
Mobile Agents and Their Use for Information Retrieval: A Brief Overview and an Elaborate Case Study

Roch H. Glitho, Ericsson Research Canada
Edgar Olougouna, Samuel Pierre, Ecole Polytechnique, Canada

Abstract

Mobile agents emerged in the mid-1990s, and have raised considerable interest in the research community. The proponents associate several benefits with their use. However, there are still very few quantitative measurements to back the claimed benefits. This article is devoted to mobile agents and their use for information retrieval. We provide a brief overview and an elaborate case study. The overview introduces the concept of mobile agent, enumerates the claimed benefits, and reviews the hindrances to widescale deployment. It also discusses the state of the art of mobile-agent-based information retrieval, including the very few quantitative studies that exist. Our case study is on information retrieval from electronic calendars for multiparty event scheduling. Many events require the participation of several parties. Prior knowledge of the date when most (if not all) targeted participants are available is often a prerequisite for scheduling them. However, identifying this date can easily turn into a nightmare, especially when the number of targeted participants is large. Nowadays, electronic agendas (e.g., MS Outlook) are stored on servers. An application can access them, retrieve information on the availability of the targeted participants, and derive the date from the information. In the case study, a mobile agent is dispatched in the network, instead of retrieving the information using the client/server paradigm. The agent visits the servers, accesses the agendas, retrieves the information, and identifies the date. Finding a date suitable for several potential participants may require the rescheduling of some events that have been previously arranged by some participants. We propose the use of agents that act as the personal agents of the participants for the negotiation inherent to this rescheduling. The measurements we have made indicate clearly that the mobile-agent-based approach outperforms its client/server counterpart even when the latter is optimized. These results can easily be transposed to most information retrieval applications, and demonstrate, for this specific application domain, the performance benefit associated with mobile agents. We now dispatch a single agent in the network. In the future, we will dispatch several agents. Mobile multi-agent systems raise several challenging issues we would like to tackle in further work.

Mobile agents emerged in the mid-1990s, and have generated a lot of interest in the research community. They have been used in applications ranging from network management to automatic software distribution, as well as information retrieval. Several benefits are associated with their use by its proponents. This article focuses on mobile agents and their use for information retrieval. We provide a terse overview and an elaborate case study.

The case study is on multiparty event scheduling. It focuses on the information retrieval aspects of the problem. A myriad of events requires the participation of several parties. These multiparty events range from social gatherings (e.g., family

reunions) to business gatherings (e.g., workshops), and include electronic multiparty sessions (e.g., videoconference). Scheduling them can easily become a nightmare, especially when a large number of participants are involved. Prior knowledge of the date when most (if not all) targeted participants are available is often required.

The scheduling process remains manual in most cases, although nowadays electronic agendas are commonly used. The event organizer usually exchanges e-mails and/or has discussions with the targeted participants to gather the information needed to identify the date when everybody is available. Electronic agendas store this very information. An application could retrieve it, and use it to identify the date.

In the case study, we dispatch a single agent in the network and contrast this approach to the client/server paradigm. The agent visits the servers, accesses the agendas, retrieves the information on availability, and identifies the date. Finding a date suitable for several potential participants usually implies that some of the participants reschedule events that have been previously arranged. Agents that act on behalf of the participants are proposed for the negotiations inherent in rescheduling.

We have prototyped our multiparty event scheduler and made measurements. The measurements show clearly that the mobile-agent-based approach outperforms the client/server counterpart even when the latter is optimally designed. Although the case study is on information retrieval for multiparty event scheduling, the results can be transposed to most information retrieval applications. It demonstrates the performance benefit associated with the use of mobile agents for the specific application domain.

The next section is devoted to the overview and covers both mobile agents and mobile-agent-based information retrieval applications. The section after that presents the application tackled by the case study. We then describe the prototype of the mobile-agent-based application and the prototype of the client/server counterparts we have built for comparison purposes. The fifth section is devoted to performance evaluation. We conclude with a summary, lessons learned, and items for future work.

A Brief Overview: Mobile Agents and Mobile-Agent-Based Information Retrieval Applications

An introduction to mobile agents is provided first. Mobile-agent-based information retrieval applications are reviewed after that.

Mobile Agents in a Nutshell

Several tutorials [1–4] have been published on mobile agents. We successively define the concept, discuss the platforms, and review the claims along with the hindrances to widescale deployment.

Mobile Agents — The term *agent* is widely used, and its meaning is often context-dependent. An agent can be broadly defined as a software entity that:

- Acts on behalf of another entity (e.g., person, another agent)
- Is autonomous and goal-oriented
- Reacts to external events

Mobile agents are agents that can migrate between physical nodes. Mobility is their primary characteristic, not intelligence. Most of them use very few of the concepts developed by the artificial intelligence community. Their mobility borrows a lot from process migration. Process migration consists of transferring a process from one computer to another. A process is an operating system abstraction that comprises the code, the data, and the state of a running application.

