima ação será a submissão dos dados de um morador para completar o cadastro de um lar. Uma vez que não existe a possibilidade de que um morador seja cadastrado sem que o mesmo esteja vinculado a uma residência, a busca de um domicílio se faz necessária. A Figura 51 apresenta a busca do domicílio recém cadastrado para a inserção de um morador.


**Figura 51 - Processo de busca de um domicílio**

Uma vez localizado o domicílio, existe a possibilidade de utilização das ações apresentadas na Figura 52. Nela constam as ações:

- Editar: Leva a visualização dos dados cadastrados previamente permitindo a sua edição e gravação na base de dados.
- Ver Todos Moradores: Apresenta uma lista contendo todos os moradores vinculados ao domicílio em questão.

- Inserir Morador: Permite a inserção de um morador no domicílio localizado.


**Figura 52 - Ações possíveis em um domicílio localizado.**

Selecionando a opção 5 do menu presente na Figura 52 irá levar a inserção de um morador. As telas presentes na Figura 53 representam algumas das telas disponíveis para a coleta de dados de um morador. Nela pode ser contatado o cadastramento do registro de um morador chamado "Antonio Carlos".


**Figura 53 - Telas para o cadastro de um morador.**

Depois de feito o cadastramento de domicílios e moradores o sistema de coleta disponibiliza um sistema de transmissão destes dados para o servidor central. A Figura 48 apresenta o item "Transmitir Dados". Este item será responsável por levar a tela responsável por transmitir os dados de lares cadastrados até o momento em um dispositivo de coleta.

A Figura 54 apresenta a inicialização do servidor da aplicação. O servidor de aplicações JBoss irá conter os EJB's criados para realizar as ações de recebimento de dados, validação dos mesmos assim como gravar os cadastros recebidos na base de dados MySQL criada.

