Gerador de Aplicações Móveis

Framework GAM

Bruno Henrique Guerreiro Cavaco

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente: Professor Luís Eduardo Teixeira Rodrigues

Orientador: Professor José Alves Marques

Vogais: Engenheiro José Afonso Vasconcelos Pires

Professor Luís Manuel Antunes Veiga

Abril 2008

Agradecimentos

Este trabalho é o culminar de um percurso académico com óptimos momentos para recordar. Recordo-me de muito mas tenho pena de não me recordar de tudo. Foram muitos anos onde fiz muitos amigos, partilhei vitórias e derrotas, felicidades e, felizmente poucas coisas correram “menos bem”.

Sempre me dei com os melhores e isso fez-me crescer para ser como eles, aprendi com eles e eles comigo, hoje sinto-me um deles.

Agradeço a todas as pessoas que colaboraram comigo e me fizeram crescer, professores, amigos, à Margarida e, principalmente, à minha família. Agradecimento muito especial à minha Mãe e ao meu Pai que tornaram tudo isto possível. Obrigado Mãe, obrigado Pai.

Nesta tese em concreto agradeço ao Professor José Alves Marques pela atenção que teve pelo meu trabalho e ao Eng.º José Afonso Pires que me guiou e direccionou para aquilo que realmente era o foco do trabalho.

Obrigado a todos. A partir deste momento a minha vida toma um novo rumo.

Abstract

In the last few years we've been witnessing a significant technological advance to the level of the mobile systems. This is due to the evolution of two factors. On one hand, the features present on mobile devices which have considerably developed its autonomy as well as its capacity of processing. On the other hand, wireless communications are getting faster and more secure every day. Today, the world is more predisposed to contact with mobile environments than it was a few years ago. Nowadays it is common to see people everywhere working with mobile devices, such as computers or PDA, enabled with wireless connections to the Internet. Mobile communications have grown considerably. These factors make organizations consider this mobile environment as an opportunity to make their business more agile. Many of them already offer services of technological basis to their collaborators and even their clients. The work carried out within the context of this master's degree dissertation follows a course final thesis – 2005/2006 – designated "Mobile Application Generator". The work developed went directly in the way of the normalization referred to early on, consisting in the creation of a standard platform, built under the OSGI Framework, to speed up and simplify the process of implementation, distribution and maintenance of applications for mobile environments. Namely, graphical interfaces, communication capacities and connectivity management of the terminal where they are lodged, security of data kept persistently and changed with external entities and data synchronization with a server. Therefore, the OSGI Framework ends up being the basis where the GAM Framework was implemented. Continuing with the developed work, some objectives for this work were drafted in the area of data consistency, such as the improvement of the management of its access or their synchronization between fixed and mobile environment. This dissertation underwent in an initial phase, an investigation of the existing constraints in mobile devices that could affect the consistency and coherence of data, a study of the developed framework and a listing of problems found in the platform made in the previous work. Afterwards, a new architecture of standard data synchronization was designed and implemented under the platform of normalization, and finally, an application was implemented that would display the synchronization mechanisms created.

Key-words: mobile systems, synchronization, OSGI Framework, standard platform

Resumo

Durante os últimos anos tem-se vindo a assistir a um avanço tecnológico significativo ao nível dos sistemas móveis. Este facto tem sido alcançado ano após ano pela evolução drástica de dois factores, por um lado as características dos dispositivos móveis que têm evoluído consideravelmente a sua autonomia, bem como a sua capacidade de processamento e por outro, as comunicações *wireless* estão cada vez mais rápidas e seguras. Estes factores fazem com que as organizações vejam neste ambiente móvel uma oportunidade para agilizar o seu negócio, muitas são já as organizações que oferecem serviços de base tecnológica aos seus colaboradores ou mesmo aos seus clientes. No entanto há a considerar que muitas soluções apresentadas por estas organizações são soluções proprietárias de difícil integração e evolução. Como tal, existe a necessidade de evoluir estas soluções para uma plataforma estandardizada no sentido da normalização. Desta forma mais facilmente se podem integrar com outros sistemas capacitando as organizações com maior poder de reacção face às vicissitudes do mercado onde se inserem.

Factos importantes a considerar são os constrangimentos inerentes aos dispositivos móveis que impossibilitam que estes terminais se encontrem constantemente ligados à rede. Assim sendo, será necessário capacitar a plataforma de normalização com mecanismos de gestão de dados que possibilitem que estes possam ser acedidos e manuseados quando os dispositivos estão desconectados da rede e sincronizados com o servidor assim que haja conectividade.

O trabalho realizado no contexto desta dissertação de Mestrado encaixa-se no seguimento de um trabalho final de curso do ano passado – 2005/2006 – com o nome “Gerador de Aplicações Móveis” [1]. O objectivo principal desse trabalho foi o desenvolvimento de uma Framework, denominada Framework GAM, sob a Framework OSGI capaz de agilizar o desenvolvimento, gestão e manutenção de aplicações em dispositivos móveis, nomeadamente PDAs [2][3]. Dando continuidade ao trabalho desenvolvido foram traçados alguns objectivos na área da consistência de dados como o melhoramento da gestão do seu acesso ou a sincronização destes entre o ambiente fixo e móvel. A dissertação em causa passou, numa primeira fase, por uma investigação dos constrangimentos existentes nos dispositivos móveis que pudessem afectar a consistência e coerência dos dados, estudo da *framework* desenvolvida e levantamento dos problemas encontrados na plataforma fruto do trabalho anterior. Seguidamente foi desenhada e implementada uma nova arquitectura de sincronização de dados *standard* sob a plataforma de normalização desenvolvida e, por fim, implementada uma aplicação que demonstrasse os mecanismos de sincronização realizados.

Palavras-chave: sistemas móveis, mobilidade, *framework*, OSGI, persistência de dados, *webservices*, replicação, sincronização

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

GAM – Gerador de Aplicações Móveis

OSGi – Open Services Gateway initiative

PDA – Personal Data Assistant

SBDM – Sistema de Base de Dados Móvel

SBDD – Sistema de Base de Dados Distribuído

BD – Base de Dados

1 Introdução

O Mundo está hoje mais predisposto ao contacto com os ambientes móveis do que há alguns anos atrás. Hoje em dia é normal vermos pessoas a trabalhar em todo o lado por meio de dispositivos móveis, como computadores ou PDAs, capacitados de ligações *wireless* capazes de se conectar à Internet. As comunicações móveis têm crescido de forma considerável. Por outro lado, algumas organizações encontram nestes dispositivos vantagens competitivas, outras vêm neles a solução para uma redução de custos de forma considerável ou simplesmente para agilizar os seus processos de negócio. Por estes factores os sistemas móveis têm vindo a ser a solução encontrada para aceder a um número significativo de serviços tanto para actividades pessoais como profissionais. No entanto, há a considerar que muitas das aplicações existentes são proprietárias das organizações que as desenvolvem tendo que criar de base muitos componentes de gestão da conectividade, segurança, sincronização, interfaces, etc. Para além deste facto, estas aplicações são de difícil evolução ou integração com outras soluções encontradas no mercado. Como tal, existe a necessidade de evoluir para uma plataforma de normalização que facilite o desenvolvimento de aplicações e que capacite estas com um conjunto de funcionalidades espectáveis de aplicações actuais e enquadradas em paradigmas recentes de programação.

1.1 Estrutura do relatório

O primeiro capítulo, intitulado “Introdução”, no qual está inserida esta secção, pretende fornecer uma visão geral sobre o problema, analisar o projecto anterior, definir os objectivos do trabalho e referir os aspectos desenvolvidos.

No segundo capitulo, “Conceitos”, são apresentados alguns conceitos subjacentes à problemática do ambiente onde se insere o trabalho.

O terceiro capítulo, “Arquitectura” pretende referir e justificar as opções tomadas a nível tecnológico na arquitectura de testes criada.

O quarto capítulo, “Sincronização”, tem como objectivos explicar as funcionalidades de sincronização desenvolvidas, bem como explicar sumariamente o funcionamento de toda a Framework.

O quinto e sexto capítulo, designados por “Framework GAM” e “Aplicação Testes” respectivamente, dão uma visão total da implementação dos mecanismos de sincronização e suporte criados descritos no capítulo anterior. São a par deste mesmo capítulo o corpo principal deste relatório.

Por fim, no capítulo “Conclusões”, são mencionadas as considerações finais do projecto, bem como as possíveis soluções de melhoramento.

1.2 Enquadramento

Nesta secção pretende-se oferecer uma visão geral do trabalho a ser realizado no contexto da Tese de Mestrado. Esta visão parte de um enquadramento que permite situar a realização da Tese e o problema a ser resolvido. É especificado o contexto da Tese, bem como um exemplo prático subjacente à investigação, que envolve o tema da sincronização de dados.

1.2.1 Enquadramento da Tese de Mestrado

Este trabalho foi efectuado no contexto da Tese de Mestrado da Licenciatura em Engenharia Informática e de Computadores do Instituto Superior Técnico pertencente à Universidade Técnica de Lisboa.

O trabalho foi desenvolvido na empresa Link Consulting SA sob orientação pedagógica do Professor José Alves Marques do Departamento de Engenharia Informática do IST e Presidente do Conselho de Administração da Link Consulting SA e coordenação do Engenheiro José Afonso Pires.

1.2.2 Enquadramento Teórico

De forma a enquadrar o leitor do problema em causa podemos referir muitos exemplos de colaboradores que trabalham através de sistemas móveis e necessitam de dados coerentes e consistentes para poderem executar as suas operações diárias. Os vendedores, por exemplo, capacitados com dispositivos móveis nos quais anotam toda a informação relevante à sua actividade, através das aplicações disponíveis ou interagindo directamente com os sistemas computacionais centrais, querem ver recriado todo o seu ambiente em qualquer local e sob quaisquer circunstâncias de conexão. Situação ideal, assumindo que se encontram completamente conectados à rede, seria poderem tomar acções sob os sistemas centrais da organização, caso contrário, deveriam ser mantidas persistentemente como operações pendentes e, assim que se restabelecesse a ligação o sistema sincronizaria as operações pendentes com os dados do servidor, bem como actualizaria os dados locais guardados no terminal móvel. Desta forma os vendedores poderiam sempre fornecer informação relevante sobre os seus produtos aos clientes ou dar baixa de stock imediatamente, ou assim que houvesse conexão, no sistema central.

1.3 Trabalho Anterior

O trabalho desenvolvido no projecto anterior é exactamente no sentido da normalização referida anteriormente [1], constando na criação de uma plataforma *standard*, construída sob a Framework OSGi [2][3], para agilizar e simplificar o processo de implementação, distribuição e manutenção de aplicações para ambientes móveis. Designadamente, interfaces gráficas, capacidades de comunicação e gestão da conectividade do terminal onde estão alojadas, segurança de dados guardados persistentemente e trocados com entidades externas e sincronização de dados com um servidor. Neste sentido a Framework OSGi acaba por ser a base onde foi implementada a Framework GAM.

Focando-nos nas questões de sincronização de dados, objectivo principal deste projecto, o projecto anterior contempla a sincronização entre duas bases de dados (Db4o) – cliente, servidor. Esta sincronização é feita a pedido do utilizador através de um serviço disponibilizado pela Framework à aplicação. As acções tomadas pelo utilizador na aplicação são guardadas como operações pendentes, caso não exista conectividade, e podem posteriormente ser sincronizadas com uma tabela *HashMap,* desenvolvida para efeitos demonstrativos, através da invocação de serviços definidos como *webservices.* Por "operações" entenda-se todas as transacções iniciadas na aplicação.

A figura seguinte apresenta o ambiente desenvolvido para comunicação com o exterior, assim como algumas tecnologias usadas:

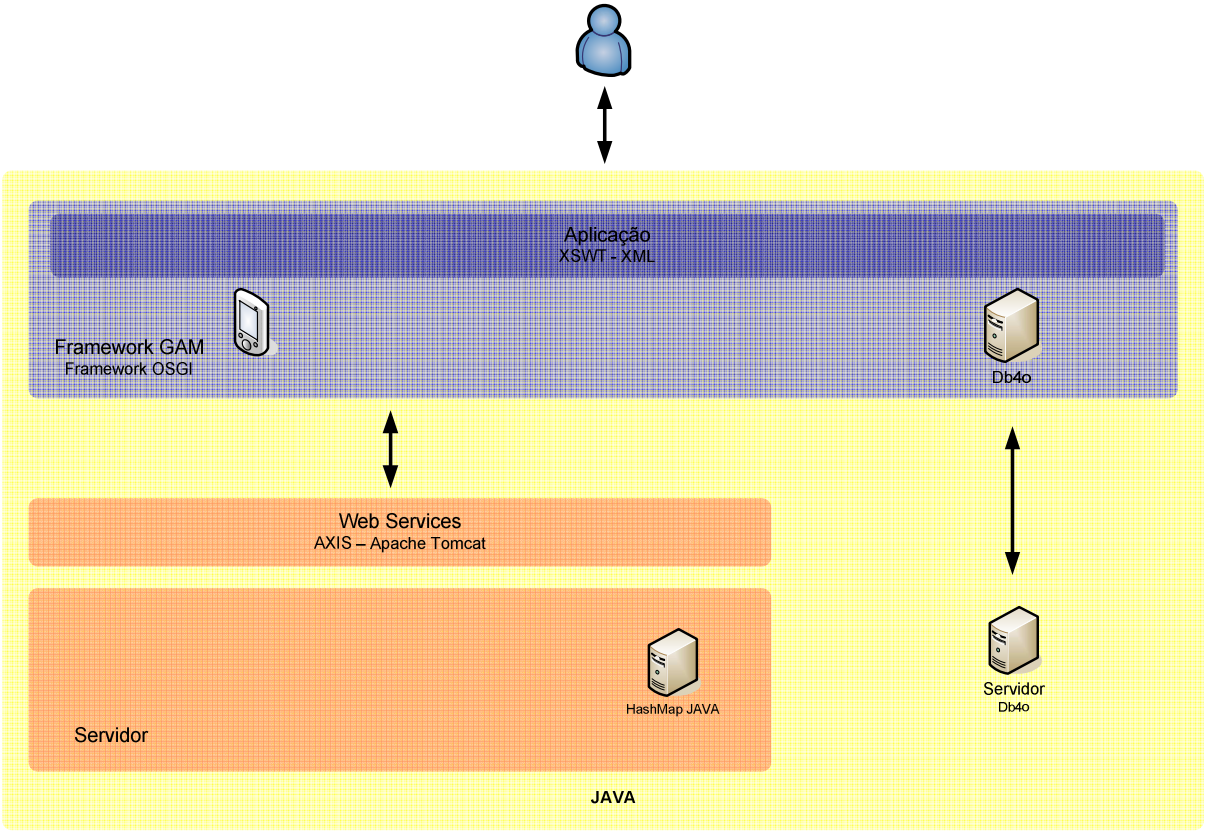

Figura 1-1: Arquitectura para sincronização de dados com o "exterior" (trabalho anterior)

De acordo com esta figura pode ser observado que a Framework GAM gere uma base de dados local por Aplicação. As comunicações com o exterior ocorrem tipicamente de duas formas distintas: por invocação de *webservices* e/ou sincronização da base de dados local com a do servidor. No que diz respeito às tecnologias utiliza bases de dados Db4o [11] e uma *HasMap* Java para a persistência dos dados, linguagem XSWT [5], no âmbito da biblioteca utilizada SWT – "Standard Widget Toolkit" [4], para desenho das interfaces gráficas para as aplicações e Axis do projecto Tomcat para a criação e implementação dos *webservices* [6]. Toda esta arquitectura será detalhadamente explicada na secção "Arquitectura" do capítulo "Sincronização".

1.4 Objectivos

O trabalho fruto desta dissertação tem como objectivo principal capacitar a Framework GAM com uma nova arquitectura de sincronização de dados genérica, ou seja que sirva qualquer tipo de aplicações, nomeadamente garantir a consistência dos dados locais à aplicação e sincronização com o servidor das operações executadas quando desconectado.

