Framework de distribuição assíncrona de aplicações móveis situadas

Miguel Borges José Creissac Campos António Nestor Ribeiro
Departamento de Informática/CCTC, Universidade do Minho
Campus de Gualtar, 4710-057 Braga
borges.miguel@gmail.com, jose.campos@di.uminho.pt, anr@di.uminho.pt

Resumo

Ao contrário das aplicações criadas para os computadores de secretária, as aplicações móveis podem tirar partido do facto de acompanharem o dispositivo em que estão a correr para diferentes locais. Estas localizações geográficas são um factor que permite que estas possam reagir e ajustar-se a realidades distintas. O trabalho apresentado neste artigo pretende ser um passo no sentido de aproximar as aplicações aos sítios físicos onde estas correm. É também uma aproximação ao problema da variabilidade dos dispositivos. Consiste numa framework para distribuição de aplicações que irão correr num gestor de aplicações pré-instalado em dispositivos móveis. Este gestor é responsável por actuar sobre a aplicação sempre que determinadas condições reais sejam atingidas. É também responsável por gerir o descarregamento das aplicações a partir das infraestruturas que as disponibilizam. Numa perspectiva qualitativa, e com base numa aplicação de teste desenvolvida, a framework revela-se adequada à distribuição de aplicações em ambientes móveis.

Palavras-Chave

Mobile Code, Smart spaces, Push

1 Introdução

A tecnologia que é disponibilizada no dispositivos móveis é cada vez mais sofisticada e aproximada à que está presente nos computadores de secretária. É possível ”navegar”em telemóveis e com isso, usar todas as possibilidades permitidas pela web. No entanto, e ao contrário do que se passa nos *desktop*, os dispositivos móveis são muitos e variados quer seja nas capacidades computacionais, nos recursos de rede, nas características dos dispositivos de entrada e saída. Também ao contrário dos *desktop*, a utilização de um dispositivo móvel pode ocorrer nos mais diversos contextos.

Ao contrário das aplicações criadas para os computadores de secretária, as aplicações móveis podem tirar partido do facto de acompanharem o dispositivo em que estão a correr para diferentes sítios. Este factor possibilita que estas possam reagir e ajustar-se a realidades distintas.

O trabalho apresentado neste artigo pretende ser um passo no sentido de aproximar as aplicações aos sítios físicos onde estas correm. É também uma aproximação ao problema da variabilidade dos dispositivos. Consiste numa framework para distribuição de aplicações que irão correr num gestor de aplicações pré-instalado em dispositivos móveis. Este gestor (abaixo referido como *container*) é responsável por actuar sobre a aplicação sempre que determinadas condições reais sejam atingidas. É também responsável por gerir o descarregamento das aplicações a partir das infraestruturas que as disponibilizam.

O artigo está estruturado do seguinte modo: na secção 2 discutimos modelos de disponibilização de aplicações em dispositivos móveis; na secção 3 discutimos técnicas que suportam a disseminação de informação pelos dispositivos; na secção 4 apresentamos a arquitectura de uma plataforma de disseminação de aplicações e na secção 5 algumas notas sobre a implementação da plataforma; na secção 6 apresenta-se um exemplo que se encontra em desenvolvimento; o artigo termina com apontadores para trabalho futuro.

2 Enquadramento

A disponibilização de aplicações em dispositivos móveis assenta fundamentalmente em dois modelos aplicacionais. Ambos devem muito a modelos semelhantes e desenvolvidos para os Computadores Pessoais (PC): o modelo *fat client* e o modelo *thin client*. Trata-se, respectivamente, de um modelo em que um objecto compilado concentra lógica aplicacional em si (no dispositivo) e um modelo *online* cuja lógica aplicacional está no servidor e que por via de uma ligação de dados disponibiliza uma interface e informação ao utilizador através de um *browser*.

A propriedade dos dispositivos móveis que melhor caracteriza o estado actual do seu desenvolvimento tecnológico é sem dúvida a heterogeneidade. Dela decorrem uma série

de problemas. Variabilidade nas capacidades computacional e de memória de trabalho, diversidade dos métodos de entrada e saída de dados, necessidade de suportar diferentes arquitecturas e sistemas operativos.

