6.
Implementação do MobiWfMS

A implementação do sistema foi desenvolvida no intuito de validar a arquitetura proposta. Ela cobre praticamente todos os pontos da arquitetura com exceção da comunicação parceiro – parceiro, referente a coreografia de *workflows*. Desta forma, o MobiWfMS pode ser definido como um sistema de coordenação (mais especificamente orquestração) de *workflow* com utilização de PDAs e com suporte a desconexão.

As soluções de implementação para dispositivos móveis com baixo recurso de processamento e pouca memória devem ter como preocupação permanente a otimização do código. Aplicações que não se atentam a este fator tendem a ser lentas, prejudicando e inviabilizando a usabilidade das mesmas. Estes tipos de soluções podem ser descartadas pelo usuário que esteja insatisfeito com o tempo de resposta. No caso de PDAs, é sempre bom evitar processamentos desnecessários. Processos de alto custo computacional como buscas em banco de dados, leitura e escrita de dados, transferência de arquivos pela Internet, devem ser efetuadas de maneira racional e quando não se tem outra saída. As funcionalidades do sistema fazem uso de todos esses processos computacionais citados, sendo assim, a implementação deste trabalho, além de ter como principal objetivo a validação da arquitetura, foi desenvolvida com essa preocupação.

A implementação foi desenvolvida utilizando alguns padrões de projeto. Mais informações sobre padrões de projetos podem ser encontradas em (Gamma, 1995; J2EE Patterns, 2001).

Este capítulo apresenta a implementação proposta no trabalho. Na seção 6.1 é apresentada toda a estrutura da implementação. Em seguida, na seção 6.2, é feita uma visualização do MobiWfMS através das suas telas. A seção 6.3 descreve as

tecnologias utilizadas, na intenção de facilitar o trabalho de pessoas que tiverem interessadas na utilização do sistema. Por fim, a seção 6.4 conclui o capítulo.

6.1. Estruturação da implementação

O sistema possui três principais módulos (Figura 24):

- Servidor: que reflete a implementação do Controlador Central. Nele foi desenvolvido as funcionalidades definidas através dos requisitos 1, 2, 3 e 4. Estes requisitos estão relacionados à definição do *workflow*, particionamento, suporte a *triggers* de emergência e controle da execução do *workflow*.
- Cliente: reflete a implementação dos parceiros. Aqui são cobertos os requisitos restantes: 5, 6 e 7. Estes requisitos estão relacionados a configuração inicial para participar do processo e execução do *workflow*.
- Integração: este módulo é o responsável pela comunicação entre o módulo cliente e o módulo servidor.

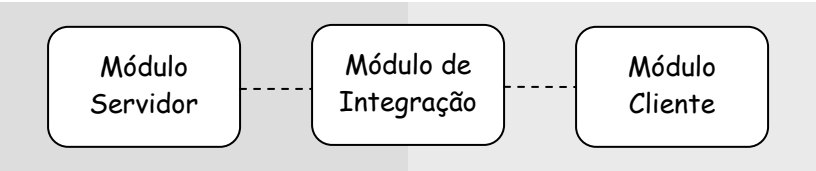

Figura 24 – Principais módulos do sistema

6.1.1. Módulo servidor

O módulo servidor foi projetado utilizando o padrão de projeto MVC (*Model/View/Controller*) (Reenskaug, 2003) (Figura 25):

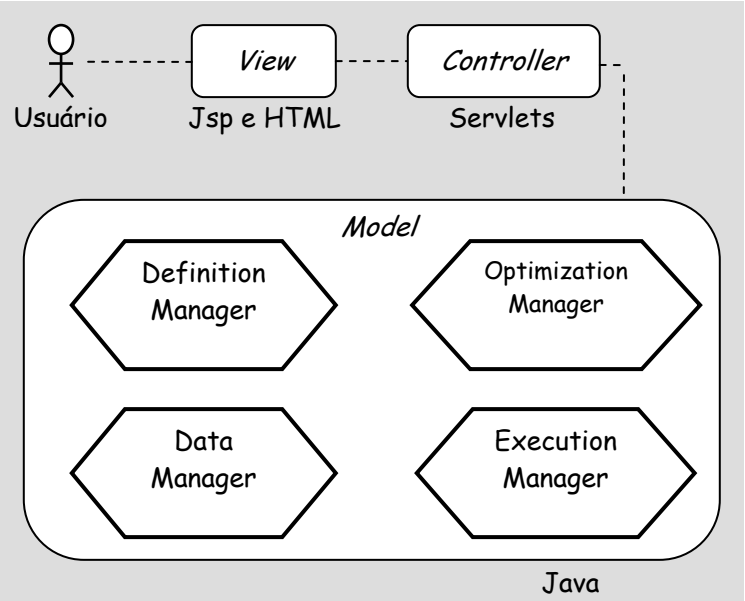

Figura 25 – Módulo Servidor

Este padrão propõe a divisão do sistema por blocos de funcionalidade. Esta divisão torna o sistema mais coeso em cada uma de suas parte, permitindo que alterações sejam feitas de forma mais simples e independente. A seguir são descritos cada um dos módulos do padrão.

View

Esta é a camada de visão que é responsável pela interação do usuário com o sistema. Através desse módulo é possível que o usuário:

- Defina tipos de dados que servirão como entrada e saída de uma tarefa. Ele pode também cadastrar dados físicos que poderão ser utilizados na construção do *workflow* e também pelo agente otimizador;
- Defina tipos para tarefas;
- Cadastre as tarefas;
- Visualize os parceiros cadastrados;
- Construa o *workflow*, crie partições e as associe aos parceiros que a executarão;

Controller

Esta é a camada de controle que provê ao módulo *view* os canais de acesso ao módulo *model*. Segundo o padrão, não é permitido outro canal de comunicação entre o *view* e o *model* que não pelo *controller*.

Model

Esta é a camada que contém o núcleo do sistema. Nela são definidos o modelo dos objetos e a lógica do sistema. Esta camada foi divida em quatro partes:

Definition manager:

Este é o gerente de definição do sistema, o qual contém os objetos do modelo (*Partition*, *Partner*, *Workflow*, etc). É no *definition manager* também que contém os métodos de cadastro dos dados solicitado pelo usuário através do *view*. Em alguns casos, objetos específicos para transferência de informação foram definidos nesse módulo para facilitar a visualização por parte do módulo *view*.

Data manager:

Este é o gerente de dados do sistema que é o responsável por todos os acessos a base de dados. Este módulo foi projetado seguindo o padrão de projeto DAO (*Data Access Object*) (J2EE Patterns, 2001). De uma forma geral, esse padrão permite o acesso aos dados através de uma interface evitando que o sistema fique dependente de um SGBD específico. Com esse padrão é possível trocar de banco de dados facilmente, sem impactar no restante do sistema.

Optmization manager:

O *optimization manager*, ou gerente de otimização, é o responsável pela otimização do *workflow*. A otimização do *workflow* é realizada com a ação do agente *Optimizer*, definido no capítulo anterior. O gerente de otimização é executado no momento que o usuário solicita a validação do *workflow*. Caso alguma otimização seja encontrada pelo o agente, o usuário é informado antes de iniciar a execução. O usuário que deve tomar a decisão de fazer ou não a otimização do *workflow* que foi sugerida pelo sistema.

Esta parte do sistema foi implementada seguindo o padrão *Template Method* (Gamma, 1995) para fornecer ao agente *optimizer* todas as otimizações possíveis. O agente *optimizer* possui sua otimização padrão, a que simplifica a estrutura do *workflow*, e novas otimizações podem ser adicionadas no momento que se estende a classe.

Execution manager:

O execution manager é o responsável pela gerência da execução do *workflow*. Ele é auxiliado pelo agente de mesmo nome, *execution manager*. Além do controle da execução, esse módulo também é responsável pelo controle de falhas que venha a ocorrer durante a execução.

A seguir serão mostrados os diagramas do módulo servidor mais relevantes. Quando necessário, eles serão mostrados de forma resumida ou omitindo alguns dados para facilitar a visualização e o entendimento.

Diagrama de pacotes

Os pacotes do módulo servidor estão estruturados da seguinte forma (Figura 26):

- MobiWfMS.controller: contém os componentes que fazem a interface para o servidor *Web* provendo o acesso à aplicação;

- MobiWfMS.model: contém classes de definições do modelo;

- MobiWfMS.model.data: contém as classes de acesso ao banco de dados. É neste pacote que está implementado o DAO.

- MobiWfMS.model.execution: contém as classes de auxilio a execução e o agente *execution manager*.