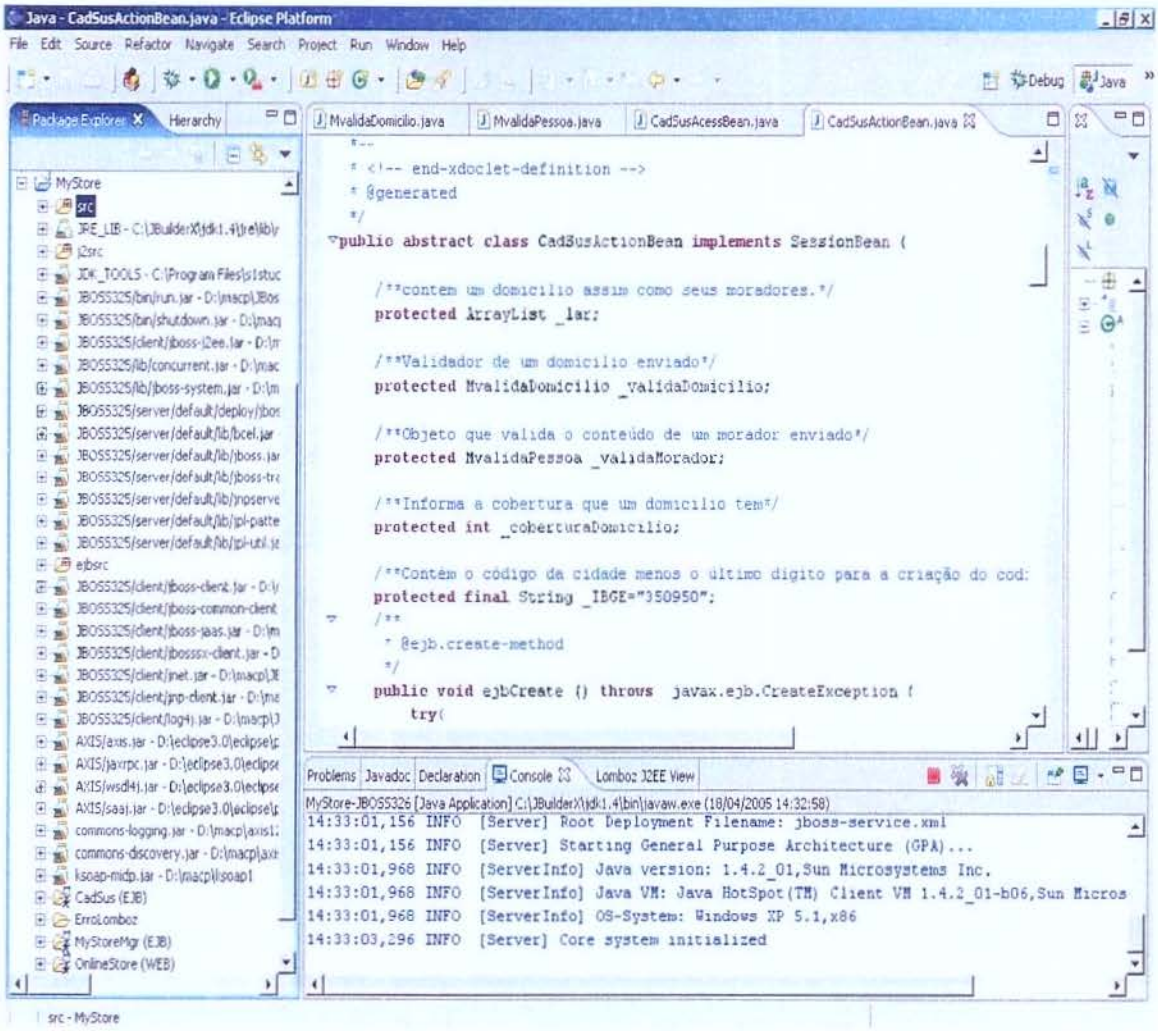


**Figura 54 - Inicialização do servidor JBoss através do IDE Eclipse.**

Uma vez disparada a transmissão dos dados pelo dispositivo móvel, o sistema *Web Service* KSOAP irá dar suporte ao envio dos dados. A Figura 55 apresenta as mensagens geradas no lado servidor durante o recebimento dos registros. Nela é possível constatar que ao ser recebido o fim de um lar, rotinas de validação de conteúdo são disparadas verificando a pré-existência de chaves na base de dados local para permitir a gravação do registro recebido. Outra validação é feita para que seja verificada se os dados estão corretos ou não segundo o leiaute definido pelo SUS. O primeiro tipo de validação inviabiliza a gravação do dado na base de dados, a segunda irá gerar registros de erro, os quais informam quais campos apresentaram problemas segundo o leiaute do SUS.

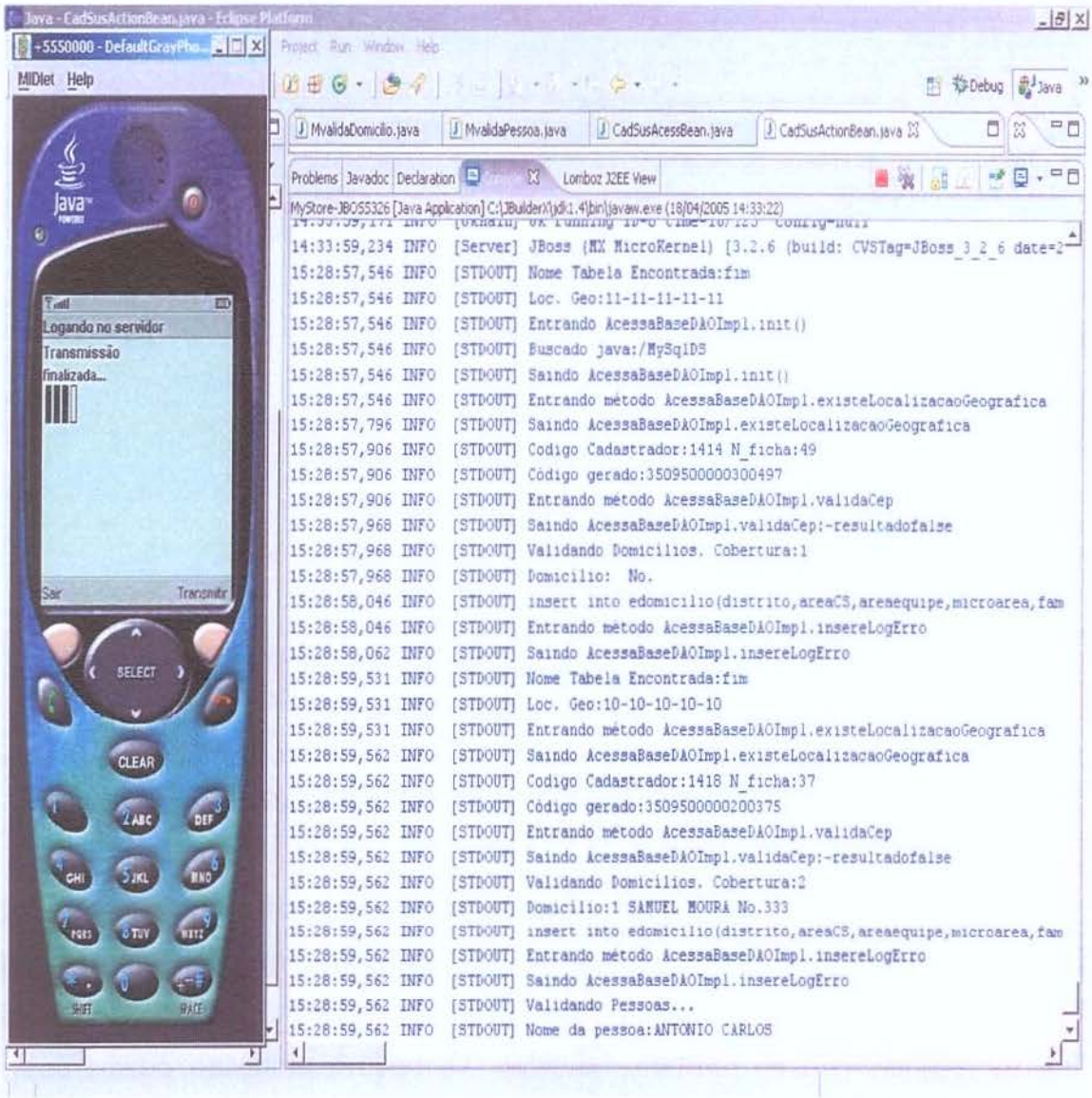


**Figura 55 - Recebimento dos dados transmitidos pelo dispositivo móvel.**

Uma vez validados os registros estarão presentes na base de dados. A Figura 56 apresenta como foram gravados na base de dados do servidor os dados do domicílio transmitido.