Numa primeira fase, foi feito um estudo sobre o que são os sistemas móveis e a sua principal característica de mobilidade. Posteriormente foram analisados os mecanismos de acesso e sincronização de dados entre o sistema móvel e fixo existente na Framework GAM desenvolvida, de forma a servir de modelo para as funcionalidades espectáveis e para as lacunas existentes. Concluído o processo de análise da consistência de dados realizada foi necessário definir e implementar uma nova arquitectura de sincronização que permita aos dispositivos poder continuar a operar em situações de desconexão. Nesse sentido quando a ligação é restabelecida as operações são sincronizadas com o servidor e actualizada automaticamente a base de dados local. Para implementar esta arquitectura foi desenvolvido um ambiente capaz de servir de suporte aos mecanismos anteriormente identificados. Por fim, foi implementada uma aplicação que demonstra a arquitectura em questão.

Definindo sumariamente, os objectivos iniciais passam por:

- Estudo/Levantamento das características dos sistemas móveis;
- Estudo da Framework GAM desenvolvida;
- Análise dos métodos/mecanismos de sincronização de dados usados;
- Definição e implementação de uma nova arquitectura de sincronização ou melhorar a arquitectura actual;
- Definição e implementação de um ambiente que sirva de suporte à arquitectura desenvolvida;
- Definição e implementação de uma aplicação que demonstre os mecanismos de sincronização realizados.

1.5 Trabalho Realizado

Analisada a componente de sincronização desenvolvida foi verificado que não resolvia alguns problemas importantes, tais como:

- Base de dados Db4o definida como *servidor*, no trabalho anterior, acaba por não ter funções de *servidor*, uma vez que opera de forma totalmente independente do resto do sistema. As acções executadas na aplicação não são sincronizadas com esta base de dados. Esta funcionalidade foi desenvolvida para demonstrar a sincronização entre o dispositivo móvel e a Framework com dados que não são criados através deste;
- Base de dados Db4o definida como *servidor* não está de acordo com os sistemas legados encontrados nas organizações. Normalmente nessa posição surgem sistemas de alta escalabilidade (Oracle [8], SQL Server [9], MySQL [10], ...);
- Operações pendentes não são guardadas de forma persistente, ou seja caso o dispositivo se desligue por qualquer motivo as operações perdem-se;
- Não existe um servidor central *fixo* que guarde toda a informação e garanta a sua coerência e consistência para utilização no ambiente móvel;
- Não existem mecanismos automáticos de sincronização da base de dados *móvel* com o servidor;
- Não existe a noção de grupos de operações encadeadas. As operações são vistas independentemente umas das outras, não existindo o conceito de “transacção" como um conjunto de operações.

1.5.1 Requisitos

Identificados os problemas, foram levantados os requisitos da arquitectura de sincronização a ser desenvolvida. Desde logo a arquitectura teria de ser genérica e funcionar com qualquer tipo de aplicação. Teriam que ser mantidos os baixos custos de desenvolvimento no seguimento do trabalho sob a plataforma realizada anteriormente.

No que diz respeito à sincronização propriamente dita era expectável que a arquitectura a desenvolver agisse sob a base de dados local e sob a do servidor, caso o dispositivo se encontre conectado. Esta realizar-se-ia sob duas formas: explicitamente pelo utilizador através da invocação de um serviço ou configurada para sincronizar de “x” em “x” tempo.

No dispositivo móvel, através de uma qualquer aplicação, as acções do utilizador deveriam ser, em todo o caso, reflectidas na base de dados local para que o utilizador possa continuar a executar as suas tarefas mesmo quando desconectado. Nesse caso, as operações em causa seriam mantidas de forma persistente para mais tarde, assim que as ligações estivessem disponíveis, sincronizar com o servidor.

Seria também necessário capacitar a Framework de outros requisitos, tais como, o utilizador ter total controlo sob as operações pendentes. Ao serem formalizadas estas seriam imediatamente reflectidas na base de dados local de forma a poder operar com dados coerentes no modo *offline*. Com estes requisitos e para que se possa considerar um total controlo por parte do utilizador seria necessário considerar, então, três outras funcionalidades: visualização, eliminação e notificação do utilizador quando na presença de erro.

Por fim a Framework teria de ter a capacidade de operar com acções formalizadas em grupo, em vez de simplesmente atómicas. Ou seja, neste caso o utilizador poderia interagir com o sistema formalizando uma série de operações "encadeadas" que só seriam definitivamente executadas caso todas se pudessem realizar. Com este requisito é possível colocar diferentes operações dependentes entre si não comprometendo o sistema caso uma delas não se possa concluir.

Seguidamente são sumarizadas as funcionalidades desenvolvidas, que de encontro com os requisitos apresentados, solucionam os desafios encontrados no trabalho anterior:

- Gestão das bases de dados locais das aplicações;
- Acções da aplicação são reflectidas na base de dados locais e nas do servidor (caso se encontre conectado);
- Quando desconectado, acções iniciadas na aplicação são guardadas de forma persistente como "Operações Pendentes";
- Constante verificação da existência de "Operações Pendentes" por sincronizar com servidor;
- Total controlo das "Operações Pendentes" (visualizar, apagar);
- *Report* de erros encontrados na execução das "Operações Pendentes";
- Possibilidade de criar grupos de operações, designadas "transacções", com características de atomicidade, consistência, durabilidade e isolamento;
- Constante actualização da base de dados local da aplicação através de três modos distintos:
    - A pedido do utilizador, por intermédio de um serviço disponibilizado pela Framework à aplicação;
    - Actualização automática com tempo variável definido pela aplicação;
    - Sempre que haja "operações" que modifiquem objectos no servidor.

1.5.2 Soluções técnicas

Perante os requisitos apresentados foi necessário desenvolver uma série de soluções que os permitissem resolver.

Sistema Base de Dados Móvel

Desde logo foi necessário, então, migrar a plataforma desenvolvida para um ambiente mais próximo dos sistemas legados das organizações. A base de dados Db4o criada como "servidor" foi substituída pelo sistema de gestão de base de dados MySQL. Este sistema foi responsável por consolidar toda a informação e garantir a sua consistência e coerência. As acções tomadas pelo utilizador através da

aplicação (operações) foram redefinidas para invocar serviços que lêem/escrevem na base de dados central (servidor).

Arquitectura Flexível Cliente-Servidor

Uma das mais importantes transformações implementadas foi a definição e implementação de uma arquitectura de suporte às comunicações denominada "arquitectura flexível cliente-servidor" (Figura 1-2) [15][16]. Neste tipo de arquitectura a Aplicação/Framework comporta-se como *cliente* responsável por executar as suas operações no *servidor,* mas também como *cliente autónomo* quando completamente desconectado da rede. Nesse sentido seria expectável que a Framework conseguisse gerir uma base de dados local por aplicação para que o dispositivo pudesse operar quando desconectado da rede. Esta gestão passa por métodos de leitura, actualização e escrita. Para permitir que a base de dados esteja constantemente actualizada foi criada uma única base de dados definida como servidor e desenvolvidos alguns mecanismos de sincronização automáticos, tais como, uma *thread* de sincronização que pode ser configurada pela aplicação e, ainda, alguns mecanismos que a Framework utiliza para actualizar os objectos necessários na base de dados local assim que as operações são executadas no servidor. Desta forma, as actualizações de objectos da base de dados local são efectuadas facilmente utilizando a técnica de replicação de objectos do servidor para o ambiente local, mantendo o ambiente com dados coerentes e consistentes.

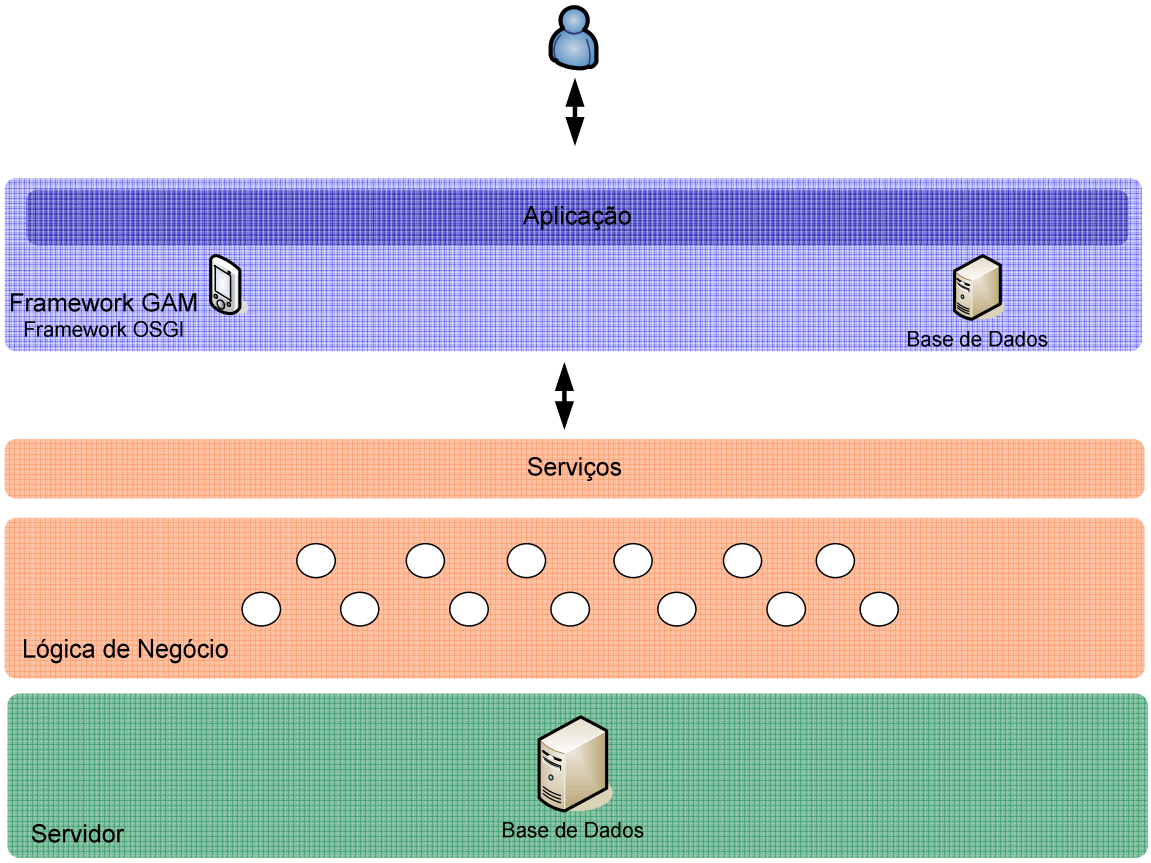

Figura 1-2: Arquitectura desenvolvida para comunicação com o "servidor"

Operações Pendentes

Quanto às operações pendentes foi criado o conceito de serviços "*críticos*" e "*não críticos*" para distinção entre um serviço que necessita de resposta imediata ou pode ser guardado e sincronizado com o servidor mais tarde. As operações tiveram que ser alteradas e guardadas de forma persistente utilizando a ferramenta para persistência de dados locais com base em objectos – Db4o. Através de serviços que a Framework disponibiliza à Aplicação, foi dado total controlo ao utilizador para poder ver/apagar as operações pendentes ou o seu *report,* em caso de falha na execução.

Transacções

Este conceito surge da necessidade de realização de um sistema que permitisse "encadear" operações que se apresentavam totalmente independentes. Com este mecanismo é possível invocar diversos tipos de operações de forma a criar dependência entre elas. Deste modo, qualquer utilizador poderá colocar no sistema duas operações de transferência que só se poderão realizar em conjunto. Ou seja, caso uma delas falhe a outra também não se poderá realizar.

2 Conceitos

O capítulo seguinte tem como objectivo compreender alguns conceitos subjacentes ao Ambiente Móvel [17]. Pretende-se deste modo apresentar a evolução da computação móvel, bem como uma breve explicação do que é um sistema móvel e um sistema de base de dados móvel de forma a situar e/ou enquadrar o leitor no trabalho realizado.

2.1 Computação Móvel

A computação móvel começou em meados de 1992, com a introdução no mercado de um *handheld* chamado Newton, pela Apple. O Newton chegou ao mercado com ecrã sensível ao toque, 1MB de memória total, e capacidade de transmissão de dados de 38.5kbps. Este modelo não teve muita repercussão, mas é considerado o início dos dispositivos móveis.

Em 1996, a U.S. Robotics lançou o (Palm) Pilot 1000 e 5000, dispositivos que tiveram uma grande aceitação no mercado e lançaram as bases de toda uma plataforma de "Palms" que chegaram a atingir 80% do mercado mundial e existem até hoje. A U.S. Robotics foi adquirida pela 3Com, que depois desvinculou dela a empresa Palm Inc., sendo esta totalmente focada nesta nova plataforma de dispositivos.

Também em 1996, começaram a surgir dispositivos com o Windows CE 1.0, da Microsoft, como o NEC MobilePro 200 e o Casio A-10. Até o lançamento do Windows CE 3.0 e da plataforma Pocket PC, em 2000, a plataforma Windows CE não teve grande aceitação do mercado. Mas a partir do Sistema Operacional Pocket PC 2000, embutido em dispositivos como o HP Jornada e o Compaq Ipaq, esta plataforma ganhou aceitação do mercado e começou a crescer.

Paralelamente a empresa Symbiam crescia. Formada em 1998 por alguns dos maiores fabricantes de telemóveis do mundo e a PSION, entregou ao mercado o sistema operacional Symbiam, que é executado na maioria dos *smartphones* e *handhelds* da Nokia, e detém a maior fatia do mercado europeu actualmente.

Actualmente o mercado está a tender para a convergência de recursos nos dispositivos móveis, criando equipamentos que concentram funções de palmtops, telemóveis, câmara fotográfica, gps, etc., além de oferecerem excelente performance, grande capacidade de armazenamento e inúmeras possibilidades de comunicação.

2.2 Sistemas Móveis

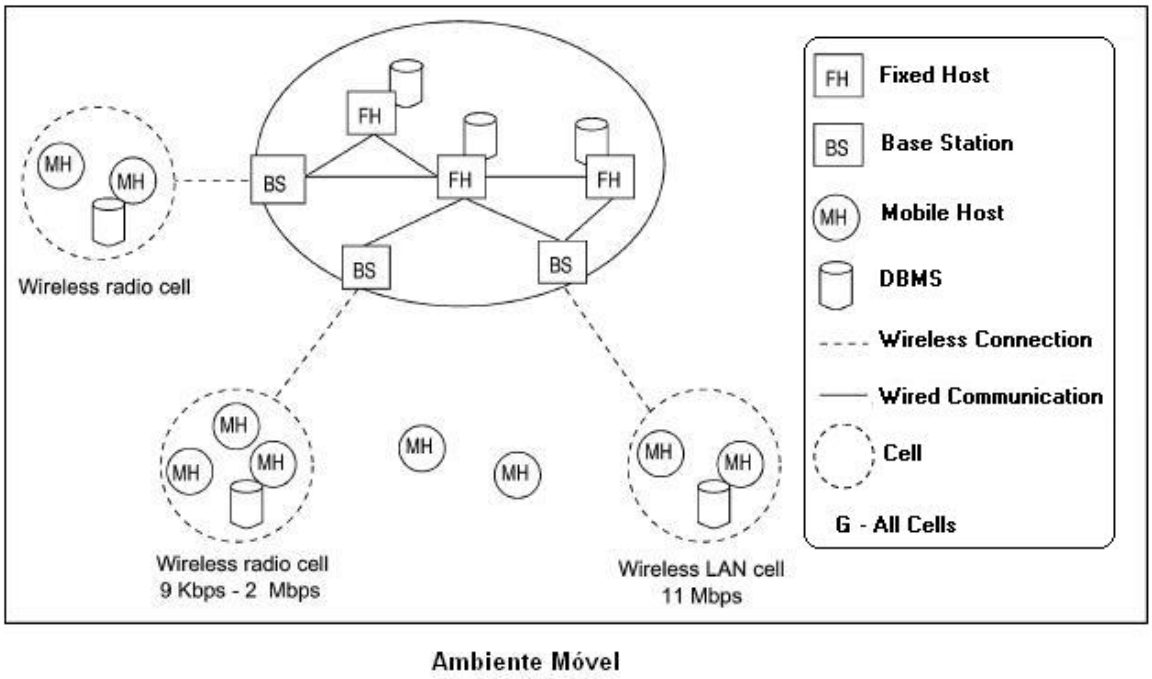

Figura 2-1: Ambiente Móvel

Um sistema móvel é um sistema distribuído, uma vez que possui unidades distribuídas ao longo de uma rede que comunicam entre si através de ligações com ou sem fio [15]. Tipicamente a rede é constituída por unidades fixas ("Fixed Host"), com uma localização conhecida e estática, unidades móveis ("Mobile Host"), que podem efectuar deslocações durante a execução das suas operações, não possuindo portanto uma localização exacta, e estações base ("Base Stations"). As unidades móveis estão conectadas aos componentes de rede, somente através das estações, via canais *wireless*. Nestes sistemas o utilizador não necessita de se encontrar sempre conectado com a restante rede, ou seja, permite-se que o utilizador se encontre desconectado por alguns períodos de tempo, operando nesta altura apenas sobre os dados que estão armazenados localmente na sua unidade. As desconexões podem ser voluntárias, para reduzir os custos de comunicação e para poupar recursos, ou involuntárias, sempre que ocorra uma falha no sistema.

