INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

OBIHOC

Middleware para Redes ad-hoc

André António Santos Conrado

Dissertação para obtenção do Grau de Mestre em

Engenharia de Redes de Comunicações

Júri

Presidente:	Prof. Rui Jorge Morais Tomaz Valadas
Orientador:	Prof. Paulo Jorge Pires Ferreira
	Prof. Luís Manuel Antunes Veiga
Vogais:	Prof. Artur Miguel do Amaral Arsénio

Outubro 2010

Resumo

O aparecimento de novos dispositivos móveis, com melhorias significativas em termos de processamento, armazenamento e capacidade de comunicação sem fios, sugere a possibilidade de criação de redes *ad-hoc*. Face a esta possibilidade, a necessidade de desenvolver aplicações para estas redes aumenta. No entanto este desenvolvimento implica que os programadores dominem áreas para as quais, na maioria das vezes, não estão preparados. Assim este trabalho propõe a criação de um *middleware* que liberte os programadores do trabalho com essas áreas, implementando-as de forma eficaz e correcta - OBIHOC.

O middleware OBIHOC é uma solução que propõe um mecanismo de replicação de dados que permite manter inalteradas as funcionalidades das aplicações. A utilização deste mecanismo permite uma gestão mais eficiente dos recursos usados pela aplicação (por exemplo memória) e a possibilidade de utilização num ambiente *ad-hoc*, recorrendo a várias soluções de comunicação sem fios (*Wi-Fi* e *Bluetooth*). Para além disso implementa um mecanismo de DGC coerente na presença de réplicas bem como a utilização da linguagem SPL, para especificação e execução de políticas de segurança.

A implementação deste mecanismo de replicação numa aplicação, baseia-se na adição de código e extensão do código das classes da aplicação. Deste modo permite a interacção entre a aplicação e as funcionalidades do *middleware*.

Este documento apresenta o trabalho de investigação sobre diversos aspectos técnicos que o *middleware* deverá cumprir: flexibilidade de paradigmas, replicação automática, *distributed garbage collector* e políticas de segurança. Para além disso é apresentada a arquitectura implementada e a avaliação da solução.

Palavras-chaves: middleware, replicação, rede ad-hoc, flexibilidade de paradigmas, distributed garbage collector, políticas de segurança

Abstract

The appearance of new mobile devices with significant improvements in terms of processing, storage and wireless communication capacity, suggests the possibility of creating ad-hoc networks. In view of this possibility, the need of developing applications for these networks increases. However this development implicates that programmers master areas of which, sometimes, they are not prepared for. Therefore this work proposes the creation of a middleware that releases the programmers of these duties, implementing them in a correct and effective way – OBIHOC.

The OBIHOC middleware is a solution that proposes a data replication mechanism that allows maintaining unaltered the application's functions. The use of this mechanism allows a more efficient management of the resources used by the application (e.g. memory), as well as the possibility of using it on an ad-hoc environment, supported by multiple wireless communication solutions (*Wi-Fi* and *Bluetooth*). Furthermore, it implements a DGC mechanism coherent in the presence of replicas as well as the use of SPL language, for specification and security policies implementation.

The implementation of this replication mechanism on an application is based in the addition of code and extension code of the application classes, in order to allow the interaction between this and the middleware functions.

This document presents the research work about various technical aspects that the middleware should satisfy: Flexibility of paradigms, automatic replication, distributed garbage collector and security policies. Beyond that, the implemented architecture is presented and also the solution's evaluation.

Keywords: Middleware, Replication, ad-hoc, Paradigm Flexibility, Distributed Garbage Collector, Security Policies

Índice

Lista de figuras

1. Introdução

Num mundo onde a tecnologia avança a passos largos, os desenvolvimentos técnicos que vão surgindo são inúmeros. São caso prático os computadores, sem os quais a realidade que hoje conhecemos seria totalmente distinta.

Com os diversos avanços tecnológicos, os paradigmas de utilização destes dispositivos sofreram imensas mutações. Um dos paradigmas mais em voga é a mobilidade, ou seja, os utilizadores deixaram de ter de se movimentar para junto do seu computador pessoal de secretária, passando esses dispositivos a estar permanente com os utilizadores. São caso prático, e de grande uso, os telemóveis. Quem não tem um?

Este novo paradigma trouxe a necessidade de novas funcionalidades e, em contra partida, novos problemas como as limitações destes dispositivos ao nível energético, processamento e memória. Para além destas limitações surgem outras prioridades como a capacidade de estar sempre contactável.

Para cumprir esta prioridade, actualmente estamos dependentes de redes estruturadas já existentes – Wi-Fi [1], Bluetooth [2], GPRS [3]. Esta limitação fixa bastante a capacidade de se criar uma rede personalizável, isto é, independente do serviço genérico que é oferecido a todos os aderentes a essa rede bem como da sua disponibilidade. Para além disso, estas redes na sua maioria têm associados custos directos para o utilizador.

Imaginemos o seguinte cenário: duas pessoas estão num jardim e uma delas pretende partilhar alguns conteúdos da sua biblioteca digital. Nesse local não têm acesso a nenhuma rede infra-estruturada externa. No cenário actual seria complexo realizar esta desejada troca de ficheiros. Mas se os ficheiros estão nos dispositivos móveis e estes dispositivos estão ao alcance um do outro, qual a necessidade de conectar-se a uma rede externa se a pessoa com quem quero trocar os ficheiros está mesmo ao meu lado?

Tendo em conta o cenário anterior alguns utilizadores experientes poderiam debelar esta dificuldade. Mas se avançarmos para cenários ligeiramente mais complexos as dificuldades de resolução aumentam bastante. Com tudo isto é fácil conceber a necessidade de desenvolver soluções que permitam aos utilizadores realizarem tarefas que envolvam dispositivos próximos entre si, sem a necessidade de uma rede infra-estruturada.

Face a este cenário, é natural considerar que as redes ad-hoc potenciam o desenvolvimento de novas aplicações. Contudo, o desenvolvimento destas aplicações é complexo, pois é necessário lidar com diversos problemas em camadas de baixo nível do sistema – *system-level* – para as quais a maioria dos programadores não está preparado. É de realçar que mesmo

face a estas dificuldades os programadores poderiam resolver estes entraves, todavia, na maioria dos casos, essas soluções não seriam as mais eficientes/eficazes. Para além disso, os programadores seriam obrigados a desviarem-se da lógica da aplicação, para a qual estão preparados e na qual se devem concentrar.

1.1. Roadmap

Este relatório está organizado no seguinte formato: no capítulo 3 estão descritos alguns aspectos técnicos relacionados com os objectivos do trabalho e alguns sistemas existentes. No capítulo 4 é apresentada a arquitectura seguido da sua implementação no capítulo 5. No capítulo 6 são apresentados os métodos para a avaliação da arquitectura implementada e os resultados obtidos. Por fim, o capítulo 7 apresenta as conclusões de todo o trabalho desenvolvido.

2. Objectivos

Para os dispositivos móveis poderem oferecer novas funcionalidades estão dependentes da criação de novos programas. Durante este processo, os programadores têm de lidar com módulos de sistema fora da lógica da aplicação. Ora esta necessidade provoca dois constrangimentos: 1) a necessidade de programadores com conhecimentos aprofundados em áreas para as quais não estão preparados, 2) a utilização de mecanismos pouco eficazes/eficientes.

Para colmatar estes problemas, este trabalho visa desenhar e implementar uma plataforma *middleware* que permita:

- **Flexibilidade de paradigmas:** permitir aos programadores desenvolverem aplicações usando RMI, replicação de objectos ou agentes móveis, de acordo com as especificações da aplicação.
- **Replicação Automática:** suportar a gestão automática de objectos replicados (replicação incremental).
- **Distributed Garbage Collection:** suporte automático à eliminação de réplicas desnecessárias.
- **Políticas de segurança:** suporte à definição e à execução de políticas de segurança, baseadas em eventos passados, adaptadas às necessidades dos agentes móveis.
- **Redes ad-hoc:** suporte à entrada e saída de dispositivos na rede.

O foco principal da implementação deste *middleware* está na componente de replicação automática numa rede ad-hoc, ou seja, de um ambiente móvel para outro móvel. Esta componente é de maior importância pois é fulcral para a utilização das restantes, isto é, sem esta componente correctamente implementada as restantes perdem a sua funcionalidade.

Deste modo o objectivo principal do trabalho está no desenho e implementação de um *middleware* que permita aos programadores desenvolverem aplicações orientadas a objectos (OO) para dispositivos móveis numa rede ad-hoc, sem que estes sejam obrigados a lidar com problemas que fogem da lógica da aplicação. Assim os programadores poderão utilizar funcionalidades correctamente implementadas sem as compreenderem, ou seja, sabem o que elas fazem, mas não como o fazem.

2.1. Requisitos

Para a correcta execução deste *middleware* ele obedece a determinados requisitos tais como:

- **Rede ad-hoc** – permitir a comunicação entre dispositivos móveis sem intervenção de nenhuma entidade externa.
- **Segurança** – controlo dos recursos a serem utilizados/disponibilizados;
- **Replicação** – permitir a replicação incremental;
- **Eficácia** – protocolo de comunicação com um número de mensagens trocadas reduzido;
- **Custos** – minimizar os custos energéticos, de processamento e de memória nos dispositivos móveis.

2.2. Contributo

Com a conclusão deste trabalho é possível constatar que o desenvolvimento de aplicações para redes ad-hoc foi simplificado. Com a integração deste middleware o programador poderá desenvolver a sua aplicação sem a necessidade de interagir com mecanismos fora da lógica da aplicação. Resumidamente este trabalho contribui para a que as aplicações possam:

- Ser integradas numa rede ad-hoc;
- Manter a correcta execução da aplicação quando o dispositivo sai da rede, ou outros dispositivos que a integram;
- Utilizar de forma eficiente os recursos no dispositivo;
- Implementar políticas de segurança;
- Manter um estado consistente na presença de réplicas;
- Aumentar as funcionalidades a serem disponibilizadas aos utilizadores.

3. Trabalho Relacionado

Para a análise do trabalho relacionado é necessário estudar diversos aspectos técnicos a ter em conta para o cumprimento de todos os objectivos propostos. Embora o presente trabalho esteja relacionado com redes ad-hoc, é impreterível que a investigação não se restrinja a este aspecto. Deste modo este capítulo está dividido de forma a abranger as diferentes áreas mencionadas nos Objectivos (ver secção 2). Na secção 3.1 são abordadas as redes ad-hoc e algumas das suas características. Na secção 3.2 são analisados vários paradigmas que permitem implementar o mecanismo de replicação. Na secção 3.3 são analisados diversos mecanismos para suportar a gestão automática de objectos replicados. Na secção 3.4 são analisados vários algoritmos para *distributed garbage collection*. Na secção 3.5 são analisadas algumas soluções que permitem implementar políticas de segurança. Por fim na secção 3.6 são apresentadas algumas soluções existentes.

3.1. Ambiente

Uma rede ad-hoc apresenta várias características que a diferencia dos restantes tipos de redes de comunicação. Algumas dessas características são [4]:

- **Autonomia** – possibilidade de criação de uma rede tendo em conta as necessidades da aplicação, não havendo portanto dependência de entidades externas. Desta forma qualquer dispositivo poderá criar uma rede ad-hoc.
- **Multi-hop** – reencaminhamento de tráfego. Como não existe nenhuma entidade central para fazer o reencaminhamento, são os dispositivos terminais que o têm de assegurar. Este reencaminhamento envolve a implementação de protocolos de encaminhamento, que estão fora do âmbito deste trabalho.
- **Topologia dinâmica** – capacidade de entrada e saída de dispositivos na rede, sem que esta perca as suas características. Este dinamismo aumenta a disponibilidade da rede, pois não existe dependência da permanência de nenhum dispositivo na rede.
- **Dispositivos heterogéneos** – possibilidade de participação na rede de diversos tipos de dispositivos. Qualquer dispositivo que reúna as características de hardware (comunicação sem fios) e software (protocolos de encaminhamento) pode entrar na rede e participar de forma construtiva na mesma.
- **Constrangimento de largura de banda –** não é possível garantir largura de banda estável comunicação entre dois dispositivos na rede. Devido ao multi-hop e à

heterogeneidade dos dispositivos não é possível garantir que todas as ligações obtenham o mesmo desempenho.

- **Segurança limitada** – dificuldade em implementar uma política de segurança devido à ausência de uma entidade externa que permita a autenticação. Outra limitação de segurança é o facto de a rede estar disponível para entrada e saída de qualquer dispositivo, mesmo os dispositivos mal intencionados.
- **Auto-gestão da rede** – capacidade da rede ser gerida pelos dispositivos que nela participam sem necessidade da presença de uma entidade central. Esta é das características mais importantes de uma rede ad-hoc.

Figura 1 - Exemplo rede ad-hoc

Tendo em conta estas características a utilização de uma rede ad-hoc é bastante aconselhada em situações onde não é possível aceder a uma rede estruturada externa. A Figura 1 representa um exemplo de uma rede ad-hoc, onde é possível visualizar algumas das características acima referidas (autonomia e dispositivos heterogéneos)

Devido à sua auto-gestão é uma rede de criação rápida e com pouca intervenção, ao contrário de uma rede estruturada onde é necessário realizar diversos estudos para saber onde implementar os equipamentos, quais os equipamentos a utilizar e a sua instalação e configuração. A autonomia destas redes não impede que os dispositivos presentes nessa rede possam aceder a serviços de outras redes, para tal basta que alguns dispositivos possam fornecer esse serviço e toda a rede poderá usufruir deles, como por exemplo acesso à Internet.

Este tema é muito rico em mais informação e detalhes. No entanto o foco deste trabalho é a capacidade de suportar replicação de objectos entre dispositivos móveis, cumprindo os requisitos apresentados em 2.1. Por esse motivo este tema é apresentado de forma mais genérica pois não é o principal tema de investigação.

3.2. Flexibilidade de paradigmas

A flexibilidade de paradigmas abordados neste trabalho está relacionada com os diversos mecanismos ao dispor dos programadores para acederem a dados remotos.

Tendo em conta os diversos paradigmas existentes, fica a cargo do programador decidir qual o paradigma a utilizar para desempenhar uma determinada tarefa na lógica da aplicação. A escolha do paradigma a ser utilizado deverá ter em conta diversas características do ambiente onde vai ser executado - memória disponível e capacidade de processamento dos dispositivos, qualidade de ligação, etc. Mas não basta somente ter em conta o dispositivo na qual a aplicação será executada, pois a própria lógica da aplicação influencia o paradigma escolhido – número de vezes que a tarefa é executada, carga computacional associada à execução da tarefa, etc.

Não obstante a especificidade da escolha do paradigma, existe também a dificuldade de implementação deste, pois na sua maioria envolve diversas camadas de baixo nível – *system-level*.

De seguida são apresentados três paradigmas: invocações remotas, replicação e agentes móveis. Para além da apresentação das suas características serão relevadas algumas situações nas quais os paradigmas se adequam, tendo em conta as limitações existentes num ambiente/dispositivo móvel.

3.2.1. Invocações Remotas

Com a utilização de memória distribuída por várias máquinas, as invocações remotas [5] surgem como um mecanismo que permite invocar métodos remotos, ou seja, interagir com objectos que não estão no dispositivo onde está a ser executada a aplicação. Em termos de lógica da aplicação, a utilização de invocações remotas é semelhante a utilização de *Remote Procedure Call* (RPC) [6]. Deste modo, o programador na posse das referências para os objectos pode-se abstrair da localização - local ou remota - do objecto.

Este mecanismo oferece muitas vantagens na programação distribuída, como por exemplo, uma perfeita junção deste mecanismo com programação OO, heterogeneidade e flexibilidade.

Tendo em conta um ambiente móvel, onde os dispositivos apresentam diversas limitações, esta solução é bastante favorável para diversas situações, como por exemplo: 1) reduzido número de invocações a um objecto, 2) pouca capacidade de armazenamento no dispositivo móvel ou 3) invocações que implicam uma forte carga computacional. Para além destas

situações a utilização deste mecanismo está muito relacionada com a qualidade da conexão, pois se esta for fraca os benefícios da utilização deste mecanismo diminuem.

3.2.2. Replicação

A replicação [7] é um das técnicas mais utilizadas para aumentar a disponibilidade de conteúdos, diminuir a largura de banda utilizada e minimizar os atrasos.

Para tal, este mecanismo copia os objectos remotos necessários para o dispositivo móvel. Após esta cópia, todas as invocações dos métodos desses objectos serão realizadas de forma local. Consoante a finalidade do objecto, este poderá ser enviado de volta à origem para nela serem realizadas as correspondentes alterações efectuadas.

Tendo em conta as limitações dos dispositivos móveis, esta solução é bastante favorável para diversas situações, como por exemplo: 1) quando o objecto é invocado diversas vezes ou 2) quando a ligação é de fraca qualidade. A utilização deste mecanismo pressupõe que o dispositivo móvel tenha capacidade para armazenar o objecto replicado, bem como capacidade para processar todas as invocações.

3.2.3. Agentes Móveis

Os agentes móveis [8] são programas que têm a capacidade de se moverem pela rede, para desempenhar determinadas tarefas autonomamente ou em nome do seu proprietário. Este mecanismo permite que uma aplicação crie o seu agente e o lance na rede para realizar as funções para as quais foi criado, por exemplo, procura de conteúdos. Neste exemplo o agente poderá viajar entre os diversos servidores realizando o pedido de procura de conteúdos, retornando no final da procura ao dispositivo onde foi criado, com o resultado da procura. Desta forma, a aplicação poderá continuar a realizar tarefas que não dependam do resultado do agente móvel, bem como reduzir os custos de processamento e memória. Para além destes recursos, existe uma diminuição dos gastos na ligação pois, ao invés de se realizar uma ligação para cada servidor, é apenas criada uma única para colocar o agente na rede e outra para o receber.

Devido às inúmeras funcionalidades dos agentes móveis existem alguns constrangimentos a ter em conta: 1) a programação difícil e complexa e 2) a segurança dos dispositivos percorridos. A complexidade da programação destes agentes está bastante relacionada com as necessidades da aplicação e do ambiente onde será executado (diversos sistemas operativos),

sendo portanto um problema difícil de parametrizar. Em relação à segurança dos dispositivos, existem alguns mecanismos (autenticação e autorização para aceder a determinado recurso) que permitem aos dispositivos determinar se o agente poderá ou não executar as tarefas pretendidas.