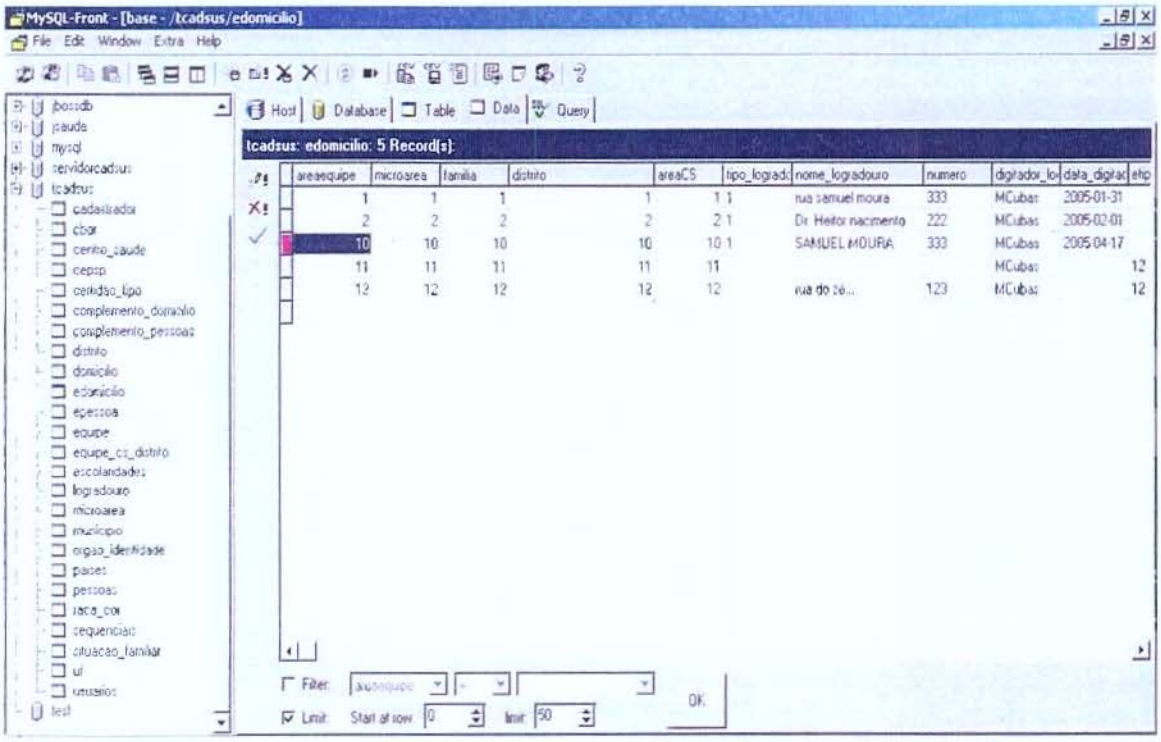


**Figura 56 - Dados do domicílio endereço "rua Samuel Moura 333" presente na base de dados na tabela Domicílios. [Figura extraída do programa MySQL-Front versão beta. 2001]**

Por ter apresentado erro segundo o leiaute do SUS, a gravação deste registro na base de dados gerou também um registro de erro como apresentado na Figura 57. Através deste registro é possível saber quais campos coletados não estão de acordo com as normas definidas. O mesmo processo ocorre com o morador "Antonio Carlos".