Também no que diz respeito aos dois modelos de disponibilização de aplicações apresentados acima a heterogeneidade é um problema[3]. No caso do modelo *fat* a variabilidade de plataformas e de recursos torna difícil a existência de um ponto comum entre arquitecturas. No caso do modelo puramente *online* (*thin client*), a questão reside no facto de a experiência de utilização ser intrinsecamente pobre por não tirar partido de todas as potencialidades da plataforma. O modelo *thin* também é limitado pela heterogeneidade dos interfaces de rede específicas das plataformas móveis e da variabilidade da sua qualidade (ligações 3G, WiFi, BlueTooth).

Quando se considera um *thin client*, a componente principal da lógica aplicacional está concentrada no servidor e a informação trocada com o servidor é geralmente de elementos visuais que dão suporte à entrada/saída de dados com a aplicação remota - interface gráfica. No caso do *fat client*, a lógica aplicacional acompanha a aplicação. Em ambos os casos o interface com o utilizador pode variar de forma profunda. Um *fat client* é compilado nativamente para a plataforma em que pretende ser executado e pode assim fazer uso de componentes gráficos e de API's específicas de uma forma mais integrada que um *thin client*. Com os *thin clients* apenas se garante a representação gráfica oferecida pela linguagem de marcação utilizada e os componentes gráficos disponibilizados pelo *browser*. Estes são quase sempre um sub-conjunto dos componentes gráficos nativos.

Para tornar mais claras as vantagens e desvantagens de ambos os modelos, veja-se a tabela 1. Nela expõem-se ambos os modelos em relação a parâmetros de necessidade de recursos, de conectividade, flexibilidade na portabilidade e riqueza de integração com o ambiente nativo.

Tabela 1. Comparação dos modelos *thin* e *fat*.

Característica	*thin*	*fat*
Necessidade de recursos no disp.	Baixa	Elevada
Necessidade de Conectividade	Elevada	Baixa
Portabilidade	Elevada	Baixa
Integração	Baixa	Elevada

3 Modelo de disseminação de dados

É na perspectiva da heterogeneidade dos dispositivos que se explorará de forma breve as vantagens e desvantagens dos dois modelos identificados anteriormente e do impacto que ambas as abordagens têm na implementação e distribuição de aplicações para as plataformas móveis.

Os métodos de distribuição (entrega) de dados e a concentração da lógica aplicacional são diferentes em ambos os modelos. A cada modelo correspondem diferentes formas de distribuição de informação (entendendo-se informação de uma forma lata e que pode significar dados ou código executável).

O *thin client* depende quase exclusivamente da existência de conectividade permanente. Actualmente as ligações de dados nas plataformas móveis é oferecida por tecnologias como o GPRS, UMTS, *bluetooth* ou o *WiFi*. No entanto, a cobertura física de qualquer uma destas tecnologias rádio não garante uma conectividade permanente do dispositivo móvel pelo que fragiliza o modo de distribuição de aplicações assentes exclusivamente em *browser*. Esta desvantagem torna o modelo *thin client* frágil em aplicações puramente *online*. No sentido de atenuar as condições reais de cobertura de ligações de dados, tem sido feita alguma investigação com, por exemplo, técnicas de *caching* móvel [4] aplicadas em particular à navegação com *browser*, entre outras abordagens (ver [2, 5]) que tentam mitigar as limitações das ligações de dados móveis.

Para além das ligações de dados *connection oriented* (o caso do TCP, orientado à conexão), as plataformas móveis dispõem também de ligações *connectionless* (não orientado à conecção) como é o caso dos SMS (para além do mais evidentes UDP e IP), que para além de meio de comunicação entre utilizadores, são também usados como mecanismo de negociação inicial de um *WAP push* [9].

Um *WAP push* é uma mensagem especial com um URL embebido que é enviado para os dispositivos. O utilizador poderá posteriormente optar por aceder ao conteúdo disponibilizado nesse URL. Assim, o estabelecimento de uma comunicação *push* não implica a existência explícita de um pedido da aplicação cliente, podendo assim ser considerada uma forma de comunicação assíncrona - ao invés do que acontece no modelo cliente servidor em que a entidade servidora responde apenas a pedido da aplicação cliente. Em particular, o modelo de *WAP push*, estabelece a existência de um servidor *Push - Push Proxy Gateway*. Este servidor é responsável pelo aprovisionamento de conteúdos e sua distribuição. Esta é feita de forma assíncrona. O aprovisionamento da informação a disseminar, é feito por aplicações/ fornecedores externos que usam a infra-estrutura de comunicações do fornecedor do serviço móvel. O modelos *WAP push* define também o protocolo e a linguagem necessárias à execução dessas operações.