As unidades móveis são equipamentos portáteis carregados a bateria, que se movimentam livremente numa área restrita. De forma a suportar toda a mobilidade destas existem pequenas

células (“cells”) que representam parte de toda a área geográfica que o sistema cobre (“geographical region”) – G. Por exemplo, na figura anterior, G é o total da área coberta por todas as estações. Esta limitação de tamanho é devido à limitação de largura de banda dos canais de comunicação *wireless.* Cada célula é gerida por uma estação em particular (“Base Station”). A cada instante as unidades móveis comunicam só com a estação responsável pela célula na qual se encontram.

As capacidades de comunicação entre as unidades móveis e as estações, que gerem a área geográfica na qual se inserem, são assimétricas: as estações não têm constrangimentos a nível de energia e podem tirar maior vantagem dos canais *broadcast* de alta largura de banda que possam existir entre elas e os clientes móveis numa dada célula. Por sua vez, as unidades móveis possuem, geralmente, recursos mais fracos do que as unidades fixas, uma vez que, para aumentar a mobilidade dessas unidades se perdem algumas capacidades ao nível do armazenamento e processamento de dados, bem como da autonomia das suas baterias. Por outro lado, como estas unidades utilizam, geralmente, comunicações sem fios, facilmente se identificam outras possíveis limitações, tais como limitada largura de banda, nível de confiança e custos de comunicação um pouco mais elevados.

2.3 Sistema de Base de Dados Móvel

Um sistema de base de dados móvel (“SBDM”) é uma extensão dos sistemas de base de dados distribuídos (“SBDD”), uma vez que possui um conjunto de unidades fixas tal como estes [19][20][21]. No entanto, neste caso estamos perante um sistema que possui uma ou mais unidades móveis com características bem diferentes dos terminais fixos. Num sistema de base de dados móvel é necessária a existência de unidades fixas que de certa forma sirvam de suporte às actividades das unidades móveis, podendo mesmo actuar como servidores de dados. O sistema em causa possui inúmeras características e facilidades distintivas que conferem aos seus utilizadores algumas vantagens tais como a mobilidade e portabilidade. A mobilidade não se confere simplesmente ao movimento das unidades fixas mas também à informação que possuem.

No entanto, as características inerentes ao ambiente móvel fazem com que um sistema deste género seja muito mais difícil de gerir do que um sistema distribuído “fixo”. Nestes a estabilidade das suas ligações e a localização das unidades fixas e da informação é conhecida e estática contrariamente ao que acontece num sistema de base de dados móvel que é dinâmico e com ligações muito instáveis.

3 Arquitectura

O capítulo seguinte tem como objectivos introduzir o leitor na arquitectura da Framework GAM e explicar sumariamente o seu funcionamento.

3.1 Arquitectura

O projecto desenvolvido anteriormente contemplava um modelo de programação por componentes (*bundles*) executados e geridos por cima de uma Framework OSGI – Knopflerfish [2] [3]. Os diferentes componentes que constituem a *framework* apresentam uma dependência forte para fazer da *framework* um todo, mas ao mesmo tempo uma dependência fraca vistos como componentes de distribuição de software no universo OSGi. Assim, com esta noção de *bundles* fracamente interligados e ainda dentro do conceito OSGI, verifica-se a existência de um modelo de cooperação entre componentes desde a partilha de código ao fornecimento e consumo de serviços entre eles. Temos então assim que cada bloco principal da *framework* é um *bundle* OSGi e as próprias aplicações também o são.

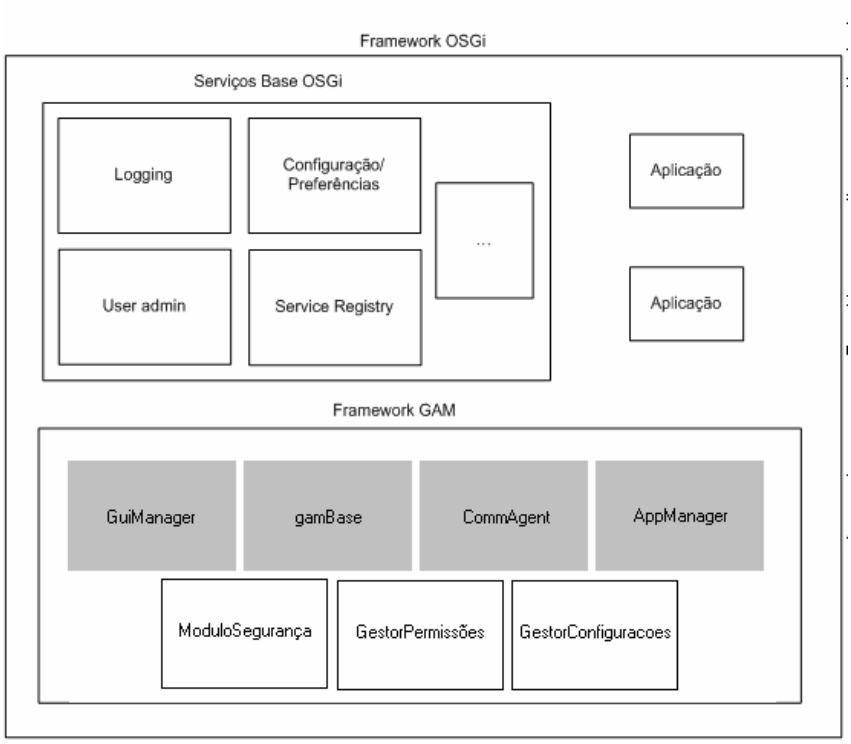

Figura 3-1: Decomposição em módulos da arquitectura da Framework GAM

A Figura 3-1 representa a vista em módulos do sistema desenvolvido, onde cada módulo representa um *bundle* (componente OSGi).

Seguidamente é apresentado os *bundles* desenvolvidos, na figura anterior a sombreado, importantes para este trabalho:

- AppManager: é o *bundle* que arranca a *framework* e como tal é o mais visível para o utilizador. É através da interface gráfica deste que é fornecido um modelo simplificado para a instalação e remoção de aplicações e a sua execução;
- GuiManager: responsável pelas interfaces gráficas das aplicações da *framework*. Efectua o carregamento das configurações do aspecto visual e das acções associadas a cada elemento da interface. Dado que as acções do utilizador partem todas da interface gráfica, é também responsável pela invocação de outros componentes quando é necessário que estes forneçam serviços específicos para as aplicações ou a execução de código particular a uma determinada aplicação;
- CommAgent: responsável pelas comunicações da *framework* com o exterior. Efectua o carregamento da configuração de comunicações descrita nos ficheiros XML específicos a cada aplicação. Guarda um modelo interno das comunicações de cada aplicação e regista um serviço na *framework* OSGi para as aplicações, poderem utilizar e efectuar as suas comunicações com o exterior, englobando IO local ao dispositivo, *webservices* e pedidos http. É também o responsável por controlar a conectividade do dispositivo e manter um registo de comunicações a efectuar consoante a existência ou não de conectividade;
- gamBase: *bundle* contendo e disponibilizando classes de base da *framework* para serem usadas pelos outros *bundles.* Estas vão desde classes abstractas para a utilização de serviços da *framework* às *interfaces* desses mesmos serviços.

Nos mecanismos de sincronização desenvolvidos, neste projecto, foram consideradas algumas alterações importantes em alguns *bundles*, tais como “CommAgent”, “gamBase”, “Gestor Configurações” e “AppManager”, no entanto as alterações realizadas mantêm as propriedades dos componentes em questão.

Desta forma a arquitectura por componentes identificada mantém-se no seu todo.

3.1.1 Ambiente de Testes

Embora a arquitectura a nível de *bundles* se tenha mantido foi necessário desenhar e implementar uma arquitectura e estrutura tecnológica capaz de suportar todo o ambiente de testes. O ambiente é caracterizado pela estrutura que suporta as actividades da aplicação desenvolvida para efeito de teste.

Após uma análise cuidada das ferramentas/tecnologias utilizadas anteriormente e feito o estudo de investigação necessário acerca dos constrangimentos existentes foram identificadas as ferramentas/tecnologias a utilizar.

A figura seguinte mostra a arquitectura tecnológica definida:

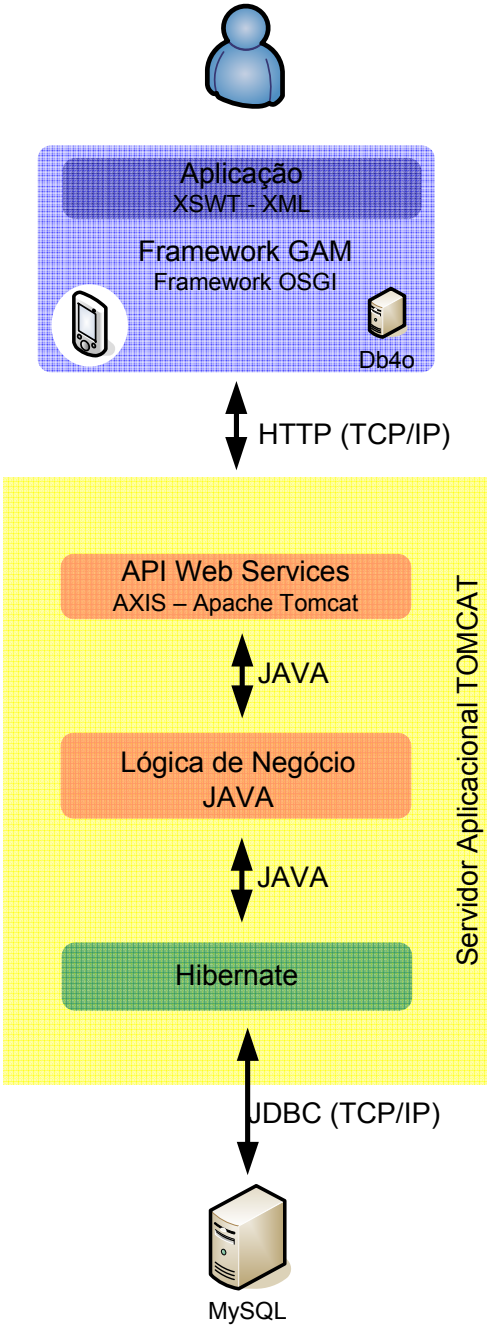

Figura 3-2: Arquitectura Tecnológica

Como linguagem de programação esta estaria à partida limitada à utilizada no trabalho anterior (JAVA) uma vez que é a linguagem utilizada pela própria Framework Knopflerfish (OSGI R3) [2][3].

3.1.1.1 Interface Gráfica

A biblioteca gráfica SWT [4], bastante rápida na resposta ao utilizador, foi mantida do projecto anterior dado corresponder às necessidades identificadas, bem como a linguagem XSWT [5], a linguagem mais popular no âmbito do SWT, que utiliza ficheiros XML para definição das interfaces gráficas para interacção com o utilizador. O facto de tanto a biblioteca gráfica como a linguagem em causa não irem de encontro ao cerne do problema a resolver neste trabalho e estando ambas completamente estáveis na Framework GAM foram dois factores que pesaram na continuação da sua utilização.

3.1.1.2 Webservices

A escolha de *webservices* como tecnologia para comunicação com a Framework define a primeira camada de interacção com o exterior. Foi utilizada uma ferramenta para uso de *webservices* – Axis do projecto Apache Tomcat [6] – que pela sua simplicidade foi, também, mantida do projecto anterior.

No entanto, a invocação dos métodos remotos para entrega das "operações pendentes" pressupunha alguns problemas no caso de falhar a comunicação entre o servidor e a Framework. Desde logo, garantir a entrega de todas as "operações pendentes" no servidor, garantir a entrega por ordem pela qual foram executadas na unidade móvel, não repetição da invocação e a existência de "transacções", como um conjunto de operações, colocava à partida objecções que teriam de ser estudadas.

Se por um lado os dois primeiros requisitos referidos são completamente salvaguardados pela própria invocação de serviços por parte da Framework, já os últimos dois (não repetição da invocação e existência de transacções) mereceram especial atenção. Nesse sentido foi definida uma solução no sentido da normalização das mensagens enviadas com base nos conceitos subjacentes à especificação baseada em mensagens SOAP: WS-Reliability [22]. Esta especificação define a entrega no contexto *standard* de *Webservices* e tem sido desenhada para utilização na combinação com outros protocolos complementares (W3C SOAP 1.1/1.2, "OASIS ebXML Message Service Specification", "OASIS Web Services Security", "WS-I Basic Profile 1.1").

A solução encontrada foi, então, uma funcionalidade desenvolvida que atribui um identificador único e coloca-o no *header* da mensagem SOAP [7] – responsável pelo empacotamento dos dados necessários à invocação do serviço. Assim, através de um *handler* colocado no servidor poderá haver um prévio controlo do identificador, antes da invocação do serviço, evitando a duplicação de mensagens. Este método desenvolvido foi também solução para a existência do conceito "transacção" quando atribuído um identificador que relacione as operações que a constituem.

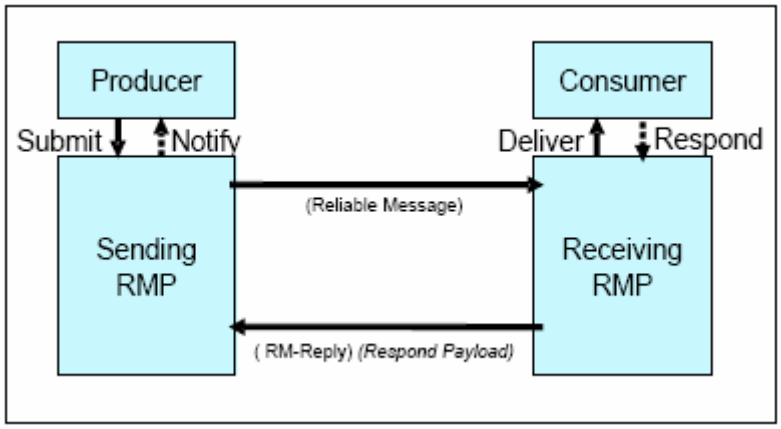

Figura 3-3: "WS-Reliable Message"

3.1.1.3 Lógica de Negócio

Foi criada uma lógica de negócio, que não é mais que uma orquestração de métodos em Java representativos das diferentes funções de negócio da aplicação criada.

3.1.1.4 Persistência de Dados

No que diz respeito à persistência de dados há a considerar dois ambientes totalmente distintos. Por um lado existe a necessidade de guardar dados de forma persistente do lado do servidor para efeitos de teste, por outro é necessário capacitar a Framework GAM com mecanismos de escrita e leitura de dados guardados persistentemente.