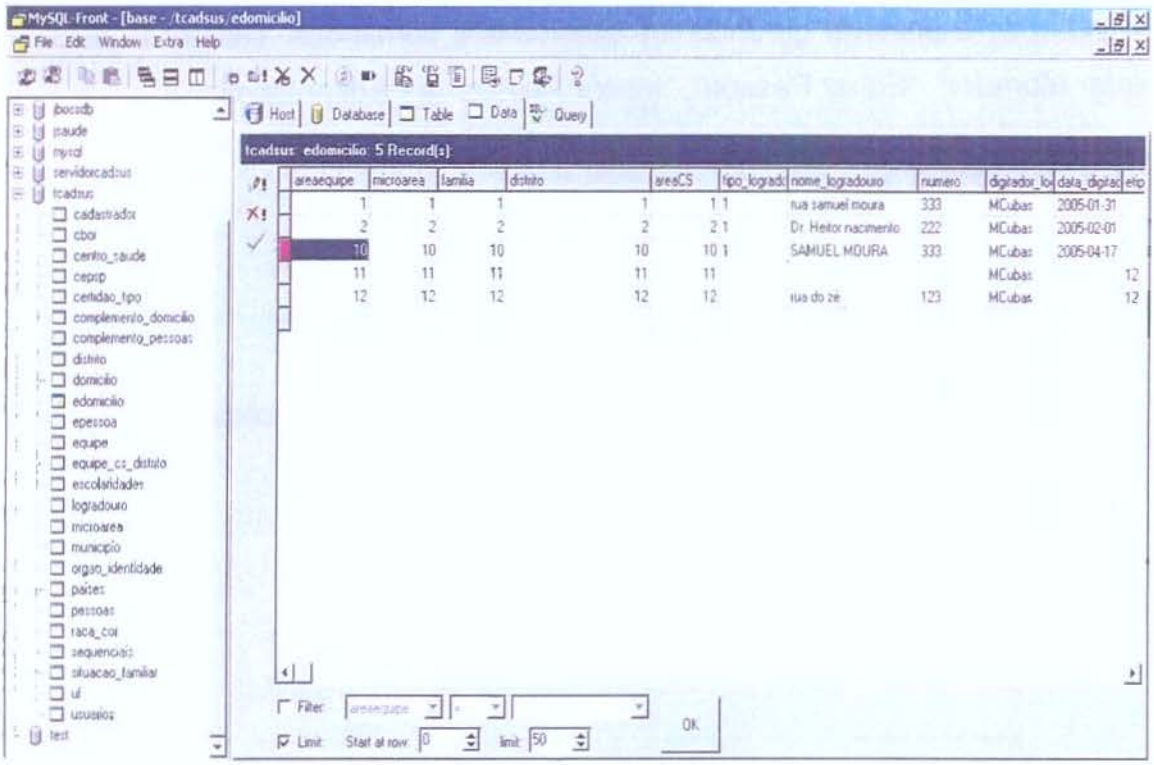


**Figura 57 - Registro de erro gerado por causa da gravação do registro de domicílio endereço "rua Samuel Moura 333". [Figura extraída do programa MySQL-Front versão beta. 2001]**

Uma vez presente na base de dados, estes registros ficam disponíveis para edição através de um programa criado para a edição, visualização, cadastramento e comunicação com o servidor central do SUS. A localização do lar transmitido ocorre como apresentado na Figura 58. Nela pode ser notado que a parte da chave geográfica de um domicílio é utilizada para localizar o domicílio cujo endereço era "Rua Samuel Moura 333." Os campos "área de equipe", "microárea" e "família" representam parte da chave utilizada. Para o domicílio citado, todos estes campos continham o valor 10. Nesta mesma tela é possível notar que existe outro fichário responsável pela busca de moradores, sendo que através desta é possível a localização do morador "Antonio Carlos" que foi transmitido como sendo morador deste domicílio. Uma vez selecionado o domicílio na tabela visual que apresenta os mesmos, todos os seus moradores irão ser apresentados na tabela visual localizada logo abaixo. Na Figura 58 estão apresentados o domicílio "rua Samuel Moura 333" e o seu morador "Antonio Carlos". Tendo estes dados sido

localizados será possível gerenciá-los através dos comandos "Deletar Domicílio", "Deletar Morador", "Editar Pessoa", "Inserir Pessoa" e "Editar Domicílio".