- MobiWfMS.model.optimization: contém as classes responsáveis pela otimização do *workflow*. O agente optimizer também se encontra neste pacote;

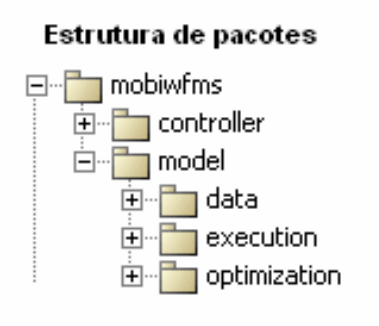

Figura 26 – Estrutura de pacotes do módulos servidor

Diagrama de Classes

Controller

O diagrama de classes do módulo *controller* está representado nas figuras Figura 27 e Figura 28. Ele foi dividido em duas partes para facilitar a visualização. As classes apresentada nas figuras que estendem a classe *AbstractCommand* são os possíveis comandos que o módulo *view* pode invocar.

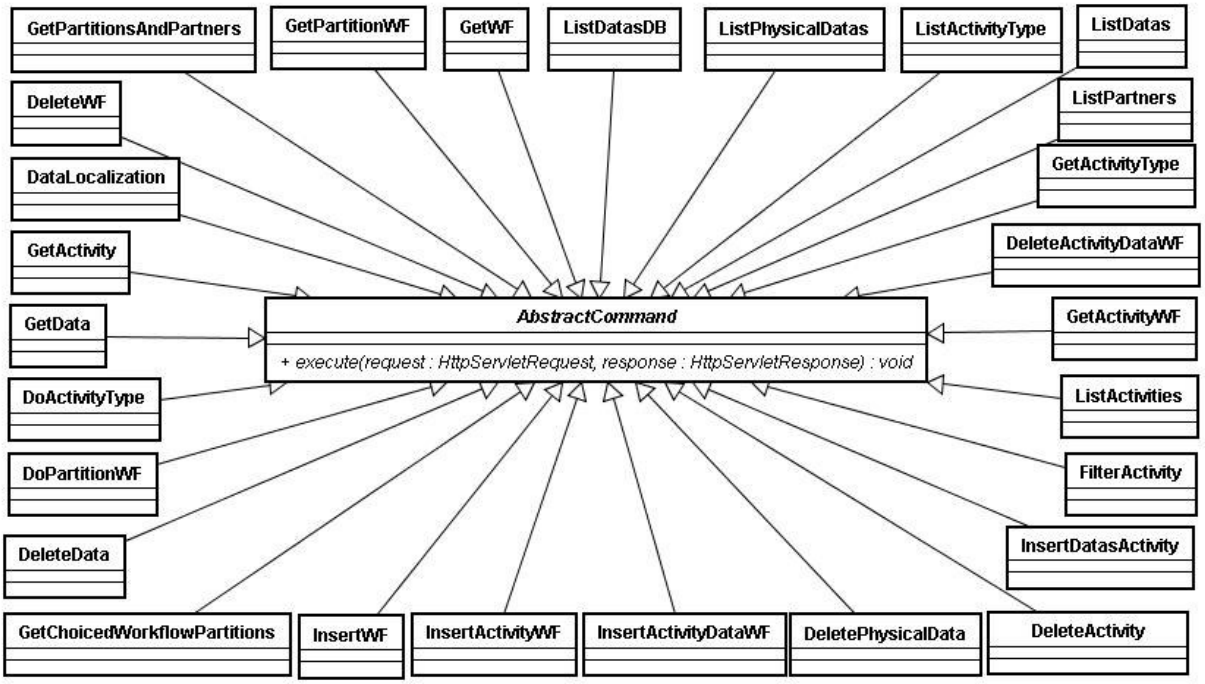

Figura 27 – Diagrama de classes: controller – parte 1

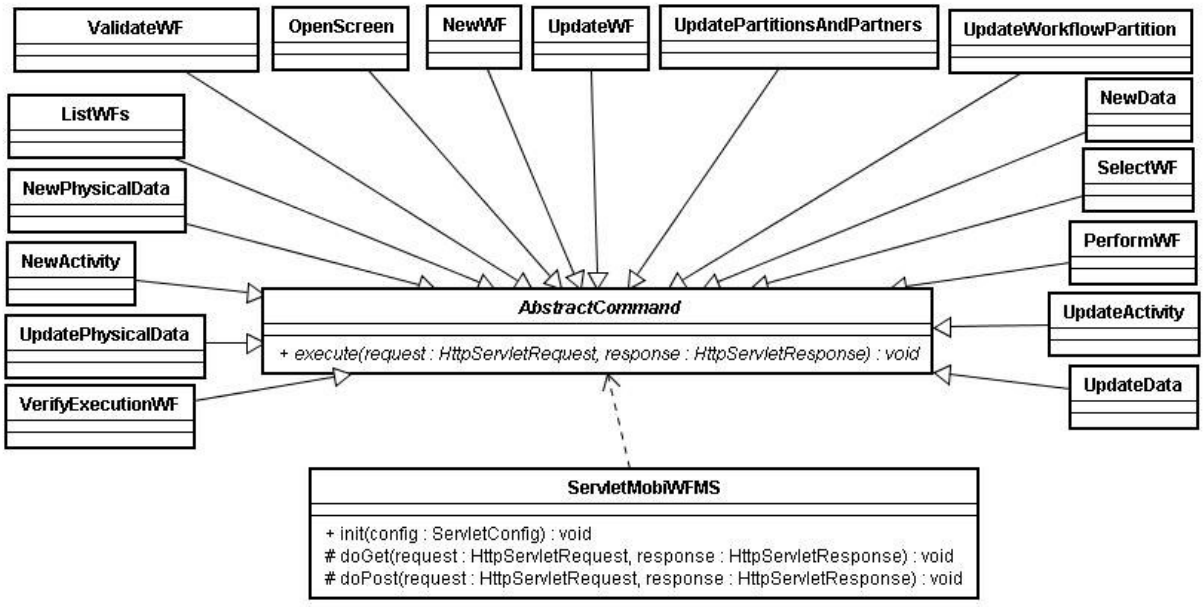

Figura 28 – Diagrama de classes: controller – parte 2

A classe ServletMobiWfMS é a classe que implementa uma interface para o servidor *Web*. Todas as requisições efetuadas pelo módulo *view* são processadas por esta classe. A partir da requisição do módulo *view*, ela instancia um dos possíveis comandos disponíveis e o executa. A classe AbstractCommand é o comando abstrato que possui o método abstrato *execute(request: HttpServletRequest, response: HttpServletRespons): void*. Todos os comandos estendem essa classe e implementam esse método, o qual foi omitido do diagrama.

Model

O diagrama de classes do módulo *model* será mostrado aqui de uma forma diferente da especificada pela UML - *Unified Modeling Language* (UML, 2007). Isto será feito para facilitar a visualização dos diagramas. Métodos e atributos das classes serão omitidos sempre que necessário, e algumas classes não relevantes aqui neste documento também serão omitidos. A Figura 29 traz uma visão geral do módulo *model*, seus principais pacotes internos, suas principais classes e a relação entre eles.

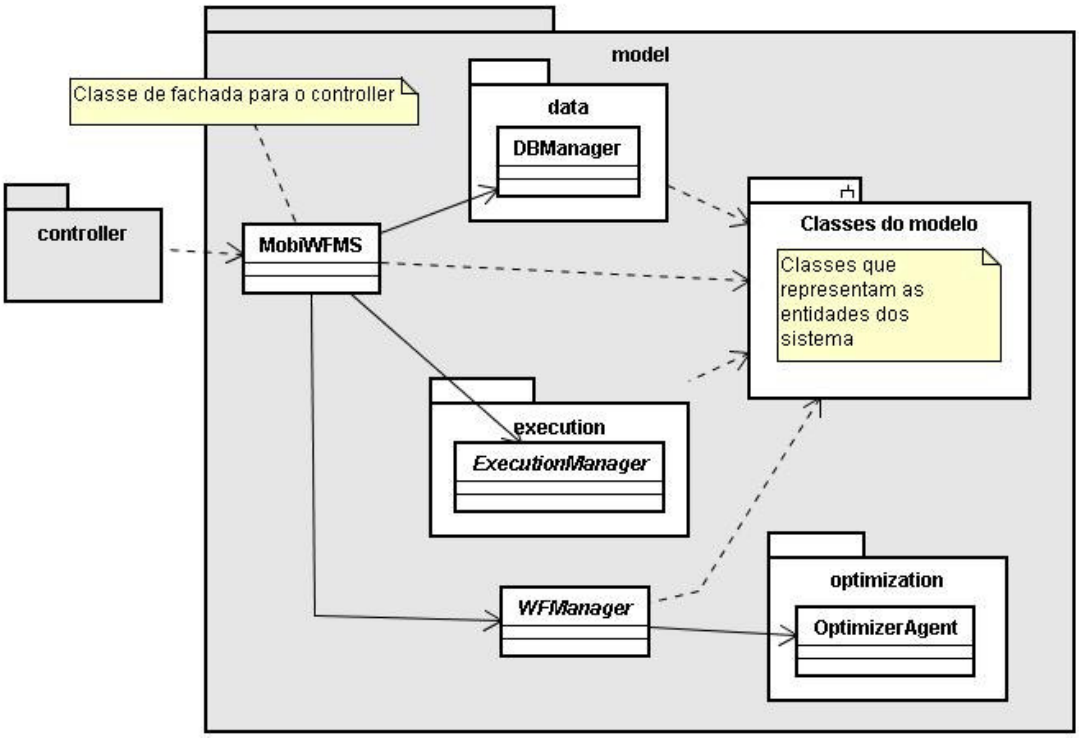

Figura 29 – Diagrama de classes simplificado do Model

Os principais componentes são descritos a seguir:

Classe MobiWfMS: classe de fachada para o pacote *controller*. Todos os comandos pertencentes ao controlador fazem acesso ao sistema através desta classe. Esta classe possui relação com todas as partes do sistema.