Os ambientes, no entanto, são totalmente diferentes com diferentes características, como referido anteriormente, nomeadamente a assimetria existente entre os dispositivos.

Ambiente Fixo

Em relação a persistência de dados em ambiente "fixo" podemos optar por ficheiros de dados ou sistemas de gestão de base de dados (SGBD). Se a escolha for a segunda coloca-se a questão das licenças.

A oferta é elevada (Oracle [8], SQLServer [9], MySQL [10], etc.) mas cada uma com as suas características. A escolha poderia recair sobre uma versão de maiores capacidades e possibilidades

mas que implica custos, ou então, o MySQL ou a versão "light" do SQLServer (MSSQL Express) que têm algumas limitações.

Análise e Solução encontrada (Ambiente Fixo)

Para o projecto em causa a opção recaiu sobre um SGBD por forma de recriar o ambiente natural das organizações. Recorrer a ficheiros de dados seria uma opção neste caso, tendo em conta que o ambiente de testes não continha um modelo de dados complexo, no entanto, a escolha recaiu sob um sistema de gestão de base de dados uma vez que os acessos são constantes e desta forma é possível recriar todo o ambiente encontrado nos sistemas legados das organizações.

Em relação ao SGBD a usar, depois de se analisar algumas soluções que o mercado apresenta, a opção acabou por recair sobre MySQL. As características que pesaram na sua escolha foram a livre utilização do sistema, grande comunidade de utilizadores por todo o Mundo, garantindo praticamente a resolução de todos os problemas que pudessem surgir, e, ainda a sua facilidade de utilização.

MySQL

O MySQL é um sistema de gestão de base de dados (SGBD), que utiliza a linguagem SQL ("Structured Query Language") como interface [10]. É actualmente um dos sistemas de base de dados mais populares, com mais de 4 milhões de instalações pelo mundo. Compatibilidade, desempenho, pouca exigência relativamente aos recursos de hardware necessários são algumas das suas principais características, no entanto também tem algumas limitações, tais como não suportar *Views*, *Triggers* ou "*Stored Procedures*". mas no geral as limitações encontradas não comprometem a sua utilização neste projecto e, mais concretamente, no ambiente criado.

Ambiente Móvel

No ambiente móvel as hipóteses teriam de ter em conta os constrangimentos encontrados nas unidades móveis, tais como a limitada capacidade de armazenamento, velocidade de processamento ou memória. Também neste caso podia se ter recorrido a ficheiros de dados ou a um SGBDs pelo que as questões de licenciamento voltam a colocar-se.

Existem algumas soluções interessantes, embora não tantas como as existentes para os sistemas fixos. As soluções analisadas foram o Db4o [11], PointBase [12], "FastOjects j2" [13] e "WebSphere EveryPlace" (IBM) [14].

Análise e Solução encontrada (Ambiente Móvel)

Igualmente como no ambiente fixo foi escolhido um SGBD, neste caso móvel, ao invés de ficheiros de dados dado o volume de acesso a dados ser elevado e por questões de performance de desenvolvimento. A escolha recaiu sob o sistema de gestão de base de dados DB4o por três importantes motivos. Desde logo a necessidade de manter os reduzidos custos de licenciamento de software. De notar que tanto a linguagem de programação JAVA, como a Framework OSGI escolhida (Knopflerfish) ou mesmo a opção de persistência de dados escolhida para o servidor (MySQL) não têm quaisquer restrições sendo de utilização livre. Outro motivo é a enorme comunidade de utilizadores/programadores que utilizam DB4o como demonstra a quantidade de fóruns existentes na Internet. Por outro lado foi considerado que embora tendo sido analisadas e testadas outras soluções, o DB4o já tinha sido utilizado na versão anteriormente desenvolvida, pelo que por mais este factor foi mantido.

Db4o

Db4o é uma ferramenta, *open-source*, para persistência de dados locais baseado em base de dados por objectos [11]. A programação é feita num ambiente totalmente orientado a objectos. Os grafos de objectos são armazenados directamente na base de dados e com possibilidades de pesquisas usando *templates* de objectos ou mesmo interrogações tradicionais SQL.

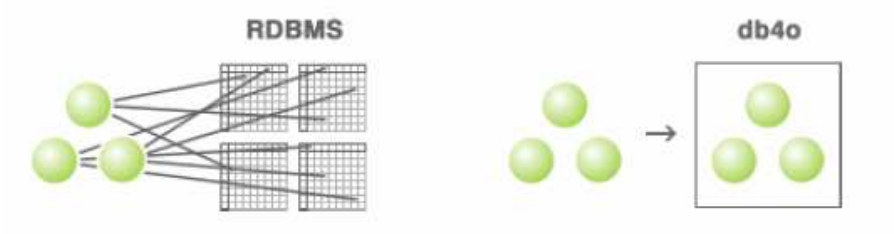

Figura 3-4: Mapeamento Db4o

Como características principais, tem no seu sistema de gestão de base de dados, a possibilidade de sincronização entre diferentes bases de dados e com um *footprint* muito leve (400KB). A sincronização é efectuada através do atribuir de versões aos objectos, e da identificação destes com UUIDs. Torna, assim, possível a actualização de apenas as diferenças entre objectos das duas bases de dados, e ao mesmo tempo, o suporte a classes gestoras de conflitos, no caso de um objecto ter sido alterado em mais de um local.

Podemos ver a utilização desta ferramenta como numa camada que serve de sincronização com os dispositivos móveis podendo haver a outro nível uma interligação com bases de dados tradicionais através do mapeamento Objecto-Relacional, presente em ferramentas para a linguagem *Java* como o *OJB* ou o *Hibernate* [23][24].

Concluindo, Db4o é, então, desenhado para providenciar um motor de gestão de base de dados que possa ser introduzido em equipamentos móveis, fixos e servidores num ambiente orientado a objectos.

3.1.1.5 Mapeamento Objecto-Relacional

Perante a escolha de Java como linguagem de programação, DB4o [11] e MySQL [10] para a persistência de dados, fez com se surgisse a hipótese de as combinar com outras ferramentas/tecnologias. Mesmo utilizando a linguagem de programação Java seria interessante utilizar algumas ferramentas que evitassem a escrita de muito código e, assim, facilitasse o desenvolvimento, bem como ajudassem a definir certas camadas:

Um exemplo flagrante é o acesso por JDBC à base de dados. A solução clássica é utilizar um *driver* JDBC para o SGBD em questão (MySQL) e escrever toda a parte de comunicação. Depois de algumas pesquisas sobre ferramentas que pudessem abranger esta área, decidiu-se utilizar Hibernate [23][24]. O seu desenvolvimento foi facilitado graças à utilização da ferramenta de desenvolvimento MyEclipse [25] construída sob a plataforma Eclipse [26]. As bibliotecas Hibernate são distribuídas com a ferramenta sem necessidade de fazer o download, possui ainda características que permitem capacitar o projecto com Hibernate ou para criar o ficheiro de configuração do Hibernate (*hibernate.cfg.xml*) e gera automaticamente classes JAVA e todo o mapeamento para a base de dados através de objectos persistentes em XML que depois são acedidos pela camada lógica de negócio. A utilização do Hibernate acabou por facilitar o desenvolvimento uma vez que criou um ambiente orientado a objectos e encapsulou, de certa forma, todas as invocações SQL à base de dados (*read*, *write*, *update*, *delete*). Esta ferramenta aliada à escolha de DB4o para a persistência de dados na Framework, criou todo um ambiente orientado a objectos facilitando o seu desenvolvimento.

3.1.1.6 Comunicações

Quanto às comunicações, com o servidor aplicacional TOMCAT [6], são realizadas através do protocolo de comunicação TCP/IP e através de um *driver* JDBC para interacções com a base de dados MySQL.

4 Sincronização

4.1 Funcionamento

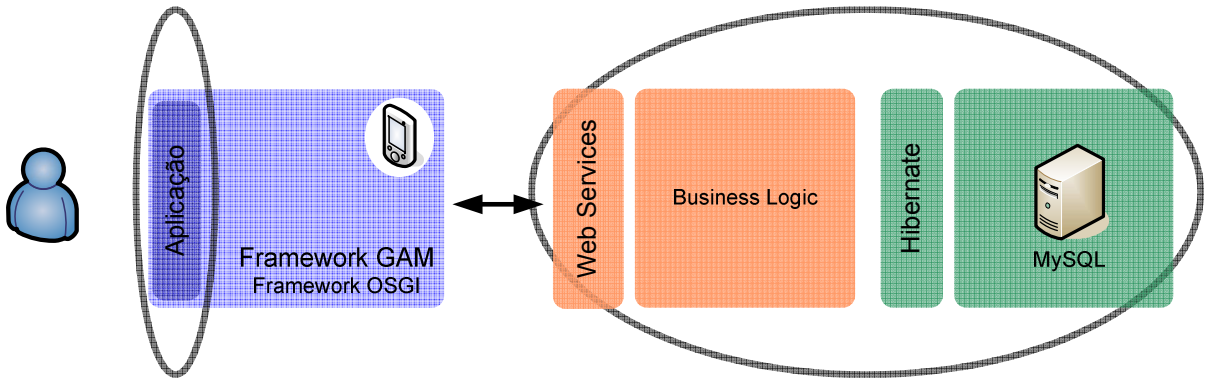

Figura 4-1: Vista da arquitectura de testes desenvolvida

Os mecanismos de sincronização foco deste projecto estão inseridos na Framework GAM. Desta forma, a utilização desses mecanismos só será possível com o seu completo funcionamento.

Tipicamente terá de ser desenhada uma aplicação, bem como, todas as camadas associadas às necessidades de execução da aplicação, designadamente toda a lógica de negócio e camada de *webservices*. De forma a facilitar o leitor consideremos todas estas camadas como "servidor".

A Framework tem, assim, um papel de mediadora de todo o processo de gestão das operações pendentes, gestão da base de dados local à aplicação, bem como, de todas as comunicações com o exterior através da configuração de ficheiros XML (*gui.xml*, *comm.xml*) por parte da aplicação [27]. Seguidamente são apresentados sumariamente os ficheiros de configuração e código necessário ao funcionamento da Framework.

4.1.1 Servidor

Desenhada toda esta estrutura de suporte à aplicação e definida a camada de interacção (*webservices*), é necessário definir nos ficheiros de configuração da aplicação o modo como esta pode invocar os *webservices*.

```
<config id="ebgmc">
            <wsdl host="127.0.0.1" port="8080" /><!-- 127.0.0.1 -->
            <locator value="pt.link.gam.ws.bank.EBGMCServiceServiceLocator" />
            <port value="pt.link.gam.ws.bank.EBGMCService_PortType" />
            <portMethod value="geteBGMCService" />
</config>

        <service id="levantamento" config="ebgmc" retry="false">
                <method value="levantamento" />
                <arg name="numCliente" type="string" />
                <arg name="user" type="string" />
                <arg name="pass" type="string" />
                <arg name="nib" type="string" />
                <arg name="valor" type="double" />
        </service>

        <service id="deposito" config="ebgmc" retry="true">
                <method value="deposito" />
                <arg name="numCliente" type="string" />
                <arg name="user" type="string" />
                <arg name="pass" type="string" />
                <arg name="nib" type="string" />
                <arg name="valor" type="double" />
        </service>
```

Figura 4-2: Configuração de serviços presente no ficheiro *comm.xml*

A aplicação escolhida para testar os mecanismos de sincronização sendo uma interface que simula interacções com uma entidade bancária era necessário desenvolver uma camada de suporte que permitisse pesquisas e actualizações de dados numa base de dados central. Nesse sentido, foram desenvolvidas quatro camadas, já anteriormente identificadas, designadamente as camadas de *webservices*, lógica de negócio, Hibernate e base de dados central.

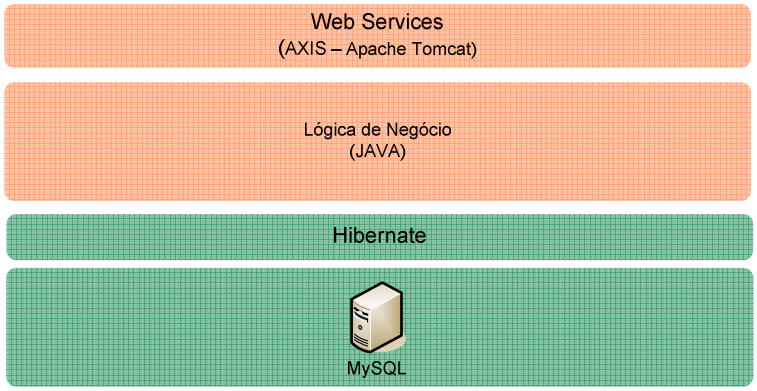

Figura 4-3: Diferentes camadas da arquitectura (servidor)

Para comunicação com os *webservices* desenvolvidos foi criada uma classe genérica (*TransactionResponse*) para retorno do serviço invocado de modo a facilitar e a normalizar as comunicações.

4.1.2 Aplicação

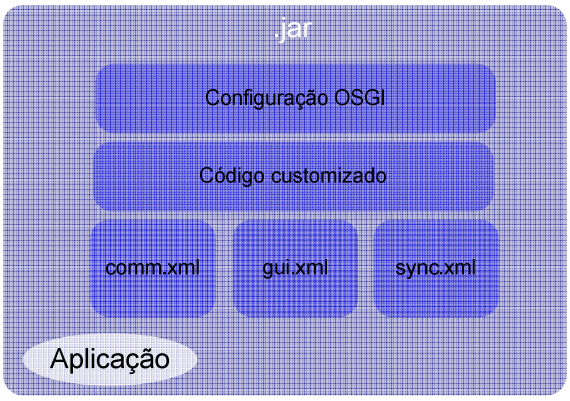

Figura 4-4: Componentes desenvolvidos pela Aplicação

Desenvolvida a aplicação através da criação de um *bundle* OSGI é criado um único ficheiro JAR. Este JAR contém diferentes ficheiros dos quais é necessário destacar o ficheiro de configuração OSGI (*bundle.manifest*), código necessário ao registo do *bundle* OSGI, ficheiros de configuração da Framework GAM e o código necessário à aplicação.

Código necessário à aplicação

A aplicação para executar as suas funções necessita código "costumizado", tal como código de execução de *stubs* para uso de *webservices*. Desta forma, a aplicação terá de indicar nos seus ficheiros de configuração, nomeadamente no "*gui.xml*", quais os pacotes Java que têm esse código para que seja disponibilizado dinamicamente a todos os outros componentes OSGI.

```
<interface id="frmClientes" filename="forms/frmDownloadDadosClientes.xswt">
     <object type="shell" event="onLoad">
          <action type="invoke">
               <param name="class" value="pt.link.gam.custom.esynctest.CustomClasses" />
               <param name="method" value="carregaClientes" />
          </action>
```

Figura 4-5: Configuração para invocações dinâmicas de código "costumizado" (*gui.xml*)

Ficheiros de configuração GAM

- Gui.xml: Interface
- Comm.xml: Comunicações
- Sync.xml: Sincronização

Os ficheiros em causa são disponibilizados dinamicamente à Framework para configuração das diferentes interfaces da aplicação, comunicações da aplicação tanto com o exterior como para acesso à sua base de dados local e sincronização da sua base de dados local com outra base de dados exterior Db4o.

O ficheiro Sync.xml é carregado pelo componente (*bundle*) SyncManager e, tal como este, não foi desenvolvido neste trabalho. A opção recaiu sobre a utilização do Comm.xml para definição de um *webservice* que retorna os objectos para actualização na base de dados local.

4.1.3 Framework

Como referido anteriormente, cabe à Framework GAM o papel de mediadora do processo de comunicação entre a aplicação e o exterior ou entre esta e o acesso à sua base de dados. A Framework torna-se, assim, um meio normalizado para qualquer tipo de aplicações que facilita o seu desenvolvimento, distribuição e gestão em ambientes móveis. Contem funcionalidades que necessitam de configuração através dos ficheiros XML indicados anteriormente e pode carregar dinamicamente e executar código "customizado" da própria aplicação. Deste modo é possível implementar qualquer tipo de aplicação sendo que parte da sua funcionalidade poderá ser executada através das características da própria Framework ou de código personalizado.