A existência de dispositivos móveis em massa com capacidades computacionais suficientes e a quase ubiquidade das tecnologias de comunicação rádio abre espaço à possibilidade de combinar o melhor dos dois modelos aplicacionais. Tendo por objectivo manter a versatilidade do modelo *thin client* e ao mesmo tempo garantir a usabilidade, robustez e versatilidade resultante do cliente nativo, com base num mecanismo de publicação activa (*push*) de aplicações, pode designar-se um paradigma exclusivo da computação móvel - o da aplicação situada. Neste contexto, entende-se a aplicação como situada por ser sensível à sua localização geográfica, por ter a noção de utilizador

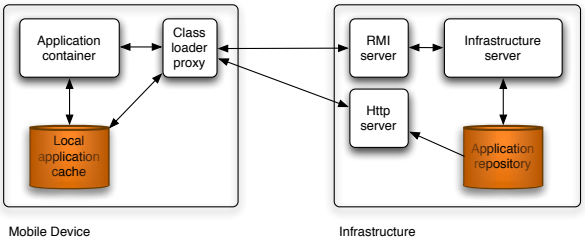

Figura 1. Arquitectura do sistema

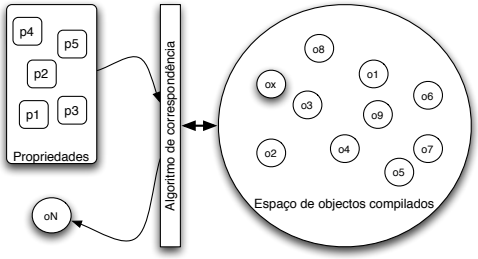

Figura 2. Disponibilização de objectos

único e por ser sensível a eventos de ordem cronológica[1].

4 Uma proposta de arquitectura

Nesta secção, enquadra-se o trabalho desenvolvido tendo em conta as contribuições de diferentes áreas disciplinares relacionadas com a interacção entre entidades cliente e entidades servidoras. Pretendemos criar uma plataforma de disseminação de aplicação para dispositivos móveis de forma assíncrona.

De uma forma global, o sistema divide-se em dois componentes fundamentais - o *container* e a infraestrutura. O *container* é um processo que irá correr no dispositivo móvel e cuja responsabilidade engloba o controlo de recepção de aplicações a partir da infraestrutura e de uma série de operações sobre as aplicações que se detalharão mais adiante. A infraestrutura corre em um ou mais computadores ligados em rede e tem a responsabilidade de gerir o processo de aprovisionamento de aplicações a distribuir. Para além disso, deve também servir de conector de comunicação para o *container* e para a aplicação que o *container* esteja a correr no momento. A infraestrutura é ainda responsável pela gestão da distribuição das aplicações por diferentes *container* e pela comunicação de regras de invalidação ao *container* para que este opere sobre a aplicação que está a correr de acordo com as mesmas regras.

As aplicações a distribuir são em linguagem Java compilado na forma de um ficheiro *jar*, estrutura de directórios (*jar* expandido) ou de um ficheiro *class*. A distribuição do código compilado é tarefa de um servidor HTTP em modo de serviço de ficheiros e gerido pela infraestrutura. A *framework* desenvolvida impõe que as aplicações produzidas para serem distribuídas implementem uma interface que vai permitir ao *container* operar sobre elas. Esta arquitectura está esquematizada na figura 1.

A infraestrutura como a entidade responsável pela distribuição de aplicações tem uma abrangência física limitada pelo espaço onde esta se insere: um edifício, um campus, uma loja ou outro. Supondo então a infraestrutura associada a um espaço físico limitado, admite-se que um dispositivo fica ao alcance da infraestrutura por um qualquer método de detecção. Ao detectar a presença de um dispositivo novo nas suas imediações, a infraestrutura publicita a sua presença. De seguida, o dispositivo móvel, por meio de mecanismos pertencentes ao *container* mostra à infraestrutura o seu vector de interesses. Este vector é um objecto que agrega um conjunto de propriedades do mesmo domínio das propriedades associadas às aplicações no repositório. Este vector de interesse é criado pelo utilizador e espelha interesses por determinados tipos de aplicações e expõe características físicas do dispositivo.

