Mobile Intermediaries Supporting Information Sharing between Mobile Users

Norliza Zaini and Luc Moreau
{ nmz00r, L.Moreau }@ecs.soton.ac.uk

Department of Electronics and Computer Science, University of Southampton
Southampton SO17 1BJ UK

Abstract. Mobile device's networking capabilities offer opportunities for a new range of applications. We consider here a service that allows mobile users taking part in virtual meeting rooms to share information and documents. The sharing is promoted by a recommender system that assists users browsing documents, by making recommendations in the form of URLs pointing to other documents, that users in the virtual room have explicitly decided to share. A multi-user recommender system is a complex application requiring communication, memory and computing resources, and does not lend itself to a port to mobile devices with limited resources and intermittent connectivity. For this reason, we decided to offload the computationally intensive part of the application to the infrastructure, and to introduce the idea of an intermediary located in the network infrastructure, which interacts with applications on behalf of the mobile device, thereby hiding away the intermittent connectivity details. Our vision is that of a mobile intermediary, called Shadow, that will always be in close vicinity with the mobile device. We show that multiple Shadows may co-exist, and we propose a protocol capable of coordinating them. We present an abstraction layer, hiding away communication and coordination details, which offers a substrate for building the distributed recommender system across mobile devices and fixed infrastructure. Implementation details of our application are also presented.

1 Introduction

The context of this paper is the "ubiquitous computing environment" [12], where embedded devices and artifacts abound in buildings and homes, and have the ability to sense and interact with devices carried by people in their vicinity. Mobile devices' networking capabilities offer opportunities for a new range of services, such as access to stock updates or latest news, or exchange of information with other mobile users discovered dynamically.

However, as devices communicate over wireless networks, they are prone to temporary disconnections and the low bandwidth communication channel they use may lead to high network latency. Moreover, although having the advantage of being small and easy to use, handheld devices such as PDAs suffer from low resource capability such as low memory capacity, limited processing power and small display area. Consequently, these limited capabilities would prevent the

large-scale deployment of advanced services to mobile users, as such services tend to be communication and computation intensive.

Instead of requiring complex applications to be installed on mobile devices and constant connectivity between mobile devices and fixed network to serve users' requests, we believe that applications can be offloaded to the fixed infrastructure, and act semi-autonomously on behalf of the user. In this context, if infrastructure applications can perform tasks without direct control and monitoring from the user, then the proposed approach does not rely on permanent connectivity with mobile devices, it can save mobile device's resources, and it can take advantage of the available resources on the wired network.

While offloading applications to the fixed infrastructure solves the problem of limited devices' resources, it does not address the issue of how communications can take place between the mobile device and the infrastructure application. To this end, we introduce an *intermediary process* in the fixed infrastructure, whose responsibility is to spawn applications in reaction to user's requests and to store and forward messages between mobile devices and applications.

Since a stationary intermediary may lead to long distance communications, we decided to adopt a more flexible approach, in which the intermediary is mobile. Our vision is to have a *mobile* intermediary, which is a mobile agent [5], acting as a *Shadow* of the mobile user, migrating to the user's vicinity when prevailing conditions permit it. The flexibility offered by the mobility can help reduce the bandwidth required for the application and improve its performance [3]. Besides, this has a number of other advantages: *(i)* Shadow and mobile device can communicate using specialised protocols, possibly dynamically chosen according to the current location or to a negotiation between parties; *(ii)* newly created applications would run in the user's vicinity, making use of the local infrastructure; *(iii)* even if the local network is not connected to the Internet, local services could be accessed; *(iv)* Shadows and applications can communicate reliably using transparent routing of messages to mobile agents [8, 9].

When a user moves to a new location, their mobile device interacting with the infrastructure will request the user's Shadow to migrate to a new location. However, this may fail when the user's local network is not connected with the user's previous location. In order to support services in the current vicinity, we opted for a solution where new Shadows can be created dynamically. As result, a user may be associated with multiple Shadows that need to be coordinated and we shall describe in this paper a coordination protocol for such Shadows.

The purpose of this paper is to introduce an application for mobile users, which supports information sharing in virtual meeting rooms. The design and implementation of this application involves complex interactions between infrastructure and mobile device. Our specific contributions are:

1. An architecture supporting multiple mobile Shadows;
2. A coordination protocol between mobile devices and Shadows;
3. An abstraction layer, encapsulating migration and coordination, offering a substrate to program applications between mobile devices and infrastructure;
4. An application supporting information sharing between mobile users.

In the next section, we overview our application scenario. Then, in Section 3, we introduce the architecture followed by Section 4, where we present the algorithms to be implemented by all its components. In Section 5, we further comment on the application's implementation and finally, in the last section we present the summary and discuss related work.

2 Application Scenario

Today's Internet offers various forms of virtual meeting rooms allowing online users to meet and interact with each other; examples include Internet Relay Chat (IRC) supporting text channels and the Access Grid (www.accessgrid.org) supporting audio and video multicast conferencing.

When business meetings are conducted with such facilities, it is useful for users to access an out-of-line mechanism for sharing information or documents. Our approach is to rely on an agent-based recommender system [11] to recommend relevant documents to users as they browse information. The recommended documents will be selected from a pool of documents that users participating to the virtual room have actively decided to share; documents are recommended according to their similarity to other documents. Practically, a user navigates information, while recommendations are displayed in a browser sidebar; recommendations also include information regarding the user who decided to make the document available to the meeting room. The sidebar also allows users to export documents for possible recommendation in the meeting room.

In the context of this paper, our goal is to allow *mobile* users to take part into virtual meetings. As mobile users roam to other networks, they remain in the virtual meeting rooms that had subscribed to, and they are given the opportunity to discover and use different virtual meeting rooms running on different networks. Here, we do not address the problem of delivering content of the meeting room (such as IRC, audio or video channels), instead, we investigate the infrastructure to allow the sharing of information through the recommender system.