No que diz respeito aos mecanismos de sincronização deste trabalho os mesmos foram desenvolvidos no *bundle* CommAgent. Neste componente é carregado todas as configurações existentes no ficheiro "*comm.xml*" da aplicação quando esta é iniciada, sendo possível, assim, articular a comunicação com os métodos da camada de *webservices* definidos ou com a sua base de dados local.

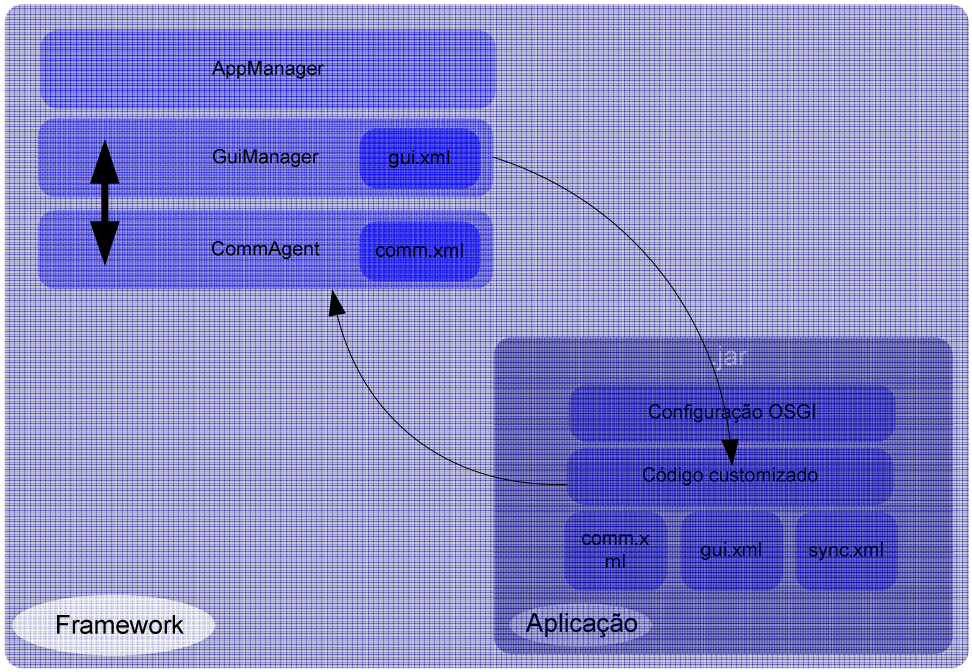

Figura 4-6: Interacção entre componentes da Framework e da Aplicação

O GuiManager controla toda a interacção do utilizador com as interfaces gráficas. Quando uma funcionalidade da interface gráfica, referente a uma comunicação com o exterior, é executada este componente invoca o serviço disponibilizado pelo CommAgent com os parâmetros indicados pelo utilizador e aguarda o seu resultado. O CommAgent por sua vez verifica se essa invocação é relativa a uma acção local ou remota. Caso seja local este age em conformidade retornando o resultado da sua acção ao GuiManager. Caso contrário está perante uma invocação de uma acção remota através de *webservices*.

Verificadas as configurações de invocação estamos perante uma das características principais dos ambientes móveis – a possibilidade de não haver conexão à rede. Caso haja conexão é invocado o método remoto e são actualizados localmente os objectos alterados no servidor, caso contrário a Framework está perante uma falha de rede guardando a operação como operação pendente para mais tarde poder ser sincronizada com o servidor. Deste modo a *thread* de sincronização das

operações pendentes pode a todo o momento sincronizar com o servidor as operações assim que exista conexão à rede e logo de seguida actualizar localmente os objectos alterados para que a aplicação possa ser executada quando completamente desconectada. De notar que pode ser parameterizado no ficheiro "*comm.xml*" quais os serviços que necessitam de resposta imediata, e desta forma não são guardados em memória, e quais os que pelas suas características podem ser realizados mais tarde. As operações pendentes são invocadas no servidor garantindo três propriedades: garantia de entrega, garantia de não duplicação e garantia de entregas em grupo (realização de todas ou nenhuma).

A Framework possui também uma *thread* de sincronização da BD local, com tempo variável definido pela aplicação.

5 Framework GAM

A seguinte secção tem como objectivo a explicação clara e concisa de toda a implementação da Framework objecto deste trabalho. No capítulo anterior foi abordado o funcionamento dos diferentes componentes desenvolvidos (Aplicação, Servidor e Framework), seguidamente serão aprofundados os temas referidos na secção Framework com maior nível de detalhe.

O foco deste trabalho está na sincronização de dados entre o ambiente móvel e o ambiente fixo. Como tal, neste contexto, não menosprezando outras características importantes, foquemo-nos simplesmente na característica da Framework como intermediária no processo de comunicação remoto entre os dois ambientes ou local quando corresponde a pedidos de escrita/leitura à base de dados da aplicação.

5.1 Comunicações Framework – Servidor

A arquitectura implementada para suporte às comunicações é a "arquitectura flexível cliente-servidor". Neste caso, o cliente (Framework) descarrega as suas operações no servidor mas também se comporta como cliente autónomo quando completamente desconectado da rede.

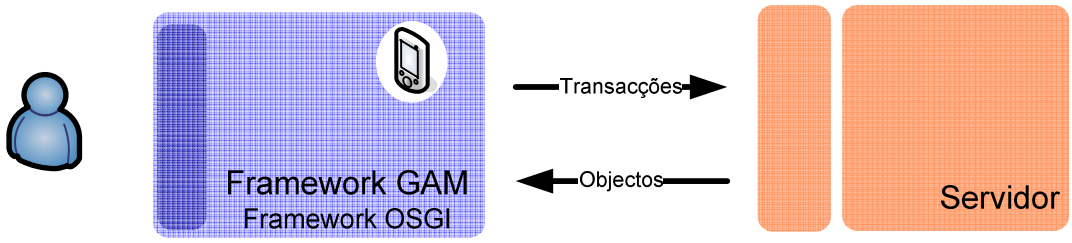

Figura 5-1: Comunicações entre Framework e Servidor

Neste tipo de arquitectura o servidor é responsável por toda a actividade computacional, no entanto também simula funções de servidor para permitir que as aplicações continuem a ser executadas sem estarem fortemente conectadas a servidores remotos. O papel do cliente, servidor, bem como, das funcionalidades das aplicações pode ser dinamicamente realojado. Tendo como objectivo a

performance e a disponibilidade a distinção entre cliente e servidor pode temporariamente não ser clara.

Por operação entenda-se uma transacção "atómica" iniciada pelo utilizador através da aplicação desenvolvida. Esta transacção corresponde à invocação de um único serviço (levantamento, depósito, transferência, consulta, etc.) e não a um conjunto de invocações encadeadas numa sequência.

Desta forma foi decidido que no sentido Framework – Servidor as operações seriam denominadas transacções simples e no sentido contrário seriam enviados objectos actualizados. Assim garante-se que o processamento significativo é realizado no servidor e que existe uma base de dados completamente consolidada e actualizada.

Invocação de Serviços

As transacções "atómicas" simples não são mais do que invocações de serviços definidos como *webservices.* Por exemplo, uma transferência é neste caso a invocação de um *webservice* denominado "transferência", assim a complexidade de subdividir a "transacção" em duas sub-operações (levantamento e depósito) está sob a responsabilidade do Servidor.

No entanto, como referido na subsecção "Ambiente de Testes" no capitulo "Sincronização", é necessário garantir algumas propriedades na sua invocação, tais como garantir a entrega de todas as operações ao Servidor, garantir a entrega por ordem pela qual foram executadas na unidade móvel e não repetição das execuções no Servidor.

A forma como a Framework executa as operações resolve os dois primeiros requisitos. A invocação das operações é executada de forma sequencial e pela ordem pela qual chegam à Framework. Por outro lado esta fica à espera pela resposta do Servidor o que garante a entrega da operação. Por último, resta o requisito mais delicado. Se aparentemente a espera pela resposta garante a entrega, pode resultar que caso haja uma falha na comunicação da resposta esta possa não ser entregue à Framework, o que faz com que a invocação da mesma se repita.

A figura seguinte demonstra o problema identificado:

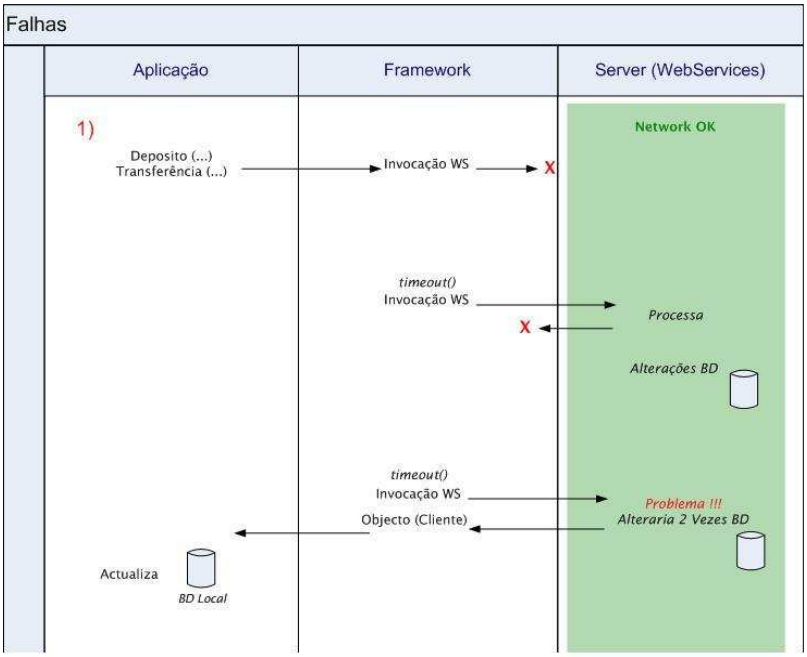

Figura 5-2: Problema encontrado por falha na comunicação da resposta

Para resolver este problema inerente à invocação de *webservices*, a Framework cria um identificador único que coloca no *header* da mensagem SOAP.

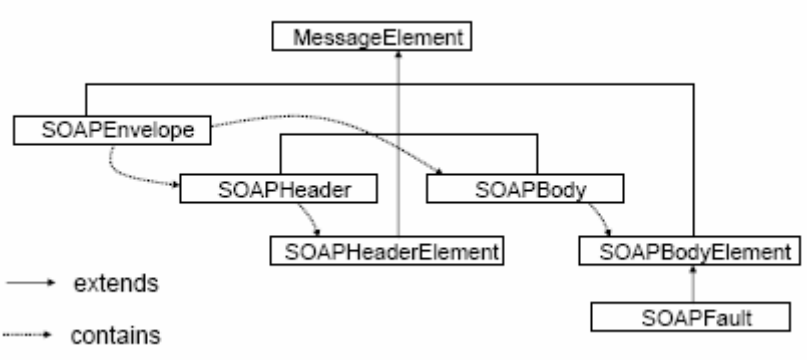

Figura 5-3: Decomposição da mensagem SOAP

Com o identificador presente o Servidor pode controlar as mensagens que lhe chegam. Para isso, foi definido um "*request global handler"* próprio no ficheiro *server-config.wsdd* do Axis [6].

Handlers são serviços invocados pelo Axis em *runtime*. Estes podem ser executados no *requestFlow* e/ou *responseFlow*. *Handlers* definidos no *requestFlow* são executados antes do método definido no *webservice*, caso sejam definidos no *responseFlow* são invocados depois do método mas antes ainda da resposta ao cliente.

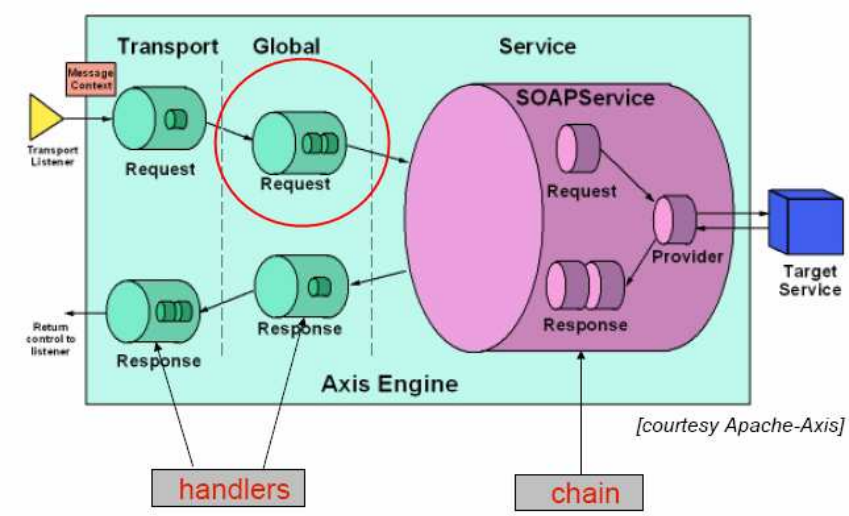

Figura 5-4: Diferentes tipos de *Handlers*

O *handler* criado e definido neste projecto é denominado "MyHandler" e foi definido ao nível da camada "*Global*" da figura anterior. A solução genérica encontrada foi pensada no sentido da normalização das mensagens, oferecendo a possibilidade ao Servidor de controlar as mensagens que já executou guardando de forma persistente os identificadores e as respectivas respostas.

5.2 Comunicações Aplicação – Framework

Foi criada uma classe denominada *InvocationClass* na componente gamBase para estandardizar os parâmetros de invocação da componente CommAgent da Framework a partir da Aplicação.

InvocationClass *(tipo,argumentos,objectos_actualizar)*

A classe disponibiliza uma *String* que define um determinado tipo de serviço descrito no "*comm.xml*" (ex.: consulta, levantamento, deposito, transferência,...), bem como dois *arrays* de *Objects* para argumentos de invocação e, caso seja necessário, objectos a serem actualizados localmente depois de executadas as operações no servidor. Desta forma completamente genérica, a Framework consulta o ficheiro "*comm.xml*" identificando que tipo de invocação se trata através do primeiro parâmetro. Seguidamente executa o serviço passando-lhe os argumentos contidos no segundo parâmetro. Por fim, verifica a existência de objectos no terceiro parâmetro e caso se verifique executa, recorrendo a estes, o serviço de sincronização genérico, definido no ficheiro "*comm.xml",* obtendo os objectos actualizados e, assim, actualizando a base de dados local à aplicação.

```
InvocationClass inv = new InvocationClass();

inv.type = "transferencia";
inv.args = new Object[]{
        c.identificador,
        c.user,
        c.pass,
        numConta,
        numClienteDest,
        numContaDest,
        new Double(valor)};
inv.obj_to_update = new Object[]{numCliente,numClienteDest};

o = ca.doLook(inv);
```

Figura 5-5: Invocação do CommAgent a partir da Aplicação (CustomClasses.java)

```
Object c = comm.read.run(null,inv.args,null);

if(inv.obj_to_update != null){
    synchronizeGenericObj(inv);
}
```

Figura 5-6: CommunicationAgent - commAgent (Framework)

Como retorno a Framework pode responder através de duas formas – *TransactionResponse* ou *Exception*. Se tudo correr bem ao nível da comunicação e da execução no Servidor, o retorno passa pela classe *TransactionResponse* criada no Servidor, tal como referido anteriormente. Neste caso a Framework adquire um papel não preponderante limitando-se a reencaminhar a classe para a Aplicação.

Por outro lado pode ser retornada uma excepção por dois motivos distintos, falha na execução no Servidor ou na comunicação com a camada *webservices*.

Perante a impossibilidade das operações serem executadas por diferentes motivos (ex.: saldo insuficiente, cliente inexistente, numero de conta inexistente,...) o Servidor retorna uma excepção previamente acordada com a Aplicação de forma a esta poder identificar o seu motivo. Este procedimento é possível uma vez que a Framework adquire novamente um papel passivo reencaminhando a excepção para a Aplicação.