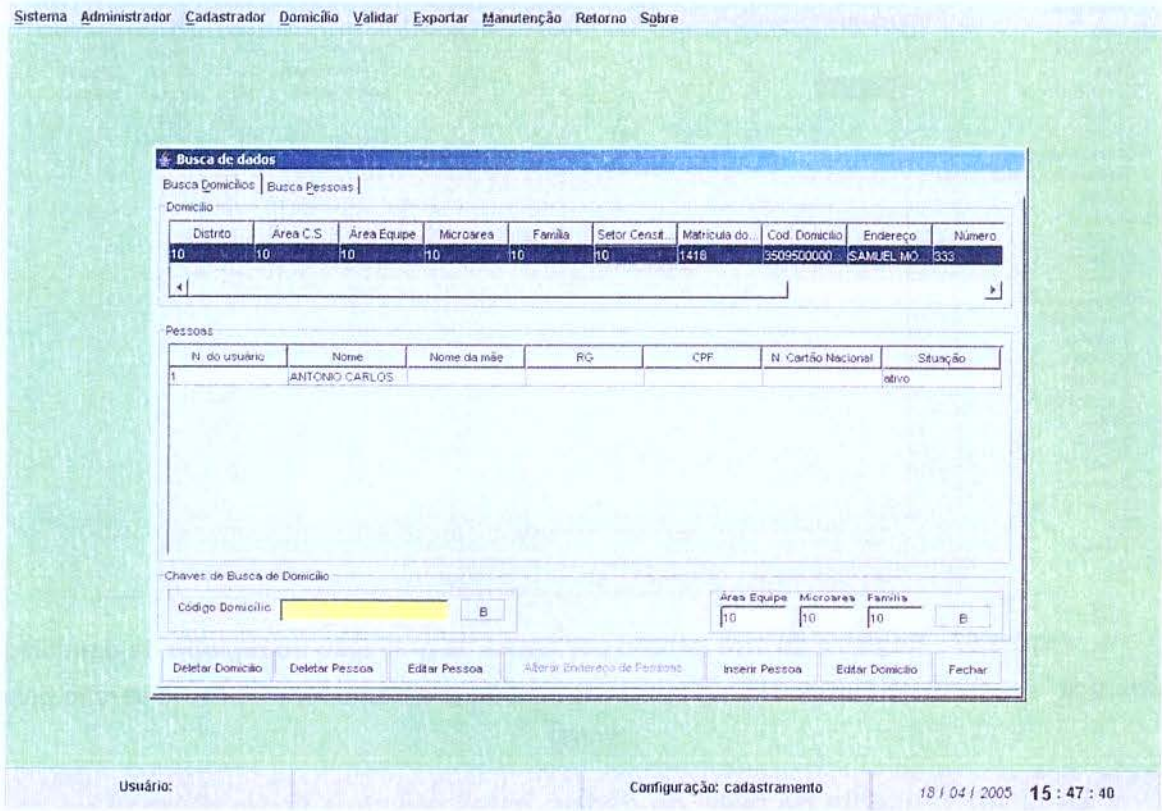


**Figura 58 - Tela de busca de registros no programa utilizado para administrar os dados recebidos no servidor.**

Selecionando a pessoa "Antonio Carlos" na tabela de moradores e clicando no botão "Editar Pessoa" será apresentada a tela presente na Figura 59. Nesta tela pode ser notado que existem campos que estão destacados através da utilização da cor vermelha nos nomes dos campos (por exemplo o campo "Escolaridade"). Estes campos destacados representam os erros encontrados nos dados transmitidos para o servidor que não estão de acordo com as normas definidas pelo Ministério da Saúde. O mesmo procedimento ocorre com os dados do domicílio recebido.

Através do correto preenchimento dos campos que apresentaram erro e posterior gravação dos mesmos, este registro ficará apto a ser transmitido para o

servidor do Ministério da Saúde que irá abrigar os registros de moradores e domicílios de todo o país.

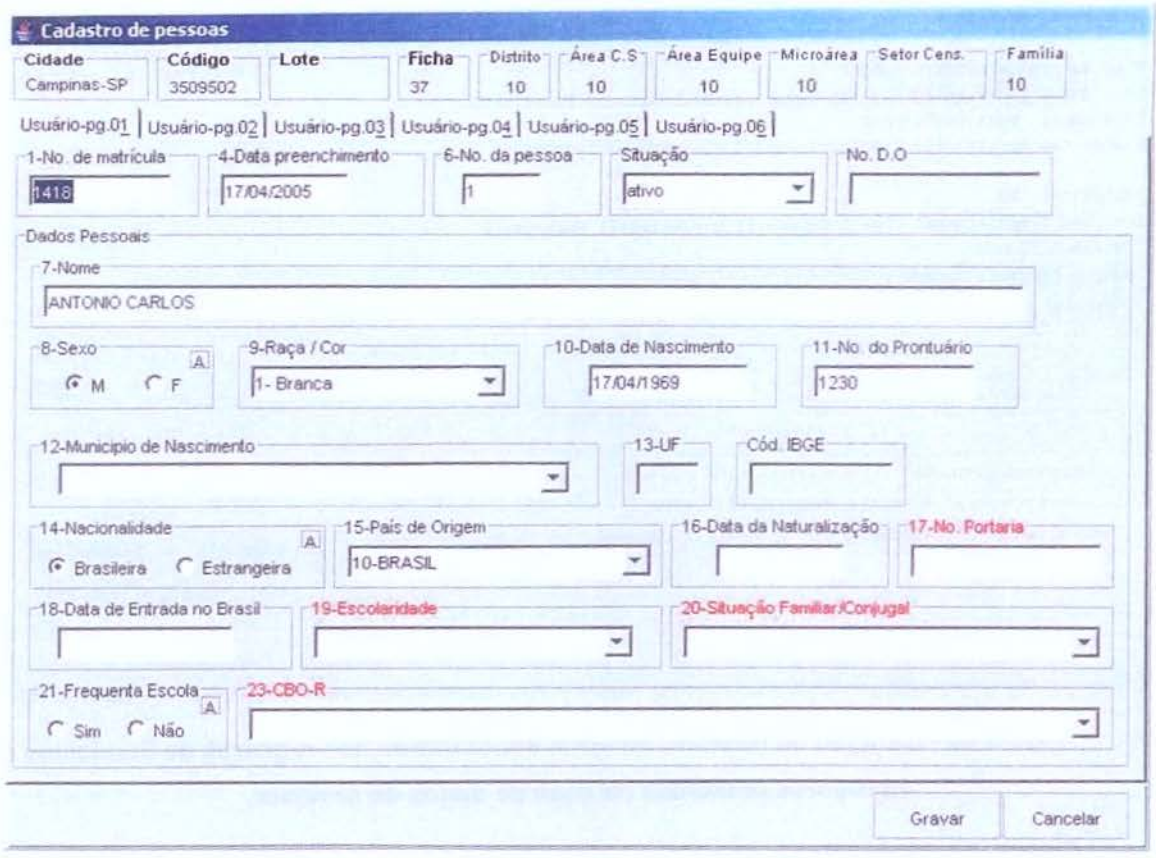


**Figura 59 - Tela que apresenta os dados de um Morador a ser editado. Seis são os fichários contendo dados de um morador. Esta figura apresenta apenas o primeiro fichário.**