In short, our application's functions include: *(i)* to support information sharing between mobile users, *(ii)* to provide recommendations based on the shared information to mobile users, *(iii)* to allow a mobile user to remain a participant in a meeting room while the user roams to different networks.

We have already developed an agent-based recommender system [11] capable of recommending documents that users have actively decided to share. Therefore, this leaves us with the challenge of developing an application that is able to support mobile users accessing virtual meeting rooms hosted by the fixed infrastucture. In the following sections, we present an architecture supporting mobile intermediaries — implemented as mobile agents — which hide communication details between mobile devices and applications on the fixed infrastructure.

3 Architecture Overview

Our proposed architecture is composed of three major components, namely a mobile device, a Shadow and a Shadow Manager, which we describe with the

assumptions we make concerning their communication capabilities. We are also investigating the security aspect of the architecture, but we do not present it at this stage. A mobile device has the ability to connect to a network in its vicinity. It may use specific methods to communicate with network hosts, e.g. infra-red or Bluetooth. We assume that the device is allocated an address, which may change as the device connects to another network, and which can be used by networked entities to communicate with it. A Shadow Manager acts as a local daemon in a local network, first contact point of a mobile device with the local network. A Shadow Manager is responsible for starting or migrating Shadows on behalf of mobile devices.

A Shadow is a mobile agent, acting as an intermediary between a mobile device and infrastructure applications. Being able to migrate allows it to move "closer" to the mobile device, and to communicate with it using the address allocated by the local network. The Shadow's functions include: *(i)* to create applications on behalf of the mobile device; *(ii)* to send messages to the applications on behalf of the mobile device; *(iii)* to store and forward messages for the mobile device; *(iv)* to migrate to a location closer to the mobile device, whenever the mobile device changes its location, network connectivity permitting.

Our architecture may be summarised as follows. When connected to a network, a mobile device makes contact with a Shadow manager, and requests it to migrate its Shadows to the manager's location. If no user's Shadow is active, the Shadow Manager creates a new Shadow for the mobile device. In the simplest case, there exists a single Shadow. If migration is successful, the Shadow interacts locally with the mobile device. The Shadow spawns new applications as requested by the device and forwards messages to and from them; in essence, the Shadow acts as a router of messages to the applications. Communications between Shadow and applications are robust to the migration of Shadows, based on a transparent routing algorithm [8, 9]; on the other hand, communications between device and Shadow may fail as the device changes location. If the migration of all Shadows fails, a new Shadow is spawned locally, and the device keeps a log of all created Shadows. When several Shadows are requested to migrate to a specific destination, the first Shadow to reach the location is assigned to be the *main Shadow;* the others coordinate with it to offload information about applications they were routing messages to.

In the following section, we describe precisely the algorithm of each component. Our goal is to define an abstraction layer, which hides the details of communication and coordination between mobile devices, Shadows and applications. On top of this abstraction layer, we have constructed an application, which supports information sharing between mobile users in virtual meeting rooms: in the mobile device, a programming API is provided to communicate transparently with fixed infrastructure applications, while applications are given the possibility to interact transparently with mobile devices; the abstraction layer takes care of all necessary routing and coordination. In addition, the coordination between multiple Shadows introduces a global property, which allows the mobile device

to interact only with a local main Shadow, while Shadows are allowed to interact directly with the mobile device only if it is connected locally.

4 The Algorithm

In this section, we describe the algorithm coordinating the interactions between mobile devices, Shadows, Shadow managers and applications. In Figure 1, we define a set of notations that we use in our algorithm.

LAN_i	(Local Network i)	β^{MD}, α^{MD}	(Mobile Device : Identifier, Address)
$P_{i,j}$	(Platform j on LAN_i)	$\beta^{MS}, \alpha^{MS}, \gamma^{MS}$	(Main Shadow : Identifier, Address,
$SM_{i,j}$	(Shadow Manager j		Counter)
	on LAN_i)	β^{S}, α^{S}	(Shadow : Identifier, Address)
LS	(List of active Shadows)	$\beta^{SM_{i,j}}, \alpha^{SM_{i,j}}$	($SM_{i,j}$: Identifier, Address)
LA	(List of applications)	$\beta^{App}, \alpha^{App}$	(Application : Identifier, Address)
		LOM	(List of outgoing messages)
		LIM	(List of incoming messages)

$send(\beta, \alpha, Msg)$	(Send message Msg to entity identified by β, α)
$receive(\beta, \alpha, Msg)$	(Receive message Msg from entity identified by β, α)

List operations:	$[M] : L$	(List composed of a head M and tail L)
	$L_1 : L_2$	(Concatenation of two lists L_1, L_2)
	$L : [M]$	(Message M added to the end of list L)
	$enqueue(M, L) \equiv L := L : [M]$	

Fig. 1. Notation

Mobile Device The behaviour of a mobile device is described in Figures 2 and 3. Each mobile device stores a main Shadow identifier and a list of Shadows (referred to by their identifier and address) that are active on the fixed network.

Every time a mobile device connects to the fixed network, it resets its main Shadow identifier; it discovers a local Shadow Manager, and sends it a request to migrate its active Shadows to the platform on which the Shadow Manager is operating. If the request fails to be sent, another local Shadow Manager is discovered and a similar request is sent. Otherwise, the mobile device starts to wait for messages. If no "$ShadowInformation$" message is received in an expected time range, a timeout occurs and a new request is sent to the Shadow Manager. A "$ShadowInformation$" message informs about the existence of a Shadow and its current address. When received, this information is updated in the list of active Shadows. If no main Shadow exists, the message's sender is assigned to be the main Shadow by sending an "$MSAssignment$" message

Variables: $LS, LOM, LIM, \beta^{MD}, \gamma^{MS} := 0$;

When MD connects to LAN_i at address α:

- $Online := true$;
- $\beta^{MS} := \perp; \alpha^{MS} := \perp$; *//resets identifier and address of main Shadow*
- D: discover $SM_{i,j}$ with identifier $\beta^{SM_{i,j}}$ and address $\alpha^{SM_{i,j}}$;
- MR: $send(\beta^{SM_{i,j}}, \alpha^{SM_{i,j}}, MigrateShadow(LS))$;
- if failed, then: discover other Shadow Manager, goto D;
- else:
  - if $receive(\beta^{SM_{i,j}}, \alpha^{SM_{i,j}}, SM^{Ack})$, then:
    - start timer$(ShadowInformation(\beta^{MD}, migrateCount^{S_x}), d)$;

Repeatedly process the incoming messages:

- if $receive(\beta^S, \alpha^S, ShadowInformation(\beta^{MD}, migrateCount^{S_x}))$, then:
  - stop timer$(ShadowInformation(\beta^{MD}, migrateCount^{S_x}), d)$;
  - $LS[\beta^S] := \alpha^S$;
  - if $\beta^{MS} = \perp$, then:
    - $\beta^{MS} := \beta^S; \alpha^{MS} := \alpha^S; \gamma^{MS}++$;
    - $enqueue((\beta^{MS}, \alpha^{MS}, MSAssignment(LS, \gamma^{MS}), LOM)$;
  - else: $enqueue((\beta^S, \alpha^S, MSInformation(\beta^{MS}, \alpha^{MS}, \gamma^{MS}), LOM)$;
- if $receive(\beta^{MS}, \alpha^{MS}, MessageFailed(\beta^{App}, Msg))$, then:
  - $enqueue((\beta^{MS}, \alpha^{MS}, Message(\beta^{App}, Msg)), LOM)$;
- if $receive(\beta^{MS}, \alpha^{MS}, CreationFailed(\beta^{App}, \epsilon^{App}))$, then:
  - $enqueue((\beta^{App}, CreationFailed(\beta^{App}, \epsilon^{App})), LIM)$;
- if $receive(\beta^{MS}, \alpha^{MS}, Message(\beta^{App}, Msg))$, then: $enqueue((\beta^{App}, Msg), LIM)$;
- if $receive(\beta^{MS}, \alpha^{MS}, ShadowTermination(\beta^S))$, then: $LS[\beta^S] := \perp$;

In parallel:

- if timeout, then:
  - stop timer$(ShadowInformation(\beta^{MD}, migrateCount^{S_x}), d)$;
  - $send(\beta^{SM_{i,j}}, \alpha^{SM_{i,j}}, StartShadow)$;

Exported API:

- $CreateApplication(\beta^{App}, \epsilon^{App})$:
  - $enqueue((\beta^{MS}, \alpha^{MS}, CreateApplication(\beta^{App}\epsilon^{App})), LOM)$;
- $SendMessageToApp(\beta^{App}, Msg)$:
  - $enqueue((\beta^{MS}, \alpha^{MS}, Message(\beta^{App}, Msg)), LOM)$;
- $GetMessageFromApp()$:
  - if $LIM = \perp$, then: return $\perp$;
  - if $LIM = [(\beta^{App}, Msg)] : LIM'$, then: $LIM := LIM'$; return (β^{App}, Msg);

Fig. 2. Mobile Device's Behaviour (i)

and if a main Shadow already exists, the Shadow is sent a "$MSInformation$" message, which notifies about the new main Shadow.

An API is provided for the application layer to create a new application, to send a message to an application, or to receive a message from an application on the infrastructure — applications are referred to by identifiers passed as argument to the API procedures. The first two procedures result in a message sent to the main Shadow. In return, the mobile device may receive a failure notification from the main Shadow indicating its failure to create the application or to send a message to the application. Incoming application messages are kept in a queue of messages, until the application layer reads them. Incoming messages are changing the internal state and queues of the mobile device. For instance, when a "*ShadowTermination*" message, which informs about a recently terminated Shadow, is received, the terminated Shadow's detail is removed from the mobile device' list of active Shadows.

Messages to be sent to Shadows are added to a queue of outgoing messages, while in parallel (cf. Figure 3), separate threads are responsible for processing the enqueued messages. Additionally, messages are validated before they are sent. For this purpose, we use a counter "γ^{MS}", which identifies the number of times the mobile device has changed location; such a counter is also added as a "timestamp" to messages. A message can be outdated if the intended recipient no longer exists or has changed its status or address. An outdated message can still be valid if the intended recipient is still holding the same status but has changed its address. Such messages are updated with the new address, while invalid messages are discarded from the queue. For instance, an "$MSInformation$" or an "$MSAssignment$" message is no longer valid if a new main Shadow is being assigned. On failure of sending a message, the message is added back to the queue.

Continuously in parallel: if *Online* and $\beta^{MS} \neq \perp$ and $\alpha^{MS} \neq \perp$, then:

- if $LOM = [(\beta^R, \alpha^R, Msg)] : LOM'$, then:
  - $LOM := LOM'$;
  - if $Msg = MSAssignment(LS', \gamma^{MS'})$, then:
    - if $\gamma^{MS'} = \gamma^{MS}$, then: $sendOut(\beta^{MS}, \alpha^{MS}, MSAssignment(LS, \gamma^{MS}))$;
      //use main Shadow's latest address α^{MS} *and latest LS*
  - if $Msg = MSInformation(\beta^{MS'}, \alpha^{MS'}, \gamma^{MS'})$, then:
    - if $\gamma^{MS'} = \gamma^{MS}$ then:
      $sendOut(\beta^R, LS[\beta^R], MSInformation(\beta^{MS}, \alpha^{MS}, \gamma^{MS}))$;
      //use β^R*'s latest address*
  - else: $sendOut(\beta^R, \alpha^R, Msg)$;

Subroutine: $sendOut(\beta^R, \alpha^R, Msg)$:

- $send(\beta^R, \alpha^R, Msg)$;
- if fail, then: $enqueue((\beta^R, \alpha^R, Msg), LOM)$;

When MD disconnects from LAN_i**:** $Online := false$;

Fig. 3. Mobile Device's Behaviour (ii)

SM_{ij} running at address $\alpha_{i,k}$:

- advertise its presence at address $\alpha_{i,k}$;
- **Repeatedly process the incoming messages:**
- if $receive(\beta^{MD}, \alpha^{MD}, MigrateShadow(LS))$, then:
  - $send(\beta^{MD}, \alpha^{MD}, SM^{Ack})$;
  - $C := 0$; *//a counter*
  - for each pair (β^S, α^S) in LS:
    - $send(\beta^S, \alpha^S, MigrateRequest(\beta^{MD}, \alpha^{MD}, \alpha_{i,k}))$;
    - if successful, then C:=C+1;
  - if $C = 0$, then:
    - $startShadow(\beta^{MD}, \alpha^{MD})$;
- if $receive(\beta^{MD}, \alpha^{MD}, StartShadow)$, then:
  - $startShadow(\beta^{MD}, \alpha^{MD})$;

Subroutine: $startShadow(\beta^{MD}, \alpha^{MD})$:

- start a Shadow S; with new identifier β^S at address α^S;
- $send(\beta^S, \alpha^S, MDInformation(\beta^{MD}, \alpha^{MD}))$; resend on failure;

Fig. 4. Shadow Manager's Behaviour

Shadow Manager In our architecture, Shadow Managers are stationary agents running on the fixed network and in order for the algorithm to work, we assume there is at least one Shadow Manager operating on each local network. When a Shadow Manager is started (cf. Figure 4), it advertises its presence through a service directory, such as Jini or LDAP, and then waits for messages. A Shadow Manager may receive a request from a mobile device to migrate Shadows; the Shadow Manager then sends a "$MigrateRequest$" message to each Shadow requesting them to migrate to the platform on which it is operating. If no Shadow was able to migrate or if it receives a "$StartShadow$" request, it starts a new Shadow, to which an "$MDInformation$" message that contains information on the requesting mobile device is sent. The message is repeatedly sent on failure until the Shadow eventually gets it.

Shadow Figure 5 describes the global behaviour of Shadows while Figure 6 and 7 are describing specific behaviour of a regular Shadow and a main Shadow. A Shadow is able to create applications on request from mobile device. Each application has an identifier and an address. A variable "LA" is used to keep a mapping of application identifier to application address, which is initially empty. A Shadow has a handOver flag, which is set to false on its creation. This flag becomes true if the Shadow's function has been transferred to the main Shadow. Messages to be sent to the mobile device are queued in a list of outgoing messages to mobile device (LOM^{MD}), while messages to be sent to the main Shadow are queued in a list of outgoing messages to main Shadow (LOM^{MS}). Incoming messages from the mobile device for the applications are queued in a list of incoming messages (LIM), while a list (LOM^{Ack}) queues acknowledgement messages to

other Shadows. In parallel, separate threads are responsible for processing the enqueued messages, which would add messages failed to be sent, back to their respective queues.

Variables: $\beta^S, \beta^{MS}, \gamma^{MS}, \alpha^{MD}, \beta^{MD}, LA, handOver, LOM^{MD}, LOM^{MS}, LIM, LOM^{Ack}, migrateCount, handOverCount, MS, LS$;

Continuously in parallel:

- if $LOM^{MD} = [Msg] : LOM^{MD'}$, then:
  - remove all $ShadowInformation(\beta^{MD}, migrateCount')$ messages from LOM^{MD} where $migrateCount' \neq migrateCount$;
  - $LOM^{MD} := LOM^{MD'}$;
  - if $\beta^{MS} = \perp$, then: $send(\beta^{MD}, \alpha^{MD}, Msg)$;
  - else: $send(\beta^{MS}, \alpha^{MS}, SendMessageToMD(Msg))$;
  - if failed, then: $enqueue(Msg, LOM^{MD})$;
- if $LIM = [(\beta^{App}, Msg)] : LIM'$, then:
  - $LIM := LIM'$;
  - $send(\beta^{App}, LA[\beta^{App}], Msg)$; if failed, then: $enqueue((\beta^{App}, Msg), LIM)$;
- if $LOM^{Ack} = [(\beta^{S_x}, \alpha^{S_x}, Msg)] : LOM^{Ack'}$, then:
  - $LOM^{Ack} := LOM^{Ack'}$;
  - $send(\beta^{S_x}, \alpha^{S_x}, Msg)$; if failed, then: $enqueue((\beta^{S_x}, \alpha^{S_x}, Msg), LOM^{Ack})$;

Subroutine: $migrate(\alpha^{P_{x,y}})$:

- if $(LAN_x \neq LAN_i)$, then: migrate to $\alpha^{P_{x,y}}$;
- $migrateCount++$;
- $\beta^{MS} := \perp; \alpha^{MS} := \perp; MS = false$;
- $enqueue(ShadowInformation(\beta^{MD}, migrateCount), LOM^{MD})$;

Fig. 5. Global Behaviour of Shadows

A migrateCount is a variable, which keeps track of the number of migrations of a Shadow. This variable is important to validate messages during the process of sending out messages from the list of outgoing messages. For example, a "*ShadowInformation*" message, which is used to inform the mobile device about the Shadow's arrival on a platform is no longer valid if the Shadow has already migrated to a new platform. In this case, the migrateCount variable recorded in the "*ShadowInformation*" message would be less than the current migrateCount, thus showing that the message is outdated and invalid. Such messages are discarded from the list. In a Shadow, there is a hook for intelligent decision making about migration; such decision is not part of this algorithm, and may depend on the state of the application or prevailing network condition. The output of this decision making process is obtained by the "callback" canMigrate(), which returns true if the application layer decides to migrate.