Tendo em conta as limitações dos dispositivos móveis, esta solução é bastante favorável para diversas situações, como por exemplo: 1) tarefas que envolvam a execução do mesmo método em diversos dispositivos ou 2) execução de um determinado número de tarefas bem definidas cuja interacção com a aplicação não se aplique, por exemplo, o pagamento de um determinado produto no qual a aplicação define os dados do pagamento e o agente contacta os diversos serviços de forma a completar o pagamento.

3.3. Replicação Automática

Tal como referido anteriormente a replicação de objectos [9] é um mecanismo utilizado para minimizar os constrangimentos causados pelas limitações dos dispositivos móveis: memória e processamento. Este mecanismo permite então aumentar a disponibilidade dos dados e melhorar o desempenho das aplicações executadas utilizando memória distribuída. A disponibilidade de dados é assegurada mesmo quando não é possível estabelecer uma ligação, situação que poderá ser frequente numa rede ad-hoc, pois os dados estão replicados localmente. Além disto o desempenho das aplicações melhora (quando comparada com a utilização de invocações remotas) pois as invocações são locais.

No entanto este mecanismo deverá ser utilizado de forma cautelosa, pois a sua utilização apesar de minimizar alguns constrangimentos, aumenta outro: a memória. Desta forma poderá haver situações em que seja necessário libertar objectos, apesar de estes ainda serem acessíveis e úteis, como por exemplo quando existem réplicas mais relevantes para as quais não existe memória disponível. Contudo este processo de libertação de memória é delicado, pois os objectos continuam a ser acessíveis pela aplicação e portanto poderão ainda ser utilizados por esta. Assim, é necessário gerir as referências de forma a garantir a integridade destas.

Para além da gestão de referências, a replicação automática [10] deverá oferecer mecanismos para que a 1) aplicação possa decidir, em tempo de execução, qual o mecanismo a ser utilizado para a invocação do objecto (chamada remota ou invocação local a uma réplica), 2) replicação incremental de um grande grupo de objectos e 3) criação de grupos de objectos dinâmicos.

Estes mecanismos permitem que a aplicação possa lidar com situações frequentes numa rede ad-hoc móvel como problemas de falhas de comunicação ou largura de banda reduzida. Por exemplo, se o acesso a uns determinados dados num dispositivo remoto não for possível, a aplicação não deverá parar a sua execução. Como alternativa deverá, pelo menos, propor ao utilizador um mecanismo alternativo de acesso aos dados de outro dispositivo com o qual consiga estabelecer ligação, mesmo no caso de os dados não estarem actualizados. Tendo em conta que este cenário poderá ser muito frequente numa rede ad-hoc é importante garantir que as aplicações o possam contornar graciosamente e o mais transparentemente possível.

3.3.1. Modelos de Replicação

A aplicação deve ser o elemento chave no processo de replicação, pois é esta que saberá as suas necessidades no futuro podendo desta forma definir o modelo pelo qual os objectos serão replicados. Por este motivo é necessário que existam diversos modelos de replicação [10] à disposição da aplicação para que esta possa escolher o que melhor se adequa a cada situação.

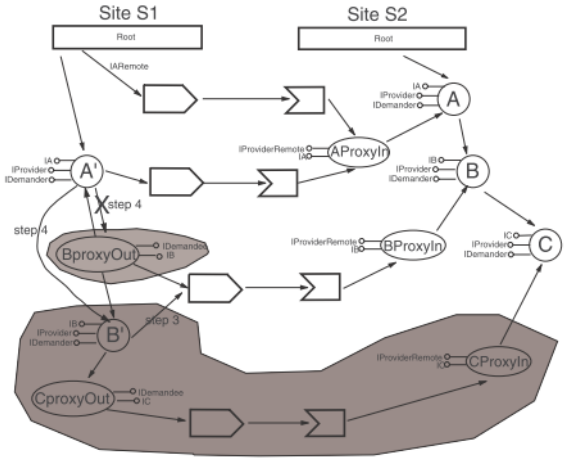

Figura 2 - Replicação Incremental[1]

Em ambos os modelos de replicação, de seguida apresentados, a implementação é semelhante: 1) são definidos pela aplicação os objectos a serem replicados, 2) são construídas as estruturas necessárias à gestão das referências (*proxies* [11]) e 3) são replicados os objectos para o dispositivo.

- **Replicação Individual** – este modelo replica os objectos de forma individual. Por cada objecto replicado são criados os respectivos *proxies*, o que implica gastos de

[1] Imagem retirada de [10].

processamento e de memória. Logo, este modelo é aceitável para situações onde é apenas necessário replicar um único objecto.

- **Replicação Incremental** – este modelo permite que a aplicação defina os objectos a serem replicados, tendo em conta um grupo de objectos. Deste modo, a replicação é bastante flexível pois permite que sejam replicados apenas os objectos que serão invocados pela aplicação, sem que seja necessário replicar todos os objectos do grupo, permitindo ainda contornar situações onde o dispositivo tem uma limitação de memória disponível para objectos replicados. Por outro lado, existem situações em que todos os objectos são necessários. Neste caso os objectos são agrupados e replicados como grupo, ou seja, apenas é criada uma estrutura de *proxies*. Por isso este modelo é mais eficiente no que diz respeito à utilização de memória e processamento. Na Figura 2 podemos ver um exemplo de replicação incremental do objecto B do Site S2 para o Site S1. Neste caso o processo de replicação está em processamento pois o objecto A' ainda referencia o *proxy* BProxyOut. Após a concretização do step 4, ou seja a troca de referência em A' de BProxyOut para B', este *proxy* poderá ser eliminado pois deixa de ser necessário.

3.4. Distributed Garbage Collection

O mecanismo *Garbage Collection* (GC) [12,13] permite eliminar automaticamente todos os objectos que deixam de ser referenciados. Desta forma a manutenção da memória deixa de estar a cargo do programador. Tendo em conta que os dispositivos móveis têm como limitação a memória, a correcta execução deste mecanismo ganha uma extrema importância.

Este mecanismo é muito útil no desenvolvimento de aplicações pois liberta os programadores da árdua tarefa da gestão de memória, tanto na alocação de memória bem como na sua libertação. Esta gestão é bastante complexa, mesmo no caso em que apenas uma aplicação acede à pilha de execução (*heap*), bem como propícia o aparecimento de erros difíceis de resolver. Tendo em conta estas dificuldades num caso simples, é fácil conceber as enormes dificuldades que surgirão em casos mais complexos. Por exemplo quando existe partilha da *heap* por diversas aplicações, quando o acesso à *heap* é feito em paralelo e no caso de aplicações distribuídas. Alguns autores [12] afirmam mesmo que esta gestão manual está completamente fora de questão.

Com o desenvolvimento de aplicações distribuídas surge a necessidade da implementação de um mecanismo de GC distribuído (DGC). Nestes sistemas o DGC oferece vários benefícios como a transparência (correcto funcionamento independentemente da localização do objecto: remota ou local) e o tratamento de tarefas complexas como a gestão de memória.

No caso de um GC local este deve estar coordenado, de forma a manter a consistência e a acompanhar as mudanças feitas entre espaços de endereço. Num ambiente distribuído existem novos problemas que devem ser tidos em conta: 1) objectos perdidos, 2) duplicação de objectos ou mensagens de atraso e 3) falhas de comunicação.

Neste sentido um DGC deverá contornar diversos problemas:

- Recuperar todos os tipos de estruturas de dados de forma eficiente;
- Escalabilidade;
- Tolerância a falhas.

De seguida são apresentados algumas soluções de DGC.

3.4.1. Reference Counting Approaches

Devido à sua natureza acíclica, a maioria dos algoritmos de DGC não são capazes de recuperar objectos ciclicamente referenciados. Para debelar este problema existem vários algoritmos baseados em contadores de referências:

Uniprocessor Adoption (UA) [14] **-** Este algoritmo é a aproximação mais simples para a implementação de um DGC. Quando é criada, duplicada ou eliminada uma referência de um objecto global, é enviada uma mensagem para o criador do objecto global. Consoante a acção realizada, o criador do objecto global terá de incrementar (criação ou duplicação) ou decrementar (eliminação) o respectivo contador para essa referência. O valor deste contador é igual ao número de referências para o objecto em causa. Quando esse valor é zero, o objecto é reciclado. Esta solução tem vários inconvenientes como a intolerância a falhas na comunicação, a duplicação de mensagens e um número de mensagens trocadas bastante elevado.

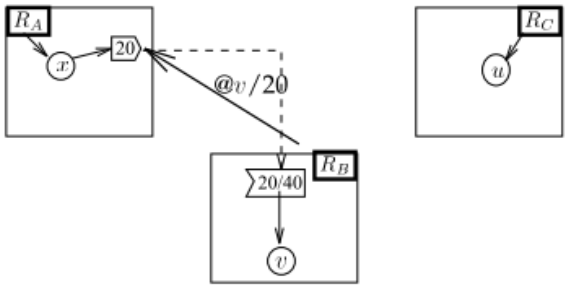

Figura 3 - Criação de uma referência (WRC)

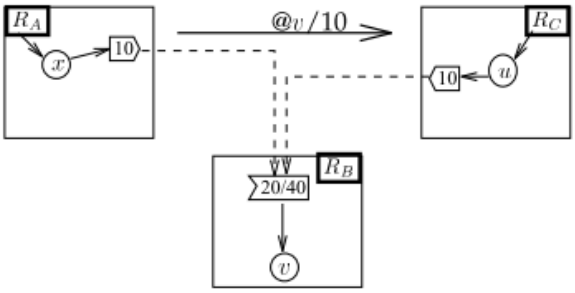

Figura 4 - Duplicação de uma referência (WRC)

Weighted Reference Counting (WRC) [14] **-** Este algoritmo apresenta uma grande melhoria em relação ao anterior, nomeadamente na redução do número de mensagens trocadas. Este melhoramento surge devido há ausência de *race conditions*[2]*.* Para solucionar este problema o WRC adiciona um novo parâmetro: o peso. Cada objecto global tem um peso máximo, sendo este dividido pelas referências que entretanto forem criadas. Como o peso apenas poderá ser decrementado não haverá situações de *race conditions*. No entanto este algoritmo apresenta alguns problemas como a limitação do número de referências a serem criadas para esse objecto global.

Na Figura 3 está representada a criação de uma referência do objecto v em R_A. Neste exemplo o objecto v tem um peso inicial de 40, portanto a referência em R_A fica com metade desse valor, ou seja 20 e em R_B a referência tem o valor de 20/40 (20 é o peso decrementado e 40 o peso total). Na Figura 4 está representada uma situação posterior em que o objecto x (réplica de v em R_A) é replicado para R_C. Neste caso o peso de x inicial (20) é dividido entre ambos ficando cada um com o peso de 10. Supondo que a referência de u é eliminada, é enviada uma mensagem para R_B indicando que este deve decrementar 10 ao valor do peso, ou seja R_B fica com 20/30. O objecto v só poderá ser apagado de R_B quando o peso for 20/20, pois nesta situação já não existem referências remotas para v.

[2] Race Conditions – Entrega não causal de mensagens. Esta situação poderá causar inconsistência, por exemplo, quando uma mensagem para decrementar gerada após uma mensagem de incrementar chega primeiro. Neste caso o objecto poderá ser recolhido pelo DGC quando na verdade não o deveria ser.

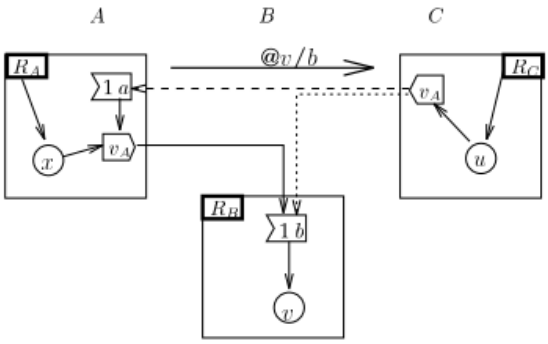

Figura 5 - Duplicação de referência (IRC)

Indirect Reference Counting (IRC) [15] **-** Neste algoritmo, para além do campo para guardar a referência ao objecto global e do campo para guardar o valor da contagem é necessário adicionar um novo campo para guardar uma referência para o objecto que criou a referência. Consequentemente, toda a informação do estado do objecto é mantida com as suas referências. Este algoritmo garante, mesmo em casos de duplicação, que a referência para o objecto global é a mais directa mantendo assim uma troca de mensagens aceitável. Este algoritmo é razoavelmente eficiente pois apenas é necessária uma mensagem nos casos de referências duplicadas ou na eliminação. Na Figura 5 podemos observar uma situação de duplicação de referências. Em R_B está o objecto global v. Aquando da criação da referência em R_A para v (v_A), R_B cria uma estrutura (1 b) que contém a referência para v e um contador iniciado com 1. Numa situação posterior, a referência v_A é duplicada para R_C. Nesta situação R_A cria uma estrutura (1 a) que contém uma referência para v_A e um contador iniciado em 1. Em R_C é criada uma estrutura com duas referências: uma para o objecto global e outra para 1 a. O objecto v só poderá ser eliminado quando o contador for igual a zero, ou seja, quando deixar de ser referenciado.

Trial Deletion [14] **-** A ideia deste algoritmo é emular a eliminação de um objecto que tudo indica será recolhido pelo GC. Depois desta operação o contador das referências desse objecto é incrementado para poder continuar activo. Desta forma é possível contornar ciclos na recolha de objectos. Este algoritmo apresenta uma grande desvantagem: a necessidade da criação de uma heurística para determinar quais os objectos com probabilidade de serem recolhidos pelo GC.

3.4.2. Reference Listing Approaches

Reference Listing (RL) difere da *Reference Counting* (RC) na forma como são geridas as estruturas que controlam as referências (*scion*[3] e *stub*[4]). No caso de RC esta estrutura de dados é única para todos os clientes, contendo um único contador por objecto, enquanto nesta nova abordagem (RL) é criada uma lista de *scion* independentes, um para cada referência ao mesmo objecto. As referências entre objectos passam então a ser controladas por estas estruturas. Por este motivo as mensagens de incrementar e decrementar da anterior solução são substituídas por mensagens de controlo de inserir e apagar o *scion*. É de realçar que a utilização de RL obriga a um aumento da memória utilizada.

A maior vantagem da utilização deste mecanismo reside no facto das mensagens serem idempotentes, portanto resistentes a falhas nas mensagens (duplicação e perda). Por exemplo, uma mensagem apagar pode ser enviada várias vezes sem consequências. Se a mensagem apagar for recebida, as mensagens que cheguem após esse momento serão ignoradas; se a mensagem apagar não chegar ao destino a próxima a chegar será processada. No entanto, a latência na entrega de mensagens pode levar a recuperar um objecto inexistente, por exemplo se uma mensagem de apagar chegar atrasada, o resultado poderá interferir com um *scion* acabado de criar. Esta situação poderá ser ultrapassada recorrendo à utilização de marcas temporais nas mensagens.

Para implementação de RL existe o algoritmo *Stub-Scion Pair Chains* (SPP Chains) [16,12]. Este algoritmo combina aplicações distribuídas com um mecanismo de referências para localizar objectos remotos. Foi desenhado para sistemas distribuídos clássicos de vários processos distintos que comunicam/utilizam invocações remotas, ou seja, sem memória partilhada, com falhas parciais e com custos de ligação.

Neste algoritmo uma referência é uma cadeia (*chain*) ponto-a-ponto, ao invés de um identificador global. Uma referência remota é representada por uma cadeia entre o par *scion-stub* (SPPs). Uma cadeia é criada como sendo uma SSP singular quando é enviada uma referência de um objecto local para o espaço ou aquando da migração de um objecto para outro espaço. No caso de a cadeia já existir esta é prorrogada nas mesmas circunstâncias.

A estrutura *stub* encapsula dois tipos de ponteiros: um fraco e um forte. O ponteiro forte indica o próximo *scion* na cadeia e serve apenas os propósitos do DGC. O ponteiro fraco é utilizado para invocações remotas permitindo o acesso a este utilizando apenas um salto.

---

[3] *scion* – estrutura de dados criada no servidor e que gere as referências para os objectos locais.
[4] *stub* – estrutura de dados criada no cliente e que gere as referências remotas (scion).

Quando é exportada uma referência é criado um *scion*, por outro lado, quando é importada é criado um *stub*. Quando o stub é localmente inacessível é recolhido pelo GC local. Periodicamente cada espaço envia mensagens para detectar quais os *stub* alcançáveis e o receptor apaga o correspondente *scion* que não está na lista.

A tolerância a falhas é feita recorrendo à conservação da ordem das acções e da troca de mensagens idempotentes; a *race conditions* (antes mencionada) é evitada recorrendo à marcação de cada mensagem com uma marca temporal. Desta forma qualquer mensagem que seja recebida fora de ordem pode ser detectada.

3.5. Políticas de segurança

A segurança de uma rede móvel ad-hoc [17,18] é uma componente importante, tendo sido ao longo dos últimos anos uma área de investigação e desenvolvimento.

Numa rede móvel a implementação de segurança é um enorme desafio por diversas razões: 1) a utilização da interface é muito susceptível a diversos tipos de ataque, 2) os utilizadores precisam de utilizar os diversos serviços quando desejam e em qualquer lugar e 3) a implantação tem de ser escalável para uma rede de larga escala.

Confidencialidade, autenticação, integridade, não repúdio e controlo de acessos são dos principais serviços de segurança que uma rede deverá garantir. No entanto, é de destacar a importância do serviço de autenticação, pois sem este os restantes deixarão de ser garantidos. Por exemplo, numa ligação cujos intervenientes estejam correctamente autenticados será possível oferecer os restantes serviços, caso não seja possível autenticar os intervenientes, não estão garantidos os requisitos para uma comunicação segura, impossibilitando a concretização dos restantes serviços.