Podemr meio de um algoritmo de correspondência, a infraestrutura decide que aplicação se ajusta ao vector de interesses publicado pelo *container* do dispositivo (ver a figura 2). Em caso de ajuste, a infraestrutura comunica ao *container* o resultado e a aplicação ou aplicações que este deverá descarregar. O *container* descarrega de seguida a aplicação ou aplicações correspondentes. No caso de se tratarem de múltiplas correspondências, o *container* descarrega-as a todas e será da responsabilidade do utilizador fazer a gestão de cada uma delas através da interface gráfica disponibilizada. Na figura 3 pode ver-se a representação em diagrama de iteracção das operações acima descritas.

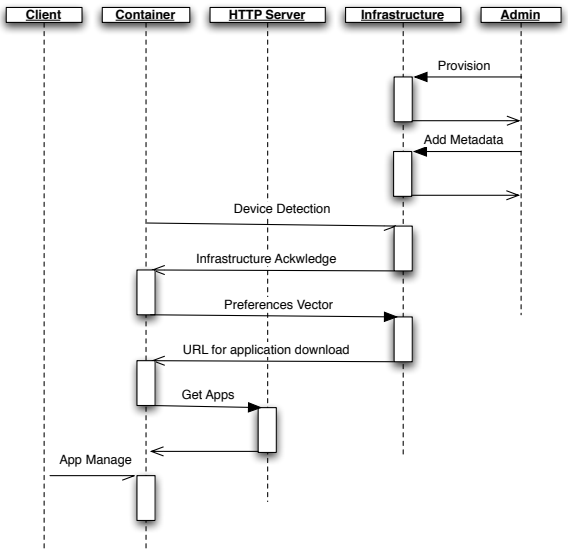

Figura 3. Diagrama de interacção do sistema

A operação de carregamento da aplicação remota processa-se do seguinte modo: Se o resultado de correspondência for positivo, a infraestrutura devolve ao *container* um URL no qual estará disponível a aplicação a descarregar. O *container* faz de seguida um pedido de GET ao URL ou múltiplos URL indicados e carrega a(s) classe(s) com re-

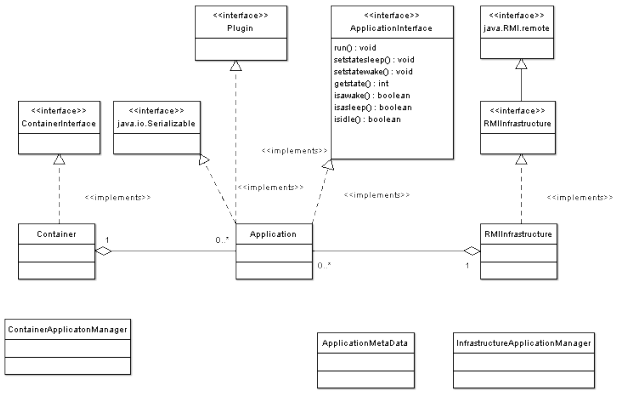

Figura 4. Diagrama de classes do sistema

curso a um *classloader*. Após o carregamento é criada uma instância da(s) classe(s) no *container* e nesse(s) objecto(s) recém-criado(s) é invocada o método *run*, definido na interface das aplicações distribuíveis. A interface das aplicações exige que a aplicação implemente os métodos ilustrados na *interface* apresentado na figura 4.

4.1 Invalidação de aplicações

Para determinar se a aplicação pode efectivamente ser entregue ao *container* no dispositivo, a infraestrutura faz correr um algoritmo de correspondência. Primeiro determina a compatibilidade entre as propriedades físicas e de software publicadas pelo dispositivo e as associadas aos objectos compilados. Nesta comparação, são os meta-dados associados aos objectos compilados que determinam quais as propriedades obrigatórias e quantas facultativas deverá o dispositivo respeitar para poder fazer correr a aplicação. O sucesso desta operação depende da correspondência entre *todas* as propriedades obrigatórias e pelo menos o número mínimo estipulado nos meta-dados da aplicação de propriedades facultativas.