Quanto à falha nas comunicações com a camada de *webservices* a Framework recebe uma *SocketException* e nesse caso tem um papel activo colocando na excepção de resposta à Aplicação uma das duas mensagens de erro descritas na classe *Constants.java* – gamBase. Os serviços podem ser "configurados" se necessitam de resposta imediata, neste caso denominados *críticos*, ou se os seus argumentos podem ser guardados como "Operações Pendentes" para posterior execução no Servidor, designados "*não críticos*".

```
<service id="levantamento" config="ebgmc" retry="false">
        <method value="levantamento" />
        <arg name="numCliente" type="string" />
        <arg name="user" type="string" />
        <arg name="pass" type="string" />
        <arg name="nib" type="string" />
        <arg name="valor" type="double" />
</service>

<service id="deposito" config="ebgmc" retry="true">
        <method value="deposito" />
        <arg name="numCliente" type="string" />
        <arg name="user" type="string" />
        <arg name="pass" type="string" />
        <arg name="nib" type="string" />
        <arg name="valor" type="double" />
</service>
```

Figura 5-7: Serviços descritos no ficheiro "*comm.xml*"

Tal como se pode verificar na figura anterior, os serviços "levantamento" e "depósito" têm dois comportamentos distintos no caso da comunicação falhar entre a Framework e o Servidor. O serviço "levantamento" é um dos exemplos representativos de serviços *críticos*, outro exemplo poderia ser a "consulta do saldo" do cliente ou a "consulta de movimentos". Quando na presença deste tipo de serviços a Framework retorna uma excepção com a seguinte mensagem de erro: "sem conectividade

disponível para realizar a operação". Por outro lado, o "depósito" é um serviço "*não crítico*", o qual pode ficar pendente à espera de conectividade. Aquando na presença deste tipo de serviços a Framework retorna uma excepção com a mensagem: "sem conectividade disponível para realizar a operação imediatamente, será realizada assim que a conectividade seja restabelecida" guardando todos os argumentos da classe *InvocationClass* como "Operação Pendente", bem como, o identificador da mensagem mencionado na secção anterior, para poder sincronizar as operações assim que obtenha conectividade.

5.3 Operações Pendentes

Esta é a característica mais importante da Framework GAM desenvolvida na classe *CommunicationAgent* do gestor de comunicações *commAgent*. A possibilidade de guardar transacções iniciadas na aplicação quando o dispositivo móvel não tem rede. No entanto, as operações guardadas na Framework desenvolvida no projecto anterior não eram guardadas de forma persistente o que permitia que fossem perdidas caso o dispositivo se desligasse. Na presença de um pedido de acesso remoto são carregadas dinamicamente as configurações necessárias à invocação do serviço. Sem conectividade, na presença de uma invocação de um dos serviços críticos, a Framework recorre a uma base de dados Db4o para guardar os dados presentes na classe *InvocationClass*.

Os dados guardados são simplesmente o identificador único da mensagem enviada e os argumentos da *InvocationClass (tipo, argumentos, objectos_actualizar)* o que permite que o carregamento dinâmico das configurações de acesso a um determinado *webservice* seja efectuado a cada vez que tenta comunicar. Neste caso, mesmo que as configurações do *webservice* mudem é possível sempre estabelecer a ligação. Ou seja, mesmo que o dispositivo se desligue e se os *webservices* forem publicados noutra máquina é apenas necessário actualizar o ficheiro de configuração "*comm.xml"* com as novas configurações.

Desta forma as operações podem ser sincronizadas posteriormente através de uma *thread* que monitoriza constantemente a existência de operações pendentes por sincronizar.

São disponibilizados, ainda, pela Framework serviços para que a aplicação possa controlar o estado das "operações pendentes". Estas podem ser verificadas, analisados os seus argumentos ou mesmo eliminadas. Caso sejam invocadas no Servidor e, por algum tipo de falha, não sejam executadas a Framework guarda-as de forma persistente e disponibiliza serviços para que o utilizador possa verificar o que se passou.

5.4 Thread Sincronização Operações Pendentes

Para sincronizar as operações pendentes já existia uma *thread* de sincronização, no entanto foi adaptada aos novos argumentos guardados e o seu tempo de espera pode ser configurável através do ficheiro "*comm.xml*". A *thread* em causa consulta a lista de operações pendentes guardada na base de dados Db4o e executa as operações existentes, deixando-as na lista caso falhem.

Executar uma operação pendente é exactamente o mesmo que executar uma operação que foi pedida nesse momento pela interface gráfica. Uma vez guardados os argumentos da *InvocationClass* a Framework executa o serviço pedido com os argumentos e caso haja objectos a actualizar actualiza-os. O envio de objectos do Servidor para a BD local (Aplicação) faz com que se sejam resolvidos as inconsistências de dados locais.

5.5 Dados Locais

Assumindo que o dispositivo se encontra conectado à rede todas as suas acções são reflectidas no Servidor, caso contrário as operações locais são reflectidas na base de dados local. Desta forma o dispositivo móvel poderá continuar a operar em quaisquer circunstâncias de conexão.

Quando a aplicação recebe uma mensagem do tipo: "sem conectividade disponível para realizar a operação imediatamente, será realizada assim que a conectividade seja restabelecida" está na presença de um serviço *não crítico*, como tal neste caso, faz sentido reflectir as alterações de dados localmente. Este processo é iniciado pela aplicação, através de código personalizado, mas totalmente executado na Framework. A Framework é responsável por toda a interacção com a base de dados local, tanto ao nível de escrita como de leitura. Para realizar estas operações a aplicação só tem que definir no seu ficheiro "*comm.xml*" um serviço com quatro canais (*read*, *write*, *update* e *delete*) dos quais os canais de escrita e leitura sejam a base de dados local.

```
<comm id="comm2">
            <!-- channel faz o mapeamento com o xml de de
            <read type="localdb" path="\cliente.yap"  />
            <write type="localdb" path="\cliente.yap" />
            <delete type="localdb" path="\cliente.yap" />
            <update type="localdb" path="\cliente.yap" />
</comm>
```

Figura 5-8: Serviço no qual os quatro canais apontam para a base de dados local

Assim sempre que a aplicação mencionar o serviço *comm2* por consulta deste ficheiro a Framework sabe que tem que consultar a base de dados local.

As alterações verificadas anteriormente colocam a base de dados local de forma temporariamente inconsistente relativamente à base de dados do Servidor. Tal como referido anteriormente a inconsistência é resolvida por envio dos objectos actualizados sempre no sentido Servidor-Framework.

5.6 Sincronização de Dados Locais

Para além do envio de objectos actualizados, no momento exactamente posterior à execução das operações iniciadas na aplicação, do Servidor para a Framework existem mais duas formas de sincronizar as duas bases de dados. Através de um serviço disponibilizado pela Framework à própria aplicação ou através de uma *thread* (*geral*) que, com um intervalo de tempo definido, actualiza os dados locais com os dados do servidor. Esta operação é feita através da replicação de objectos para a base de dados local. Se na primeira opção a sincronização é feita a pedido do utilizador, na segunda esta é automática com o tempo de intervalo a poder ser "configurado" no ficheiro "*comm.xml".*

As três formas definidas de sincronização da base de dados local permitem ir de encontro a todas as necessidades encontradas. A *thread de sincronização geral* permite que constantemente o ambiente local esteja actualizado. A Framework invoca um serviço definido no ficheiro "*comm.xml"* como sendo o serviço de sincronização de objectos genérico e actualiza localmente esses objectos. A qualquer momento poderá existir alterações de objectos realizadas por outras aplicações no Servidor, desta forma é possível constantemente actualizar a base de dados local. Foi criado, ainda, um serviço na Framework que pode desactivar/activar esta *thread* de sincronização para que a aplicação possa tomar total controlo da sua actividade. Sendo a *thread* facultativa a Framework disponibiliza, a cada execução de uma operação, a possibilidade de se actualizar objectos na base de dados local para que seja possível garantir a sua consistência durante o maior tempo possível. Por último, foi criado um serviço no qual o utilizador pode actualizar a sua própria base de dados, assim o utilizador

garante que naquele exacto momento pode operar sobre uma base de dados completamente actualizada.

5.7 Grupo de Operações

Esta funcionalidade permite formar grupos de operações independentes que de certa forma estão interligadas entre si formando a mesma unidade lógica. O utilizador pode assim considerar que determinado número de operações que deseja executar estão de tal forma dependentes que cria um grupo de operações. Por conseguinte, no caso de uma das operações não poder ser realizada então todas delas não o poderão ser. Ou seja, este grupo de operações só deve acontecer no seu todo, ou por e simplesmente não acontecer.

A Framework ao criar o identificador único atribui um número de grupo aos primeiros três caracteres desse identificador.

Figura 5-9: Constituição do identificador único

O servidor, por intermédio do *handler* definido, verifica os primeiros três caracteres e se for do tipo definido está na presença de uma operação que faz parte de um grupo de operações. Nesse caso, a operação é executada guardando num *log* as alterações efectuadas à base de dados. Se for recebida uma outra operação do mesmo grupo que falhe a sua execução o Servidor através do *log* mencionado faz a compensação de todas os registos verificados.

Este mecanismo pode ser usado para diferentes fins. Se uma pessoa, por exemplo, utilizar o seu PDA para carregar o telemóvel mas se para isso tiver que realizar uma transferência de forma a creditar a conta com o valor necessário, o carregamento do telemóvel só se realiza caso a transferência se conclua com sucesso. Outro caso será, por exemplo, um aluno ter que entregar um trabalho a dois professores para avaliação, assim o sistema garante a entrega aos dois professores. No caso de não ser possível enviar a um deles então o outro também não recebe.

Finalmente é apresentada uma figura que demonstra a execução de alguns dos diferentes mecanismos apresentados anteriormente.

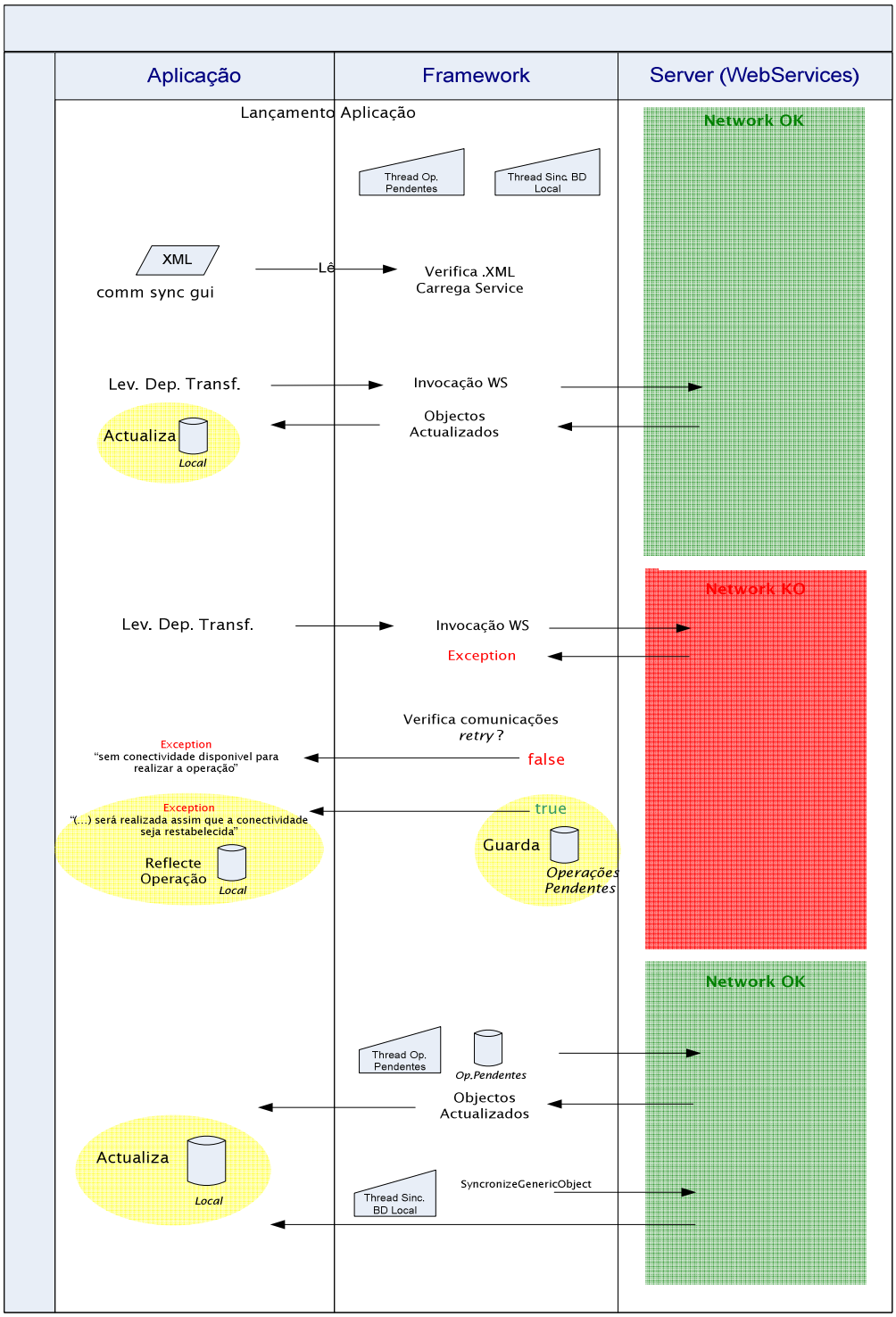

Figura 5-10: Interacções entre Aplicação-Framework-Servidor

6 Aplicação de Testes

A aplicação criada é uma aplicação móvel que simula interacções com uma entidade bancária a pedido do utilizador. As execuções no ambiente móvel interagem com uma plataforma de serviços, tecnologicamente implementados por *webservices*, que dá suporte às intenções dos utilizadores. Desta forma é impossível dissociar a aplicação de testes da camada "servidor" criada para acesso à base de dados. Assim, esta secção tem como objectivo descrever de forma clara a camada servidor, bem como, a aplicação de testes e a configuração dos respectivos ficheiros estritamente necessários ao seu funcionamento.

Figura 6-1: Interface da Aplicação

6.1 Funcionalidades e Mecanismos

Essas acções podem ser desde levantamentos a depósitos passando pela consulta dos dados dos clientes, consulta de saldos, consulta de movimentos, transferências, etc. Esta aplicação poderia ser utilizada pelos funcionários de um banco para tomarem acções sobre as contas dos clientes.

Foram desenvolvidas diferentes operações possíveis tais como:

- Consultas
    - Detalhes dos Clientes
    - Saldos
    - Movimentos
- Transacções
    - Levantamentos
    - Depósitos
    - Transferências

Uma vez que a aplicação foi desenvolvida para um sistema móvel foi também necessário capacitar a aplicação com outros mecanismos, tais como:

- Capacidade de armazenar “Operações Pendentes”;
- Relatório/*Log* de erros na sincronização das “Operações Pendentes”;
- Reflectir na base de dados local todas as operações, inclusivamente as que por razões de desconexão não se puderam efectivar no servidor (estas aparecem como “NOT SYNC” na aplicação);
- Cancelar/Eliminar “Operações Pendentes” com respectiva reposição do estado anterior da base de dados local;
- Sincronização (BD Local com BD Servidor).

De forma a manter completamente sincronizado o dispositivo móvel com o servidor, foram criadas *threads* com intervalos configuráveis que sincronizam automaticamente não só a base de dados local à aplicação como as operações que momentaneamente não puderam ser descarregadas no servidor.

Para que se possa efectivar o processo genérico de sincronização de dados, através da *thread*, é necessário que a aplicação defina um *webservice* com o nome "*synchronize*". Este nome colocado por defeito permite à Framework invocá-lo e carregar a BD local com os dados actualizados do servidor.

Por fim, o conceito "grupo de operações" também necessita um mecanismo na aplicação que possibilite a sua activação/desactivação. Como tal, sumariamente, os mecanismos são:

- Activar/Desactivar sincronização automática "Operações Pendentes";
- Activar/Desactivar sincronização automática da BD Local com BD Servidor;
- Activar/Desactivar "Grupo de Operações".