Além do procedimento descrito acima, o aplicativo de gerenciamento de dados recebidos fornece para o administrador da base de dados do município a possibilidade de geração de um relatório contendo todos os domicílios e moradores que apresentaram erro durante o processo de validação segundo as normas do SUS. Este procedimento pode ser visto na Figura 60. Nesta figura é visto que o domicílio "Rua Samuel Moura No. 333" apresentou erro no campo 17, e que o morador "Antonio Carlos" apresentou erro em vários campos entre eles o campo 17, 19 e 20. Na Figura 59 pode ser visto que os campos em questão estão destacados com vermelho. Uma vez tendo este relatório de erros, é possível notificar o cadastrador que enviou os dados para que o mesmo realize a correção do registro junto ao morador.

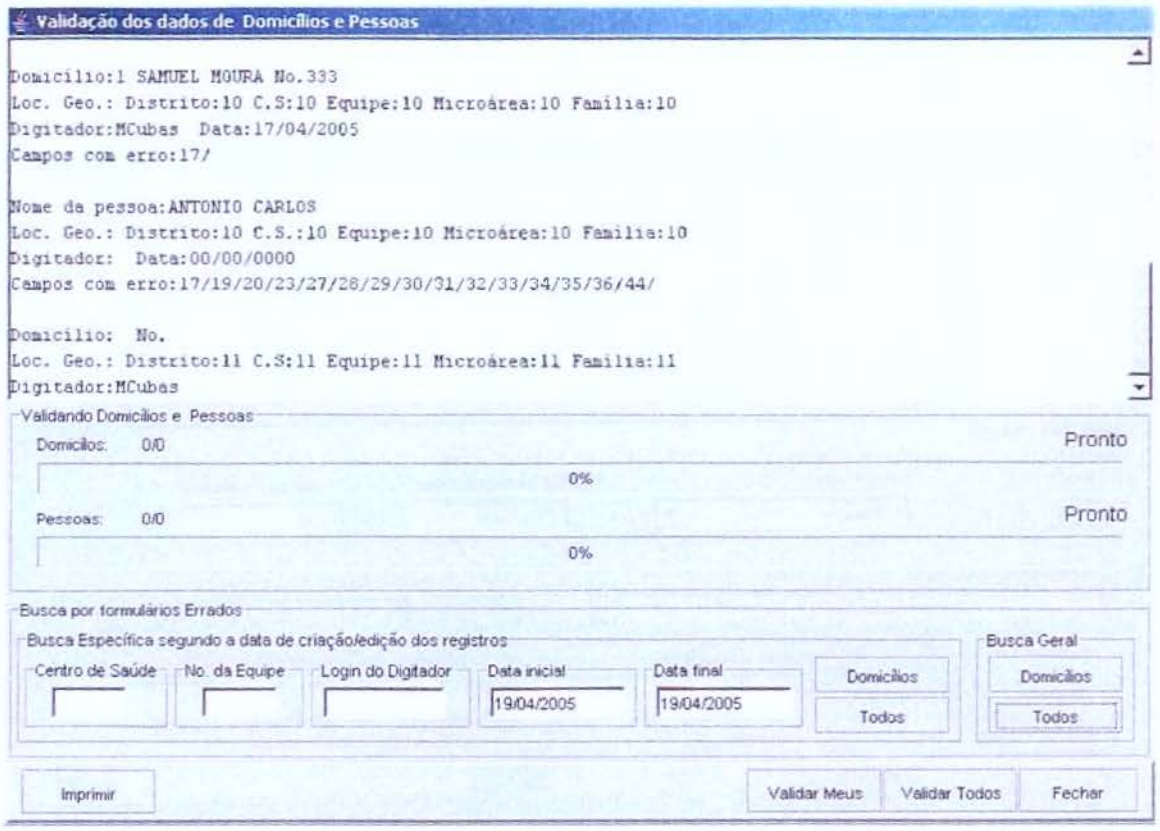


**Figura 60 - Geração de relatório de erros encontrados nos registros de domicílios e moradores presentes na base de dados do servidor.**

Outra funcionalidade fundamental deste sistema é a possibilidade de comunicação com o sistema central do SUS que irá receber os registros provenientes do município, validar os mesmos e, se tudo estiver correto, será gerado um numero que servirá como forma de acesso do cidadão aos serviços médicos disponibilizados pela federação brasileira. Este procedimento é exemplificado na seção visualizar e editar os registros recebidos do capítulo sistema servidor.