A operação seguinte consiste na correspondência entre as propriedades de interesse publicadas pelo dispositivo e as propriedades descritivas associadas à aplicação. Os esquemas XML das estruturas de dados para cada um dos tipos de propriedades são apresentados nas figuras 5 e 6 (note-se que a estrutura de dados da figura 6 é parametrizada em parte pelo utilizador). Neste caso, é o dispositivo que determina quais das propriedades são obrigatórias e quantas das propriedades descritivas facultativas têm de ser respeitadas para que esta corresponda a uma aplicação desejada. A parte das propriedades relativas a interesses pessoais e de software são especificadas pelo utilizador.

O sucesso desta operação de correspondência depende da satisfação de todas as propriedades marcadas pelo dispositivo como obrigatórias e do número mínimo de propriedades facultativas determinado pelo dispositivo. Caso ambas as operações tenham sucesso, a aplicação é disponibilizada para descarga.

As regras e acções de invalidação são pedidas à *infraestrutura* após esta publicar o URL de onde poderá ser descarregado o objecto compilado. As regras definidas correspondem à ocorrência de determinados eventos geográficos ou temporais:

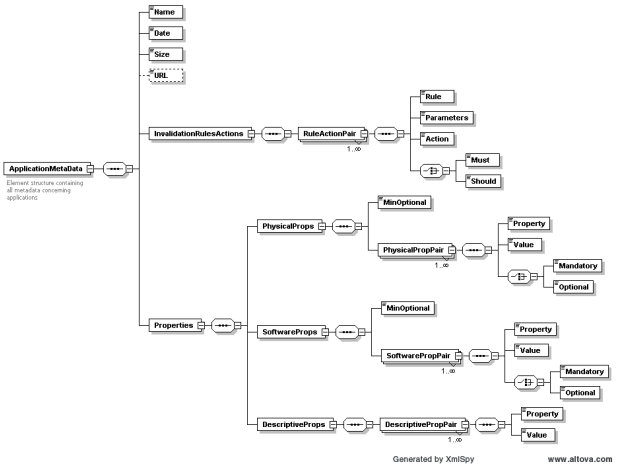

Figura 5. Estrutura de dados de um objecto compilado

- **on_inside_area** — Regra parametrizada com quatro coordenadas geográficas que definem a diagonal de um qualquer rectângulo. A acção associada a esta regra é despoletada sempre que o sub-sistema de informação geográfica dê a indicação de que o dispositivo se encontra no rectângulo geográfico definido pelos parâmetros.

- **on_outside_area** — Regra parametrizada com quatro coordenadas geográficas que definem a diagonal de um qualquer rectângulo. A accão associada a esta regra é despoletada sempre que o sub-sistema de informação geográfica dê a indicação de que o dispositivo se encontra fora do rectângulo geográfico definido pelos parâmetros.

- **on_served_by_cell** — Regra parametrizada com um identificador genérico e cuja semântica depende do sub-sistema de localização usado (pode estar a referir-se a uma espia *bluetooth* um ponto de acesso *Wifi* ou outros) e cuja acção associada é executada caso o subsistema de localização indique que o dispositivo está a ser servido pela célula dada em parâmetro.

- **on_not_served_by_cell** — Regra parametrizada com um identificador genérico. A acção associada é executada quando o sub-sistema de localização dá indicação de não estar a ser servido explicitamente pela célula (identificador) indicada em parâmetro.

- **on_time_out** — Regra parametrizada com tempo de vida da aplicação em minutos. Este evento é disparado caso seja excedido o tempo determinado no parâmetro.