Classe DBManager (pacote data): esta é uma classe de fachada para os acessos ao banco de dados. A classe *MobiWfMS* acessa o banco através dela. A Figura 30 mostra um maior detalhamento do pacote data.

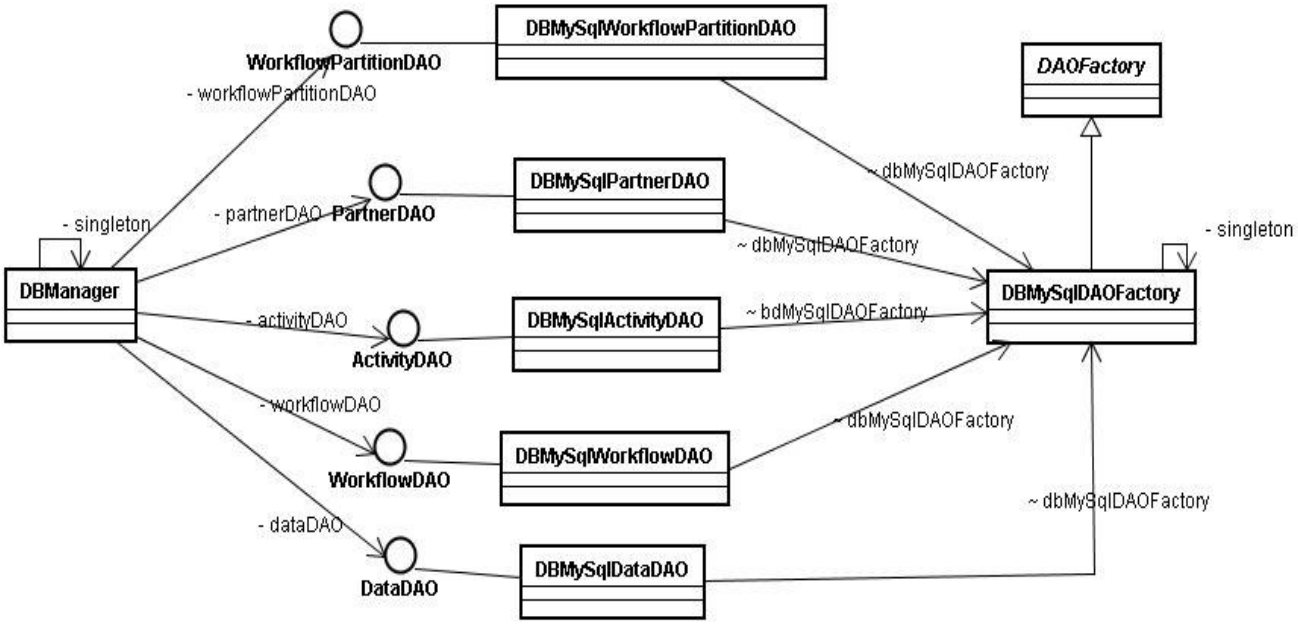

Figura 30 – Diagrama de classes simplificado do componente de dados

Os métodos e atributos foram novamente omitidos do diagrama. Para a implementação do componente de dados foi utilizado o padrão de projeto DAO-Factory que é a união de dois padrões:

O DAO, padrão da tecnologia JEE - *Java Enterpriser Edition* (Java, 2007), é usado para separar a camada de dados da camada de negócio através de uma interface que especifica um contrato de construção das classes que implementam o padrão. Ele está representado pelas interfaces *WorkflowPartitionDAO*, *PartnerDAO*, *ActivityDAO*, *WorkflowDAO*, *DataDAO*. Cada uma dessas interfaces contém os métodos necessários para o completo funcionamento do sistema. Para trocar de banco de dados no sistema, basta implementar cada uma dessas interfaces. Neste sistema, foi utilizado o sistema de gerência de banco de dados MySql e, para este banco, as implementações das interfaces são, respectivamente as classes, *DBMySqlWorkflowPartitionDAO*, *DBMySqlPartnerDAO*, *DBMySqlActivityDAO*, *DBMySqlWorkflowDAO*, *DBMySqlDataDAO*;

O Factory é um padrão usado para permitir a construção seguindo o modelo especificado.

Com a utilização desses padrões a troca de banco poderá ser feita de forma transparente e confiável.

Classes do Modelo: as classes do modelo são todas as classes que representam as entidades do sistema. São elas que compõem a parte do sistema denominada de *Definition Manager*, descrito anteriormente. Na Figura 31 são apresentadas algumas delas sem os métodos, ao passo que outras foram omitidas. As classes apresentadas estão associadas a definição do *workflow*.

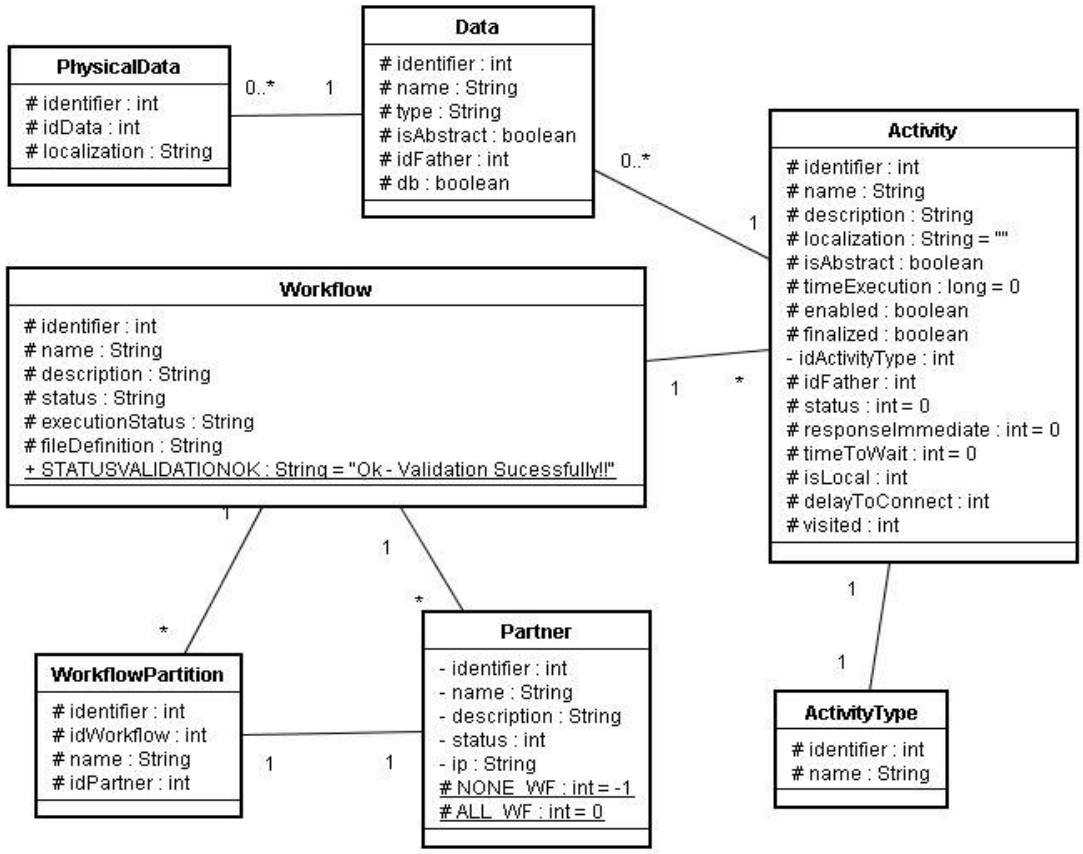

Figura 31 – Diagrama de classes simplificado das entidades

Como pode ser observado um *workflow* (classe *Workflow*) pode possuir várias partições (classe *WorkflowPartition*), vários parceiros (classe *Partner*) e tarefas (classe *Activity*). A tarefa pode conter dados (classe *Data*) de entrada e saída, possuir um tipo (classe *ActivityType*). Por fim, um dado pode ter um ou mais dados físicos associados (classe *PhysicalData*).