A regular Shadow S is recently started or migrated on platform $P_{i,k}$ at address $\alpha_{i,k}$ on LAN_i:
$handOver := false$; $handOverCount := length(LA)$; $LS := \perp$; $MS := false$; $\beta^{MS} := \perp$; $\alpha^{MS} := \perp$;
Repeatedly process the incoming messages:

- if $receive(\beta^{SM_{ij}}, \alpha^{SM_{ij}}, MDInformation(\beta^{MD'}, \alpha^{MD'}))$, then:
  - if $\beta^{MD} = \perp$ and $\alpha^{MD} = \perp$, then:
    - $\beta^{MD} := \beta^{MD'}$; $\alpha^{MD} := \alpha^{MD'}$;
    - $enqueue(ShadowInformation(\beta^{MD}, migrateCount), LOM^{MD})$;
- if $receive(\beta^{MD}, \alpha^{MD}, MSAssignment(LS, \gamma^{MS'}))$, then:
  - MS := true; $LS := LS'$; $\gamma^{MS} := \gamma^{MS'}$; $\beta^{MS} := \perp$; $\alpha^{MS} := \perp$; //behave as in Figure 7)
- if $receive(\beta^{MD}, \alpha^{MD}, MSInformation(\beta^{MS'}, \alpha^{MS'}, \gamma^{MS'}))$, then:
  - if $\gamma^{MS'} > \gamma^{MS}$, then: $\beta^{MS} := \beta^{MS'}$; $\alpha^{MS} := \alpha^{MS'}$; $\gamma^{MS} := \gamma^{MS'}$;
  - $enqueue(Transfer^F(\beta^{MD}, LA, LOM^{MD}, LIM), LOM^{MS})$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, MSInformation(\beta^{MS'}, \alpha^{MS'}, \gamma^{MS'}))$, then:
  - if $\gamma^{MS'} > \gamma^{MS}$, then: $\beta^{MS} := \beta^{MS'}$; $\alpha^{MS} := \alpha^{MS'}$; $\gamma^{MS} := \gamma^{MS'}$;
- if $receive(\beta^{SM_{u,q}}, \alpha^{SM_{u,q}}, MigrateRequest(\beta^{MD}, \alpha^{MD'}, \alpha^{P}))$, then:
  - $\alpha^{MD} := \alpha^{MD'}$; if canMigrate(), then: $migrate(\alpha^P)$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, LocationInformation(\beta^{MD}, \alpha^{MD'}, \alpha^P, \gamma^{MS'}))$ and $\gamma^{MS'} > \gamma^{MS}$, then:
  - $\alpha^{MD} := \alpha^{MD'}$; $\beta^{MS} := \beta^{S_x}$; $\alpha^{MS} := \alpha^{S_x}$;
  - if $canMigrate()$, then: $migrate(\alpha^P)$;
  - $enqueue(Transfer^F(\beta^{MD}, LA, LOM^{MD}, LIM), LOM^{MS})$;
- if $receive(\beta^{MS}, \alpha^{MS}, Transfer^F\_Ack)$, then: $handOver := true$;
- if $receive(\beta^{MS}, \alpha^{MS}, Termination\_Ack)$, then: terminate();
- if $receive(\beta^{MS}, \alpha^{MS}$, $SendMessageToNewMS(MSInformation(\beta^{MS'}, \alpha^{MS'}, \gamma^{MS'}), Msg))$, then:
  - if $\gamma^{MS'} > \gamma^{MS}$, then: $\beta^{MS} := \beta^{MS'}$; $\alpha^{MS} := \alpha^{MS'}$;
  - $enqueue(Msg, LOM^{MD})$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, Msg)$, then:
  - if $Msg = Transfer^F(\beta^{MD}, LA^{S_x}, LOM^{MD_S x}, LIM^{S_x})$ or $Msg = TerminationMessage()$, then:
    - $enqueue((\beta^{S_x}, \alpha^{S_x}, MSInformation(\beta^{MS}, \alpha^{MS}, \gamma^{MS})), LOM^{Ack})$;
  - if $Msg = SendMessageToMD(M)$, then:
    - $enqueue((\beta^{S_x}, \alpha^{S_x}$, $SendMessageToNewMS(MSInformation(\beta^{MS}, \alpha^{MS}, \gamma^{MS}), M)), LOM^{Ack})$;
- if $receive(\beta^{App}, LA[\beta^{App}], Msg))$, then: $enqueue(Msg, LOM^{MD})$;
- if $receive(\beta^{App}, LA[\beta^{App}], App^{Ack}))$, then: $handOverCount - -$;

Continuously in parallel:

- if $handOver$, then: for each application mapping $(\beta^{App}, \alpha^{App})$ in LA:
  - $enqueue((\beta^{App}, NewShadowInformation(\beta^{MS}, \alpha^{MS})), LIM)$;
- if $handOverCount = 0$, then: $enqueue(TerminationMessage(\beta^{MD}), LOM^{MS})$;
- if $LOM^{MS} = [Msg] : LOM^{MS'}$, then:
  - $LOM^{MS} := LOM^{MS'}$;
  - $send(\beta^{MS}, \alpha^{MS}, Msg)$; if failed, then: $enqueue(Msg, LOM^{MS})$;

Fig. 6. Regular Shadow's Behaviour (MS=false)

A main Shadow S is running on platform $P_{i,k}$ at address $\alpha_{i,k}$ on LAN_i:
Repeatedly process the incoming messages:

- if $receive(\beta^{SM_{u,q}}, \alpha^{SM_{u,q}}, MigrateRequest(\beta^{MD}, \alpha^{MD'}, \alpha^{P}))$, then:
  - $\alpha^{MD} := \alpha^{MD'}$; MS := false; //behave as in Figure 6)
  - if canMigrate(), then: $migrate(\alpha^{P})$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, LocationInformation(\beta^{MD}, \alpha^{MD'}, \alpha^{P}, \gamma^{MS'}))$ and $\gamma^{MS'} > \gamma^{MS}$, then:
  - $MS := false; \alpha^{MD} := \alpha^{MD'}; \beta^{MS} := \beta^{S_x}; \alpha^{MS} := \alpha^{S_x}$; //behave as in Figure 6)
  - if $canMigrate()$, then: $migrate(\alpha^{P})$;
  - else: $enqueue(Transfer^{F}(\beta^{MD}, LA, LOM^{MD}, LIM), LOM^{MS})$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, Transfer^{F}(\beta^{MD}, LA^{S_x}, LOM^{MD_{S^x}}, LIM^{S_x}))$, then:
  - $LA := LA : LA^{S_x}$; $LIM := LIM : LIM^{S_x}$;
  - $LOM^{MD} := LOM^{MD} : LOM^{MD_{S^x}}$;
  - $enqueue((\beta^{S_x}, \alpha^{S_x}, Transfer^{F}\_Ack), LOM^{Ack})$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, SendMessageToMD(Msg))$, then:
  - $enqueue(Msg, LOM^{MD})$;
- if $receive(\beta^{S_x}, \alpha^{S_x}, TerminationMessage(\beta^{MD}))$, then:
  - $enqueue(ShadowTermination(\beta^{S_x}), LOM^{MD})$;
  - $enqueue((\beta^{S_x}, \alpha^{S_x}, Termination\_Ack), LOM^{Ack})$;

Continuously in parallel, if $LS \neq \perp$ then:

- if $LS = [(\beta^{S'}, \alpha^{S'})] : LS'$ and $\beta^{S'} \neq \beta^{S}$, then:
  - $send(\beta^{S'}, \alpha^{S'}, LocationInformation(\beta^{MD}, \alpha^{MD}, \alpha_{i,k}, \gamma^{MS}))$;
  - if fails, then: $LS := LS' : [(\beta^{S'}, \alpha^{S'})]$; else: $LS := LS'$;

Interface with Applications:

- if $receive(\beta^{MD}, \alpha^{MD}, CreateApplication(\beta^{App}, \epsilon^{App}))$, then:
  - $StartApplication(\beta^{App}, \epsilon^{App})$; its address is α^{App};
  - if success, then: $LA[\beta^{App}] := \alpha^{App}$;
  - else: $enqueue(CreationFailed(\beta^{App}, \epsilon^{App}), LOM^{MD})$;
- if $receive(\beta^{MD}, \alpha^{MD}, Message(\beta^{App}, Msg))$, then:
  - if $LA[\beta^{App}] \neq \perp$, then: $enqueue((\beta^{App}, Msg), LIM)$;
  - else: $enqueue(MessageFailed(\beta^{App}, Msg), LOM^{MD})$;
- if $receive(\beta^{App}, \alpha^{App}, Msg))$, then:
  - if $LA[\beta^{App}] \neq \perp$, then: $enqueue(Message(\beta^{App}, Msg), LOM^{MD})$;
  - else: $enqueue((\beta^{App}, MessageFailed(Msg)), LIM)$;

Fig. 7. Main Shadow's Behaviour (MS=true)

When started (cf. Figure 6), a Shadow waits for "$MDInformation$" message, which contains information about a mobile device. Then, the Shadow sends the mobile device a "$ShadowInformation$" message informing its existence at its current address. The Shadow waits for messages; if it receives an "$MSAssignment$" message, it sets its main Shadow (MS) flag to true as it is

being assigned by the mobile device to be the main Shadow. Instead of receiving an "$MSAssignment$" message, a Shadow may receive an "$MSInformation$" message, which signifies that another Shadow has been assigned to be the main Shadow. The Shadow sets its MS flag to false and updates its information about the main Shadow accordingly. A Shadow may receive a "$MigrateRequest$" message from a Shadow Manager, which requests the Shadow to migrate to the platform on which the Shadow Manager is operating. If canMigrate() returned true, the Shadow migrates to the new platform. On arrival at the new platform the Shadow resets its MS flag to false and sends a "$ShadowInformation$" to the mobile device. Then it waits for messages. If a Shadow cannot migrate, it stays on the same platform and continues to wait for further messages.

A regular Shadow has to hand over its function to the main Shadow by sending its LA, LOM^{MD} and LIM in a "$Transfer^{F}$" message. Then, the Shadow sets its handOver flag to true when it received a "$Transfer^{F}\_Ack$" message, and sends messages to all applications it is interacting with that the main Shadow is the new intermediary to communicate with the mobile device. Every message sent to the applications requires an acknowledgement to ensure that the recipient has successfully received the message. Subsequently, the Shadow is ready for termination; before terminating itself, it sends a "$TerminationMessage$" to the main Shadow and waits for an acknowledgement.

"$MSAssignment$", "$MSInformation$" and "$LocationInformation$" are types of messages, which carry information about the main Shadow. Each of this messages contains main Shadow assignment counter "γ^{MS}". This is to avoid Shadows to use an outdated information about the main Shadow. For instance, if a Shadow received an "MSInformation" message with a counter that is less than the one contained in a previously received message, the message is considered as outdated and discarded. Sometimes, a regular Shadow may receive a message intended for a main Shadow, such as when a Shadow receiving a "$Transfer^{F}$" or a "$TerminationMessage$" message from another Shadow. In this case, an "$MSInformation$" message is returned informing about the current main Shadow. If a "$SendMessageToMD$" message is received, which requests it to send a message to the mobile device, a "$SendMessageToNewMS$" message containing an "$MSInformation$" and the message for the mobile device is replied to the sending Shadow.

A main Shadow sends a "$LocationInformation$" message to all Shadows of the mobile device. The message indicates current location of the mobile device. If a main Shadow received a "$Transfer^{F}$" message, the LA, LOM^{MD} and LIM of another Shadow, contained in the message are extended to the Shadow's local lists. A main Shadow may also receive a "$SendMessageToMD$" request from a Shadow, which requires it to relay the included message in the request to the mobile device. A "$TerminationMessage$" received notifies about the termination of a Shadow. This information is relayed to the mobile device in a "$ShadowTermination$" message. For every message received from another Shadow, an acknowledgement is returned to the sender. As for messages coming from the mobile device, a main Shadow may receive requests to create an

application or send a message to an application on the fixed infrastructure. The details of a newly created application are added to LA. If the Shadow failed to create an application or to send a message, a failure notification is returned to the mobile device. The Shadow also relays messages from the applications to the mobile device.