Tendo em conta uma rede móvel ad-hoc, a implementação de mecanismo de segurança é ainda mais complexo devido a: 1) instabilidade das ligações, 2) protecção física limitada de cada dispositivo, 3) natureza esporádica da ligação, 4) topologia muito dinâmica, 5) ausência de uma entidade externa de autenticação e de gestão.

Neste sentido, as políticas de segurança a implementar dependem bastante dos propósitos com que rede foi criada. Por exemplo uma rede criada com o fim de fornecer apoio a um jogo ad-hoc deverá ter mecanismos de segurança bastantes diferentes de uma rede para troca de informação confidencial.

Alguns autores afirmam que as políticas de segurança devem ser implementadas em diversos níveis. Tendo em conta o âmbito deste trabalho, ou seja, a implementação de um *middleware* que suporta agentes móveis, as políticas de segurança devem implementar [19]:

Protecção do dispositivo contra o agente - Proteger o dispositivo contra utilização indevida de recursos e informação por parte do agente móvel.

Protecção contra outros agentes - Proteger um agente honesto para não sofrer ataques de um agente malicioso.

Protecção do agente contra o dispositivo - Proteger o agente para não sofrer ataques desencadeados pelo dispositivo visitado.

Para cumprir estes objectivos existem algumas soluções:

- **Ajanta** [20] – sistema de agentes móveis baseado em Java. A principal preocupação de segurança deste sistema é proteger os dispositivos da visita de agentes móveis, implementando um mecanismo de controlo de acessos de forma a proteger o dispositivo. Não tem em consideração acontecimentos passados.
- **Deeds** [21] – sistema de controlo de acesso baseado em eventos passados para protecção do dispositivo visitado pelo agente. Este sistema, escrito em Java, baseia o controlo de acesso tendo em conta eventos passados que o agente realizou. Este sistema tem uma limitação na forma como identifica o agente, não permitindo a execução de novas classes criadas dinamicamente.
- **Ponder** [22] – sistema que fornece uma implementação de segurança e gestão de políticas de segurança. Utiliza uma linguagem declarativa e permite definir políticas de segurança para a migração dos agentes de forma independente da aplicação. Não tem em consideração acontecimentos passados.
- **Security Policy Language (SPL)** [23] – sistema que permite definir um vasto grupo de políticas de autorização e obrigação, muito eficiente e controlado por um monitor de segurança. Permite a criação de modelos de segurança complexos como o controlo de acesso baseado em eventos anteriores, o controlo de acesso discricionário ou políticas baseadas em obrigações. Permite definir vários tipos de entidades: objectos, grupos, regras e políticas. Note-se que as regras restringem as relações entre objectos e grupos e as políticas resultam da decomposição de múltiplas regras e grupos.

Estão assim apresentados os diversos aspectos técnicos relacionados com este trabalho. De realçar que objectivo final é a criação de um *middleware* que ofereça suporte às aplicações numa rede ad-hoc e que integre e adopte soluções já existentes, com ênfase no mecanismo de replicação incremental.

3.6. Sistemas Existentes

Tendo em conta os aspectos referidos anteriormente, foi efectuada uma pesquisa por diferentes soluções que as implementem. De seguida são apresentados alguns dos trabalhos investigados mais relevantes.

3.6.1. Common Object Request Broker Architecture

O *Common Object Request Broker Architecture* (CORBA) [24] foi desenvolvido pelo grupo OMG[5] com o objectivo de desenvolver e especificar uma infra-estrutura para sistemas distribuídos promovendo a utilização de sistemas baseados em objectos (reutilização, portabilidade e interoperabilidade entre sistemas distribuídos heterogéneos).

É um sistema desenhado para fornecer suporte para objectos replicados numa rede de larga escala, como por exemplo a World Wide Web. A característica chave está na noção de transparência criada na comunicação cliente/objecto. Para tal o CORBA oculta:

- **Localização do objecto** – o cliente não sabe a localização do objecto, pois este objecto pode estar presente por exemplo noutro processo ou noutra máquina;
- **Implementação objecto** – o cliente não sabe como o objecto foi implementado, qual linguagem de programação em que foi escrito, o sistema operativo e o hardware onde é executado;
- **Estado execução do objecto** – quando é invocado um objecto, o cliente não precisa de saber se o objecto está actualmente activo (a processar outra invocação) ou se está disponível.
- **Mecanismo de comunicação com o objecto** – o cliente não precisa de saber qual o mecanismo de comunicação utilizado: TCP/IP, memória partilhada, invocação local, etc.). O CORBA assegura a invocação do objecto, bem como a entrega da resposta ao cliente.

[5] Object Management Group - Organização internacional, sem fins lucrativos, fundada em 1989.

Para a implementação do mecanismo de replicação, sempre que um objecto é gerado é criado juntamente um referência para esse objecto. Esta referência está sempre associada a esse objecto, independentemente do tempo de vida do objecto. Somente o CORBA tem conhecimento das propriedades das referências, os clientes apenas as podem utilizar. Por este motivo os clientes podem obter estas referências de diversas formas: 1) criação do objecto pelo próprio cliente, 2) utilização de um serviço de directórios e 3) armazenamento de referências de forma persistente para posterior utilização.

Este sistema não oferece mecanismo de DGC, políticas de segurança nem capacidade de implementação numa rede ad-hoc.

3.6.2. Javanaise

Javanaise [25] é uma plataforma, desenvolvida em Java, com o objectivo de oferecer suporte para colaboração entre aplicações distribuídas na Internet. Esta plataforma permite que os programadores desenvolvam as suas aplicações como se fossem aplicações centralizadas. Quando finalizada, o programado configura a aplicação para se tornar distribuída, sem que seja necessário fazer alterações ao código. Após este passo um mecanismo automático gera toda a estrutura de dados necessária.

No entanto esta plataforma não fornece suporte para replicação incremental (embora permita a definição de grupo de objectos mas têm de ser realizada pelo programador), nem para a definição de políticas de segurança, agentes móveis ou DGC.

3.6.3. Thor

Thor [26] é um sistema de gestão de base de dados OO. Foi desenvolvido para ser utilizado por sistemas distribuídos heterogéneos, permitindo que programas escritos em diferentes linguagens, possam partilhar objectos de maneira conveniente.

Thor permite que os utilizadores possam armazenar e manipular objectos que capturam a semântica das suas aplicações. Para além disso fornece aos utilizadores um conjunto de objectos persistentes. Os objectos desse conjunto podem referenciar outros objectos permitindo a utilização de estruturas de objectos, tais como gráficos e árvores. Para suportar a heterogeneidade o Thor fornece uma linguagem própria, independente da utilizada pelos programas. Esta linguagem permite o desenvolvimento de sistemas hierárquicos para suporte à evolução das aplicações.

Thor é um sistema distribuído no sentido em que os objectos são guardados em servidores, ou seja, máquinas distintas da localização da aplicação cliente. Contudo, o acesso aos objectos é rápido uma vez que parte do Thor é executada nas máquinas dos utilizadores, de modo a permitir a utilização de invocações locais, reduzindo a latência observada pelos utilizadores. Esta arquitectura fornece um desempenho elevado e escalabilidade. A escalabilidade é possível pois algum do processamento é efectuado nas máquinas dos clientes, libertando os servidores desse trabalho. Existe também a possibilidade de aumentar o número de servidores. Note-se que Thor oferece um mecanismo de DGC limitado pois não contempla a existência de réplicas.

3.6.4. Object Space Voyager

Object Space Voyager (OSV) [27] é um sistema de agentes móveis baseados em Java. Neste sistema um agente não é mais do que um objecto com capacidade de se mover numa rede. No OSV é introduzido o conceito de *virtual object*, ou seja, a representação de um objecto ou agente utilizando um proxy. Este sistema permite transformar qualquer objecto num agente móvel utilizando um compilador virtual.

Um objecto ao ser processado exibe algumas características de um agente: 1) pode-se mover de máquina para máquina, 2) ser acedido remotamente por outro objecto virtual e 3) pode ter o seu próprio ciclo de vida. Por definição estes objectos móveis são passivos, ou seja apenas se podem mover e ser manipulados remotamente. No entanto eles podem ser desenhados para desempenhar mais funções.

Ao utilizar o sistema OSV no desenvolvimento da aplicação, o programador poderá escolher entre a utilização de agentes móveis ou invocações remotas. Este sistema permite também a implementação de políticas de segurança, ficando essa função a cargo do sistema operativo. No entanto, o OSV não fornece mecanismos de replicação automática nem de DGC.

3.6.5. Object Broker Infrastrucutre for Wide Area Network

Object Broker Infrastrucutre for Wide Area Network (OBIWAN) [28] é uma plataforma middleware que permite desenvolverem aplicações distribuídas sem que para tal o programador seja obrigado a desviar-se da lógica da aplicação.

Com o OBIWAN a aplicação poderá ser programada para em tempo de execução mudar o paradigma de invocação: chamada remota, chamada local ou agente móvel. A criação e gestão

de réplicas é feita de forma transparente, no entanto o programador poderá controlar, em tempo de execução, a quantidade de objectos replicados.

OBIWAN tem um módulo de replicação que é responsável por lidar com todos os aspectos relacionados com a criação de réplicas pois permite: 1) que a aplicação escolha, em tempo de execução, o paradigma de invocação, 2) a replicação incremental de um grande grupo de objectos e 3) a criação de grupos de objectos dinâmicos.

No que diz respeito a DGC, o OBIWAN fornece um mecanismo que resolve alguns problemas de DGC causados pela replicação de objectos, por exemplo, a possibilidade da existência de réplicas. Para tal o algoritmo DGC é independente do protocolo que mantém a coerência dessas réplicas. Destaque-se que a execução deste mecanismo tem um impacto mínimo na aplicação.

A utilização de agentes móveis levanta diversas preocupações de segurança, como por exemplo, controlo do fluxo de informação e autorização. Uma das questões mais importantes é a definição de quais as operações que o agente está autorizado a realizar e quais as operações cujo agente está impedido ou obrigado a realizar. Como solução para este problema o OBIWAN fornece suporte à definição e execução de políticas de segurança baseadas em eventos anteriores. Este suporte é extremamente importante no sentido em que permite a organização de políticas de segurança, nas quais o comportamento do agente influencia as permissões. Deste modo o agente local pode permitir ou impedir, baseado no comportamento passado do agente móvel nessa ou noutra máquina, o acesso a recursos protegidos.

3.6.6. Seven Degrees of Separation

Seven Degrees of Separation (7DS) [29] é um sistema que permite troca de informação entre dispositivos que têm uma ligação intermitente à Internet. Neste sistema alguns dos participantes da rede obtêm dados de diversos servidores na Internet e armazenam esses dados. Após esta operação podem trocar os dados armazenados com dispositivos que não têm acesso à Internet. Os dados trocados podem ser páginas Web, mapas, vídeos de curta duração, músicas e qualquer outro tipo de objectos com tamanho modesto.

Este sistema foi criado de forma a aproveitar os benefícios das redes sem fios, implementando para tal um protocolo de cooperação entre dispositivos, para assim ambos obterem benefícios. Um facto explorado por este sistema é a localização da informação se concentrar em determinadas zonas. Por exemplo é muito provável que os passageiros do

metropolitano queriam ler, pela manhã, os jornais diários. No entanto pessoas que estão em movimento têm preferências mais diversificadas.

7DS fornece alguns mecanismos de replicação de dados numa rede ad-hoc. Todavia, não fornece mecanismo de DGC, nem políticas de segurança ou de flexibilidade de paradigmas.

3.6.7. Babylon

Babylon [30] é um sistema que fornece mecanismo de desenvolvimento, execução e gestão paralela e distribuída de aplicações Java móveis. Este sistema suporta migração de objectos, invocação de métodos assíncronos e *download* de classes remotas sem necessidade de autenticação. Adicionalmente, é dada a possibilidade da aplicação criar e interagir com objectos remotos, sendo possível também proteger esses objectos de outras aplicações.

Babylon fornece um modelo de programação para objectos distribuídos, que pode ser implementado de forma eficiente numa grande variedade de aplicações.

A implementação do Babylon está centrada na utilização de *proxies* dinâmicos. A utilização destes proxies permite a utilização de invocações remotas e criação de objectos remotos, em tempo de execução. Uma importante característica destes *proxies* é a possibilidade da criação de objectos remotos a partir de classes cujo código fonte não é fornecido.

As invocações assíncronas são possíveis devido à utilização de *tickets* assíncronos. A invocação de métodos assíncronos utilizando *tickets* é sintacticamente idêntica às invocações locais, mas executadas de forma assíncrona.

Este sistema permite a criação de objectos remotos de duas formas: 1) baseado unicamente no nome da classe, especificada pelo programador e 2) tornando um objecto local previamente criado num objecto remoto. Para além disto permite a criação e invocação de grupos de objectos.

Babylon suporta a migração de objectos inactivos entre máquinas. Esta migração é suportada pelo mecanismo *safe-point migration* que permite migrar objectos remotos em execução utilizando para tal protocolos de *checkpointing* e *rollback*.

O mecanismo de DGC é fornecido pela *Java's Remote Method Invocation* [31,32]. Não existe suporte a políticas de segurança nem a execução num ambiente ad-hoc.

De seguida apresenta-se uma tabela que sintetiza a informação descrita neste capítulo.

	Flexibilidade de paradigmas	Replicação Automática	Distributed Garbage Collection	Políticas de segurança	Rede ad-hoc
CORBA	✓	✓	✗	✗	✗
Javanaise	✓	✓ (limitado)	✗	✗	✗
Thor	✓	✓	✓ (limitado)	✗	✗
OSV	✓	✗	✗	✓ (limitado)	✗
OBIWAN	✓	✓	✓	✓	✗
7DS	✗	✓	✗	✗	✓
Babylon	✓	✓	✓	✗	✗

Tabela 1 – Propriedades das soluções pesquisadas

Como é possível verificar na Tabela 1 as várias soluções pesquisadas visam resolver diversos aspectos. Tendo em conta os objectivos deste trabalho e todas as componentes a serem tratadas, a solução OBIWAN é a mais aproximada com o trabalho a ser desenvolvido. No entanto as restantes soluções também serão tidas em conta, pois apresentam abordagens diferentes para alguns problemas semelhantes

4. Arquitectura

Tendo em conta os objectivos apresentados anteriormente foi pensada uma arquitectura para suplantar esses desafios.

Figura 6 - Exemplo rede ad-hoc

O OBIHOC[6] (nome atribuído ao sistema implementando) é um *middleware peer-to-peer*, no sentido em que todos os dispositivos poderão funcionar como clientes e servidores a qualquer momento, Figura 6. Desta forma uma aplicação a executar num dispositivo poderá desempenhar tarefas de servidor (fornecer objectos para replicação) ou de cliente (pedir replicação de objectos de outros dispositivos).

Tal como anteriormente referido OBIHOC é um *middleware* e por esse motivo a sua arquitectura foi pensada de forma genérica para que qualquer programador a possa integrar na sua aplicação de forma o mais transparente possível. Para além disso é suficientemente adaptativa às inúmeras opções de hardware existentes.

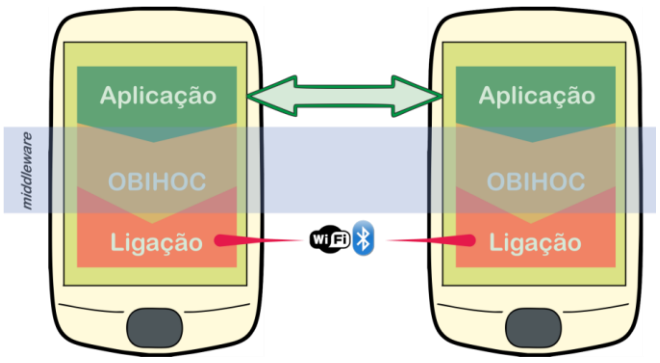

Figura 7 - Localização OBIHOC

[6] OBIHOC – **O**bjecto **B**roker **I**nfrastruture for Ad-**Hoc** Networks

Do ponto de vista do programador a sua aplicação irá comunicar com outra aplicação de forma directa (ligação a verde na Figura 7), no entanto esta comunicação é efectuada pelo OBIHOC, permitindo uma abstracção do método de comunicação, bem como na gestão dos objectos a serem replicados. No entanto, esta abstracção não poderá ser total pois continua a ser a aplicação a definir alguns parâmetros, tais como a tecnologia a ser utilizada na comunicação (*Wi-Fi* ou *Bluetooth* por exemplo) bem como os objectos que puderam ser replicados (escolha da profundidade replicação por exemplo).

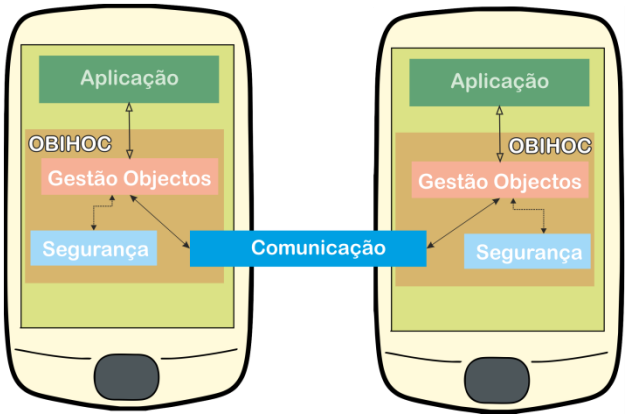

Figura 8 - Arquitectura com visão de alto-nível

Como é possível verificar na Figura 8, a arquitectura do OBIHOC tem 3 módulos:

- **Gestão de Objectos** – módulo responsável por gerir os objectos que podem ser replicados. Para além disso é neste módulo que é feito o controlo do DGC.
- **Segurança** – módulo responsável por validar as políticas de segurança.
- **Comunicação** – módulo responsável pela transferência de objectos entre dispositivos. Permite estabelecer ligações com outras aplicações e realizar todas as trocas de informação necessárias.

De seguida cada um destes módulos será abordado com mais pormenor, sendo apresentado a arquitectura de cada um deles de forma mais detalhada. No final do capítulo é apresentada uma visão de mais baixo-nivel da arquitectura geral.