Classe *ExecutionManager* (pacote *execution*): esta classe representa o gerente de execução e a ação do agente *ExecutionManager*. O controle da execução e principalmente os mecanismos de controle à falhas são implementados nesse pacote.

Classe *WFManager*: esta é uma das principais classes do sistema (Figura 32). Nesta classe são encontrados métodos de validação, construção da linguagem de definição e otimização do *workflow*. A classe *WFManager* possui uma referência para a classe abstrata *WFDefinition* que possui a estrutura de definição exigida para a definição de uma linguagem de *workflow*. Neste sistema esta classe foi estendida com a classe concreta *MobiWfMS_XML* que contém as definições da linguagem de *workflow* utilizada neste trabalho. A classe *WFManager* possui uma referência para o agente *OptimizerAgent* que é responsável pelo otimização do *workflow*.

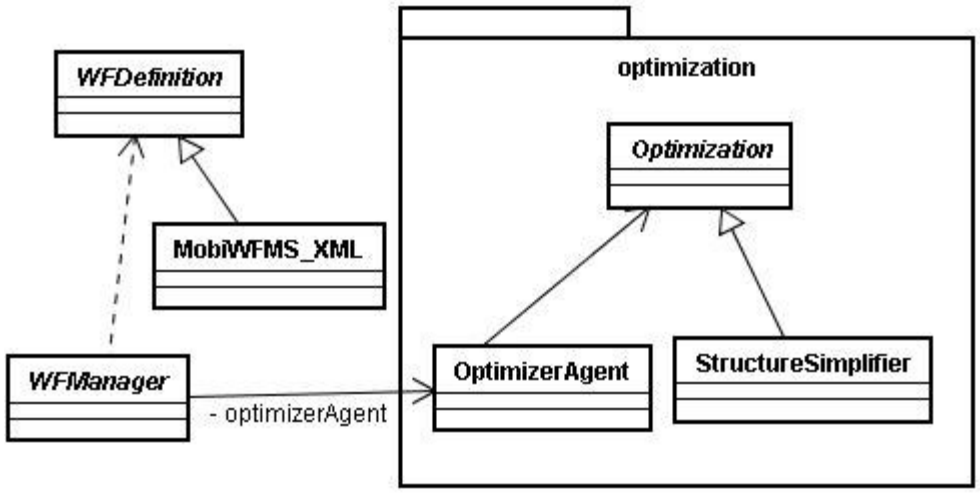

Figura 32 – Diagrama de classe simplificado do Gerente de *workflow*

Classe *OptimizerAgent* (pacote optimization): esta classe pertence ao pacote *optimization* que contém as classes de otimização do *workflow* (Figura 32). A classe abstrata *Optimization* possui a estrutura de métodos que serão utilizados pelo *WFManager*. Neste sistema, foi implementada a S*tructureSimplifier* que é responsável pela análise e simplificação da estrutura do *workflow*. Neste ponto do sistema, mais especificamente em *WFManager*, foi implementado o padrão de projeto *Template Method* (Gamma, 1995). Com este padrão é possível fornecer facilmente ao agente *Optimizer* a otimização existente ou que vierem a existir futuramente.

Diagrama entidade-relacionamento

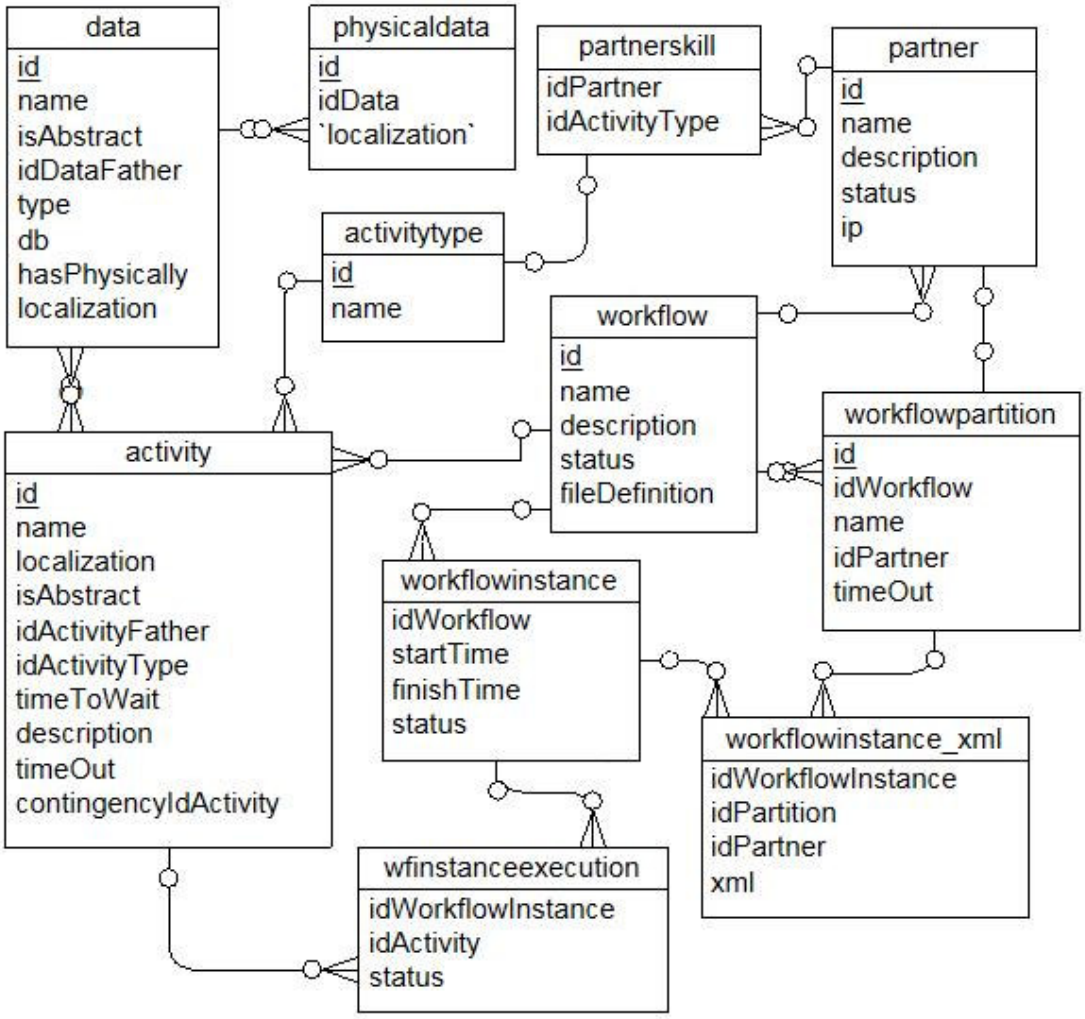

Figura 33 – Diagrama entidade-relacionamento do módulo servidor

A Figura 33 apresenta o modelo de dados para o módulo servidor, em que são mostradas as principais tabelas e seus relacionamentos. Elas são detalhadas resumidamente a seguir:

A tabela *data* contém os tipos de dados que servem de entrada e saída para as tarefas;

A tabela *physicaldata* armazena os dados concretos existentes na rede. Estes dados podem ser encontrados pelo agente *Find Resource* ou pode ser cadastrado diretamente pelo usuário. Um dado concreto deve estar associado a um tipo de dado contido na tabela *data*;

A tabela *activity* contém as tarefas cadastradas no sistema;

Cada tarefa deve ser classificada segundo um tipo, o que é armazenado na tabela *activitytype*;

A tabela *workflow* contém as definições dos *workflows* do sistema. Um *workflow* é composto de uma ou mais tarefas. Ele possui diversos parceiros associados (tabela *partner*) e ainda pode ser dividido em varias partições (tabela *workflowpartition*);

A tabela *workflowpartition* armazena as partições dos *workflows*. Ela está associada a um parceiro (tabela *partner*) que irá executá-la.

A tabela *partner* contém os parceiros que participam do processo. As habilidades de cada parceiro, que são os tipos de tarefas que eles podem executar, são armazenadas em *partnerskill*;

As tabelas *workflowinstance* e *wfinstanceexecution* contêm as informações de execução de uma instância do *workflow*;

A tabela *workflowinstance_xml* guarda os XML das partições gerados quando se inicia a execução de uma instância do *workflow*.

6.1.2. Módulo cliente

O módulo do cliente foi projetado visando principalmente a eficiência do código. A estruturação deste módulo está representada na Figura 34.