Platforms — Mobile agents require a special environment for execution, the agent execution environment. This environment is also known as a platform. It provides basic facilities such as:

- Mobility — This facility caters to the transportation of the agent between physical nodes.
- Communications — This facility caters to the communication of the agent with the external world.
- Naming and location — Mobile agents, like all entities, need to be named. Furthermore, it is important to know at which nodes they are at any given time.

- Security — Mobile agents need to be protected against hosts, and hosts need to be protected against mobile agents.

Nowadays, most mobile agent platforms are implemented as Java applications that run on top of operating systems. Attempts have been made in the past to implement platforms as extensions to operating systems, but they were not very successful. Platforms commonly used today include Aglets [5], Grasshopper [6], and Voyager [7].

Claims and Hindrances to Widescale Deployment — The claims are numerous. Some examples taken from [1] are given below:

- Mobile agents can continue roaming and working on behalf of users even when users are disconnected.
- Mobile agents can improve performance over the client/server approach, if specific conditions are met. They can move to the data source, process the data locally, and return with the results. Performance is improved when the code to be transferred (i.e., the mobile agent's code) is small in size compared to the number and length of messages exchanged in a traditional client/server approach. In the case study we focus on this very claim.
- Mobile agents make programming easy because they allow an intuitive and natural representation of many applications. It is, for instance, natural and intuitive to model electronic commerce applications as agents that roam from site to site.
- Mobile agents allow easy customization of applications. The client/server approach is too static to adapt to rapid changes. Mobile agents have so far been used in experimental settings. In our opinion, deployment has been hindered by two chief factors:

- The lack of maturity — Although the technology has been around for a few years, some technical issues such as security have not yet been solved in a satisfactory manner.
- The lack of interoperability standards. The Object Management Group (OMG) has produced a specification, known as Mobile Agent System Interoperability Framework (MASIF) [8]. However, most of today's mobile agent platforms do not support it. As a consequence, they cannot interoperate.

Mobile-Agent-Based Information Retrieval Applications

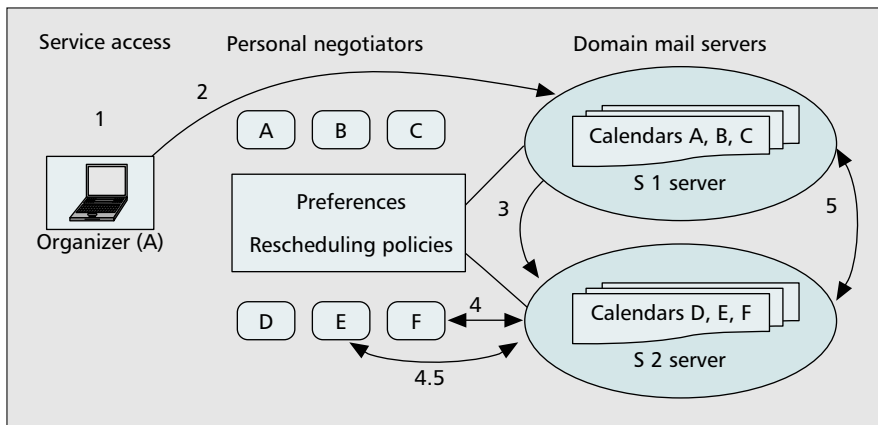
Scores of applications are based on information retrieval. Several attempts have been made to implement them using mobile agents. Some examples are given first. We focus on what the application does from the end user point of view. Studies made over the years to evaluate performance are reviewed second.

A Sample of Applications

Search Engines — An example is provided by [9]. A mobile agent consults a yellow page service to locate the servers that may contain a piece of information of direct interest to the end user. It then consults network-sensing agents to decide, depending on network conditions, if it should create "children agents" and send them to the servers, or if it should send a client/server request. It finally analyzes the results and presents the most pertinent information to the end user.

Telemedicine — Mobile agents can aid in retrieving the information needed to make a diagnosis. In the study made by Smith [10], mobile agents migrate to medical libraries that store mammograms. They identify and return the mammograms that are most pertinent to the case at hand.

Weather Forecast — Mobile agents can aid in personalizing weather forecasts, as shown by D. Johansen [11]. In the prototype he built, weather sensors supply an alarm server with the



■ Figure 1. A snapshot of the mobile scheduler scenario.

The Elaborate Case Study: Multiparty Events Scheduling Application

This section starts by stating the problem addressed by the case study. It then articulates what we expect to gain by using mobile agents.