4.1. Gestão de Objectos

Para a gestão de objectos a arquitectura foi pensada tendo em conta os seguintes objectivos:

- Permitir armazenar informação sobre quais os objectos passíveis de serem replicados para outros dispositivos;
- Gerir o mecanismo de replicação e invocação de objectos locais e/ou remotos;
- Garantir o correcto funcionamento de mecanismo de DGC.

Em toda a arquitectura do OBIHOC esta é a componente central, pois é esta que armazena e gere os objectos passíveis de serem replicados, devidamente indicados pela aplicação.

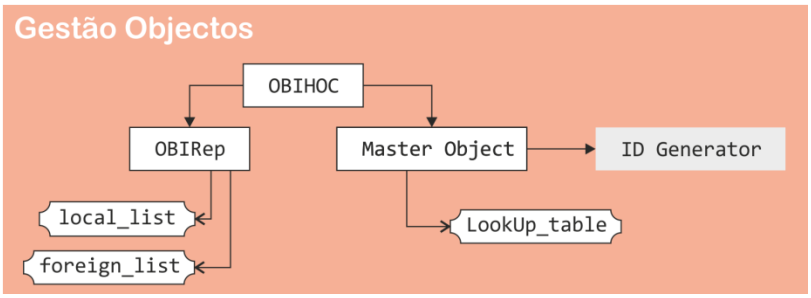

Figura 9 – Arquitectura Gestão de Objectos

A Figura 9 representa de forma esquemática a arquitectura desta componente. Para cumprir com os objectivos propostos existem cinco módulos:

- **OBIHOC** – módulo que faz a ligação entre a aplicação e o middleware. É o módulo central de toda a arquitectura pois é este que desencadeia a maioria dos processos no funcionamento do middleware.
- **OBIRep** – módulo que armazena as estruturas de dados que permite gerir a replicação de objectos. Para além disso este módulo tem um papel activo no mecanismo de DGC.
- **Master Object** – módulo que armazena informação sobre os grafos de objectos passíveis de replicação.
- **ID Generator** – módulo responsável por gerar os identificadores únicos de cada objecto.

O middleware OBIHOC permite que o programador da aplicação defina quais os objectos que deverão ser invocados remotamente e localmente. Desta forma o programador tem a hipótese de definir qual a melhor forma de invocação do objecto, tendo em conta as necessidades da aplicação e os recursos a serem utilizados no dispositivo.

Tal como referido anteriormente o módulo OBIHOC tem um papel central em toda a arquitectura. Este módulo desencadeia todos os processos necessários para o correcto funcionamento do middleware. Tendo em conta que uma aplicação tanto pode ser cliente como servidor ou ambas, este módulo engloba todas as propriedades para qualquer um dos casos,

por exemplo, a adição de grafos de objectos será um processo tipicamente de um servidor enquanto o pedido de replicação de objectos é tipicamente um processo de um cliente. Para o funcionamento deste são utilizados dois módulos – OBIRep e Master Object.

O Master Object é responsável pela gestão dos identificadores dos objectos, ou seja, sempre que um novo objecto é adicionado este módulo gera um identificador único. Para além disto possui uma lista de grafos de objectos (LookUp_table). Esta lista associa o nome do grafo ao identificador do objecto da raiz desse grafo.

O OBIRep é responsável pela gestão dos objectos locais e remotos, ou seja, faz a gestão dos objectos locais bem como dos objectos que foram replicados de outros dispositivos. Este módulo é constituído por duas listas. A local_list permite guardar, tal como o nome indica, as estruturas de dados dos objectos locais e que poderão ser replicados para outros dispositivos. A foreign_list é uma lista de objectos que foram replicados para outros dispositivos. A utilização desta lista permite que seja controlado o mecanismo de DGC, pois enquanto um objecto se mantiver nesta lista significa que foi replicado e por esse motivo não deverá ser apagado sem a confirmação que é de facto passível de ser apagado.

Todos os objectos a serem guardados nesta lista têm:

- Um identificador numérico (Long);
- Uma PropEntry – estrutura de dados que permite associar diversos atributos a esse objecto;
    - PropState: *flag* que permite diferenciar os vários estados que o objecto pode ter – NONE, PROP_IN e PROP_OUT.
    - ObjRef: Uma referência para o objecto local;
    - foreignID: identificador da posição ocupada pelo objecto na foreign_list, após este objecto ser replicado. Se este identificador tiver o valor -1 significa que ainda não foi replicado.

Com esta estrutura é possível fazer a gestão de todos os objectos, bem como dos diversos estados pelos quais ele pode passar.

Se um objecto for marcado com a *flag* NONE significa que ainda não foi carregado, ou seja, é necessário obter o objecto do local onde foi guardado, por exemplo uma base de dados. Esta flag é utilizada em aplicações que necessitam que os dados sejam guardados persistentemente. A flag PROP_OUT permite identificar um objecto como sendo local e passível de ser replicado. A flag PROP_IN é utilizada nos dispositivos para os quais o objecto foi replicado, ou seja, durante o processo de replicação um objecto ao ser correctamente replicado é marcado com esta flag. Para o mecanismo de replicação basta a utilização destas

três *flag's*. No entanto poderá ser adicionadas mais *flag´s* consoante a política de segurança a ser implementada (por exemplo para impedir que o objecto seja replicado fora do dispositivo que o criou).

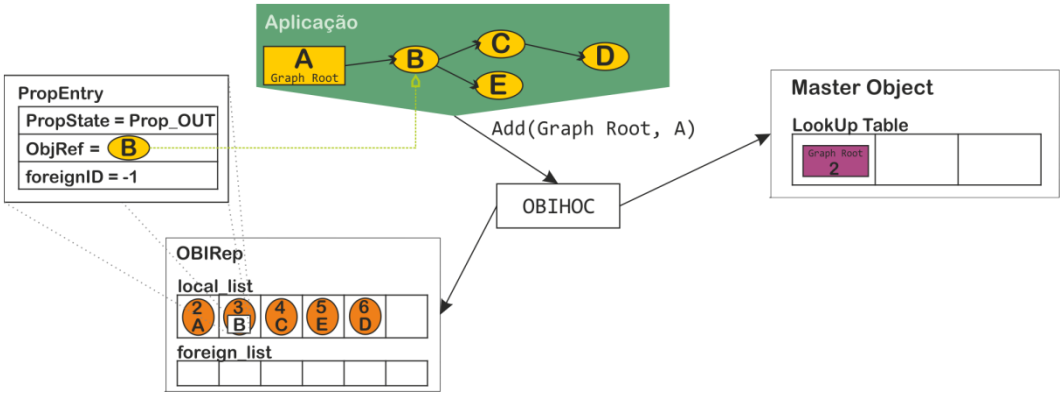

Figura 10- Exemplo de inserção de um grafo de objectos

Para a replicação de objectos é necessário que alguma aplicação os disponibilize. A Figura 10 exemplifica um desses casos em que uma aplicação explicitamente indica a raiz do grafo de objectos a ser replicado. Este pedido é composto pelo nome do grafo e a referência para a raiz do mesmo. Após esta indicação é desencadeado automaticamente um processo que analisa todos os objectos desse grafo, e se for identificado como sendo um objecto replicável, é-lhe atribuído um identificador único gerado pelo ID Generator e criada uma estrutura de dados PropEntry, predefinida com a flag PROP_OUT. De seguida é adicionado à local_list que associa esse identificador à estrutura de dados. No exemplo da Figura 10 pode-se ver em mais pormenor que o objecto B foi adicionado com o identificador 3 e com a respectiva PropEntry.

No final deste processo o grafo é finalmente adicionado à lista LookUp_Table, no módulo Master Object. A partir deste momento este grafo de objectos fica disponível para ser replicado para outras aplicações.

4.1.1. Gestão de referências dos objectos

Um dos grandes focos deste trabalho é a replicação de objectos entre dispositivos. No entanto este processo poderá causar alguns problemas no comportamento da aplicação, pois é necessário a execução de mecanismo de um nível mais baixo, em relação à aplicação. Do ponto de vista da aplicação todos os objectos são invocados localmente, no entanto em algumas situações isto não se verifica, pois o objecto invocado poderá ainda não ter sido replicado. É nestas situações que a gestão de referências é importante, pois desta forma é possível manter integro o comportamento da aplicação.

Nos casos em que os objectos são locais a gestão de referências é escusada, pois a aplicação poderá invoca-los de forma directa. No entanto nos casos de replicação de objectos esta gestão ganha extrema importância e foi esse o um dos focos deste trabalho.

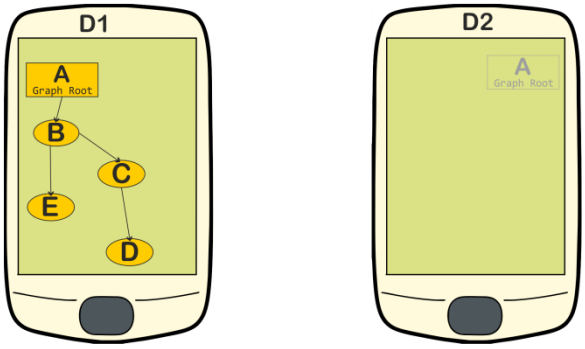

Figura 11 - Cenário inicial da Replicação de um grafo de objectos

Consideremos a situação apresentada na Figura 11. Um dispositivo D1 contém um grafo de objectos preparado para puder ser replicado para o dispositivo D2. Este grafo de objectos tem o nome “Graph Root” sendo a raiz do grafo o objecto A.

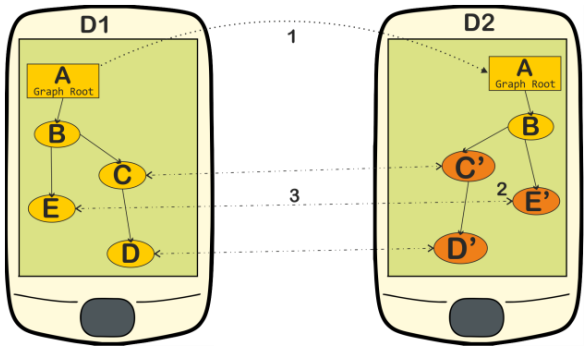

Figura 12 – Replicação de um grafo de objectos com profundidade 2

Num determinado momento D2 efectua todos os passos para se iniciar a replicação do grafo “Graph Root” de D1 para D2, com profundidade 1 (situação **1**). Após a conclusão deste processo de replicação ficamos com a situação apresentada na Figura 12 em que a raiz do grafo bem como o objecto B foram replicados na sua totalidade de D1 para D2. A profundidade desejada era de 1 e por esse motivo os restantes objectos do grafo foram trocados por proxy’s (representados por C’, D’ e E’ na Figura 12) em D2 (situação **2**). Estes proxy’s permitem que a aplicação continue a funcionar mesmo não tendo o objecto em si, pois estes implementam todas as interfaces do objecto de origem. Para além disso os proxy’s mantêm uma referência

para o objecto do dispositivo de origem (situação **3**), permitindo à aplicação fazer invocações remotas nesse objecto.

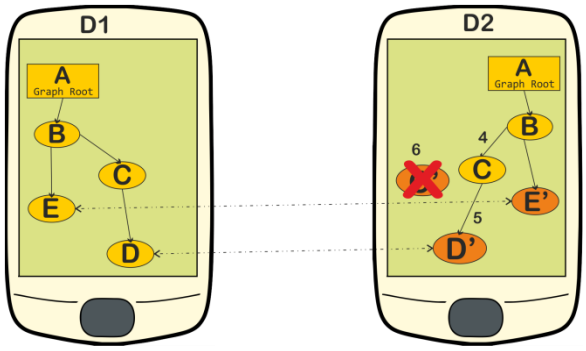

Figura 13 - Replicação de um objecto e actualização de referências

Caso a invocação a um objecto representado por um proxy, não seja explicitamente definida como invocação remota, automaticamente é desencadeado um novo processo para replicação desse objecto. No entanto existe a possibilidade desta replicação abranger um conjunto de objectos tendo em conta novamente a profundidade desejada. Continuando o cenário anterior, mas referente à Figura 13, a aplicação faz uma invocação ao objecto C, que na verdade é o proxy C'. É enviado um pedido de replicação desse objecto a D1. Após a recepção do novo objecto é iniciado o processo de actualização de referências. Neste caso é invocado um método no objecto B para que este passe a referenciar o objecto C, recentemente replicado (situação **4**) e o próprio C passará a apontar para D', um proxy do objecto original D de D1 (situação **5**). No final deste processo, o proxy C' deixa de ser referenciado por qualquer objecto e poderá ser eliminado (situação **6**). A partir deste momento qualquer invocação ao objecto C será efectuada localmente, sendo possível a sua repercussão no objecto C de D1, para tal é enviado um pedido de actualização.

Para evitar que o processo de replicação seja efectuado diversas vezes, o OBIHOC permite que uma aplicação envie um pedido de replicação de um conjunto de objectos minimizando desta forma os recursos a serem utilizados no dispositivo, pois ao invés de cada objecto ser tratado individualmente, são agrupados e replicados em conjunto, evitado actualização de referências desnecessárias.

4.1.2. DGC

Tal como referido anteriormente a utilização de recursos nos dispositivos deve ser controlada de forma a não serem utilizados de forma indevida. Esta atitude poderá levar a algumas situações em que os objectos sejam marcados como lixo, pois deixam de ser referenciados localmente. No entanto esta situação deverá ser cuidadosamente tratada, pois o

objecto não deverá ser tratado apenas localmente, mas sim num contexto global, ou seja, tendo em conta todas as réplicas que possam existir pelos diversos dispositivos. Esta situação implica que seja mantida a integridade referencial, ou seja, que os objectos mantenham as suas referências para os objectos certos.

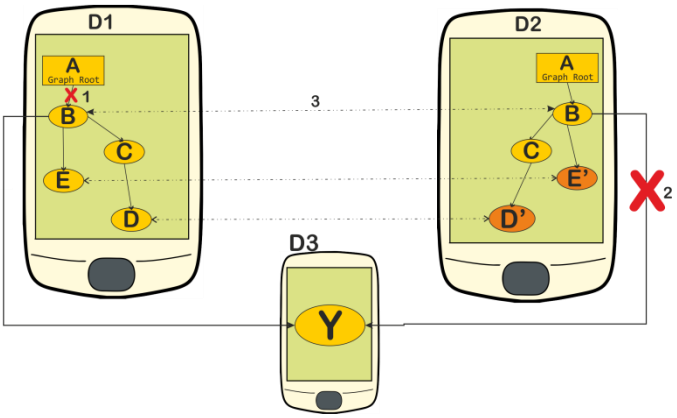

Figura 14 - Exemplo da utilização de DGC

Consideremos o cenário apresentado na Figura 14, em que o objecto B no dispositivo D1 foi replicado para D2. Neste caso o objecto B em D2 também terá uma referência para Y no dispositivo D3, tal como sucede em D1. Pode acontecer que durante a execução da aplicação em D1 o objecto B deixa de ser referenciado (situação **1**) e em D2 o objecto B deixa de referenciar Y (situação **2**). Neste caso será que Y deverá ser considerado inacessível e consequentemente considerado lixo? Na verdade o objecto Y não deverá ser considerado lixo, ao contrário do que a maioria dos mecanismos de GC diriam. O objecto Y não deverá ser considerado lixo pois o objecto B em D2 poderá ser actualizado e tendo em conta que em D1 o objecto B mantêm a referência para Y, esta actualização irá recriar a referência em D2 entre B e Y. Desta forma Y não deverá ser considerado lixo, apesar de em D1 B deixar de ser referenciado e em D2 B deixar de referenciar Y. Assim Y só deverá ser considerado lixo quando a união de todas as réplicas deixem de referenciar Y.

Em relação ao ambiente local o tratamento de objectos inacessíveis é deixado para o *garbage collector* (GC) local. No caso de um ambiente distribuído para além do GC local é necessário que haja colaboração entre este e o DGC.

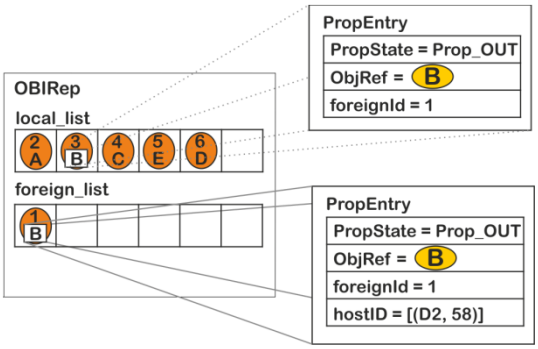

Figura 15 - Actualização do OBIRep após replicação de um objecto

Para o correcto funcionamento deste algoritmo são utilizadas as listas existentes no módulo OBIRep – local_list e foreign_list. Quando um objecto é replicado para outro dispositivo o módulo OBIRep executa duas acções sobre as suas listas, tal como na Figura 15:

1. Cria uma nova PropEntry semelhante à existente na local_list, no entanto está tem um novo campo – hostID. Este campo é uma lista de todos os dispositivos para os quais o objecto foi replicado. Para além disso cada elemento desta lista associa o identificador do dispositivo como o identificador local do objecto nesse dispositivo. No exemplo da Figura 15 podemos verificar que o objecto B foi replicado para o dispositivo D2 sendo-lhe atribuído o identificador 58 em D2.

   Esta nova PropEntry é adicionada à foreign_list sendo identificada pela sua posição na lista, neste caso a posição 1.

2. A PropEntry do objecto replicado da local_list é actualizada, ou seja, o valor do campo foreignID passa a ser a posição do objecto na foreign_list.

Com estas duas acções é possível a colaboração entre o GC local e o DGC pois quando um objecto é localmente inacessível é marcado como sendo lixo. No entanto antes de o objecto ser definitivamente eliminado o mecanismo de DGC vai verificar se existe alguma referência para esse objecto na foreign_list. Se existir significa que o objecto foi replicado para outro dispositivo e portanto não deverá ser eliminado sem a verificação que é globalmente inacessível. Para esta verificação é enviada uma mensagem para o dispositivo para o qual o objecto foi replicado por forma verificar se o objecto é inacessível. Caso a resposta seja positiva será contactado o próximo dispositivo para o qual o objecto foi replicado. Se a resposta for positiva de todos os dispositivos o objecto é considerado globalmente inacessível e, portanto, passível de ser eliminando localmente. Basta um dos dispositivo responder negativamente e o objecto não deverá ser eliminando pois poderá ser actualizado e desta forma voltar a estar acessível, tal como referido no exemplo da Figura 14.