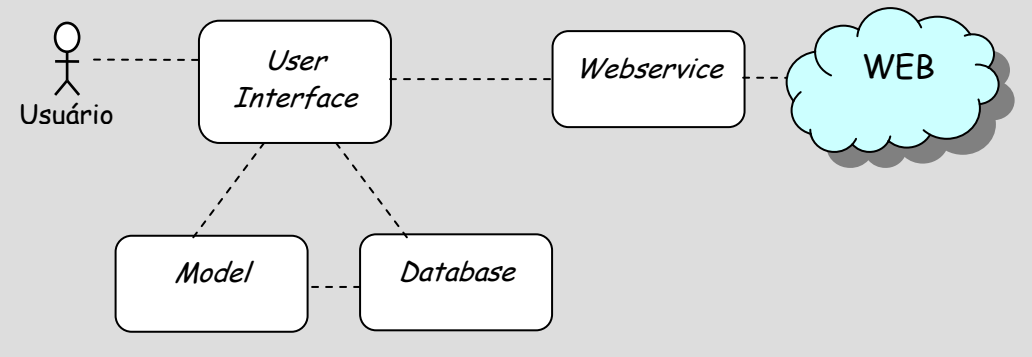

Figura 34 – Módulo cliente

O sistema foi dividido em quatro componentes, descritos a seguir. A Figura 35 apresenta as classes que compõem cada um desses componentes.

User Interface

Este componente é o responsável pela interface do usuário. Todas as telas de entrada de dados são implementadas neste local. São elas:

- *BaseMenu*, *MainMenu* e *MobiWDMSPda*: classes que definem a tela inicial e os parâmetros globais do sistema;
- *DoWorkflow* e *ResponseActivity:* classes que implementam a interface para a execução do *workflow* por parte do usuário;
- *ActivityTypes*: classe que implementa os tipos de atividade que o dispositivo pode executar;
- *RegistrerScreen* e *BecomePartner*: classes responsáveis pelo cadastro do dispositivo junto ao servidor como parceiro.

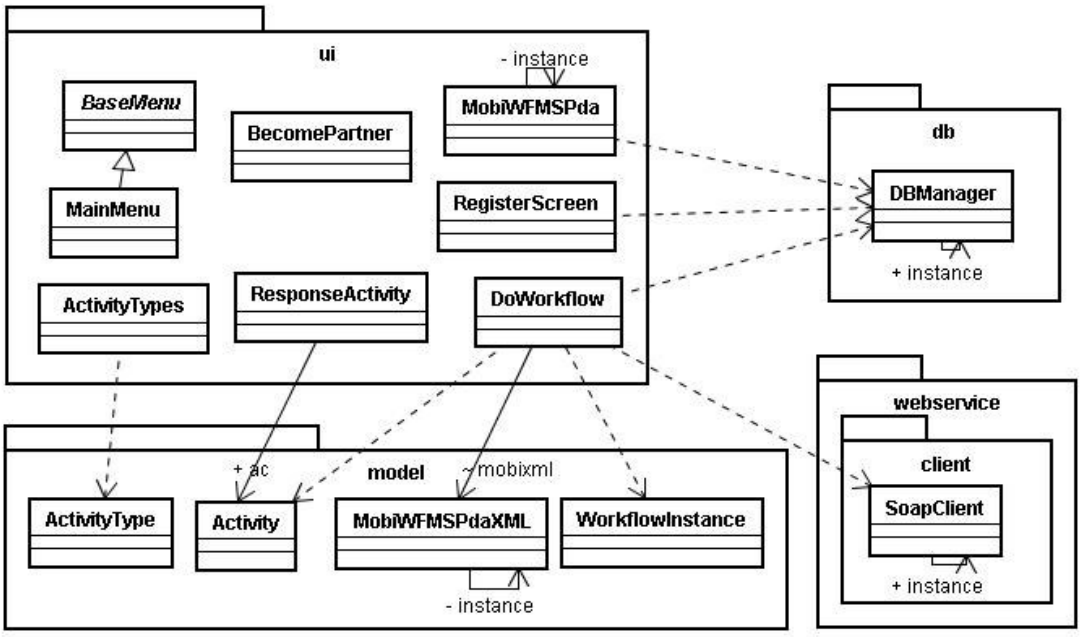

Figura 35 – Diagrama de classes simplificado do módulo cliente

Model

O *model* contém as classes que representam as entidades do sistema. São elas:

- *Activity*: representa as tarefas;
- *WorkflowInstance*: representa uma instância do *workflow* que está sendo executado nesse cliente;
- *MobiWfMSPdaXML*: classe que interpreta a definição do *workflow* vindo do servidor;
- *ActivityType*: classe de definição de tipos de tarefas. Está classe foi criada apenas para facilitar a transferência de dados dentro do sistema.

Database

O *database* contém a classe *DBManager*. Esta é a classe que gerencia todo o acesso ao banco de dados do dispositivo.

Webservice

O *webservice* contém a classe *SoapClient*, que faz as requisições aos serviços Web disponíveis. É através dela que o cliente transmite e recupera dados do servidor.

Diagrama de pacotes

Os pacotes do módulo cliente estão estruturados da seguinte forma (Figura 36):

- MobiWfMSpda.db: contém as classes de acesso ao banco de dados;

- MobiWfMSpda.model: contém classes de definições do modelo;

- MobiWfMS.ui: contém as classes que definem a interface com o usuário;

- MobiWfMS.webservice.client: contém as classes de acesso as serviços Web disponíveis pelo módulo de integração;

Figura 36 – Estrutura de pacotes do módulos cliente

Diagrama entidade-relacionamento

A Figura 37 apresenta o modelo de dados para o módulo cliente.

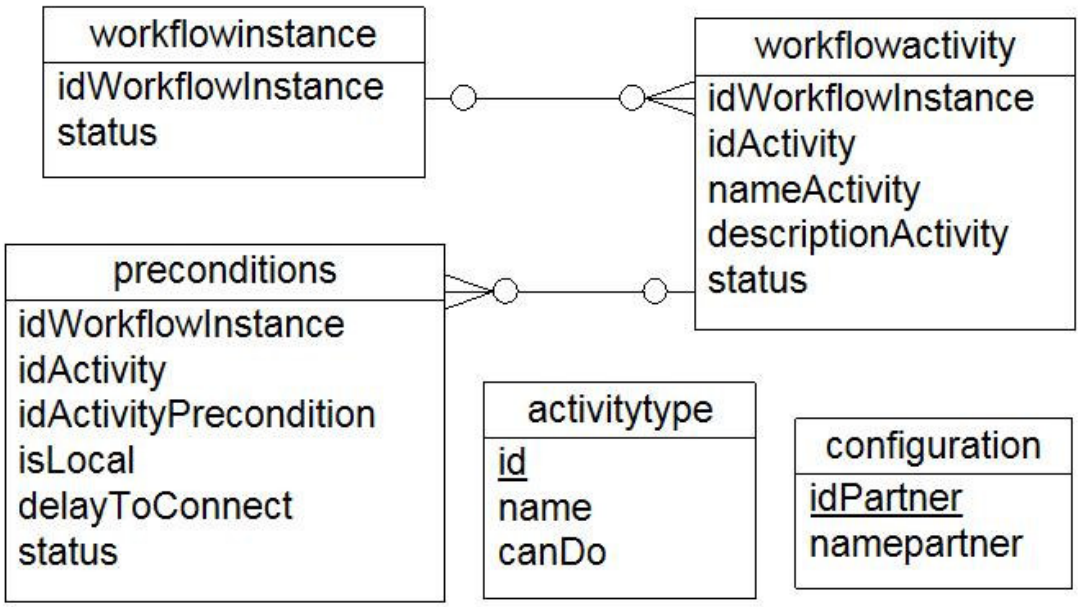

Figura 37 – Diagrama entidade-relacionamento do módulo cliente

As tabelas apresentadas na Figura 37 são detalhadas sucintamente a seguir:

A tabela *configuration* guarda as informações de cadastro do parceiro junto ao servidor;

A tabela *activitytype* armazena os tipos de tarefas que contém no servidor, além de guardar quais tipos o parceiro pode executar;

A tabela *workflowintance* contém a informação das instâncias de execução do *workflow*. Uma instância de *workflow* pode conter varias tarefas (tabela *workflowacitivity*). As tarefas podem conter precondições (tabela *preconditions*) para serem executadas.

A tabela *workflowacitivity* guarda as tarefas que serão executadas pelo parceiro;

Na tabela *preconditions* são armazenadas as informações de precondições das tarefas.

6.1.3.
Módulo de integração

O módulo de integração é o responsável pela comunicação entre o servidor e os clientes (Figura 38). Este módulo possui basicamente os serviços Web (*webservices*) que permitem a troca de informação. Ele acessa a camada *model* do servidor e processa as informações requisitadas pelo cliente através do componente *webservice* do módulo cliente. A classe que implementa o módulo de Integração é a *MobiWfMSWS*.

Figura 38 – Módulo de Integração

Descrição dos serviços Web

A seguir são descritos todos os serviços desenvolvidos nesse módulo.