Summary In our algorithm, we make sure that messages are not lost, in which case whenever communication failures occur, messages involved are put in queues. For instance, when a mobile device is disconnected from the network and no longer able to send messages to the main Shadow, those messages are added to the queue of outgoing messages. Once the mobile device reconnects to the network, messages from the queue are sent out. The same applies to the Shadows; once a Shadow failed to send a message to another Shadow or to the mobile device, the message is stored in a queue to be sent out again later. The algorithm also tries to terminate Shadows that no longer act as routers for applications and have handed over their functions to the main Shadow. Terminating such Shadows is important as it clears garbage in the system. The outcome is that some Shadows maybe temporarily disconnected from the main Shadow, and therefore may loose the route to deliver messages to the mobile device. Messages are not lost; they remain in the queue and will be forwarded when connectivity get re-established again. We are considering another approach where termination of Shadows is not as eager, in order to ensure some redundancy in the routing along the lines of [9]. Similarly, such an approach may be considered for handling host failures.

The coordination layer really benefits from Mobile Agent technology. First, mobile code can be transported to a remote platform and activated, in order to perform its tasks. Second, mobile agents also incorporate a state in addition to mobile code. Such a mobile state is needed to hold all information required to perform the coordination algorithm, which includes information on the applications the Shadow is interacting with, the mobile device location, the main Shadow information, handOver flag and queues of messages.

5 Application Implementation

We have developed our application using the Southampton Framework for Agent Research (SoFAR) [10], which supports weak mobility. The algorithm of the abstraction layer is implemented by three agents, namely a Mobile Device Agent, a Shadow Manager Agent and a Shadow Agent. Our application is currently applicable for high capability mobile machines such as laptops. We host a Mobile Device Agent on the mobile machine. The Shadow Agent is mobile, while Mobile Device Agent and Shadow Manager Agent are stationary. Although stationary on the hosting laptop, a Mobile Device Agent benefits from the physical mobility of its hosting environment. On top of the abstraction layer, we are prototyping the application mentioned earlier in Section 2. The Recommender system [11] is a stationary application located on the fixed infrastructure.

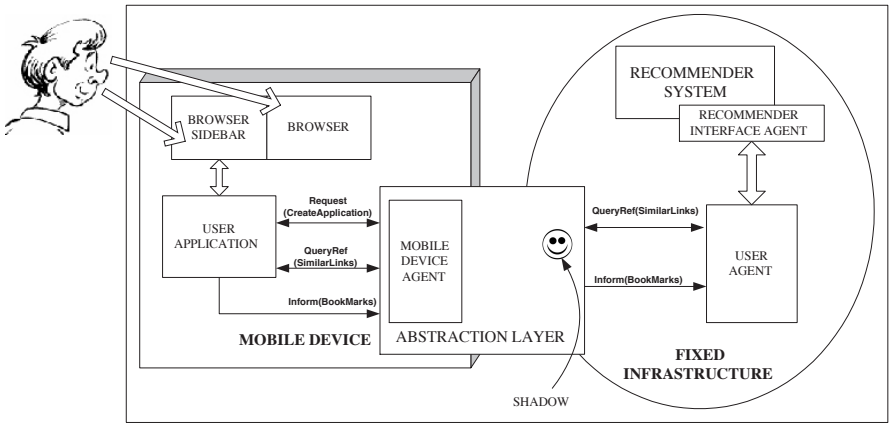

Fig. 8. Application Interactions

The application interactions are illustrated in Figure 8. A mobile user uses a browser on a laptop to access information. The browser sidebar interacts directly with a User Application, which is also an agent running on the laptop. User's requests are sent by the browser sidebar to the User Application. If no User Agent is started on the fixed infrastructure, the User Application requests the abstraction layer to start a User Agent. Then, the User Application is ready to route user's requests to the User Agent using the delivery mechanism we described in Section 4. For a request to get recommendations on related urls or to get other users' bookmarks, a "*SimilarLinks*" query is constructed. An inform "*Bookmarks*" message is created if the User Application received a request to export a set of bookmarks to the infrastructure. These messages are then forwarded to the User Agent. For a "*SimilarLinks*" query, the User Application would get a set of urls to similar documents or a set of other users' bookmarks in return. These urls or bookmarks are then forwarded to the browser sidebar to be displayed.

On the infrastructure, a User Agent queries a Registry Agent for a Recommender Interface Agent, with which it interacts in order to use services offered by the Recommender system. Every "*SimilarLinks*" query or "*Bookmarks*" inform message received from the mobile device is forwarded to the Recommender Interface Agent. Results of "*SimilarLinks*" queries are returned to the mobile device. Interactions between a User Application on the mobile device and a User Agent on the fixed infrastructure are supported by our abstraction layer.

At this stage, we have not completed a formal evaluation of our architecture, but we are collecting observations about it. Given the application, we informally compared the use of a laptop with our abstraction layer and without it. In both cases, the laptop can interact with applications on the fixed infrastructure. In the second case, when the laptop is disconnected, messages may be lost and resending of messages will has to be programmed at the application level. In the

first case, the use of a mobile agents and the transparent routing of messages to them [8] solved the problem of delivering messages to the laptop; alternatively in the second case, full IPv6 may be necessary to route messages to the laptop mobile address. Finally, by addressing the problem of the reliable delivery and of the routing of messages in an abstraction layer, we have designed a generic solution reusable by other applications having to support mobile users.

6 Related Work

In [6], mobile agents are used to move between resources on the fixed infrastructure to take advantages of those resources in order to accomplish tasks for a mobile user. An Agent Gateway, which is a stationary host, is acting as the mediator between the wireless device and fixed infrastructure resources. This is different from our approach as we adopt a mobile agent to be such a mediator. Having a mobile mediator is more flexible as it can move closer to the current location of the user, which allows local communication to be established.