# 7 Conclusões

Com o objetivo de dar os primeiros passos para a criação de uma arquitetura integrada para gestão de governo, foi analisado o sistema de cadastro de domicílios e moradores utilizado pelo ministério da saúde do Brasil. Pontos falhos foram detectados tendo em vista alguns requisitos de governo eletrônico levantados:

- Processo demorado, pois exigia a ida de uma pessoa responsável pelo cadastramento mais de uma vez a mesma residência em caso de invalidade dos dados coletados. Este processo poderia ser ainda mais lento se o morador não estivesse em casa na segunda visita.
- Processo custoso, pois necessitava de gastos com papel e deslocamentos repetidos.
- Gera atrasos na liberação dos cartões definitivos para acesso a serviços de saúde disponibilizados pelo governo federal.

Como forma de melhor inserir este sistema nos conceitos pregados pelo governo eletrônico, foram criados:

- Sistema móvel de coleta de dados: Sistema criado em J2ME focado em aparelhos celulares e PDA's cujo benefício é a redução dos custos e prazos de geração dos registros, pois uma vez localizado o morados, seus dados serão coletados, transmitidos para o servidor municipal, validados e serão gerados relatórios de erro que poderão ser corrigidos no momento.
- Sistema servidor de dados coletados: Sistema gerado em J2EE que foca as regras de negócio levando a um gerenciamento municipal de dados, tornando os mesmos mais rapidamente acessíveis.
- Sistema de comunicação com o servidor federal de registros: Sistema criado em J2SE cujo objetivo foi permitir a comunicação entre o servidor municipal de registros de saúde e o servidor federal.

Além disso, este sistema permite o acesso aos dados por parte dos administradores municipais levando a rápida tomada de decisões.

- Tecnologia de comunicação: A tecnologia de *Web-Services* foi empregada no intuito de tornar o servidor do município acessível por todos os tipos de clientes que consigam processar XML.

Além da geração dos sistemas acima, como contribuição desta tese, podem ser citados:

- Criação de um *framework* para aplicativos de cadastro em dispositivos móveis que dão suporte a plataforma J2ME.
- Primeiros passos para a criação de um sistema único de governo eletrônico, que vise empregar a tecnologia da informação de forma eficiente para maximizar a atuação do estado junto a seus cidadãos.

## 7.1 Objetivos Futuros

Como futuros passos deste sistema:

A. Permitir que clientes móveis obtenham seus registros junto ao servidor municipal para a edição do mesmo junto ao munícipe.

B. Expandir o cadastro para todos os setores que requerem tal funcionalidade para o controle do acesso aos serviços por ele disponibilizados. Esta expansão é suportada pelo *framework* criado para a coleta de registros junto aos moradores com a utilização de dispositivos móveis.

C. Gerar uma forma de acesso padronizada para os cadastros criados, permitindo o seu acesso a outros desenvolvedores de sistemas, tornando a base de dados, que contem os cadastros, propriedade do município, levando a otimização dos processos de cadastramento e evitando duplicações de registros.

D. Gerar uma camada de software responsável por filtrar dados para a geração de relatórios que servirão para a tomada de decisões por parte dos gerenciadores do município.

# 8 Referências bibliográficas

[Aoema] AOEMA (Ásia Oceania Eletronic Marketplace Association). "E-Government from User's perspective". http://www.aoema.org/E-Government/Definitions and Objectives.htm . Último acesso abril/2005.

[Armstrong] Armstrong, Eric. Ball, Jennifer. Bodoff, Stephanie. Carson, Debbie. Evans, Ian. Green, Dale. Haase, Kim. JenDrock, Eric. "The J2EE 1.4 tutorial.". Sun Microsystems. 2004.

[Axis] **A**pache E**X**tensible **I**nteraction **S**ystem (AXIS), "Web Services - AXIS". Apache. http://ws.apache.org/axis/. Último acesso em abril/2005.

[Bond] Bond, Martin. Haywood, Dan. Law, Debbie. Longshaw, Andy. Roxburgh, Peter. "Teach Yourself J2EE in 21 days". Sams Publishing. 2002.

[Chapelli] Chappell, Davis A. Jewell, Tyler. "Java Web Services.". O' Reilly. 2002.

[Datasheet] Sun Microsystems, "J2ME Technologies Overview. DataSheet java 2 platform, Micro Edition". http://java.sun.com/j2me/docs/j2me-ds.pdf. Sun Microsystems, 2002. Último acesso em abril/2005.

[Datasus] Datasus, "Layout para municípios cartão nacional de saúde. Versão 2.2.0.0 release 1". http://www.datasus.gov.br/ (produtos e serviços/cartão sus/downloads).2002.Último acesso em abril/2005.

[Demichiel] DeMichiel, Linda G. Yalçinalp, L. Ümit. Krishnan,Sanjeev. "Enterprise Java Beans Specification, version 2. Final Release.". Sun Microsystems. 2001.