String[] wsListActivityTypes(): este serviço retorna um vetor de string com a lista de tipos de tarefas (Nomes) e seus respectivos identificadores (Id). Os índices impares do vetor contém os Ids e os pares os respectivos nomes;

boolean wsSetPartnerSkill(int idPartner, int []activityTypes): através deste serviço é possível que o cliente informe ao servidor quais os tipos de atividades que ele pode desenvolver. Se a operação foi completada com sucesso é retornado "true", e "false" caso contrário;

int wsRegisterPartner(String name, String description): este serviço registra o parceiro pela primeira vez. O identificador do parceiro no servidor é retornado ou -1 caso ocorra algum erro;

String wsGetXMLActivities(int idPartner): através deste serviço é possível que o cliente receba do servidor o documento em xml, especificado na linguagem do MobiWfMS com as informações da partição do *workflow* que ele terá que executar;

int[] wsGetStatusFromActivities(int idWorkflowInstance, int[] activitiesCode): este serviço retorna um vetor de inteiros com a lista de tarefas da instância do *workflow* em execução e seus respectivos status. Os índices impares do vetor contém os Ids das tarefas e os pares os respectivos status;

boolean wsUpdateStatusWorkflowActivity(int idWorkflowInstance, int idActivity, int status): através deste serviço é possível que o cliente informe ao servidor o status da execução de uma determinada tarefa. Se ela foi concluída com êxito ou não.

6.2. Interface do MobiWfMS

Nesta seção serão mostradas algumas telas do MobiWfMS, tanto a aplicação servidor (Server) quanto a aplicação cliente (Client), que ilustram a interface do usuário implementada

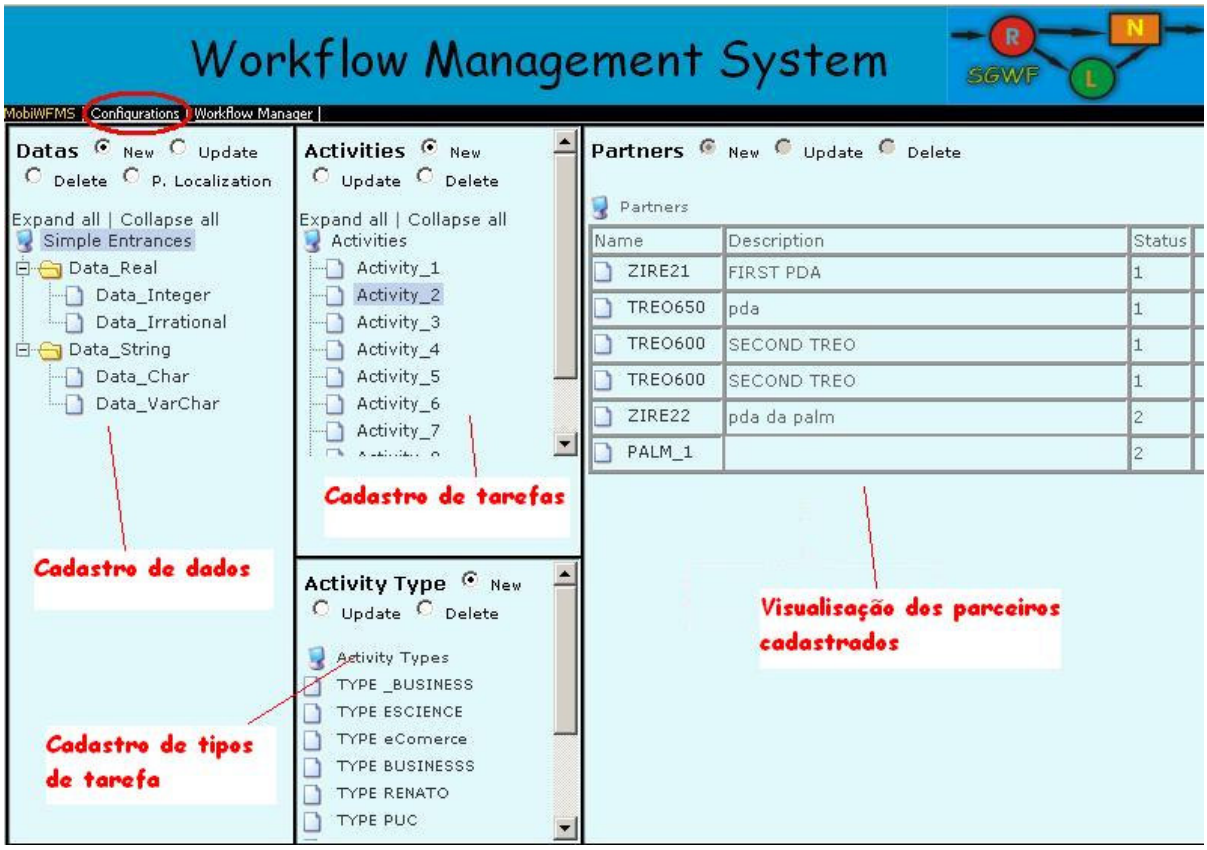

Figura 39 – MobiWfMS Server – Configurações

A Figura 39 apresenta a tela inicial do MobiWfMS Server. Nesta tela é possível visualizar diversas opções de configuração do sistema. Podem ser feitos cadastro dos dados de entrada e saída para a tarefa, cadastro de tarefas, tipos de tarefas e visualizar os parceiros cadastrados no sistema.

Todos os itens de configuração têm opções como cadastrar um novo (*new*), atualizar (*update*) e apagar(*delete*). No item de dados (*Data*), que cadastra as classes de dados, observe que existe uma opção "*P. Localization"*, que permite ao usuário cadastrar dados físicos para uma determinada classe de dado. Esses dados físicos são utilizados para uma possível otimização do *workflow*.

A Figura 40 representa a tela de cadastro de tarefas. Nesta tela o usuário informa o Nome (*Name*) e a descrição (*Description*) da tarefa, o tipo (*Type*), especificar um tempo de espera, o *timeToWait* definido anteriormente, caso esta seja uma atividade com *triggers* de emergência; especificar a contingência (*Contingency*) caso exista, definir o *timeOut*, bem como os dados de entrada e saída da tarefa.

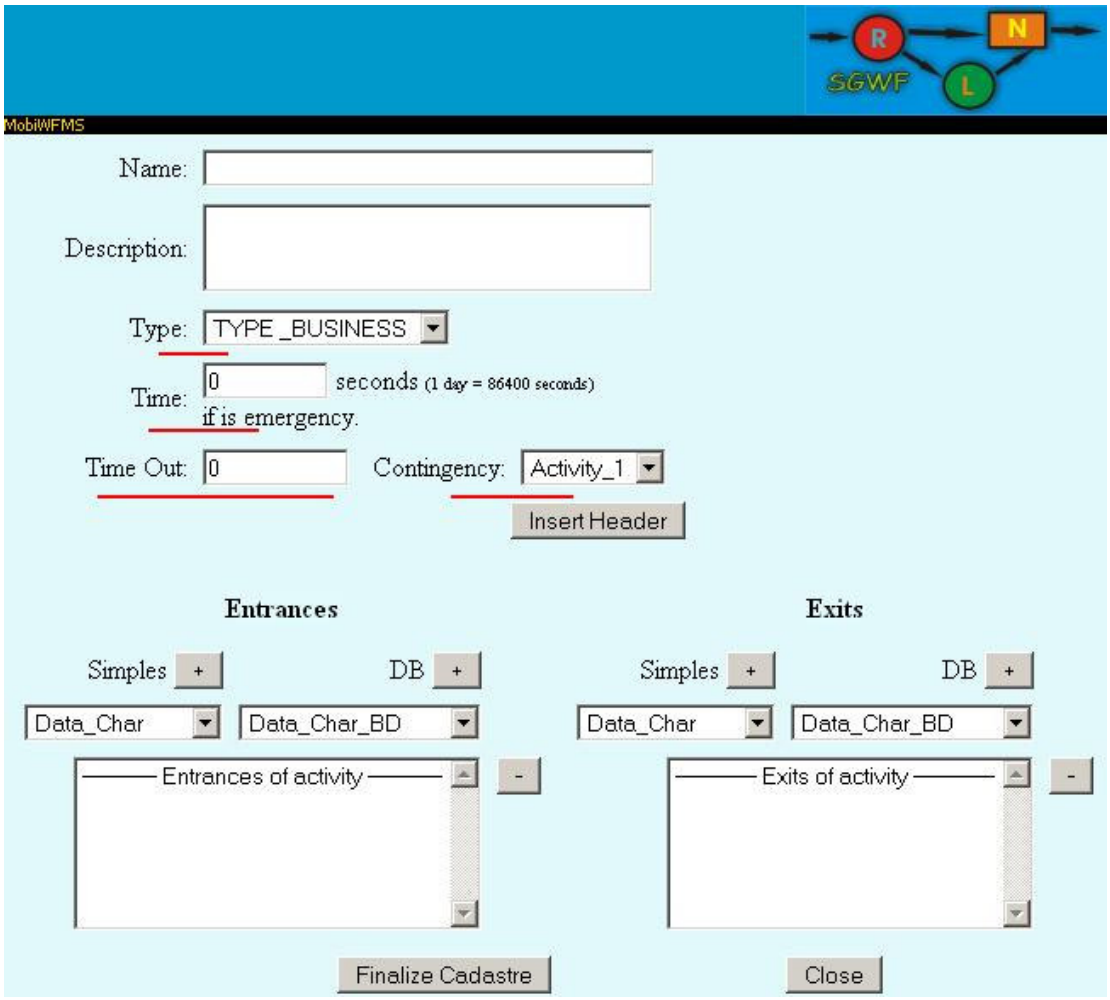

Figura 40 – MobiWfMS Server – Cadastro de tarefas

Feito todos os cadastro necessários o usuário pode prosseguir para a tela de gerência do *workflow* (Figura 41). Nesta tela o usuário tem opções de criar ou abrir um *workflow* existente, inserir tarefas no *workflow*, definir partições e associá-las aos parceiros, além de validar e iniciar a execução do *workflow*.