Podem associar-se as seguintes acções às regras de invalidação:

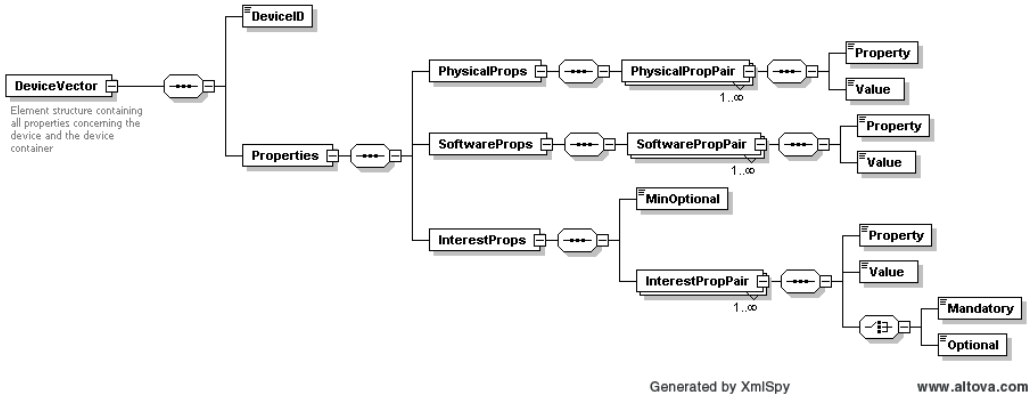

Figura 6. Estrutura de dados de um container

- **run** — Invoca na aplicação o método de *run*. Faz a aplicação iniciar a sua execução.
- **sleep** — Resulta no adormecimento da aplicação que está a correr. Esta acção invoca o método de adormecimento da aplicação que está a correr.
- **wake** — Resulta no acordar da aplicação que estaria préviamente adormecida.
- **kill** — Determina o termino imediato da aplicação. Esta acção pode ocorrer quer a aplicação esteja a correr quer esteja adormecida. Resulta na perda total de toda a informação relativa à aplicação que estava a correr.
- **alert** — Esta acção não sinaliza directamente a aplicação que está a correr mas sim o *container* para que este comunique o evento à *infraestrutura*
- **bypass** — Nega ao motor de invalidação a possibilidade de tomar uma acção que esteja definida para o mesmo domínio passado como parâmetro nesta acção.

Todas estas acções apenas invocam na aplicação que está a correr os métodos correspondentes e que foram implementados de acordo com a interface das aplicações. No entanto, isso não impede que o comportamento que foi implementado seja diferente daquele que a semântica da acção determina. Ou seja, assume-se que a invocação do método sleep (imposto pela interface da API) corresponda efectivamente a um adormecimento da aplicação mas é da inteira responsabilidade do programador assegurar a correcta implementação do mecanismo e das implicações da sua invocação para a aplicação em particular: persistência de variáveis, persistência de estado(s), etc.

5 Implementação tecnológica

Do ponto de vista da implementação, a migração de código em Java, atendendo às limitações dos dispositivos em que a aplicação irá correr, oferece duas possibilidades: Uma por invocação remota de métodos e outra por carregamento dinâmico de classes remotas.

Optou-se pelo carregamento dinâmico de classes remotas pois o primeiro método limitava o sistema à migração de apenas uma classe o que inviabilizaria a distribuição de aplicações complexas com múltiplas classes (JAR).

Esta abordagem tem por base um serviço de RMI apenas para passar entre *container* e infraestrutura dados de identificação, autenticação e outros. O mecanismo que permite ao *container* o acesso às classes a distribuir é garantida por um web server no qual estão alojadas as aplicações. Assim, ao invés de ser serializada a classe a distribuir (imposição no caso de se passar aplicações em bytecode por RMI), usa-se um modo de transmissão externo à plataforma Java - HTTP. Este método é também simples pois usa APIs comuns nos dispositivos CDC[1]. Tem ainda a vantagem de permitir o carregamento de múltiplas classes agregadas em JARs.

5.1 Limitações tecnológicas

Uma das limitações encontradas na ferramenta de simulação da Sun é o facto de não permitir a utilização efectiva de APIs que de acordo com a especificação CDC sejam de carácter opcional. Esta limitação abrange o *package* java.rmi.registry. De acordo com a especificação CDC, este *package* só está disponível em implementações com o *foundation profile* (JSR 219 v1.1.2)[6]. O RMI é englobado no que se consideram ser *packages* opcionais (JSR 66 v1.0)[7].

O recurso a *webservices* por parte do *container* também está limitado no simulador pois é também listado como sendo um *package* opcional (JSR 172 v1.0)[8].

Para ultrapassar estas limitações, o trabalho foi desenvolvido usando o *package* RMI-OP que acompanha a máquina virtual CreME da empresa NSICOM. Esta máquina virtual Java está disponível para as plataformas Microsoft Windows Mobile.