[Eclipse] Eclipse - http://www.eclipse.org/. Último acesso em abril/2005.

[EclipseME] eclipseME - http://eclipseme.org/index.html. Último acesso em abril/2005.

[Flanagan] Flanagan, David. "Java in a Nutshell. 4ª edição". O'Reilly. 2002.

[Gosling] Gosling, Jasmes. McGilton, Henry. "The Java Language Environment: Contents". http://java.sun.com/docs/white/langenv/. Sun microsytems.1996. Último acesso em abril/2005.

[Governo Eletrônico] Governo Eletrônico. http://www.governoeletronico.gov.br/governoeletronico/index.html. Último acesso abril/2005.

[Java-Language] Sun Microsystems. "The Java Language: An Overview". http://java.sun.com/docs/overviews/java/java-overview-1.html. Sun Microsystems. Último acesso em abril/2005.

[Java-Language 2] Gosling, James. Joy, Bill. Steele, Guy. Bracha, Gilad. "The Java Language Specification, Second Edition". Addison Wesley. 2000.

[Jboss] JBOSS – http://www.jboss.org/products/index. Último acesso em abril/2005.

[JSR-30] Sun Microsystems, "Connected Limited Devices Configuration. Specification version 1.0a".Java Specification Request(JSR) JSR-30. Sun Microsystems. 2000.

[JSR-37] Sun Microsystems, "Mobile Information Device Profile. JSP Specification 1.0a". Java Specification Request(JSR) JSR-37.Sun Microsystems.2000.

[JSR-118] Sun Microsystems, "Mobile Information Device Profile. JSP Specification 2.0". Java Specification Request(JSR) JSR-118.Sun Microsystems.2000.

[JSR-139] Sun Microsystems, "Connected Limited Devices Configuration. Specification version 1.1". Java Specification Request(JSR) JSR-139.Sun Microsystems 2003.

[Ksoap] KSOAP - http://ksoap.objectweb.org/ . Último acesso em abril/2005.

[Ksoap 2] KSOAP 2.0 - http://sourceforge.net/projects/kobjects/ . Último acesso em abril/2005.

[Lomboz] Plugin Eclipse Lomboz - http://www.objectlearn.com/index.jsp. Último acesso em abril/2005.

[Maclaugblin] MacLaugblin, Brett. "Building Java Enterprise Applications. Volume 1: Architecture". O'Reilly. 2002.

[Marchionini] Marchionini, Gary. Samet, Hanan. Brandt, Larry. "Digital Government". Communications of the ACM. 2003.

[Mchugh] McHugh, Jeff. "Low bandwidth SOAP". http://webservices.xml.com/pub/a/ws/2003/08/19/ksoap.html. Agosto, 2003. Último acesso em abril/2005.

[Monson] Monson-Haefel, Richard. "Enterprise JavaBeans 3rd edition.".O'Reilly. 2001.

[Mysql] MySQL – http://www.mysql.com/ . Último acesso em abril/2005.

[Nagappan] Nagappan, Ramesh. Skoczylas, Robert. Sriganesh, Rima Patel. "Developing Java Web Services. Architecting and developing secure Web Services using Java.", Wiley Publishing.2003.

[Niemeyer] Niemeyer, Patrick. Peck, Joshua. "Exploring Java, 2nd edition". O'Reilly. 1997.

[Parreiras] Parreiras, T. A. S.; Cardoso, A. M.; Parreiras, F. S. "Governo eletrônico: uma avaliação do site da assembléia legislativa de Minas Gerais." In: CINFORM, 5, 2004, Salvador. **Anais.** Salvador: UFBA, 2004.

[Pereira] Pereira, Marcelo A. Cubas. Sharoni, Dan. Mendes, Leonardo. Manzolli, Jonatas. "JavaTM Technology Sounds Good: Designing Very Low-Cost Embedded Java Software Solution for MIDI-Based Interactive Music Game Devices".Sessão Técnica. Tópico "Intriguing and Unexpected: New and Cool". ID= TS-1679. Conferência JavaOne 2004, julho 2004, San Francisco, EUA .

[Pilioura] Pilioura, T.. Tsalgatidou, A.. Hadjiefthymiades. "Scenarious of using Web Services in M-Commerce". University of Athens. Grece. ACM SIGecom Exchanges, Vol. 3, No. 4, 2003.

[Piroumian] Piroumian, Vartan. "Wireless J2ME Platform Programming". The Sun MicroSystems Press. 2002.

[Roman] Roman, Ed. Ambler, Scott. Jewell, Tyler."Mastering Enterprise JavaBeans. Second Edition". Wiley ComputerPublishing. 2002.

[Snell] Snell, James. "Programming Web Services with SOAP". O'Reilly. 2001.

[Soap] SOAP V1.2. "SOAP Specification Version 1.2". W3C. http://www.w3.org/TR/soap12-part1/.2003. Último acesso em abril/2005.

[White] White, James. Hemphill, David. "Java 2 Micro Edition. Java in Small Things". Manning Publications. 2002.