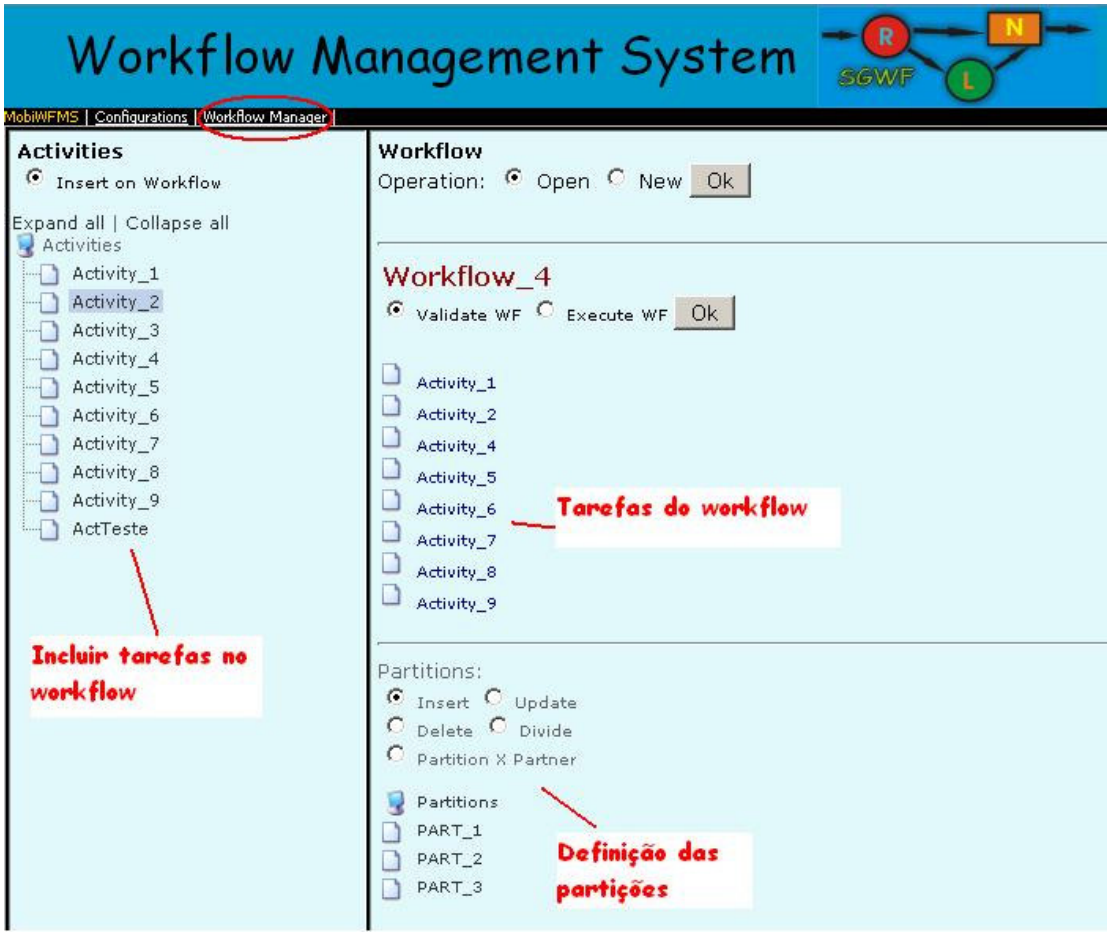

Figura 41 – MobiWfMS Server – Gerência do *workflow*

A Figura 42 apresenta a tela que permite ao usuário inserir uma tarefa no *workflow*. Na figura em questão o usuário está inserindo a tarefa "*Activity_4*" no *workflow* "*4 – Workflow_4*". Ele deve definir as entradas e saídas, sendo que uma entrada pode ser uma saída produzida por outra tarefa. Isto criará uma dependência entre estas tarefas. O usuário escolhe um nome para a saída que poderá ser utilizada por outra tarefa.

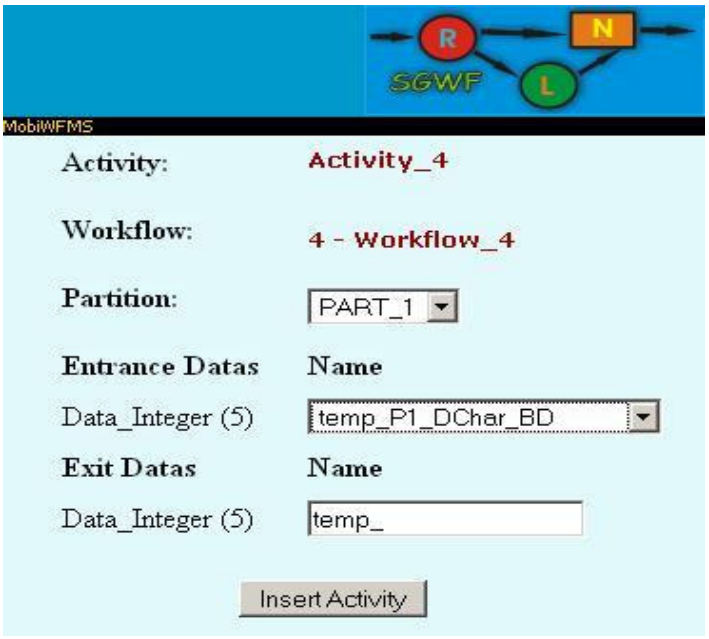

Figura 42 – MobiWfMS Server – Inserir tarefa no *workflow*

A Figura 43 mostra a tela que faz a associação entre partições e parceiros.

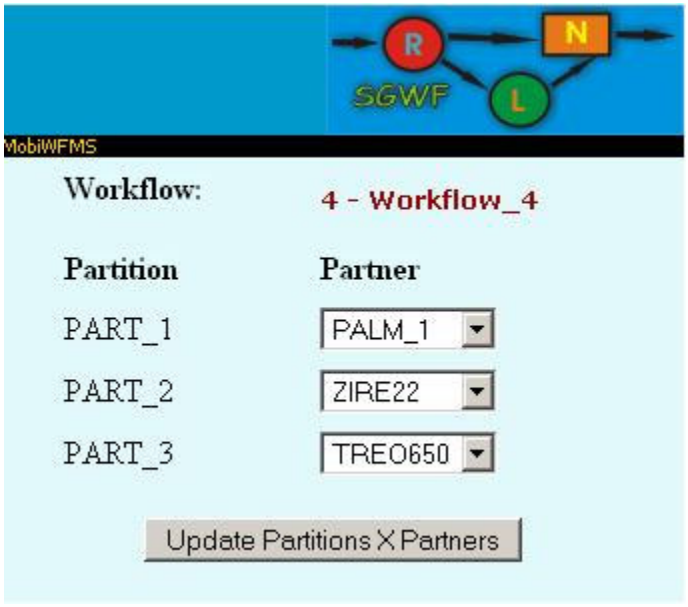

Figura 43 – MobiWfMS Server – Associando partições com parceiros

Após toda a construção do *workflow*, o usuário deve validar e iniciar a execução.

As telas seguintes estão relacionadas à aplicação cliente, o MobiWfMS Client. A Figura 44 apresenta a tela inicial do sistema. Nesta tela é possível definir os tipos de tarefa e iniciar a execução do *workflow*.

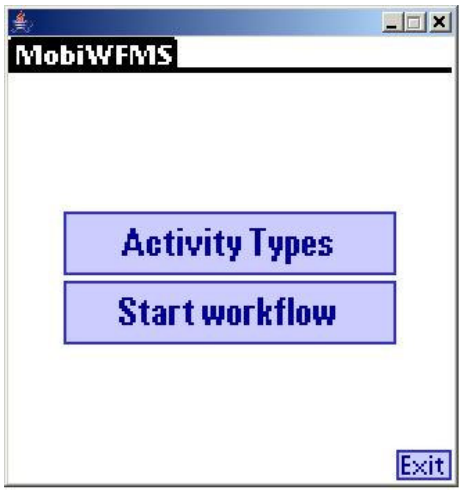

Figura 44 – MobiWfMS Client – Tela inicial

A tela de definição dos tipos de tarefas que podem ser feitos pelo dispositivo é apresentada na Figura 45. O usuário pode recuperar a lista atualizada do servidor "Get from Server", selecionar os tipos desejados, e através da opção "Update" atualizar a base local e o servidor.