Caso um objecto a ser replicado já esteja inscrito na foreign_list, ou seja, já foi replicado para outro dispositivo, essa informação também será transferida no processo de replicação. Para além disso todos os dispositivos para os quais o objecto foi replicado também serão notificados. Esta situação cria duas regras:

1. Antes de uma réplica ser enviada todos os dispositivos para os quais o objecto foi replicado deverão ser notificados, para adicionar no campo hostID da PropEntry respectiva esse facto.
2. Após uma réplica ser entregue, o dispositivo deverá notificar todos os dispositivos para os quais o objecto foi replicado, informado o identificador local desse objecto, confirmando assim a recepção do objecto.

Resumidamente este mecanismo de DGC permite que um objecto só seja considerado lixo, quando é globalmente inacessível, ou seja, quando todas as replicas existentes não referenciem esse objecto.

4.1.3. Migração de Objectos

Para o suporte à migração de objectos, nomeadamente a utilização de agentes móveis, a arquitectura utilizada é semelhante à situação de replicação de um objecto. No entanto existem algumas alterações motivadas pelas características destes agentes.

Um das principais características dos agentes móveis é a capacidade de movimentarem pela rede para desempenhar as suas funções. É portanto indispensável que esta movimentação seja tida em conta na arquitectura do OBIHOC.

Quando a aplicação cria um agente móvel são criadas mais duas estruturas de dados associadas a esse agente:

- **Agent Home** – este módulo permite a interacção da aplicação com o agente móvel e está localizado no dispositivo a partir do qual o agente foi lançado. É neste módulo que se mantenha a informação sobre a localização do agente.
- **Agent Mobile** – este modulo interage com o modulo Agent Home e desloca-se juntamente com o agente, ou seja, está alojado no mesmo dispositivo que o agente móvel.

Estas estruturas de dados permitem que a aplicação possa interagir com o agente móvel sem que para tal esta conheça a localização do agente, pois a comunicação é mediada pelo Agent Home e Agent Mobile. Desta forma pode-se ver o Agent Mobile como uma extensão do Agent Home no dispositivo visitado pelo agente.

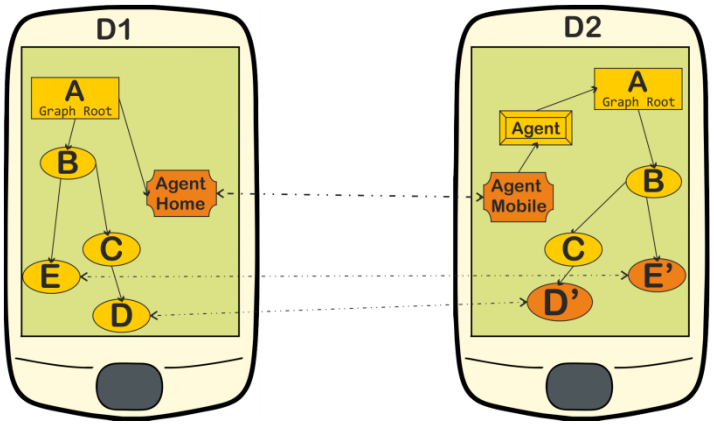

Figura 16 - Migração de Objectos

Consideremos o cenário em que uma aplicação no dispositivo D1 envia um agente móvel para a rede (Agent) e juntamente com este segue o Agent Mobile respectivo. A determinada altura o Agent encontra-se alojado no dispositivo D2, Figura 16, e a aplicação em D1 pretende interagir com o agente. Como a informação sobre a localização do agente está no módulo Agent Home a aplicação interage com este módulo supondo que esta a interagir directamente com o agente. No entanto essa interacção na realidade realiza-se entre os módulos Agent Home e Agent Mobile. O Agent Home ao receber o pedido por parte da aplicação em D1, reenvia esse pedido para Agent Mobile, localizado em D2, que posteriormente executa esse pedido no Agent. O processo inverso é realizado nos mesmos moldes.

Com este mecanismo a aplicação não precisa de saber a localização do agente para puder interagir com este. Esta interacção é realizada pelo OBIHOC, tornando-a completamente transparente para a aplicação.

4.2. Comunicação

A replicação de objectos implica que existam duas componentes: (1) um servidor para disponibilizar objectos e (2) um ou mais clientes que possam interagir com esse servidor.

Figura 17 - Arquitectura de comunicação

Tendo em conta que este trabalho está focado para um ambiente *ad-hoc* existem vários aspectos a ter em conta:

- Uma aplicação deverá ter a possibilidade de desempenhar o papel de servidor e cliente ao mesmo tempo ou então um dos casos anteriores em separado.
- As características dos dispositivos móveis nomeadamente algumas limitações em termos de processamento e memória.
- A tecnologia de comunicação a ser utilizada.

A arquitectura para esta componente, representada na Figura 17, foi desenhada tendo em conta três módulos:

- **Sessão** – módulo responsável pela ligação. Neste módulo são definidos alguns parâmetros tais como a tecnologia a ser utilizada e propriedades da ligação.
- **Cliente** – módulo que implementa o protocolo de comunicação, ou seja, criar os tipos de dados a serem enviados no pedido ao servidor.
- **Servidor** – módulo que recebe pedidos vindos dos vários clientes. No caso de haver retorno aos pedidos recebidos é este módulo que está responsável pela criação dos tipos de dados a serem enviados.

Dos módulos apresentados anteriormente, o módulo Sessão é o que necessita de uma configuração por parte da aplicação. Tendo em conta a existência de várias formas de comunicação sem fios – *Bluetooth* e *Wi-Fi* – é necessário que OBIHOC seja o mais flexível possível de forma a permitir a utilização destas tecnologias de comunicação. A escolha da tecnologia deverá ser feita pela aplicação, pois somente esta saberá as suas necessidades, por isso este módulo necessita de ser criado e configurado pela aplicação. Durante este processo a aplicação deverá indicar qual a tecnologia a ser utilizada, bem como as respectivas propriedades.

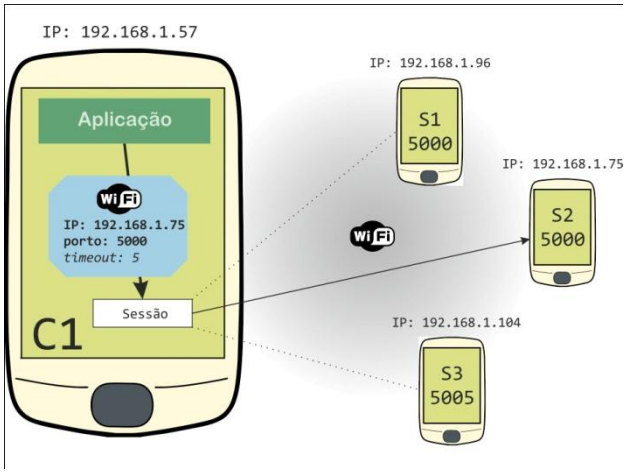

Figura 18 - Exemplo definição da tecnologia de comunicação

Imaginemos a situação em que a tecnologia escolhida é *Wi-Fi*. Neste caso a aplicação deverá indicar que essa é a tecnologia a ser utilizada bem como o endereço IP do dispositivo a ser contactado e o respectivo porto do servidor. Na Figura 18 é possível ver um pequeno exemplo desta situação, onde uma aplicação cliente no dispositivo C1 inicia uma comunicação com o servidor S2 cujo IP é 192.168.1.75 e porto 5000.

Aquando da execução deste passo existem alguns parâmetros obrigatórios, no caso anterior o IP e porto do servidor. No entanto existem mais alguns parâmetros facultativos que poderão ser definidos:

- ***timeout*** – tempo total em que a sessão é válida. No exemplo anterior esse tempo seria de cinco minutos. A utilização deste parâmetro permite que a aplicação definia um tempo razoável para a ligação se manter activa. Desta forma é possível reduzir a utilização de recurso de forma desnecessária. No caso de não ser definido nenhum valor, será utilizado um valor predefinido (5 minutos).
- **Utilizador** – identificador da aplicação na rede. No caso de este parâmetro ser omisso ele será gerado utilizando uma propriedade do dispositivo, por exemplo o endereço IP.
- **Protocolo de segurança** – protocolo de segurança a ser utilizado durante a comunicação. Este protocolo deverá ser definido pela aplicação e poderá incluir por exemplo autenticação e/ou cifra de dados trocados. No caso de este parâmetro ser omisso não será utilizado nenhum protocolo de segurança.

Tendo em conta as inúmeras aplicações que poderão utilizar este middleware foi necessário criar uma arquitectura de comunicação que pudesse ser o mais flexível possível, de forma a adaptar-se às necessidades específicas de cada aplicação. Por exemplo, a utilização de um protocolo de segurança numa aplicação em que a troca segura de dados não seja necessária, estaria a consumir recursos dispensáveis do dispositivo.

Para se iniciar um servidor a aplicação deverá indicar o porto onde este ficará à espera para receber pedidos. A partir deste momento o servidor está apto a receber grafos de objectos da aplicação e disponibiliza-los para os restantes dispositivos, sem que para tal a aplicação necessite de interagir com o servidor.

Para o funcionamento do protocolo de comunicação foram definidos seis tipos de pedidos:

- **BeginSession** – pedido para ser iniciada uma sessão. Neste pedido são definidos alguns parâmetros da sessão.
- **RefreshSession** – pedido para renegociar a sessão. Este pedido tipicamente é utilizado quando é necessário prolongar o tempo da sessão.
- **EndSession** – pedido para finalização da sessão.
- **LookUp** – pedido para obtenção do identificador da raiz do grafo de objectos, consoante o nome da raiz do grafo pretendido.
- **XMLGet** – pedido de replicação de um determinado objecto. Neste pedido a aplicação explicitamente indica o identificador do objecto bem como a profundidade de replicação que deseja.
- **XMLPut** – pedido para actualização de um determinado objecto replicado anteriormente.

Para além destes pedidos poderão ser utilizados mais alguns métodos provenientes da utilização de um protocolo de segurança. A implementação destes métodos deverá ser feita pelo programador da aplicação pois somente este saberá quais os melhores mecanismos de segurança a serem utilizados (por exemplo, método de autenticação e escolha de algoritmo de cifra). Neste caso poderão ser utilizados quatro pedidos:

- **LogIn** – pedido de autenticação.
- **LogOut** – pedido para finalização de autenticação. Este pedido não implica o fim da sessão.
- **BeginCipher** – pedido para ser iniciada uma transmissão de dados cifrados.
- **EndCipher** – pedido para finalizar comunicação cifrada.

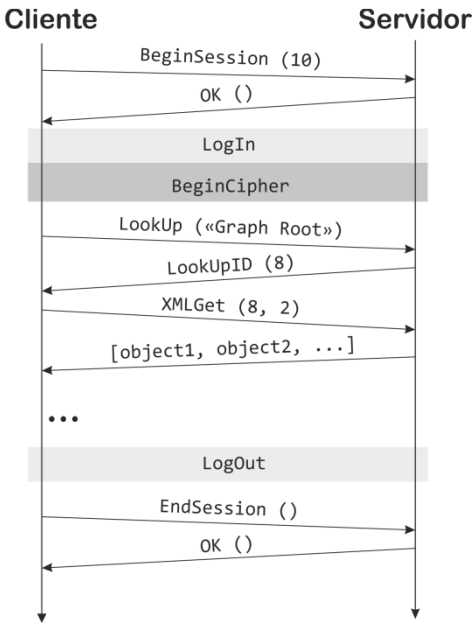

Figura 19 - Exemplo protocolo de comunicação

Um exemplo da utilização do protocolo de comunicação está patente na Figura 19. Neste caso um Cliente pretende iniciar uma sessão com um servidor pelo período de 10 minutos. O servidor recebe esse pedido e informa o cliente que a sessão está iniciada. A partir deste momento é possível a troca de mensagens entre os intervenientes para as diversas situações abrangidas pelo OBIHOC. No entanto antes desta troca de mensagens poderá ser necessário a troca de mais mensagens para cumprimento do protocolo de segurança. Neste caso essas mensagens serão trocadas no período de LogIn e BeginCipher. Durante este período poderão ser trocadas mensagens de forma a por exemplo autenticar o cliente perante o servidor e vice-versa bem como troca de dados para que as restantes mensagens da sessão sejam cifradas.

Continuando o exemplo da Figura 19, após o inicio da sessão, o cliente começa por pedir ao servidor o identificador da raiz do grafo de objectos que pretende. Este grafo de objectos é identificado por um nome, que deverá ser indicado no pedido. Este nome é atribuído pela aplicação e como tal pode ser visto como uma medida de protecção dos objectos pois apenas quem souber esse nome poderá pedir a sua replicação. Neste caso o grafo “Graph Root” é o pretendido e como resposta o cliente recebe o identificador 8. De seguida o cliente envia um pedido para replicação desse grafo de objectos com profundidade 2. Como resposta o cliente recebe a réplica do grafo de objecto.

Durante o tempo em que a sessão é válida podem continuar a ser trocadas mensagens a qualquer momento. No entanto podem surgir situações em que a sessão deixará de ser necessária, por exemplo quando a aplicação é fechada. Neste caso a sessão deverá ser

concluída de forma explícita, ou seja, com o envio da mensagem EndSession. Desta forma evita-se a utilização de recursos desnecessários no dispositivo. Como é possível verificar na Figura 19 o envio do pedido LogOut não indica que a sessão seja finalizada, pois este pedido de LogOut é definido pela aplicação e a sua utilização não implica obrigatoriamente que aplicação deixe de comunicar com o servidor.

4.3. Segurança

Para a definição de políticas de segurança, a arquitectura OBIHOC utiliza a linguagem SPL. Devido às características enunciadas anteriormente (Políticas de segurança, pág. 22) a utilização desta linguagem foi a mais indicada tendo em conta o ambiente de utilização das aplicações.

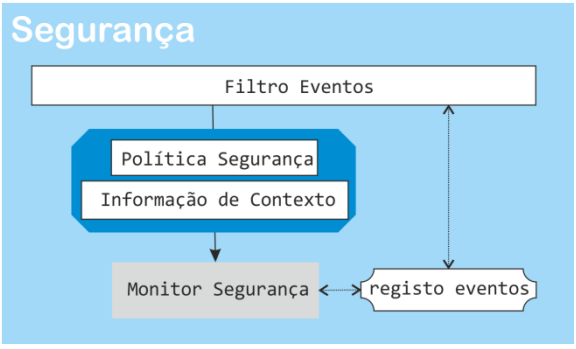

Figura 20 - Arquitectura de segurança

Devido às características deste trabalho, nomeadamente a migração de objectos entre dispositivos, é fulcral poder oferecer mecanismo de protecção do próprio dispositivo e dos objectos presentes nas migrações. No caso especifico do OBIHOC a principais precauções estão relacionadas com:

- Interacção entre agentes móveis e dispositivos – utilização indevida de recursos por parte de um agente mal intencionado num dispositivo e vice-versa;
- Interacção entre agentes – interacção indevida entre agentes móveis.

Tal como podemos verificar na Figura 20 a arquitectura de segurança tem cinco módulos:

- **Filtro Eventos** – módulo que permite detectar quais os eventos que podem desencadear a necessidade de analisar as políticas de segurança;
- **Política de Segurança** – política de segurança a ser apresentada para validação;
- **Informação de Contexto** – informação associada à política a ser analisada;

- **Monitor de Segurança** – módulo responsável por analisar a política de segurança apresentada. Para esta análise este módulo poderá recorrer à informação de contexto apresentada conjuntamente com a política de segurança. Após a análise será dado o retorno se a política de segurança foi validada ou rejeitada;
- **Registo de eventos** – módulo para armazenamento de eventos importantes e que será utilizado para situações em que acontecimentos do passado influenciem a validação.

Durante a sua execução o OBIHOC pode desencadear diversos tipos de eventos e por esse motivo a utilização do Filtro Eventos é bastante importante, pois impede a utilização de recursos no dispositivo de forma desnecessária. Quando é desencadeado um evento que possa colocar em risco o dispositivo (migração de objectos ou invocações remotas são alguns exemplos) o Monitor de Segurança recebe um pedido de autorização para a execução desse evento. Após análise da política de segurança e da respectiva informação de contexto o Monitor de Segurança informa o sistema se esse evento é autorizado ou não, notificando também o Registo de Eventos da decisão tomada. Para a avaliação das políticas de segurança recorre-se à linguagem para definição de políticas de segurança SPL.

A linguagem SPL é baseada em quatro entidades: objectos, grupos, regras e políticas. As regras estabelecem restrições na relação entre objectos e grupos. As políticas resultam na decomposição de múltiplas regras e grupos, e são o principal módulo da linguagem SPL.

```
type object {
  string name;       // The name of the object
  user owner;        // The owner of the object
  string type;       // A string identifying the type
  object set groups;// The sets containing the object
  string homeHost;  // The host where the user
}                    // is defined

type user extends object {
  rule set userPolicy; // User private policies
}
type operation extends object {
  number ID;           // operation Id
}
type event extends object {
  user author;         // The author of the event
  object target;       // The target of the event
  operation action;    // The performed action
  object set parameter;// The set of parameters
  number time;         // The time instant
  object task;         // The task to which the event
}                      // belongs to
```

Figura 21 - Exemplo da definição de um tipo[7]

Os objectos em SPL são gerados tendo em conta uma determinada interface, cujas propriedades podem ser acedidas ou alteradas. Estes objectos podem representar o modelo de autorizações interno bem como objectos vindos de outras plataformas. A maioria dos objectos é de origem externa como por exemplo agentes móveis. A cada objecto externo está associado

um tipo. Este tipo é usado para definir a interface cujo objecto implementa conjuntamente com as suas propriedades. Na Figura 21 está um exemplo da criação de um tipo object com diversas propriedades – name, owner, type, etc. – e a definição de mais três tipos. Outra propriedade da linguagem SPL é a possibilidade de criar tipos cujas propriedades são tipos anteriormente definidos. Esta propriedade ganha uma importância relevante pois permite a criação de grupos. Estes grupos permitem a associação de entidades com propriedades semelhantes, facilitando a definição das políticas de segurança.

```
OwnerRule:  ce.target.owner = ce.author ::  true;
```

Figura 22 - Exemplo da criação de uma regra[7]

As regras na linguagem SPL permitem estabelecer restrições sobre as operações de autorização. Desta forma uma política de autorização pode-se considerar um conjunto de regras a serem validadas. Para esta validação as regras podem representar três valores lógicos: *allow*, *deny* ou *not apply*. Estes valores definem a validação ou não dos eventos desencadeados pelo OBIHOC. A Figura 22 exemplifica a criação de uma regra simples. Neste caso a regra *OwnerRule* define que qualquer evento desencadeado pelo dono de um objecto, sobre esse mesmo objecto, é permitido.