Figura 6-2: Interface da Aplicação (Configurações)

6.2 Estrutura

Tal como referido na secção Funcionamento do capitulo intitulado por Sincronização, no *bundle* criado (ficheiro JAR) representativo da Aplicação encontram-se diferentes ficheiros importantes, tais como, o ficheiro de configuração OSGI (*bundle-manifest*), código necessário ao registo do *bundle* OSGI, ficheiros de configuração da Framework ("*comm.xml"*, "*gui.xml"*, "*sync.xml"*) e código que a aplicação necessite.

Foi criado um esqueleto funcional da aplicação para a Framework GAM consistindo numa interface gráfica simples e capaz de invocar *webservices* descritos. O esqueleto necessário à execução de uma aplicação sob a Framework passa por:

- ***layout's.xswt***: ficheiros de descrição da interface gráfica;
- ***gui.xml***: configuração das acções da aplicação e suas funcionalidades;
- ***comm.xml***: configuração das comunicações (locais, *webservices*);
- ***Activator.java***: Classe de activação do *bundle*. Implementa a classe BundleActivator do package org.osgi.framework;
- ***MobileAplication.java***: Classe de implementação do serviço *MobileApplication* para a *framework* poder obter os ficheiros de configuração da aplicação. Implementa a classe AbstractMobileApplication do *package pt.link.gam* do gamBase (*bundle* que contem e disponibiliza classes base da *framework* para serem usadas por outros *bundles*);
- ***CustomClasses.java***: Classes necessárias à execução de código "customizado" pela aplicação;
- ***bundle.manifest***: ficheiro a ser usado para empacotar dos ficheiros da aplicação num *bundle* OSGI.

layout's.xswt

Foi utilizada a biblioteca implementada pela Framework GAM – "Standard Widget Toolkit" (SWT), e a linguagem de programação XSWT. A linguagem XSWT descreve as interfaces por intermédio de ficheiros XML (*eXtensible Markup Language*) [27] sendo a mais popular no âmbito da biblioteca gráfica SWT [4].

```
<xswt xmlns:x="http://sweet_swt.sf.net/xswt">
    <x:import>
        <package name="java.lang"/>
        <package name="org.eclipse.swt.widgets"/>
        <package name="org.eclipse.swt.layout"/>
        <package name="org.eclipse.swt.graphics"/>
    </x:import>

    <getShell text="eBanka GMC" />
    <x:children>
    <composite>

        <layoutData x:class="gridData"
        grabExcessHorizontalSpace="true"
        grabExcessVerticalSpace="true"
        horizontalAlignment="GridData.CENTER"
        verticalAlignment="GridData.CENTER"/>

        <x:children>
            <button x:id="btnClientes" text="Clientes" bounds="0 0 180 30" />
            <button x:id="btnTransaccoes" text="Transaccoes" bounds="0 40 180 30" />
            <button x:id="btnSincronizacao" text="Sincronizacao" bounds="0 80 180 30" />
            <button x:id="btnConfiguracoes" text="Configuracoes" bounds="0 120 180 30" />
        </x:children>

    </composite>
    </x:children>
</xswt>
```

Figura 6-3: Exemplo ficheiro representativo de uma interface gráfica. XSWT

Não foi considerada nenhuma outra biblioteca, bem como nenhuma outra linguagem de programação visto que ambas as soluções correspondem plenamente aos objectivos propostos. Os ficheiros XML são, assim, carregados pela Framework GAM, através da sua componente guiManager, para o seu ambiente e traduzidos para código Java utilizando-se a biblioteca SWT descrita anteriormente. Tal como referido no capitulo da Sincronização (secção Arquitectura) o componente guiManager efectua o carregamento das configurações das interfaces e das acções associadas a cada elemento da interface contido no ficheiro *gui.xml*. É também responsável pela invocação de outros componentes que forneçam determinados serviços ou de código "customizado" pela aplicação.

gui.xml

Contem as acções associadas a cada elemento da interface, nomeadamente lançamento de outras interfaces gráficas ou métodos próprios desenvolvidos pela aplicação. Quando o utilizador interage com a aplicação a componente guiManager consulta as configurações existentes neste ficheiro e age em conformidade, lançando uma nova interface ou invocando código da aplicação.

```
<interface id="frmPrincipal" filename="forms/frmGestaoMicrocreditosPrincipal.xswt">
    <object id="btnClientes" type="button" event="onClick">
        <action type="newChildDialog">
            <param value="frmClientes" />
        </action>
    </object>
    <object id="btnTransaccoes" type="button" event="onClick">
        <action type="newChildDialog">
            <param value="frmTransaccoes" />
        </action>
    </object>
    <object id="btnSincronizacao" type="button" event="onClick">
        <action type="newChildDialog">
            <param value="frmSincronizacao" />
        </action>
    </object>
    <object id="btnConfiguracoes" type="button" event="onClick">
        <action type="newChildDialog">
            <param value="frmConfiguracoes" />
        </action>
    </object>
</interface>
```

Figura 6-4: Invocação de outras interfaces gráficas (“*gui.xml*”)

```
<interface id="frmClientes" filename="forms/frmDownloadDadosClientes.xswt">
    <object type="shell" event="onLoad">
        <action type="invoke">
            <param name="class" value="pt.link.gam.custom.esynctest.CustomClasses" />
            <param name="method" value="carregaClientes" />
        </action>
    </object>
    <object id="btnPesquisa" type="button" event="onClick">
        <action type="invoke">
            <param name="class" value="pt.link.gam.custom.esynctest.CustomClasses" />
            <param name="method" value="pesquisaClientes" />
        </action>
    </object>
```

Figura 6-5: Exemplo de invocação de métodos “costumizados” pela aplicação (“*gui.xml*”)

comm.xml

Configurações das comunicações entre a aplicação e a Framework GAM. Neste ficheiro encontram-se informações necessárias à invocação de *webservices* (IP, porta, localização dos *stubs,* argumentos necessários aos respectivos serviços) e invocações locais. A componente CommAgent responsável pelas comunicações quando invocada pela aplicação consulta o ficheiro “*comm.xml*” de forma a saber o tipo de invocação (local ou remota) e confirmar os o tipo de argumentos de invocação.

Foi ainda desenvolvida na Framework GAM uma *thread* de sincronização geral que actualiza os objectos contidos na base de dados local à aplicação. A parametrização do intervalo de tempo da

thread pode ser definido neste ficheiro. A figura seguinte demonstra as características mencionadas anteriormente.

```
<config id="ebgmc">
            <wsdl host="127.0.0.1" port="8080" /><!-- 127.0.0.1 -->
            <locator value="pt.link.gam.ws.bank.EBGMCServiceServiceLocator" />
            <port value="pt.link.gam.ws.bank.EBGMCService_PortType" />
            <portMethod value="geteBGMCService" />
</config>

<service id="transferencia" config="ebgmc" retry="true">
        <method value="transferencia" />
        <arg name="id" type="string" />
        <arg name="clt" type="string" />
        <arg name="user" type="string" />
        <arg name="pass" type="string" />
        <arg name="nib" type="string" />
        <arg name="numClienteDest" type="string" />
        <arg name="numContaIn" type="string" />
        <arg name="valor" type="double" />
</service>


<thread id="yes" time="15000" />

<comm id="transferencia"> <!-- proxys sao definidos como propiedades de sistema lidas pelo axis -->
            <!-- read types = ws, get --> <!-- protocols (ws = http(s)) (get = http(s),ftp) -->
            <read type="ws" path="transferencia" /> <!-- invocacao remota WS -->
            <write type="comm" path="comm2" /> <!-- usa o repositorio local -->
            <delete type="comm" path="comm2" /> <!-- usa o repositorio local -->
            <update type="comm" path="comm2" /> <!-- usa o repositorio local -->
</comm>
```

Figura 6-6: Ficheiro "*comm.xml*"

6.3 Serviços

Todo o fluxo de informação passa pela Framework. Sejam leituras, escritas, actualizações de dados, invocação de *webservices*, sincronização, etc. De forma a providenciar todos estes serviços foram desenvolvidos serviços que possibilitam a interacção entre a aplicação e a Framework.

Seguidamente é apresentada uma tabela com todos esses serviços, bem como os argumentos necessários à sua invocação:

Serviço	*Argumentos*	*Descrição*
doRead	InvocationClass	Leituras locais/remotas
doWrite	InvocationClass	Escritas BD local
doUpdate	InvocationClass	Actualizações BD local
doDelete	InvocationClass	Eliminar (Dados) BD local
Synchronize	-	Sincronizar BD local com BD servidor
getPendingOps	-	Obter “operações pendentes”
removePendingOp	String	Apagar “operação pendente”
getProblemPendingOps	-	Verificar *report*s erro “operações pendentes”
pendingOperations_ONOFF	-	Activar/Desactivar sincronização automática de “operações pendentes”
syncThread_ONOFF	-	Activar/Desactivar sincronização automática da BD Local
group_ONOFF	-	Activar/Desactivar opção “grupo mensagens”

Tabela 1 – Serviços

7 Conclusões

O trabalho realizado iniciou-se com um estudo sobre o ambiente onde se inseria o projecto designadamente o ambiente móvel, nomeadamente os constrangimentos inerentes a este tipo de ambiente e, mais concretamente com os objectivos propostos, analisado os conceitos sob os quais se regem os sistemas de bases de dados móveis.

Ainda nesta fase foi estudada a *framework* desenvolvida no trabalho anterior. Toda a *framework* foi analisada e estudada, no entanto, de acordo com os objectivos deste projecto, foi dado maior atenção aos mecanismos de sincronização desenvolvidos. Esta análise foi muito produtiva uma vez que se revelou um bom ponto de partida tanto ao nível de código desenvolvido como de tecnologia utilizada. Este trabalho acaba por utilizar grande parte da tecnologia utilizada pela *framework* anterior sendo a alteração a este nível mais significativa a aproximação aos sistemas legados utilizados pelas maior parte das organizações com a utilização de um sistema de gestão de base de dados como “servidor” e a definição de uma arquitectura de testes com camadas bem definidas de acesso a dados. A maior parte do trabalho realizado acaba por ser a definição e criação de novas funcionalidades de sincronização de dados ou alteração de funcionalidades existentes na Framework GAM.

Uma das dificuldades encontradas prendeu-se com o facto de todo o desenvolvimento ter que ser feito de uma forma *standard* para servir qualquer tipo de aplicação. Ou seja, as funcionalidades definidas de sincronização de dados teriam de ser normalizadas a ponto de servirem as necessidades de diferentes tipos de aplicações.

A nível tecnológico é necessário realçar o uso da *framework* OSGI. Se a nível de desenvolvimento dos objectivos deste trabalho nem sempre houve problemas na sua utilização rapidamente se verificou que teria que haver uma nova forma de pensar antes de desenvolver alguma coisa. A fraca e, ao mesmo tempo forte dependência entre componentes existente no conceito OSGi, referido na secção “Arquitectura”, cria um modelo de cooperação tão importante, entre componentes, que “obriga” a repensar todo o desenvolvimento.

Para além do OSGI e da persistência de dados para o ambiente móvel Db4o nunca terem sido usados, a utilização de *webservices*, através do projecto Axis, a persistência de dados com MySQL, ou mesmo a utilização de Hibernate para mapeamento objecto-relacional já tinham sido anteriormente utilizados em projectos académicos. Assim, a maior dificuldade encontrada foi interligar as diferentes tecnologias, pensando em componentes que fornecem serviços entre si, e criar uma solução genérica que permitisse oferecer funcionalidades de sincronização de acordo com os objectivos propostos.

Os objectivos definidos inicialmente acabam por ser totalmente alcançados. Persistência local para que o dispositivo continue a operar quando desconectado da rede, constante actualização da base de dados local, noção de operações pendentes quando desconectado e sincronização destas com o servidor, total controlo das operações pendentes por parte da aplicação são algumas das funcionalidades desenvolvidas por este trabalho que permitem às aplicações terem dados consistentes e coerentes necessários às suas operações. No entanto a plataforma pode evoluir em diferentes sentidos. Na secção seguinte é apresentada uma possível direcção a seguir.

Em retrospectiva o trabalho realizado começou por ser extremamente complicado quando confrontado com a realidade da framework OSGI e o trabalho realizado anteriormente. Sendo o trabalho realizado em contexto de estágio na Link Consulting as responsabilidades também aumentaram pelo que de certa forma foi com enorme motivação que o projecto foi evoluindo, sendo cada pequeno passo entendido com muito maior dimensão.

7.1 Trabalho Futuro

Existem algumas direcções que podem ser seguidas de forma a acrescentar valor à plataforma desenvolvida. Tendo sempre em linha de pensamento a sincronização de dados, uma das hipóteses seria a partilha de dados entre diferentes dispositivos móveis. Nesse sentido a Framework teria de ter a capacidade de comunicar com outras *frameworks* e, assim, resolver alguns problemas, como por exemplo o servidor estar em "baixo". Neste caso, dispositivos que pudessem partilhar entre si dados poderiam faze-lo sem necessitar dos dados do servidor. Outro dado importante, seria um dispositivo poder sincronizar os seus dados com um outro dispositivo que acabou de receber dados do servidor e, desta forma, aliviar as comunicações do próprio servidor.

Outro ponto interessante seria desenvolver e capacitar a Framework GAM com um componente (*bundle*) que permitiria localizar o dispositivo móvel por coordenadas geodésicas (GPS – "Global Positioning System"). Desta forma o dispositivo poderia "ver alimentada" a sua base de dados local com dados dependentes da localização física. Assim qualquer aplicação poderia fazer uso deste mecanismo até para poder optimizar o uso da sua base de dados local.

8 Referências

[1] D. Henriques, Pedro Pita. *Gerador de Aplicações Móveis, Relatório do Projecto Final de Curso*. Instituto Superior Técnico, 2005/2006

[2] OSGi Alliance. *The OSGi Service Platform – The Dynamic Module System for Java* http://www.osgi.org

[3] Knopflerfish Open Source OSGI

http://www.knopflerfish.org/

[4] SWT Library

http://www.developers.net/tsearch?searchkeys=SWT+library

[5] The Visual Editor - XSWT
http://xswt.sourceforge.net

[6] Webservices – Axis Project
http://ws.apache.org/axis/

[7] Simple Object Access Protocol (SOAP) Specifications

http://www.w3.org/TR/soap/

[8] Oracle Database 11g

http://www.oracle.com/database/index.html

[9] Microsoft SQL Server

http://www.microsoft.com/sql/default.mspx

[10] MySQL – *Open Source Database*

http://www.mysql.com/

[11] Db4o – *Native Java & .NET Object Database*

http://www.db4o.com/

[12] PointBase – *Java Database, Mobile Database, Embedded Java Database*

http://www.pointbase.com/

[13] FastObjects

http://www.versant.com/en_US/products/fastobjects

[14] IBM WebSphere Everyplace Access

http://www-306.ibm.com/software/pervasive/ws_everyplace_access/

[15] Patricia Serrano-Alvarado, Claudia Roncancio, Michel Adiba: A Survey of Mobile Transactions, Distributed and Parallel Databases, 16, 193-230, 2004, Kluwer Academics Publishers

[16] Jin Jing, Abdelsalam Helal, Ahmed Elmagarmid: Client-Server Computing in Mobile Environments, ACM Computing Surveys (CSUR) 31 Issue 2 117-157, June 1999

[17] Tomas Imielinski and B. R. Badrinath: "Data Management for Mobile Computing", ACM SIGMOD Record Volume 22, Issue 1 Pages 34-39, March 1993

[18] Silvia Maria Rodrigues da Cunha: Sistemas de Bases de Dados Móveis, Dissertação apresentada à Universidade do Minho em Informática Informática, na especialidade de Sistemas Distribuídos, Comunicações por Computador e Arquitectura de Computadores, 2004

[19] Tomas Imielinski and B. R. Badrinath: "Data Management for Mobile Computing", ACM SIGMOD Record Volume 22, Issue 1 Pages 34-39, March 1993

[20] Margaret H. Dunham, Abdelsalam (Sumi) Helal: MOBILE COMPUTING and DATABASES:ANYTHING NEW?, SIGMOD Record, Vol. 24, No. 4, December 1995

[21] Panos K. Chrysanthis. Transaction processing in mobile computing environment. In IEEE Workshop on Advances in Parallel and Distributed Systems, pages 77-82, October 1993

[22] Web Services Reliability (WS-Reliability) Ver1.0, January 8, 2003

[23] Hibernate – Relational Persistence for Java and .NET

http://www.hibernate.org/

[24] Hibernate: "Hibernate Reference Documentation" Version: 3.2.2

[25] MyEclipse – *Eclipse plugin development tools for Java, JSP, XML, Struts, HTML, CSS and EJB*

http://www.myeclipseide.com/

[26] Eclipse – *Open Development Plataform*

http://www.eclipse.org/

[27] XML – *What is XML?*

http://www.xml.com/pub/a/98/10/guide0.html

9 Anexos

9.1 Open Services Gateway Iniciative (OSGI)

A "OSGI Alliance" é um consórcio mundial de empresas que aposta na tecnologia, especialmente num processo suficientemente maduro, que assegura a interoperabilidade de aplicações e serviços baseado na sua plataforma de integração de componentes. A Plataforma de Serviços OSGI [2] – "*OSGI Service Plataform*" – é usada em produtos e serviços de muitas das 100 maiores empresas a nível mundial nos mais diversos mercados desde móvel, fixo, sistemas de telemetria, etc.