6 Aplicação experimental

Um visitante entra num departamento universitário. Um espaço novo para si. Pretende chegar ao gabinete de uma

[1] CDC: *Connected Device Configuration* — Uma plataforma para desenvolvimento de aplicações Java para dispositivos embebidos.

determinada pessoa. Ao entrar no departamento, num dos seus dispositivos móveis surge a proposta para descarregar uma aplicação. O utilizador sabe que esta é do seu interesse pois o *container* pré-instalado no seu dispositivo age como um *broker* do utilizador e "sabe"de que tipo de aplicações este gosta. Trata-se de uma aplicação que de forma interactiva o permite chegar ao seu destino.

A aplicação disponibiliza num interface que se ajusta ao seu dispositivo as fotos dos docentes do departamento. O utilizador escolhe uma delas e a aplicação vai dando ao utilizador indicações de como avançar para chegar ao gabinete pretendido. As indicações são apresentadas à vez e quando o utilizador cumpre uma delas, dá indicação ao software para que este dê a próxima indicação. Ao utilizador é permitido navegar para trás e para diante na informação mostrada se assim o desejar.

Ao passar junto a um laboratório específico, é-lhe oferecida uma aplicação piloto para testar um novo interface com o utilizador. O nosso visitante rejeita-a e prossegue. Entretanto, chega ao destino e a aplicação passa para modo adormecido. O utilizador abandona depois o edifício e a aplicação apaga-se assim que este abandona o perímetro estabelecido para o seu tempo de vida sem questionar o utilizador. Estes comportamentos resultam directamente da expiração do tempo de vida da aplicação por parte do motor de invalidação. Como as premissas de localização foram ultrapassadas, o *container* reage adormecendo e apagando a aplicação, respectivamente.

Com a finalidade de avaliar o protótipo implementado está a ser criada uma aplicação piloto. Trata-se de um sistema de auxilio à navegação em espaços públicos. Pretende-se que esta se aproxime do cenário equacionado acima e sirva de guia ao utilizador indicando o gabinete de uma determinada pessoa.

A concepção do interface com o utilizador desta aplicação teve como ponto de partida as seguintes restrições, decorrentes da natureza intrínseca de dispositivos móveis:

- Propriedades do interface de visualização desconhecidas
- Modo de interacção variável

A figura 7 apresenta um protótipo da aplicação em desenvolvimento.

7 Conclusões e trabalho futuro

De uma perspectiva qualitativa, o trabalho desenvolvido revela-se adequado à distribuição de aplicações em ambientes móveis. Não é neste momento possível aferir qual o grau de adequação à distribuição de aplicações em larga escala pois os testes efectuados foram o sentido da prova do conceito. Como trabalho futuro, dever-se-à explorar a distribuição em massa de aplicações situadas. Deve também desenvolver-se uma (ou mais aplicações) que tirem mais partido dos mecanismos de invalidação que a API disponibiliza, por forma a validar até que ponto eles cobrem todas as necessidades de sistemas deste tipo.

Figura 7. Protótipo da interface

Referências

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99*, pages 304–307, London, UK, 1999. Springer-Verlag.

[2] A. Demers, K. Petersen, M. Spreitzer, D. Ferry, M. Theimer, and B. Welch. The bayou architecture: support for data sharing among mobile users. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 2–7, 1994.

[3] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, April 1994.

[4] S. Hadjiefthymiades and L. Merakos. The operation of the www in wireless environments. Technical report, University of Athens, 1999.

[5] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *SIGOPS Oper. Syst. Rev.*, 25(5):213–225, October 1991.

[6] Sun Microsystems. Foundation profile, version 1.1.2. `http://java.sun.com/javame/reference/apis/jsr219/`, 2006 (acedido em 6 Fevereiro, 2008).

[7] Sun Microsystems. J2me rmi optional package specification v1.0. `http://java.sun.com/javame/reference/apis/jsr066/`, 2006 (acedido em 6 Fevereiro, 2008).

[8] Sun Microsystems. Jsr 172 j2me web services. `http://java.sun.com/javame/reference/apis/jsr172/`, 2007 (acedido em 6 Fevereiro, 2008).

[9] Wireless Aplication Protocol Forum, Ltd. Wap push architectural overview, July 2001.