As políticas na linguagem SPL são um conjunto de regras e grupos que determinam a autorização ou proibição para um determinado evento. Em cada política existem uma regra especial (*Query Rule*) que é responsável pelo funcionamento da política bem como da sua decisão. A *Query Rule* poderá invocar outras regras de forma a cumprir devidamente a política de segurança.

Tal como explicado anteriormente o OBIHOC permite a definição de políticas de segurança para agentes móveis e dispositivos. Para além disso é necessário oferecer mecanismo que possam impedir alguns tipos de ataques – *denial-of-service*[8] ou fluxo de dados indesejados. Desta forma OBIHOC suporta:

- Políticas para agentes móveis que permitem proteger esses agentes de operações prejudicais. Estas operações podem ter origem numa programação pouco correcta desses agentes e na interacção de entidades não confiáveis como outros agentes ou dispositivos.

[7] Imagem retirada de [23]

[8] Este tipo de ataque consiste em bloquear um dispositivo impedindo que possa fornecer os seus serviços. Por exemplo abusar dos recursos do dispositivo impedindo que responda a tempo devido a pedido legítimos.

- Políticas que permitam proteger os dispositivos de operações prejudicais desencadeadas por agentes móveis.

No entanto podem surgir situações em que se torna necessário manter um histórico de forma a autorizar um evento.

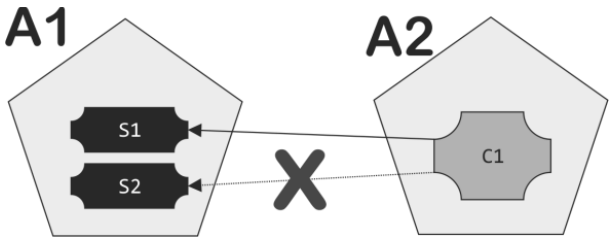

Figura 23 - Exemplo da utilização de política de segurança entre agentes móveis

Consideremos o cenário apresentado na Figura 23, em que um agente móvel (A1) oferece dois serviços (S1 e S2). Este agente implementa uma política de segurança que não permite que outro agente (A2) possa utilizar os seus serviços em simultâneo. Neste caso a politica a ser especificada é no sentido de proteger o agente de outro agente. No entanto, para ser assegurada essa protecção é necessário recorrer a possíveis eventos anteriores, de forma a saber se A2 já está a utilizar algum dos serviços de A1. Neste exemplo o acesso ao serviço S2 por A2 será negado pois já está a utilizar S1. Uma possível definição desta política é:

```
Policy ControloServicos {
        ?ControloServicos:
                NOT EXIST e IN PastEvents {
                        ce.source = “A2” & ce.operation = “Invocacao” &
                        ce.operation.service = S2
                        ::
                        e.source = “A2” & e.operation = “Invocacao” &
                        e.operation.service = S1
                };
}
```

A definição destas politicas de segurança deverá ser alvo de consideração pelo programador da aplicação, pois desta forma poderá evitar diversos tipos de ataques de forma relativamente simples.

Para ajudar o programador nessa tarefa o ideal seria que a criação destas políticas fosse possível através de uma interface gráfica. Para além de ajudar seria uma forma de reduzir erros na criação das políticas de segurança. Durante este trabalho não foi possível desenvolver essa interface, ficando o seu desenvolvimento como um trabalho a ser realizado no futuro.

4.4. Arquitectura Geral

Como resumo deste capítulo é apresentada na Figura 24 a arquitectura geral da solução. Esta figura conjuga as três componentes referidas anteriormente.

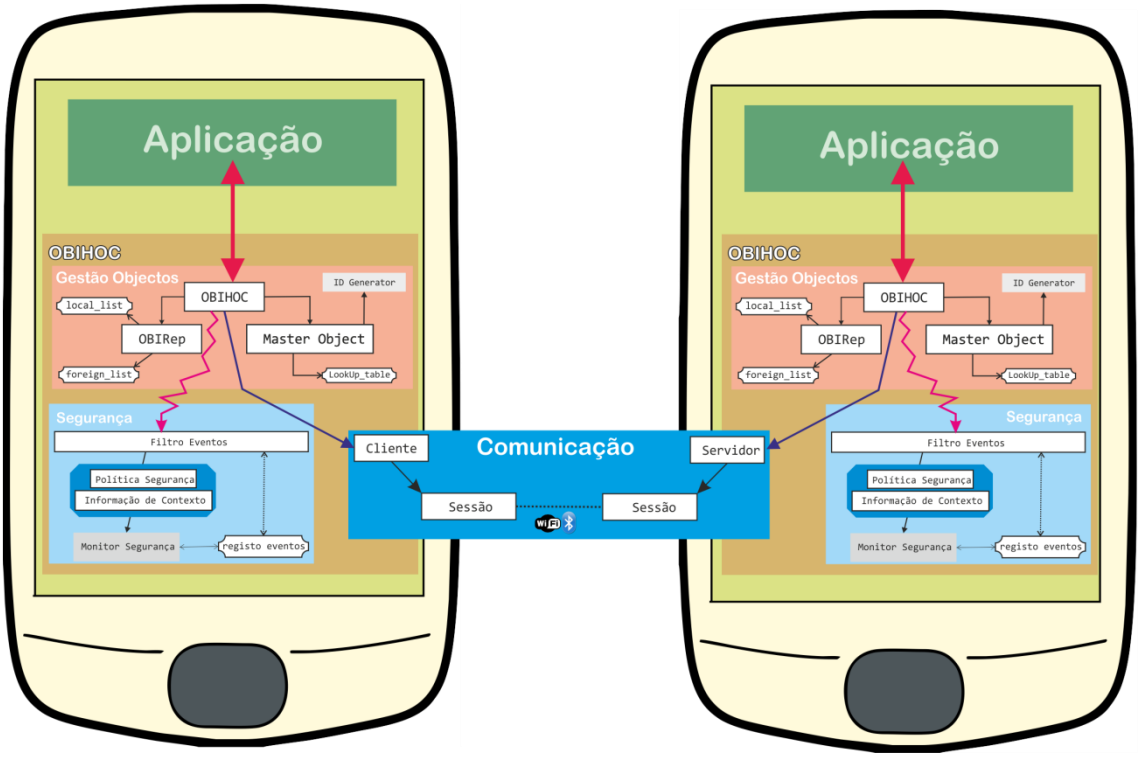

Figura 24 - Arquitectura geral da solução

Como se pode verificar, a aplicação interage somente com o módulo OBIHOC. Esta interacção permite adicionar objectos para replicação, tipicamente no caso do servidor, e obter grafos de objectos de outro dispositivo, tipicamente no caso do cliente.

Para estas acções é necessário o módulo Gestão de Objectos para gestão das referências dos objectos locais, gestão dos objectos replicados e controlo do mecanismo de DGC.

O módulo Comunicação trata do envio de todo o tipo de mensagens entre dispositivos. Este módulo na sua criação necessita de ser configurado por parte da aplicação, nomeadamente na escolha da tecnologia de comunicação a ser utilizada e os respectivos parâmetros.

O módulo de Segurança é desencadeado somente em determinados eventos, como por exemplo na migração de agentes entre dispositivos. Este módulo valida se esses eventos são passíveis de ser executados, tendo em conta as politicas de segurança definidas.

5. Implementação

Este trabalho foi implementado para o sistema operativo Android[9]. Este recente sistema operativo está destinado principalmente para dispositivos móveis e como tal foi escolhido para ser a plataforma de implementação. Tem a característica de ser na sua maioria *open-source* e por esse motivo bastante aliciante para os programadores desenvolverem novas aplicação. Para além disso tem disponível um *Software Development Kit* (SDK) que permite emular o comportamento do sistema operativo como se fosse um dispositivo móvel (Figura 25). Por exemplo é possível definir o espaço disponível, bem como o controlo de diversos sensores.

Figura 25 - Emulador Android

O desenvolvimento de aplicações para este sistema operativo é feito recorrendo à linguagem de programação Java [34]. Por esse motivo OBIHOC também foi desenvolvido nessa linguagem, sendo executado no topo da máquina virtual Java [35] (JVM – Java virtual machine). Para além disso a JVM oferece suporte a diversas funcionalidades a serem utilizadas – RMI, *sockets*[10]. Um dos factores positivos do OBIHOC é que não necessita de alterações na JVM e este motivo torna-o facilmente portável.

Tendo em conta que este sistema operativo foi desenvolvido para dispositivos móveis a JVM disponibilizada é compactada, ou seja, não apresenta todas as funcionalidades da versão *desktop*. No entanto para a implementação deste *middleware* essa situação não se revelou um problema pois todas as funcionalidades necessárias estavam contempladas nesta versão compactada.

---

[9] Google Inc. (2007, Abril) Android.com - Android at Google I/O. - http://www.android.com/

[10] Abstracção computacional que mapeia directamente uma porta de transporte (TCP ou UDP) com um endereço de rede. Com esse conceito, é possível identificar unicamente uma aplicação ou serviço na rede de comunicação.

5.1. Classes e Interfaces

Este *middleware* pretende facilitar o desenvolvimento de aplicações aos programadores. Por esse motivo o programador deverá estar focado no desenvolvimento das classes e métodos inerentes ao funcionamento da aplicação, deixando para o OBIHOC a geração de todos os mecanismos necessários à sua execução numa rede ad-hoc.

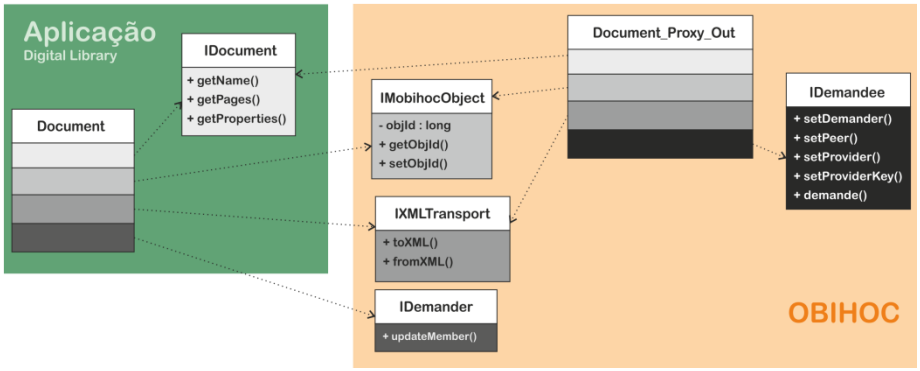

Figura 26 - Diagrama de classes que ilustra as interfaces OBIHOC

Imaginemos que um programador está a desenvolver a aplicação "Digital Library" (Figura 26). Para tal cria uma interface – IDocument – que será implementada pela classe Document. A definição destas duas componentes da aplicação fica a cargo do programador, pois a sua utilização está somente dependente da aplicação.

Para facilitar a implementação do OBIHOC nas aplicações é necessário que essas classes estendam algumas interfaces específicas do OBIHOC e a criação de uma nova classe – *proxy*. Tal como patente na Figura 26 as interface necessárias são:

- **IMobihocObject** – permite que os objectos tenham uma propriedade que será alterada pelo identificador gerado pelo OBIHOC. As classes que implementem esta interface são consideradas passíveis de replicação e portanto também deverão implementar as interfaces IXMLTranspor e IDemander/IDemandee.
- **IXMLTransport** – permite que seja possível converter a classe num objecto passível de ser replicado pelo OBIHOC bem como a sua recuperação no dispositivo para o qual foi replicado.
- **IDemander** – permite que sejam trocadas as referências após a chegada de um objecto replicado para troca com o proxy que o substituiu.
- **IDemandee** – permite que seja executado o processo de replicação automática. Desta forma quando um proxy é invocado, automaticamente é iniciado o processo de replicação do objecto correspondente.

Para a correcta implementação do OBIHOC na aplicação o seu programador deve:

1. Definir a Interface (IDocument) e criar a classe (Document).
2. Implementar na classe Document as interfaces (1) IMobihocObject, (2) IXMLTransport e (3) IDemander.
3. Criar o proxy (Document_Proxy_Out) que implementa a interface do próprio objecto (IDocument) bem como (1) IMobihocObject, (2) IXMLTransport e (3) IDemandee.

O processo de extensão de código bem como a criação dos respectivos proxy´s poderá ser automatizado, ficando a implementação desta funcionalidade como uma prioridade para trabalhos futuros.

Para a implementação do processo de comunicação foi utilizada um solução bastante simples e minimalista. No sentido de reduzir os recursos a serem utilizados nos dispositivos a comunicação é realizada recorrendo directamente a funcionalidades de comunicação oferecidos pela JVM (socket´s). Foram pensadas outras soluções (servidor Web, servidor XMLRPC) que não se revelaram serem as ideais pelos consumos extras inerentes e por necessitaram da instalação e/o configuração de software externo ao *middleware*.

Devido às semelhanças entre a linguagem SPL e Java a maior parte das acções do compilador são de tradução: cada política SPL é traduzida numa classe Java, cada regra numa função sem parâmetros (excepto *Query Rule* que recebe como parâmetros o evento actual) e os objectos SPL são traduzidos em interfaces Java.

O trabalho foi desenvolvido recorrendo ao software Eclipse, versão Galileo[11]. Este IDE (*Integrated Development Environment*) permite adicionar *plugins* para auxiliar os programadores no desenvolvimento dos seus projectos. Neste caso foi utilizado o plugin ADT (*Android Development Tools*) pois permite a integração do IDE com o SDK, nomeadamente duas ferramentas: (1) ADB (*Android Debug Bridge*) para debug e (2) DDMS (*Dalvik Debug Monitor Server*) para controlo e gestão dos processos e recurso utilizados no emulador. Foi utilizada a versão 2.2 do SDK, que é a versão disponibilizada mais recente.

[11] The Eclipse Foundation. (2010) Eclipse.org home - http://www.eclipse.org/

6. Metodologia de avaliação do trabalho

A avaliação deste trabalho foi desenvolvida em duas fases: (1) definição de microbenchmark e (2) análise de desempenho de uma aplicação desenvolvida sobre o OBIHOC. Desta forma é possível obter resultados credíveis de forma a avaliar de forma realista o trabalho realizado. Nesse sentido a avaliação vai-se basear nos seguintes aspectos:

- **Recursos utilizados no dispositivo** – devido às limitações de memória e processamento nos dispositivos móveis é vital analisar o impacto que a execução do OBIHOC produz nos dispositivos.
- **Informação excedente trocada entre dispositivos** – uma vez que a utilização de redes ad-hoc implica a troca de dados, é importante minimizar os dados excedentes.
- **Impacto na aplicação** – apesar do aumento de funcionalidades da aplicação, estas não deverão influenciar a fluidez da aplicação, para que não se perca a noção de transparência pretendida.

Neste capítulo é apresentada a aplicação desenvolvida e como os cenários de testes executados podem contribuir para a obtenção de resultados relevantes para a avaliação do trabalho. Em cada um dos seguintes subcapítulos é apresentada a metodologia realizada para obtenção dos dados, a sua relevância e análise dos mesmos.

6.1. Aplicação de demonstração

A aplicação de demonstração, intitulada *Digital Library*, é uma aplicação desenhada e implementada para o Android versão 2.2. Esta aplicação permite que sejam consultados, adicionados, alterados e apagados diversos conteúdos dessa biblioteca. Estes conteúdos podem ser de três tipos: (1) áudios (músicas, gravações de voz, etc.), (2) documentos (livros, relatórios, etc.) e (3) vídeos.

Esta aplicação foi escolhida pois permite evidenciar as novas funcionalidades que a utilização de aplicações numa rede ad-hoc permite. De entre as novas funcionalidades salienta-se:

- Partilha de conteúdos da biblioteca sem necessidade de utilizar um serviço externo, sendo criado para tal uma rede ad-hoc.
- Possibilidade de manter a execução da aplicação, através da replicação de conteúdos, em situações de falha de comunicação. Numa rede ad-hoc esta situação

poderá ocorrer com naturalidade, pois estas redes caracterização pela entrada e saída de dispositivos da rede.

- Utilização mais eficiente dos recursos do dispositivo. Como esta aplicação pode envolver a transferência de grandes quantidades de dados é importante que os dados transferidos sejam realmente necessários para as intenções do utilizador.

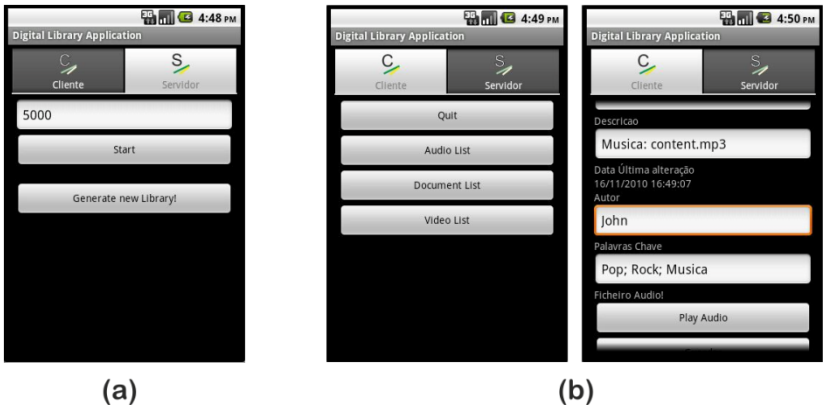

Figura 27 – Aplicação Digital Library

Na Figura 27 é apresentada a interface da aplicação. Esta aplicação, recorrendo ao OBIHOC, permite que os seus conteúdos sejam disponibilizados para outros utilizadores. Esta partilha de conteúdos é feita recorrendo a uma ligação Wi-Fi pois, por limitação do emulador, a ligação Bluetooth não está disponível. A aplicação tem duas interfaces distintas: (1) interface Servidor (Figura 27 (a)) que permite controlar as opções do servidor (porto a ser utilizado e disponibilização do serviço) e (2) a interface Cliente (Figura 27 (b)) que permite aceder a uma biblioteca remota e respectivos conteúdos.