The "Personal Agent System" [1] provides a mobile user with a personalised information retrieval service. The Personal Agent is a mobile agent that resides on the fixed infrastructure and communicates with agents residing on the mobile device. The system is similar to ours in the sense that it involves migration of the Personal Agent to other stationary servers so that it follows the mobile user around in the wired network, while the user moves around in the wireless network. In comparison, our abstraction layer provides more flexibility since we allow multiple mobile agents to exist when the user's current local network is not connected with the user's previous location.

The Mobile Agents Platform (MAP) architecture [2] involves data servers to store results acquired by a mobile agent for a mobile user once the mobile user is disconnected from the network. When reconnected, the user has to undergo multiple communication steps to get the result, like having to query the lookup table for data server address and then to query the data server for the result. Our approach is much simpler since we provide a store and forward mechanism built in the abstraction layer, which allows the results of user's queries to be forwarded to the mobile user once the mobile user is reconnected to the network.

In the M-Commerce Framework [7], a mobile agent called Service Agent is moving around the wired network to gather information for a mobile user, while another mobile agent called Courier Agent is migrating to the mobile device to establish an interaction with the Service Agent on the fixed infrastructure. Migrating the Courier Agent to the mobile device in order to interact with the Service Agent puts more burden on the network connection than a migration between two hosts on the fixed network, as it involves the ability to move the agent state and code, which includes the serialisation and deserialisation of the transferred data through the low bandwidth wireless communication channels. In our abstraction layer, we adopt a simpler approach, where an application residing on the mobile device is responsible for interacting with applications on the fixed infrastructure.

In the Tacoma Architecture [4], a support specific to PDA application is provided using an entity called "hostel", which is the host that a PDA normally uses to synchronise data with. The hostel is also assumed to act as the network provider or proxy for the PDA, i.e. the hostel is a networked workstation. In this architecture, mobile agents are used to gather information on the wired network assuming the presence of a host that they can inquire in case the PDA is not connected. This approach is suitable for a PDA user that has the hostel as the only connection point needed for the PDA. But in the case of a user who is always on the move and needs to connect to different hosts, a more flexible approach such as having a mobile agent acting as the "hostel" is more suitable.

7 Conclusion

A mobile agent able to migrate around the network trying to stay as close as possible to the mobile user, gives a major advantage by allowing local communication to be established with the mobile device. With this capability, the mobile agent is designed to be the main component in our abstraction layer, which allows transparent interactions between fixed infrastructure applications and applications on mobile devices.

This paper has presented an application, which supports information sharing between mobile users in virtual meeting rooms. The main challenge in developing the application is to construct an intermediary layer, which supports seamless communication between a traveling mobile user and virtual meeting rooms hosted by the fixed infrastructure. We have introduced an architecture and algorithm of the intermediate layer, based on a mobile agent called Shadow. This intermediary layer takes care of coordination of multiple Shadows, as well as the communication between a mobile device and its Shadows. It is defined as an abstraction layer, which hides the details of communication and coordination, allowing transparent interactions between fixed infrastructure applications and applications on a mobile device.

Having the intermediary layer has made the implementation of the application straightforward, in which the layer takes care of complex interactions with mobile devices. An agent-based Recommender system [11] is used in our application to provide an information sharing environment between User Agents, which are mobile-users' representative in the virtual meeting room. The User Agents interact transparently with a mobile device through the intermediary layer. We believe such ability is important to allow more applications for mobile users to be easily developed.

8 Acknowledgement

This research is funded in part by QinetiQ and EPSRC Magnitude project (reference GR/N35816).

References

1. Debbie Chyi. An Infrastructure for a Mobile-Agent System that Provides Personalized Services to Mobile Devices. Technical Report TR2000-370, Dartmouth College Computer Science, 2000.
2. A. La Corte, A Puliafito, and O. Tomarchio. An Agent-based Framework for Mobile Users. In *Proceedings of European Research Seminar on Advances in Distributed Systems 1999*, Madeira, Portugal, 1999.
3. Robert S. Gray, David Kotz, Ronald A. Peterson, Joyce Barton, Daria Chacon, Peter Gerken, Martin Hofmann, Jeffrey Bradshaw, Maggie R. Breedy, Renia Jeffers, and Niranjan Suri. Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task. In *Mobile Agents*, pages 229–243, 2001.
4. Kjetil Jacobsen and Dag Johansen. Mobile Software on Mobile Hardware – Experiences with TACOMA on PDAs. Technical Report 97-32, Department of Computer Science,University of Troms, Norway, 1997.
5. Danny B. Lange and Mitsuru Ishima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.
6. Q.H. Mahmoud. MobiAgent – An Agent-based Approach to Wireless Information Systems. In *Proceedings of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2001)*, Montreal, 2001.
7. Patrik Mihailescu and Walter Binder. A Mobile Agent Framework for M-Commerce. *Computer Science 2001*, GI/OCG annual Convention:2:959–967. .
8. Luc Moreau. Distributed Directory Service and Message Router for Mobile Agents. *Science of Computer Programming*, 39(2–3):249–272, 2001.
9. Luc Moreau. A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers. In *The 17th ACM Symposium on Applied Computing (SAC'2002) — Track on Agents, Interactions, Mobility and Systems*, Madrid, March 2002.
10. Luc Moreau, Nick Gibbins, David DeRoure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, and Danius Michaelides. SoFAR with DIM Agents An Agent Framework for Distributed Information Management. In *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 369–388, 2000.
11. Luc Moreau, Norliza Zaini, Jing Zhou, Nicholas R. Jennings, Yan Zheng Wei, Wendy Hall, David De Roure, Ian Gilchrist, Mark O'Dell, Sigi Reich, Tobias Berka, and Claudia Di Napoli. A Market-Based Recommender System. In Paolo Giorgini, Yves Lespérance, Gerd Wagner, and Eric Yu, editors, *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems at AAMAS 2002 (AOIS'02)*, Bologna, Italy, July 2002. http://CEUR-WS.org/Vol-59/.
12. Mark Weiser. Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM*, 36(7):74–84, July 1993.