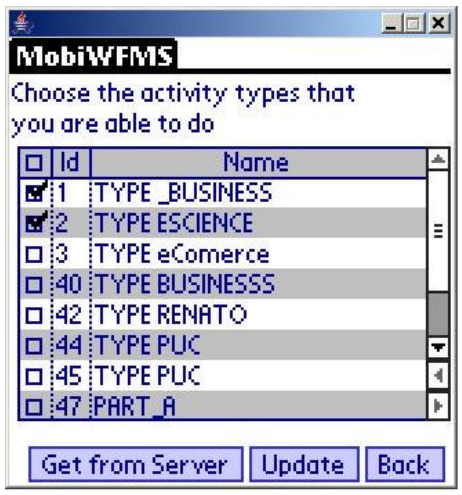

Figura 45 – MobiWfMS Client – Definir os tipos de tarefas

Na Figura 46 é apresentada as tarefas do *workflow*. Nesta tela é possível selecionar uma tarefa para executar.

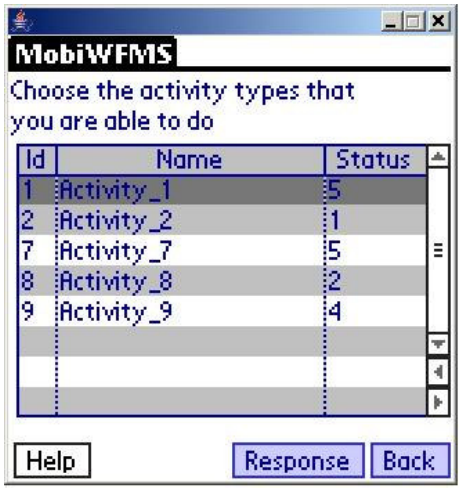

Figura 46 – MobiWfMS Client – Visualizando as tarefas do *workflow*

A coluna status está relacionada à situação das tarefas em relação as suas pré-condições. Os possíveis valores são:

- 0 – Finalizada, já executada;
- 1 – Pode ser executada, todas pré-condições foram satisfeitas;
- 2 – Possui pelo menos uma pré-condição local, que pertence a mesma partição;
- 3 – Possui pelo menos uma pré-condição remota, que pertence a outra partição. Deve contactar o servidor;
- 4 – Possui pelo menos uma pré-condição local e pelo menos uma remota;
- 5 – Tarefa de emergência com o tempo de espera espirado. As pré-condições foram quebradas.

Por fim, a Figura 47 apresenta a tela de execução da tarefa, onde o usuário visualiza o nome, a descrição da tarefa e finaliza a mesma dizendo se foi finalizada com sucesso ou não.

Figura 47 – MobiWfMS Client – Respondendo uma determinada tarefa

6.3. Tecnologias utilizadas

O intuito dessa seção é apresentar ao leitor todas as tecnologias utilizadas para o desenvolvimento do sistema, de forma que ele possa instanciar toda a aplicação fazendo uso das tecnologias aqui descritas. A Figura 48 apresenta uma visão geral do MobiWfMS e as tecnologias utilizadas em cada um dos pontos do sistema.

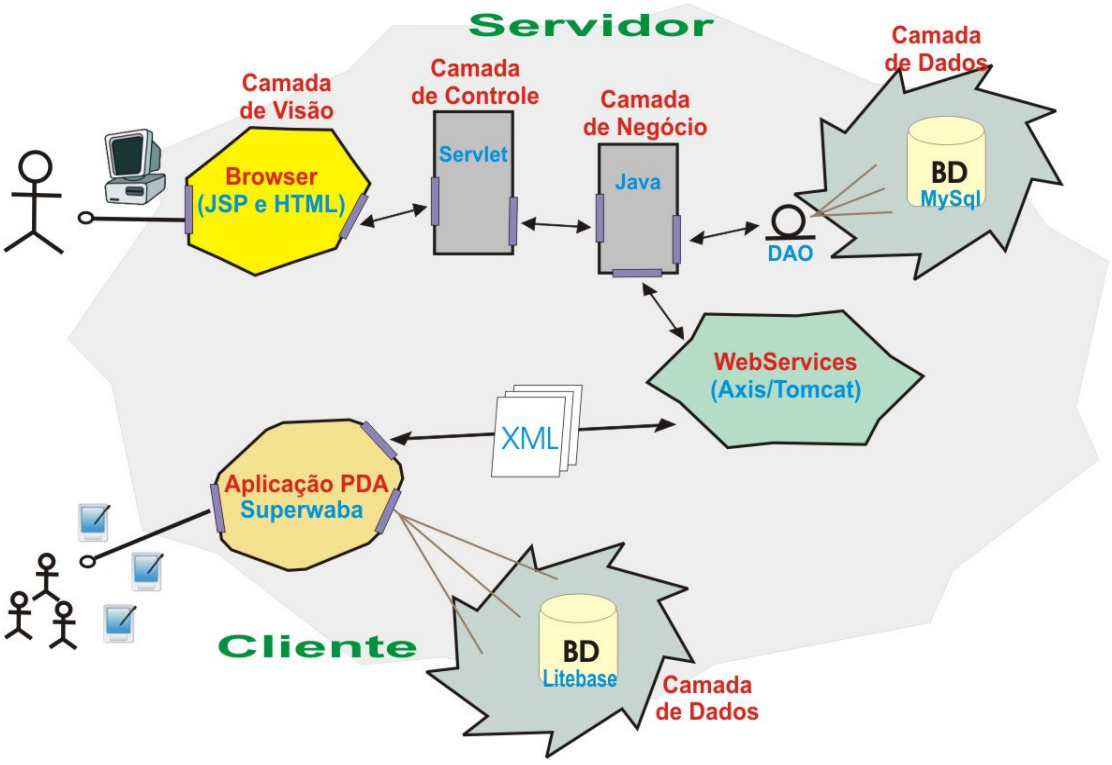

Figura 48 – MobiWfMS: Visão geral e tecnologias utilizadas

O servidor foi desenvolvido utilizando tecnologia JEE (Java, 2007). Este servidor roda sobre o *container* web APACHE TOMCAT 5.5.9 (Tomcat, 2007). O Tomcat é um servidor de aplicações web que permite o desenvolvimento utilizando a linguagem Java.

As camadas de visão, controle e negócio foram desenvolvidos utilizando a plataforma Java. Foi utilizado HTML e JSPs para a camada de visão, componentes servlets, na camada de controle e JSE - *Java Standard Edition* (Java, 2007) na camada de negócio.

Foi visto anteriormente que a camada de dados foi projetada de tal forma a permitir uma maior independência do banco de dados. Para o desenvolvimento desse trabalho foi utilizado o banco de dados MySQL (MySQL, 2007).

Para a criação de componentes *Webservices*, é necessário que o servidor de aplicação provenha alguma API (*Application Programming Interface* ou Interface de programação de aplicativos) para o desenvolvimento desses serviços. Este trabalho utilizou a AXIS (AXIS, 2007) que é a principal API para desenvolvimento de *webservices* dentro do Tomcat. A AXIS permite que os usuários desenvolvam de forma simples os *webservices*.

Para a transferência de informação entre o servidor e os clientes foi desenvolvida a linguagem de *workflow* do MobiWfMS. Esta linguagem foi definida baseada em XML - eXtensible Markup Language (XML, 2007), que é a linguagem de marcação da W3C. O XML é considerado por muitos como a principal linguagem de marcação existente.

O desenvolvimento da aplicação cliente foi feita utilizando a linguagem Superwaba (Superwaba, 2007). Esta é uma linguagem para desenvolvimento de aplicações portáveis para PDAs e Smartphones. O MobiWfMS tem sua portabilidade atrelada a portabilidade da Superwaba, podendo rodar em diversos dispositivos com sistemas operacionais como PALMOS, WINDOWS CE/POCKET PC, SIMBIAN OS.

O banco de dados utilizado na aplicação cliente foi o Litebase (Litebase, 2007). Este é um banco de dados que vem embutido dentro do Superwaba. Está atualmente na versão 1.0 e caminha para, num futuro próximo, poder suportar todo o padrão SQL, além de permitir o acesso através de outras linguagens para PDAs.

6.4. Conclusão

Este capítulo abordou a implementação do MobiWfMS. Esta implementação tem como objetivo a validação da arquitetura proposta neste trabalho. Através da descrição dos módulos do sistema e dos diagramas apresentados foi possível observar que o MobiWfMS cobre o seu propósito.