O desenho desta aplicação foi pensado de forma a permitir que a integração com OBIHOC potencie a sua utilização e/ou programação.

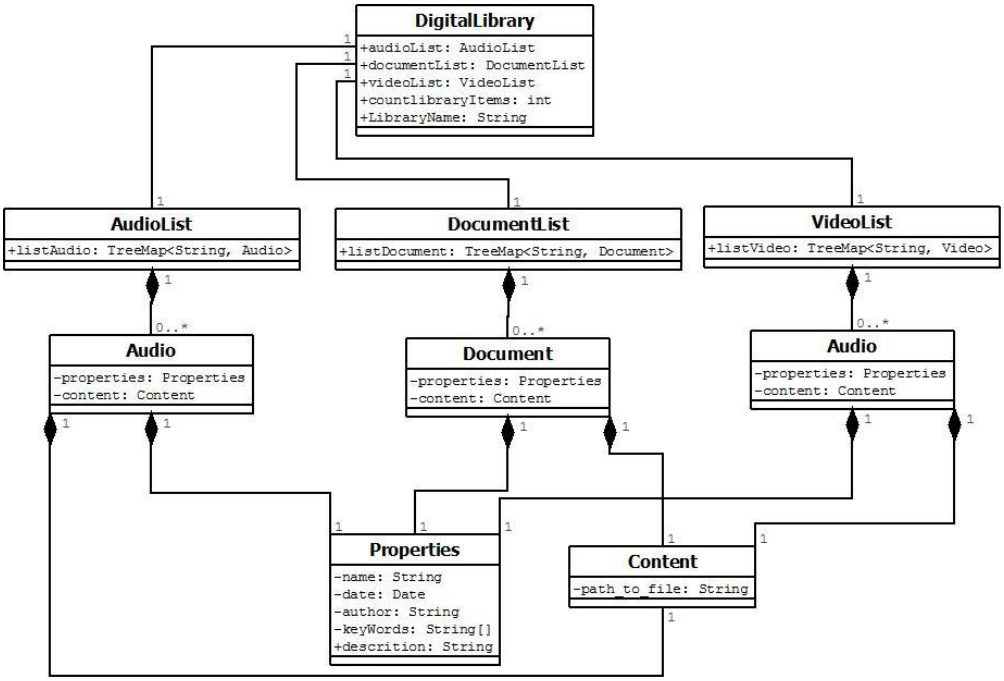

Figura 28 - Diagrama de classes da aplicação Digital Library

Como é possível verificar no diagrama de classes da Figura 28, a aplicação tem quatro níveis. Uma das características mais importantes desta aplicação é a possibilidade de aceder a conteúdos tendo em conta o seu tipo, ou seja, se um utilizador estiver interessado em conteúdos áudio poderá apenas aceder a essa parte da biblioteca. Esta característica ganha maior relevo no caso de um ambiente distribuído, pois permite aceder remotamente aos conteúdos desejados sem que para tal seja necessário aceder à biblioteca na sua totalidade.

É de referir também que os dados dos conteúdos, ou seja a sua informação binária, são guardados em ficheiros locais ficando a aplicação apenas com a indicação do caminho para esse ficheiro (atributo *path_to_file* da classe Content). No caso de esta classe ser replicada é automaticamente adicionado o conteúdo desse ficheiro para que a aplicação que recebe a réplica possa criar o ficheiro com os mesmos dados binários.

6.2. Cenário de teste

Para a avaliação deste trabalho foi elaborado um cenário de teste que utiliza a aplicação Digital Library. Este cenário de teste consiste no acesso a todos os conteúdos de uma biblioteca remota de forma sequencial, ou seja são acedidas as propriedades e conteúdo (Properties e Content) de todos os áudio e após esses os documentos e por fim os vídeos. A

cada um destes conteúdos está associado um ficheiro guardado localmente com as seguintes propriedades:

- Audio – ficheiro .mp3 com um 484KB;
- Documento – ficheiro .pdf com 100KB;
- Video – ficheiro .mp4 com 988KB;

A aplicação foi parametrizada para que durante a sua execução se um determinado objecto for invocado, mas na realidade este representa um proxy, automaticamente é desencadeado o processo de replicação desse objecto e posteriormente é invocado o método pretendido no objecto recentemente replicado. A profundidade a ser aplicada neste processo depende do tipo de teste que está em execução.

Este cenário foi realizado recorrendo a dois emuladores a serem executados num PC com processador Intel® Core(TM)2 Duo com 2.1GHz, 3GB de memória RAM e com sistema operativo Windows 7 Professional. Devido ao facto dos emuladores estarem a ser executados na mesma máquina a velocidade de comunicação atingida não corresponde a um cenário real. Por esse motivo foi definido nos emuladores que a comunicação deveria apresentar uma velocidade semelhante à utilização do protocolo de comunicação HSDPA [37], vulgarmente conhecido no ambiente móvel como 3.5G.

6.3. Recursos utilizados

Tal como referido anteriormente, os dispositivos móveis apresentam algumas limitações de recursos. Neste caso são analisados alguns microbenchmark tais como: tempos de acesso a um determinado objecto que ainda não foi replicado e a memória total utilizada no dispositivo.

6.3.1. Tempo de acesso

Para a obtenção destes dados foi executado o cenário de teste anteriormente referido com uma profundidade de 0, ou seja cada objecto é replicado individualmente, não sendo abrangido a situação de replicação de grupos de objectos. A biblioteca replicada era composta por 10 conteúdos áudio, 10 documentos e 10 vídeos. Este cenário foi executado 10 vezes sendo que os resultados apresentados representam a média da execução dos cenários.

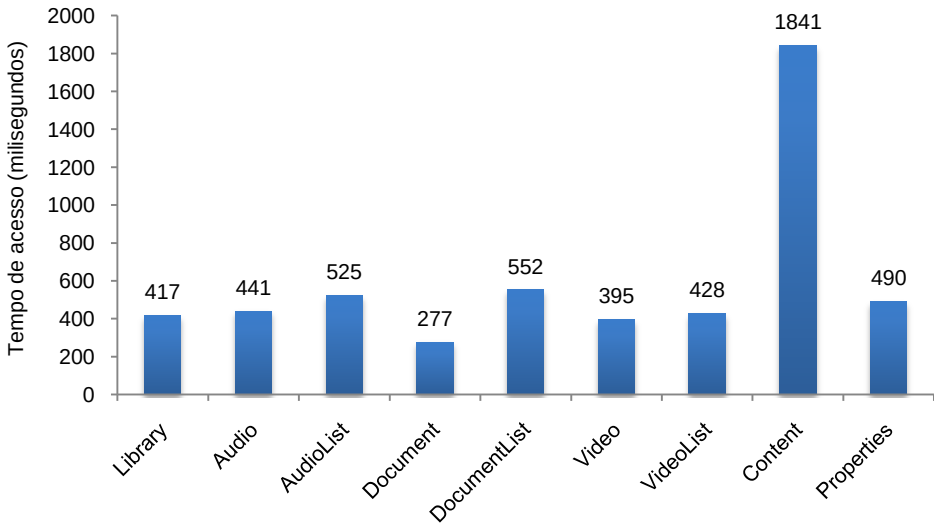

Gráfico 1 - Tempo total de acesso

O Gráfico 1 apresenta os tempos de acesso aos vários objectos existentes no grafo de objectos da aplicação (Figura 28). Estes valores representam o tempo de processamento para acesso ao proxy, a replicação do objecto do dispositivo remoto, a troca de referências e por fim a execução do método pretendido.

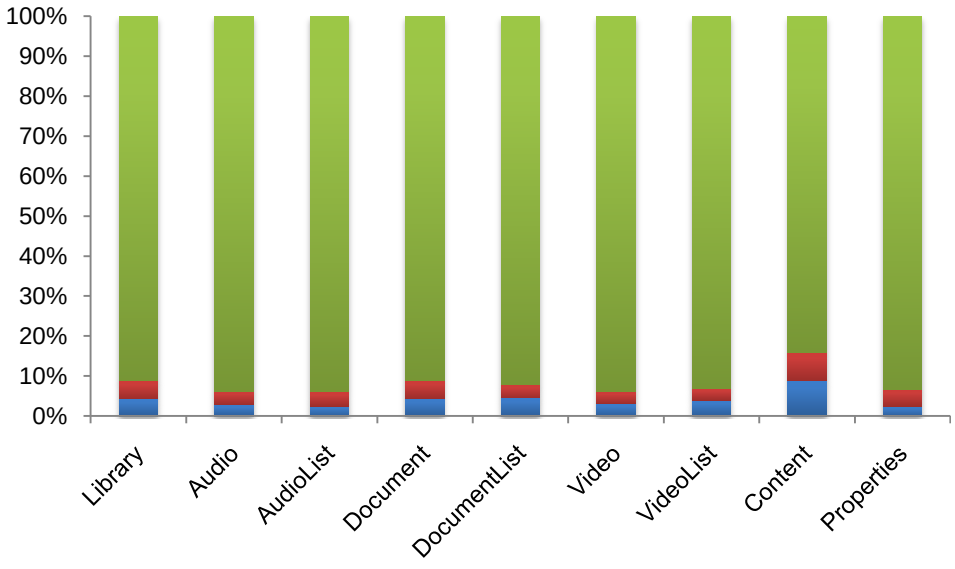

Gráfico 2 – Percentagem de tempo em cada processo de replicação

Complementando a informação do Gráfico 1, o Gráfico 2 apresenta a percentagem do tempo total utilizada por cada acção no decorrer do processo de replicação. Os processos representados neste gráfico são: (1) criação das estruturas de dados a serem enviadas do

servidor para o cliente (a azul), (2) conversão da informação recebida do servidor para objectos a serem utilizados pela aplicação, que inclui a gestão de referencias (a vermelho) e (3) transferência dos dados na rede (a verde). Como se pode observar uma grande percentagem do tempo necessário para o acesso a um objecto que ainda não foi replicado é desembolsado na transferência de dados entre os dispositivos. Este dado permite concluir que a utilização deste middleware não afecta directamente o tempo de resposta da aplicação, no sentido que os dados trocados são essenciais para o correcto funcionamento da aplicação, sendo esta troca independente do middleware.

6.3.2. Memória utilizada

Para a obtenção destes dados foi executado o cenário de teste anteriormente referido com uma profundidade de 0, ou seja cada objecto é replicado individualmente, não sendo abrangido a situação de replicação de grupos de objectos. A biblioteca replicada era composta por 20 conteúdos áudio, 20 documentos e 20 vídeos.

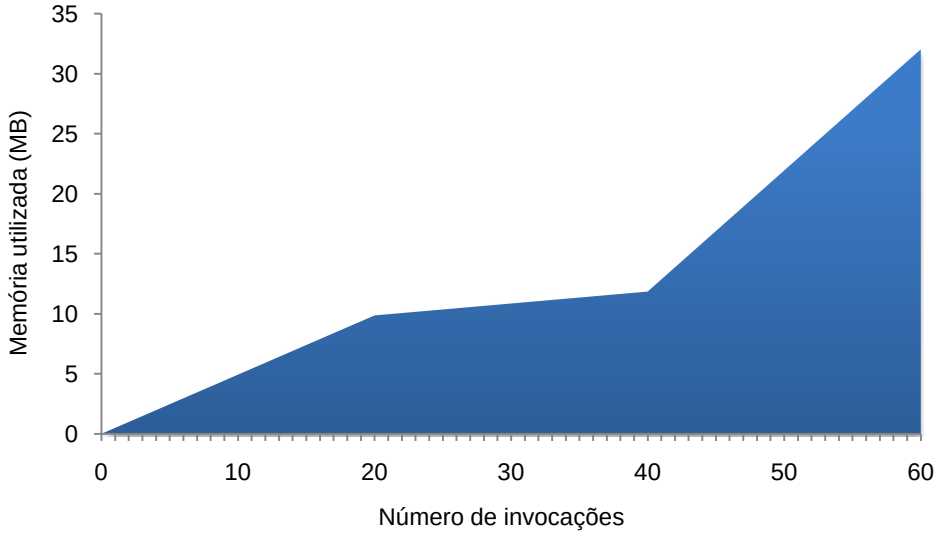

Gráfico 3 - Memória utilizada

O Gráfico 3 apresenta a evolução da memória utilizada no dispositivo durante a execução do cenário anteriormente descrito. Como é possível observar o gráfico apresenta três inclinações contastes. Estes três momentos representam a invocação dos conteúdos de cada um dos tipos, ou seja, entre a invocação 0 e 20 são as invocações em conteúdos do tipo Audio, entre 20 e 40 do tipo Documento e as restantes para Video. O facto de no caso de teste se ter utilizado ficheiros com o mesmo tamanho para o mesmo tipo de conteúdo implica que a utilização de memória no dispositivo seja constante entre invocações desse tipo de conteúdos.

Por este mesmo motivo os três momentos têm inclinações diferentes, dependente, tal como anteriormente, do ficheiro associado a esse tipo de conteúdo.

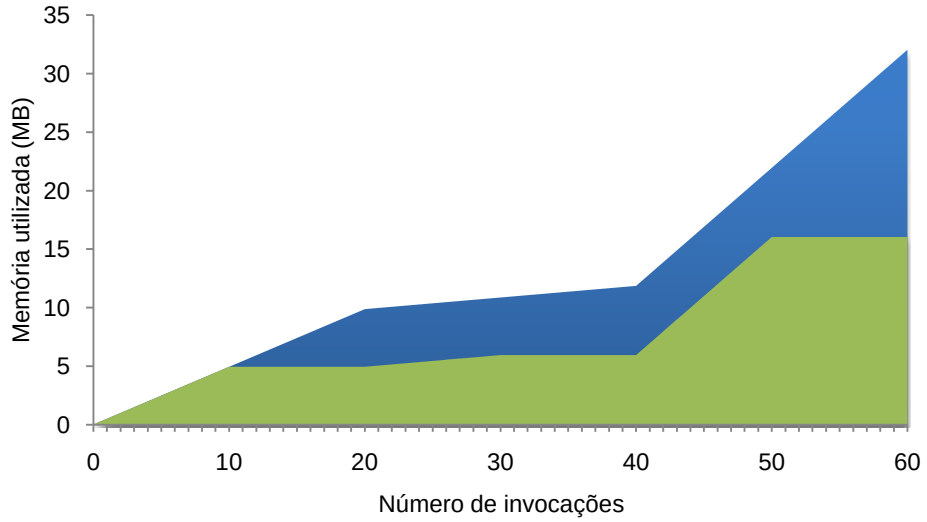

Gráfico 4 - Comparação memória utilizada entre dois casos de teste

O Gráfico 4 representa a comparação da utilização de memória da situação referida anteriormente (situação azul) e um novo caso de teste (situação verde). Este novo caso de teste é semelhante ao anterior apenas difere no facto de serem apenas invocados os primeiros dez elementos de cada um dos tipos de conteúdos. Neste caso a utilização de memória aumenta durante as primeiras dez invocações de cada tipo de conteúdo e mantêm-se constante nas restantes dez.

Com estes dados é possível observar que a integração do OBIHOC na aplicação permite reduzir a utilização de recursos desnecessários no dispositivo, no sentido em que apenas são transferidos e armazenados neste, os dados necessários para a interacção do utilizador com a aplicação. Esta situação poderia ser tratada directamente pela aplicação, no entanto essa solução implicaria um esforço maior por parte do programador. Com a integração do OBIHOC esta funcionalidade é facilmente associada, com resultados satisfatórios.

6.4. Informação excedente trocada

Tendo em conta o ambiente móvel na qual esta aplicação poderia ser utilizada é necessário conferir a informação excedente trocada entre os dispositivos. Por esse motivo foi executado o

cenário de teste 10 vezes e recolhidos dados que indicam a quantidade de dados transferidos e desses quais os que realmente são úteis para a aplicação.

	Dados recebidos (bytes)	Dados úteis (bytes)
Library	**428**	**393**
AudioList	**782**	**642**
Audio	**395**	**273**
DocumentList	**788**	**642**
Document	**398**	**273**
VideoList	**782**	**642**
Video	**395**	**273**
Content	**93394**	**92994**
Properties	**624**	**499**

Tabela 2 - Dados recebidos e úteis

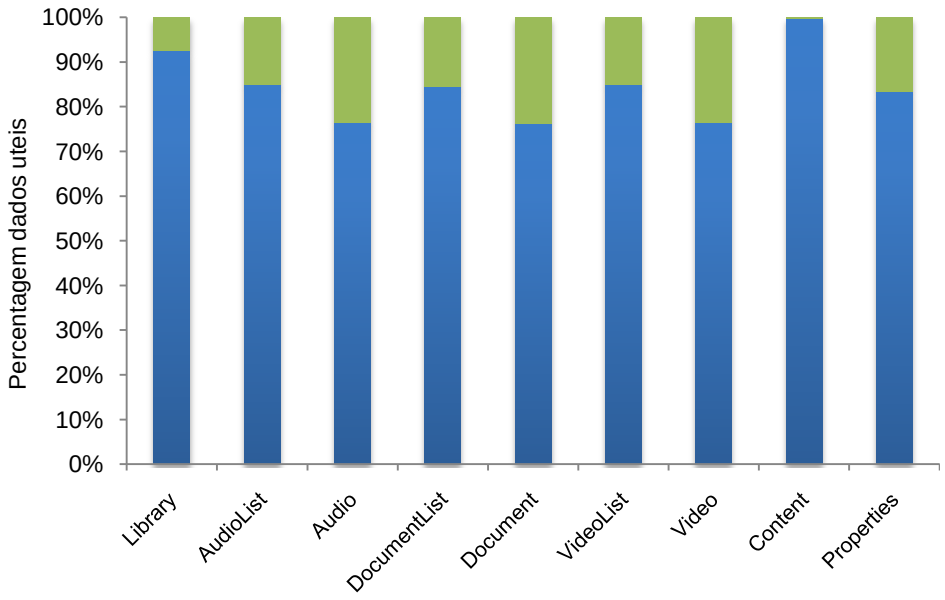

Gráfico 5 - Percentagem de dados uteis trocados

Na Tabela 2 é apresentado o número de bytes de cada tipo de dados trocados entre dispositivos, ou seja, foram analisados a quantidade de bytes recebidos para cada objecto e o número de bytes que acabaram por ser úteis para a aplicação. Cada um destes valores foi obtido pelo valor médio obtido em todas as execuções do cenário de teste. Para uma melhor percepção do valor de dados excedentes é apresentado no Gráfico 5 a percentagem de dados recebidos (a azul) e a percentagem de dados excedentes (a verde). Estes dados excedentes

resultam de uma estrutura de dados criada pelo OBIHOC, na qual o objecto a ser replicado é inserido. Esta estrutura tem diversos parâmetros, que permitem a correcta execução das funcionalidades do OBIHOC, como por exemplo: identificadores do tipo de pedido enviado, identificador do objecto e lista de dispositivos para os quais o objecto já foi replicado, etc.