A sua missão é:

"(…) create a market for universal middleware. The OSGi Alliance, therefore, promotes widespread adoption of the OSGi Service Platform to assure interoperability of applications and services delivered and managed via networks."

A aliança fornece especificações, referências de implementação, casos de teste e certificações para promover um ecossistema favorável à utilização da plataforma em diferentes indústrias. Todas as companhias que fazem parte deste consórcio colaboram no mesmo ambiente promovendo a adopção da tecnologia OSGI através de fóruns, experiências dos próprios utilizadores e vantagens da sua utilização no próprio negócio.

A opção por uma plataforma baseada em componentes – *bundles* – reduz o tempo de chegada ao mercado dos produtos e diminui os custos de desenvolvimento, uma vez que, permite a integração de módulos previamente testados. Reduz, também, os custos de manutenção e providencia oportunidades já depois de estar o produto no mercado porque as redes podem ser usadas para dinamicamente actualizar e/ou disponibilizar novas aplicações e serviços ao sistema.

Os membros da Aliança representam diversos mercados como o mercado móvel, comercial, componentes electrónicos para automóveis, etc. Representam inúmeras indústrias como empresas que disponibilizam acesso à Internet, lideres no fornecimento de serviços, *software-houses*, fornecedores de equipamentos electrónicos (*wireless* ou fixos) ou, mesmo, instituições de investigação.

A Alianças OSGI é uma organização sem fins lucrativos fundada em Março de 1999.

9.1.1 O Problema

A complexidade do software está a aumentar a uma velocidade alarmante. Hoje, uma grande parte desta complexidade é causada pelo breve e curto ciclo de vida dos produtos, requisito mais que suficiente para um dramático aumento da funcionalidade e um aumento do número de variações do mesmo produto (*ex.*: diferente hardware e sistemas operativos). Estas tendências têm causado custos de software a ponto de se tornar uma percentagem considerável de quase todo o custo de desenvolvimento dos fabricantes.

Nos dias que correm, o desenvolvimento de software consiste normalmente na adaptação de funcionalidades já existentes de forma a alcançar um novo ambiente. Nos últimos 20 anos, uma grande parte dos "*standard building blocks*" ficaram disponíveis e são bastante utilizados em todos os produtos; um exemplo visível é o sucesso do "*open software*". No entanto, a utilização dessas bibliotecas não se faz sem problemas. Integrar diferentes tipos de bibliotecas pode tornar-se bastante complicado porque muitas bibliotecas têm se tornado complexas e chegam a necessitar de funcionalidades delas próprias para operar, mesmo que as funcionalidades incluídas nunca sejam sequer necessárias para o produto.

Na realidade, integrar código existente tornou-se a grande parte do trabalho dos programadores de software. Por consequência, são necessárias ferramentas que criem *standards* dos aspectos de integração de software para que a reutilização de "componentes" se torne confiável, robusta e barata.

9.1.2 A Solução

A tecnologia OSGI oferece um dinâmico sistema modular para a linguagem Java. A plataforma de serviços OSGI disponibiliza funcionalidades ao ambiente Java que torna esta linguagem, ela própria, o primeiro degrau para a integração de software e, como tal, para o seu desenvolvimento. Java permite a portabilidade que é necessária para suportar diferentes produtos em diferentes plataformas. A tecnologia OSGI dá as primitivas *standards* que permite às aplicações serem construídas a partir de componentes pequenos, reutilizáveis e que podem colaborar entre si.

A plataforma dispõe de funções que podem alterar a composição dinamicamente no próprio dispositivo móvel, sem que este tenha sequer que reiniciar. De forma a minimizar a dependência entre componentes, bem como poder gerir essas ligações, a tecnologia OSGI oferece uma arquitectura orientada a serviços que permite que esses componentes se possam descobrir dinamicamente uns aos outros. A Aliança OSGI tem desenvolvido muitas interfaces de componentes *standards* para funcionalidades comuns como servidores *HTTP*, configuração, *logs*, segurança, administração de utilizadores, *XML* e muitos outros. Diferentes tipos de implementações destes componentes podem ser obtidos dos próprios vendedores com diferentes optimizações e custos. No entanto, interfaces de serviços podem ser desenvolvidas no proprietário.

A tecnologia OSGI tem alguns pontos a favor no que diz respeito ao tempo de resposta ao mercado e na redução de custos de desenvolvimento porque a plataforma oferece uma base completamente estabilizada para integração de sub-componentes previamente testados. A tecnologia inerente poderá permitir a redução de custos de manutenção e permitir boas oportunidades já depois dos produtos estarem no mercado porque os componentes podem ser dinamicamente instalados nos equipamentos mesmo no terreno.

9.1.3 Framework

O núcleo da especificação OSGi é a sua *framework*. Esta fornece um ambiente *standard* para aplicações (designadas *bundles*). A *framework* está dividida em várias camadas (Figura 9-1) e adicionalmente ao descrito na figura, existe também um sistema de segurança fortemente interligado com todas as camadas.

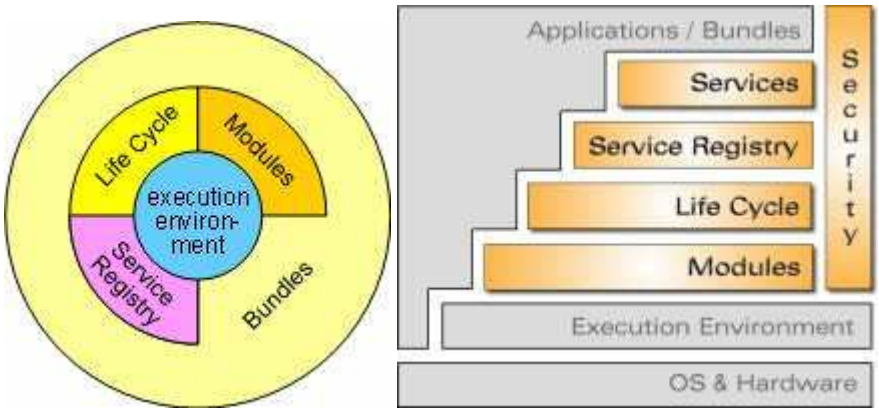

Figura 9-1: Modelo de camadas da Framework OSGI

A camada mais baixa – "*Execution Environment*", é a especificação do ambiente Java. Configurações e perfis da plataforma Java2.

A camada de módulos – *Modules*, específica as políticas de carregamento de classes. A *framework* OSGi tem um modelo de carregamento de classes bastante rígido baseado em Java mas adiciona modularização. Em Java, existe apenas normalmente um único *classpath* que contém todas as classes e recursos necessários. A camada de módulos do OSGi adiciona classes privadas a um módulo e a possibilidade de controlar a ligação entre módulos. Os *bundles* necessitam da camada de módulos para o carregamento de classes.

A camada de ciclo de vida – "*Life Cycle*", gere o ciclo de vida dos *bundles*, permitindo que estes sejam instalados, iniciados, parados, actualizados e removidos, sempre dinamicamente. Esta camada introduz dinamismo às aplicações e, ao mesmo tempo, garante o funcionamento correcto do ambiente de operação através de mecanismos de dependências.

A camada de "*Service Registry*" fornece um modelo cooperativo para os *bundles*. Os *bundles* podem cooperar através do mecanismo habitual de partilha de classes, embora tal não seja muito compatível com a instalação e remoção dinâmica de código. Este "*Service Registry*" fornece um modelo compreensivo para a partilha de objectos entre *bundles*. É definido um modelo de eventos, sendo definidos eventos para processar a entrada e saída de serviços. Neste contexto, serviços são apenas objectos Java que podem representar qualquer coisa do mundo real. Muitos destes são actualmente servidores, enquanto que outros representam objectos do mundo real.

Por cima da *framework*, a "OSGi Alliance" especificou vários serviços de base para permitir o funcionamento da *framework*, bem como as funcionalidades básicas de cada serviço disponibilizadas para as aplicações correndo sobre a *framework*.

As especificações OSGi poderão ser aplicáveis devido a tratar-se de uma pequena camada que permite que vários componentes baseados em Java cooperem eficientemente numa única máquina virtual de Java. Ao mesmo tempo, fornece um modelo exaustivo de segurança para que estes componentes possam correr num ambiente isolado, mas, que com as permissões adequadas também possam ser reutilizados e cooperar entre si. A *framework* OSGi fornece um conjunto bastante exaustivo de mecanismos para possibilitar esta cooperação e torná-la segura.

9.2 Persistência de Dados

Seguidamente são apresentados outros sistemas de gestão de bases de dados analisados para persistência de dados em ambiente móvel.

9.2.1 Ambiente Móvel

PointBase

É uma base de dados relacional criada para sistemas móveis. Directamente imbebível em aplicações móveis, pode operar numa unidade móvel incluindo computadores portáteis, *table* PCs e PDAs. O *PointBase* é completamente programado em Java com um *footprint* (ficheiro *Jar*) entre *45KB* e *90KB* para a versão *micro* e *1MB* para a versão *embedded* [12].

A versão micro é uma base de dados relacional independente, completamente em *Java*, optimizada para correr no *Java 2 Micro Editio*n e plataformas *J2SE*. É a solução para ser executada em computadores portáteis ou PDAs. Usando *PointBase UniSync*, as bases de dados podem ser sincronizadas bi-direccionalmente com bases de dados *Oracle* ou "*Microsoft SQL Server"*.

FastObjects j2

FastObjects j2 é uma base de dados orientada a objectos [13]. Fornece componentes de base de dados imbebíveis num *package* de *450KB*. *F*ornece funcionalidades de gestão de base de dados, como a gestão da *cache* e transacções. Permite, também, o acesso *multi-thread* no modo *off-line* à base de dados através de transacções ACID (*Atomic*, *Consistent*, *Isolated*, *Durable*). A replicação de dados é efectuada com base numa "*shadow database*". Os dados podem ser modificados pelas transacções sendo essas alterações colocadas em *logs* e aplicadas à primeira base de dados.

WebSphere Everyplace e DB2 Everyplace

IBM tem desenvolvido estes produtos para a gestão de dados em ambientes móveis. *WebSphere Everyplace Access* permite aos terminais móveis o acesso a dados (e-mail, dados de aplicações de negócios). Entrega páginas *web* e aplicações *e-business* a PDAs e telefones portáteis. IBM também fornece o produto DB2 *Everyplace*, uma base de dados relacional com um *footprint* de *180KB* que garante a persistência local de dados nos terminais móveis. Contem um servidor de sincronização bi-direccional que sincroniza dados relacionais entre o equipamento móvel e fontes de dados fixas (*Oracle*, *Informix*, *Sybase*, MS *SQL Server* e *Lotus Domino*). Suporta "*flat transactions*" (*commit*, *auto-*

commit e operações de *rollback*) e conexões à base de dados *multi-thread* (*ODBC*/*JDBC*) de forma "serializada" [14].

9.3 Mapeamento Objecto-Relacional

9.3.1 Hibernate

Ferramenta que faz o mapeamento objecto/relacional para ambientes Java. O termo mapeamento objecto/relacional refere-se à técnica de "mapear" uma representação de um determinado dado de um modelo orientado a objectos para um modelo relacional sob um esquema baseado em SQL. O Hibernate "mapeia" os objectos persistentes em XML que depois são acedidos pela aplicação [23][24].

Hibernate não só se preocupa com o mapeamento de classes Java em tabelas de base de dados (e de tipos de dados Java para tipos de dados SQL), mas também fornece *querys* que facilitam significativamente a leitura e reduzem o tempo de desenvolvimento, uma vez que não é necessário consumir tempo com manipulação de dados em SQL e JDBC.

Esta ferramenta utiliza o "*runtime reflexion*" do JAVA para determinar as propriedades persistentes de uma classe. Os objectos persistentes são definidos em documentos de mapeamento, que servem para descrever os campos persistentes e respectivas associações de cada objecto. Através desses documentos o Hibernate gera *stubs* em JAVA para realizar todo o tipo de operações na base de dados.

Como podemos verificar na figura seguinte o Hibernate é a plataforma que permite a comunicação entre a lógica de negócio com a base de dados e que mantém toda a persistência do servidor.

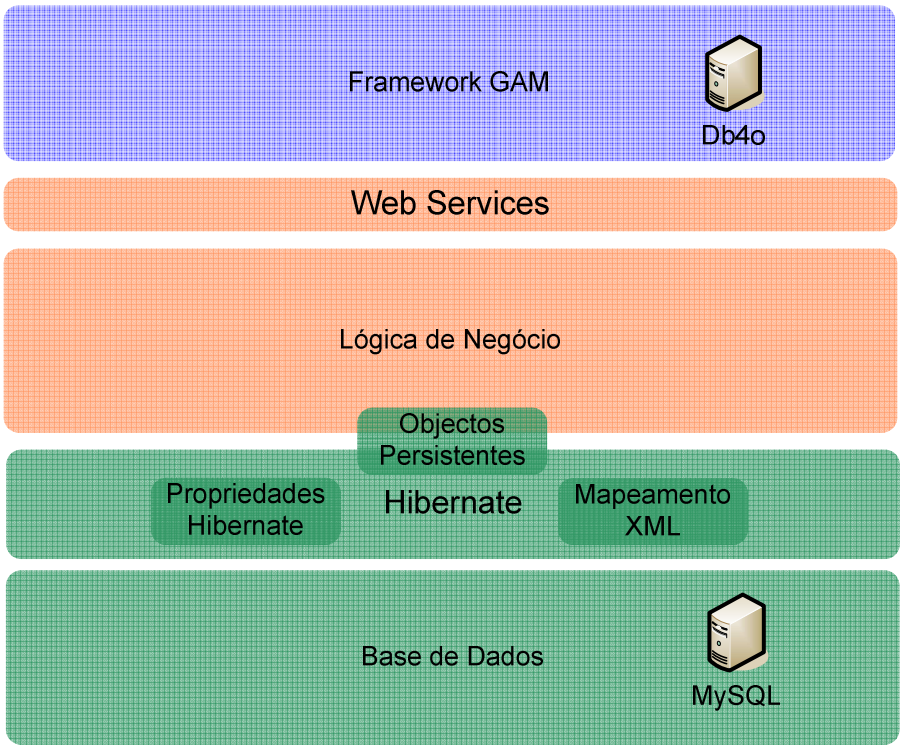

Figura 9-2: Camada *Hibernate*

O objectivo do Hibernate é reduzir o tempo de desenvolvimento relacionado com as tarefas da persistência de dados. É uma solução possível para modelos de domínio orientados a objectos e camadas de lógica de negócio em Java.