Como é possível verificar os objectos que estão no mesmo nível do grafo (por exemplo Audio, Document e Video) apresentam sensivelmente as mesmas percentagens de dados úteis. Mais uma vez o objecto Content apresenta um comportamento distinto, pois apresenta uma percentagem muito próximo de 100% de dados úteis. Este valor é normal pois a replicação deste objecto implica a transferência de grande quantidade de dados (dados associados a um ficheiro) e a informação excedente, apesar de ser a mais elevada em relação aos restantes objectos, é disfarçada.

6.5. Impacto na Aplicação

Para além dos recursos a serem utilizados no dispositivo uma componente importante a ser analisada é a interferência deste middleware no funcionamento da aplicação. Esta interferência terá de ser analisada com especial atenção pois poderá influenciar muito a aceitação da aplicação pelos utilizadores. Para tal é importante que a aplicação mantenha um desempenho aceitável em execução local e distribuída. De forma a cumprir com este objectivo a aplicação deverá ser configurada de forma utilizar da melhor forma as funcionalidades oferecidas pela OBIHOC. Para analisar qual a melhor configuração foi realizado o cenário de teste recorrendo a diferentes valores de profundidade, aquando a replicação de objectos. A utilização deste parâmetro permite a replicação de grupos de objectos, agilizando o processo de replicação.

Os dados obtidos resultam da execução do cenário de teste alterando a profundidade. Estes resultados representam o tempo que o utilizador tem de esperar para aceder às Properties e Content de cada conteúdo da biblioteca. Para além disso também é apresentado a utilização da memória do dispositivo ao longo do cenário de teste.

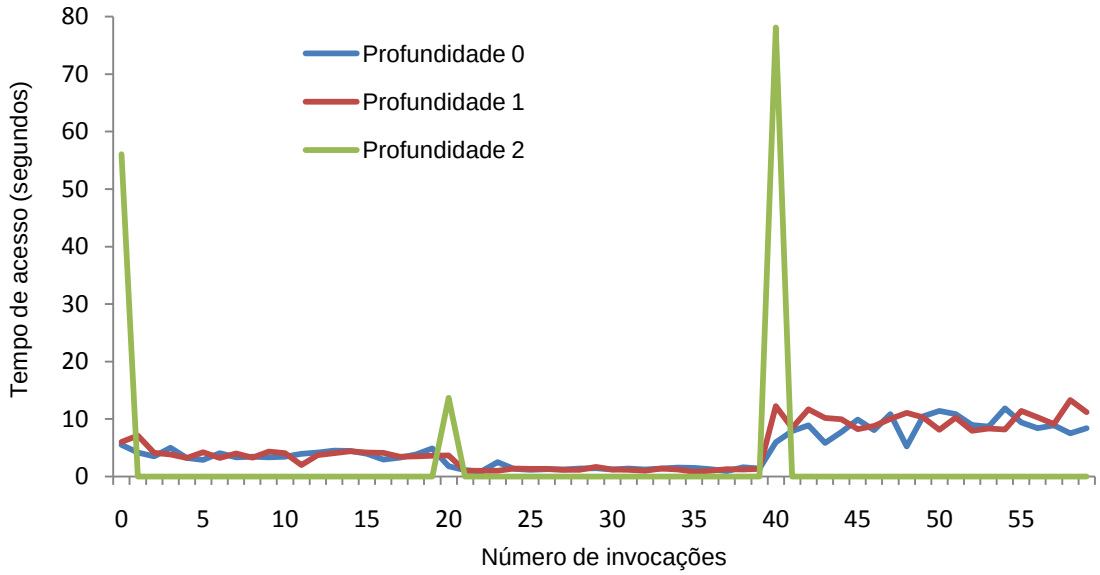

Gráfico 6 - Tempo de acesso com diferentes profundidades

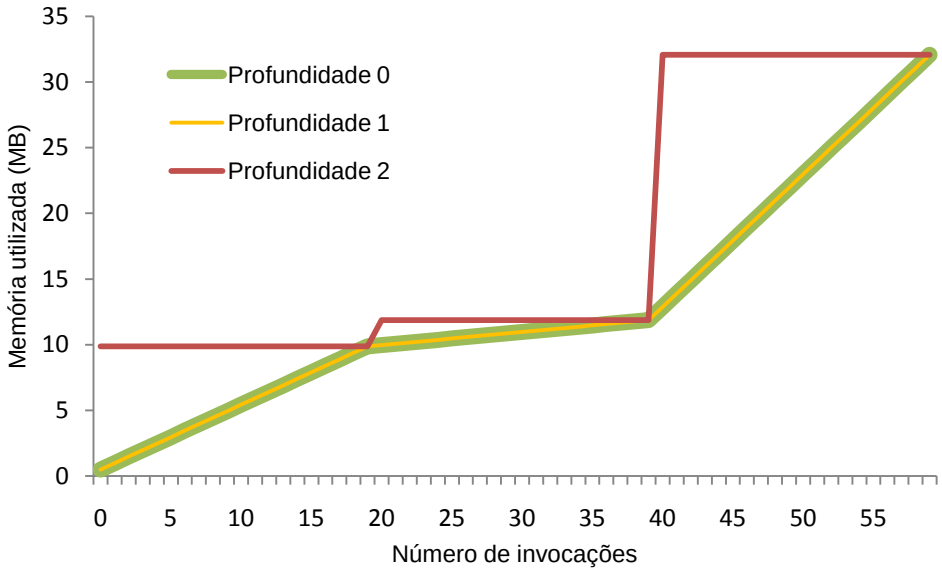

Gráfico 7 - Memória utilizada no dispositivo com diferentes profundidades

No Gráfico 6 é possível observar o tempo de acesso a um determinado elemento da biblioteca. Como é possível observar as diferenças entre a utilização de profundidade 0 e 1 são mínimas. A principal diferença que se observa é quando se inicia o acesso a um novo tipo de conteúdo (invocações 0, 20 e 40). Nestes três casos o tempo de acesso com profundidade 1 é superior pois existe mais informação a ser replicada, nomeadamente todos os objectos dessa lista. Por exemplo, o acesso ao primeiro elemento da AudioList implica que seja replicado o

próprio objecto AudioList bem como todos os objectos Audio. Só após este processo é que se desencadeia o processo de replicação das Properties e Content do Audio em causa. Nos acessos às Properties e Content dos restantes Audio apenas estes dois últimos são replicados pois o objecto Audio em si já fora replicado anteriormente.

Em relação à utilização de profundidade 3 os tempos de acesso, Gráfico 6, apresentam três picos extremamente elevados. Este valores anómalos são aceitáveis pois antes de aceder a esse conteúdo todos os conteúdos desse tipo são replicados para o dispositivo, o que implica a transferência de uma enorme quantidade de dados. No entanto esta transferência de dados, e respectivo tempo de acesso elevado, não está relacionado com a execução do OBIHOC mas sim com as características da aplicação, nomeadamente a transferência de um número elevado de dados. Esta situação tem a vantagem de diminuir bastante o tempo de acesso aos restantes conteúdos desse tipo, pois já foram replicados. Se por outro lado os restantes conteúdos não foram necessários esta situação revela-se pouco eficiente, pois estão a ser utilizados recursos desnecessários.

No Gráfico 7 complementa a situação apresentada anteriormente. A utilização de profundidade 0 e 1 resulta numa ocupação da memória equivalente pois o processo que ocupa mais memória, replicação do Content, é executado apenas quando esse tipo de conteúdo é acedido.

Em relação à utilização da profundidade 3 existem três situações em que a memória no dispositivo utilizada aumenta repentinamente. Estas situações estão relacionadas com a replicação de toda a informação respeitante a esse tipo de conteúdo, tal como referido anteriormente. É de referir também que estes aumentos têm diferentes amplitudes pois, os ficheiros binários associados a cada conteúdo têm um tamanho diferente.

A análise destes dados permite perceber que a utilização de um valor de profundidade adequado pode influenciar bastante o desempenho da aplicação. A escolha deste valor deverá ser tida em conta pelo programador para se adequar à utilização mais frequente da aplicação. Por exemplo se a aplicação for utilizada para maioritariamente aceder a um único tipo de conteúdo, a utilização de uma profundidade de 3 poderá ser benéfica, pois o acesso ao primeiro conteúdo é bastante mais lento, no entanto os restantes serão praticamente imediatos. Se por outro lado um comportamento mais previsível dos utilizadores seja navegarem por toda a biblioteca a utilização de um profundidade 1 ou 0 será a mais indicada.

7. Conclusões

Com os diversos avanços tecnológicos os dispositivos móveis estão cada vez mais preparados para integrar novas funcionalidades. Devido às melhorias significativas de processamento, armazenamento e capacidade de comunicação sem fios os dispositivos móveis estão cada vez mais presentes nas nossas vidas profissionais e pessoais. No entanto ainda existem bastante alternativas a serem exploradas, nomeadamente a criação de aplicações que beneficiem de um ambiente distribuído, como por exemplo uma rede ad-hoc.

No entanto o desenvolvimento de aplicações neste ambiente implica que os programadores dominem áreas para as quais, na maioria das vezes, não estão preparados. Para além disto, a incursão nestas áreas afasta-os da lógica da aplicação, na qual se devem concentrar. Assim este trabalho permitiu desenvolver um middleware que liberta os programadores dessas áreas, implementando-as de forma eficaz e correcta.

Ao longo do trabalho foram analisados diversos aspectos técnicos: 1) flexibilidade de paradigmas, 2) replicação automática, 3) DGC e 4) políticas de segurança. Para além desta análise foi efectuado um trabalho de pesquisa sobre diversos sistemas/trabalhos existentes: por exemplo OBIWAN.

Após a assimilação do trabalho referido anteriormente, foi desenvolvida e implementada uma arquitectura, de forma a satisfazer todos o objectivos e requisitos propostos. É de realçar que esta arquitectura dá maior ênfase à replicação incremental, pois este é um requisito essencial para a implementação dos restantes requisitos.

A avaliação do trabalho foi realizada num ambiente controlado, recorrendo a emuladores. No entanto esta situação não é impeditiva de concluir, com base nos resultados obtidos, que a utilização deste middleware permite potenciar bastante as características das aplicações, nomeadamente:

- Integração numa rede ad-hoc;
- Controlo de recursos a serem utilizados nos dispositivos móveis;
- Utilização de políticas de segurança direccionadas para cada aplicação;
- Manutenção de um estado consistente na presença de réplicas;
- Aumento de funcionalidade a serem disponibilizadas aos utilizadores.

Após a conclusão deste trabalho há a destacar algumas situações que poderão ser implementadas no futuro e que irão potenciar ainda mais este middleware, são elas:

- Desenvolvimento de um plug-in (para Eclipse) para auxiliar o desenvolvimento das aplicações.

- Integração de um mecanismo que permita automatizar a geração e extensão de código.
- Desenvolvimento de uma interface gráfica para a definição de políticas de segurança.
- Possibilidade de integração de um protocolo para consistência de dados.
- Integração deste middleware noutros sistemas operativos.

8. Referências

[1] M S Gast and M Loukides, *802.11 wireless networks: the definitive guide*.: OReilly & Associates, Inc. Sebastopol, CA, USA, 2002.

[2] Brent A Miller and Chatschik Bisdikian, *Bluetooth Revealed*.: Prentice Hall PTR, 2001.

[3] R Kalden, I Meirick, and M Meyer, "Wireless Internet access based on GPRS," *IEEE Personal Communications*, vol. 7, pp. 8--18, 2000.

[4] J Hoebeke, I Moerman, B Dhoedt, and P Demeester, "An overview of mobile ad hoc networks: applications and challenges," *JOURNAL-COMMUNICATIONS NETWORK*, vol. 3, pp. 60--66, 2004.

[5] "Efficient Java RMI for parallel programming," *ACM Trans. Program. Lang. Syst.*, vol. 23, pp. 747--775, 2001.

[6] WE Weihl, "Remote procedure call," *Acm Press Frontier Series*, pp. 65--86, 1990.

[7] H Stockinger et al., "File and object replication in data grids," *Cluster Computing*, vol. 5, pp. 305--314, 2002.

[8] C G Harrison, D M Chess, and A Kershenbaum, "Mobile agents: Are they a good idea," *Mobile Object Systems: Towards the Programmable Internet*, vol. 1222, pp. 25--47, 1997.

[9] L Veiga and P Ferreira, "Object-Swapping for Resource-Constrained Devices," in *International Conference on Distributed Computing Systems Workshops*, 2007, p. 7.

[10] L. Veiga and P. Ferreira, "Incremental replication for mobility support in OBIWAN," , Vienna, Austria, 2002, p. 249.

[11] J Xu, B Li, and D L Lee, "Placement Problems for Transparent Data Replication Proxy Services," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, vol. 20, p. 1383, 2002.

[12] D Plainfossé and M Shapiro, "A survey of distributed garbage collection techniques," in *Memory management: international workshop, IWMM'95*, Kinross, UK, 1995, p. 211.

[13] J D Eckart and R J LeBlanc, "Distributed garbage collection," *SIGPLAN Not.*, vol. 22, pp. 264--273, 1987.

[14] J Hughes, "A distributed garbage collection algorithm," in *Functional Programming Languages and Computer Architecture*, 1985, pp. 256--272.

[15] J M Piquer, "Indirect reference counting: A distributed garbage collection algorithm," in *Proceedings on Parallel architectures and languages Europe: volume I: parallel*

architectures and algorithms: volume I: parallel architectures and algorithms table of contents, Springer-Verlag New York, Inc. New York, NY, USA, 1991, pp. 150--165.

[16] M Shapiro, P Dickman, and D Plainfossé, "Robust, distributed references and acyclic garbage collection," , New York, NY, USA, 1992, pp. 135--146.

[17] H Luo, S Lu, and L Zhang, "Providing robust and ubiquitous security support for mobile ad hoc networks," in *Network Protocols. Ninth International Conference on*, 2001, pp. 251--260.

[18] J P Hubaux, L Buttyán, and S Capkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, 2001, p. 155.

[19] G Karjoth, D B Lange, and M Oshima, "A security model for aglets," *IEEE Internet Computing*, vol. 1, pp. 68--77, 1997.

[20] A Tripathi and N Karnik, "Protected resource access for mobile agent-based distributed computing," in *Proceedings of the 1998 ICPP Workshop on Wireless Networks and Mobile Computing*, 1998, pp. 144--153.

[21] G Edjlali, A Acharya, and V Chaudhary, "History-based access control for mobile code," in *Proceedings of the 5th ACM Conference on Computer and Communications Security*, New York, NY, USA, 1998, pp. 38--48.

[22] N Dulay, E Lupu, M Sloman, and N Damianou, "A policy deployment model for the Ponder language," in *Proc. IM*, vol. 2001, 2001, pp. 14--18.

[23] C Ribeiro, A Zuquete, P Ferreira, and P Guedes, "SPL: An access control language for security policies with complex constraints," in *Proceedings of the Network and Distributed System Security Symposium*, 2001, pp. 89--107.

[24] S Vinoski, "CORBA: Integrating diverse applications within distributed heterogeneous environments," *IEEE Communications Magazine*, vol. 35, pp. 46--55, 1997.

[25] D Hagimont and D Louvegnies, "Javanaise: distributed shared objects for Internet cooperative applications," in *Middleware'98*, 1998.

[26] B Liskov, M Day, and L Shrira, "Distributed object management in Thor," *Distributed Object Management*, pp. 79--91, 1993.

[27] A Silva, M Mira da Silva, and J Delgado, "An overview of AgentSpace: a next-generation mobile agent system," *Lecture Notes in Computer Science*, pp. 148--159, 1998.

[28] P Ferreira, L Veiga, and C Ribeiro, "OBIWAN: design and implementation of a middleware platform," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14,

pp. 1086--1099, 2003.

[29] M Papadopouli and H Schulzrinne, "Seven degrees of separation in mobile ad hoc networks," *GLOBECOM-NEW YORK-*, vol. 3, pp. 1707--1711, 2000.

[30] W van Heiningen, S MacDonald, T Brecht, and M S Witter, "Babylon: middleware for distributed, parallel, and mobile Java applications," *Concurrency and Computation: Practice & Experience*, vol. 20, pp. 1195--1224, 2008.

[31] Sridharan P., Rieken B., and Peterson L., *Advanced JAVA Networking*.: Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1997.

[32] W Grosso, *Java RMI*.: O'Reilly Japan, 2002.

[33] J Gosling, B Joy, G Steele, and G Bracha, *Java (TM) Language Specification, The (Java (Addison-Wesley))*.: Addison-Wesley Professional, 2005.

[34] T Lindholm and F Yellin, *Java virtual machine specification*.: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[35] H Holma and A Toskala, *HSDPA/HSUPA for UMTS: High speed radio access for mobile communications*.: John Wiley Chichester, Eng., 2006.