The Motivating Problem

The key issues inherent to multiparty event scheduling are the availability of targeted participants and the availability of resources (e.g., a data projector). This article focuses on the first issue.

current weather condition data. The end user dispatches a mobile agent to the server. The agent periodically retrieves the weather condition data from the server, processes it, and sends an alarm to the end user when the conditions he/she has specified are met.

Performance Evaluation — The common characteristic of all the applications previously described is that a mobile agent is dispatched to a server, where it locally retrieves data. They all could have been built using the client/server paradigm, and the data would have been transferred over the network. This raises the question of comparative advantages. A few studies have been conducted over the years to compare mobile agents and client/server.

An interesting analytical study was performed by R. Jain *et al.* in [12]. The effects of wireless losses and the number of information servers are assessed as a function of the expected latency. The study indicates that both client/server and mobile agent models provide similar latencies if the search is done sequentially/randomly with no filtering. However, mobile agents reduce the latency by more than half when the search is done with filtering.

Two empirical studies are reviewed below:

- Reference [13] contrasts stationary approaches (e.g., client/server, local static agent to remote static agent) to mobile agents for distributed database queries. Network load measurements show that the mobile agent approach is the best option, when the number of queries is not a two- or several-digit figure.
- In [11], a mobile agent is dispatched to a server, where it locally retrieves MPEG videos in order to identify a specific video sequence. The response time is measured as a function of the data size. The measurements indicate clearly that the mobile-agent-based application outperforms its client/server counterpart for almost all the data sizes considered in the experiment.

In the rest of this article, we focus on empirical measurements. The key weakness of the measurements described in the literature is that the client/server application used as a yardstick is not optimized. This lack of optimization biases the assessment in favor of the mobile agent model. The coarse granularity used for the data downloaded by the client is the root of the problem.

In the measurements made in [11], for instance, the data could have been downloaded partition by partition, allowing the client application to stop downloading as soon as the sequence is found. We implemented such an algorithm in the client/server application we used as a yardstick in the case study presented in the next sections. This allows us to show that a mobile-agent-based information retrieval application can outperform its client/server counterpart, even when the latter is optimized.

This issue is subject to several constraints: participants' preferences, priority of the event, duration, and timeframe. The general problem we want to address is to identify the date and time slot where all targeted participants (or a quorum) are available, under the constraints previously mentioned.

The problem is complex due to its multifaceted nature (e.g., negotiation, rescheduling). Events with low priority may be rescheduled or even canceled to make space for events with higher priority. In intranet environments, existing solutions rely on electronic agendas shared through centralized calendar/mail management systems. Most are client/server-based (e.g., MS Outlook). These tools allow event organizers to download the calendars of potential participants in order to detect the time slots when the potential participants are available.

The detection is visual, which makes the process tedious and time-consuming. There is an obvious issue of scalability. When a large number of participants are involved, the setting up array provided to the event organizer (i.e., participants, dates, schedules) gets too big and unmanageable. Besides this, there is also a privacy issue. Participants need to release free access rights to their whole calendars to all potential event organizers.

These issues can be addressed by automating the processing of the information contained in the array downloaded by the event organizer. Preinstalled modules can download the calendars, process the information, and identify the date. Mobile agents can also be dispatched to the calendar servers, retrieve the information, and process it locally.

Benefits Expected from the Mobile Agent Approach

We first present a scenario to concretely show how single agents can be used to schedule multiparty events. The expected benefits are presented afterward.

A Scenario — Figure 1 depicts the scenario. The videoconference is organized by A, and the targeted participants are A (the organizer), B, C, D, E, and F. We assume that:

- A, B, and C reside in domain S1, while D, E, and F reside in domain S2. The electronic agendas are stored on domain servers, and there is a server per domain.
 - Each potential participant has a personal agent (on his/her host or in the network) who can negotiate potential rescheduling on his behalf.
 - The videoconference will take place in a given month and will last a whole working day.
 - The videoconference will take place if at least four participants are available.
1. A decides to schedule a videoconference. He gives the names and e-mail addresses of the targeted participants to the application, along with the month when the event should take place.

2. The mobile-agent-based scheduler connects itself to S1. It successively retrieves the calendars of A, B, and C for the given month, and processes the information in order to identify the days of the month where all three targeted participants are available. In this scenario we assume that a few days are found.
3. The scheduler moves to S2, then retrieves the calendars of D, E, and F. It behaves as above, then tries to figure out whether D, E, and F are available any of the days where A, B, and C are available.
4. The scheduler detects three dates that are suitable for D, but fails to find a date that satisfies E and F. It proposes these three dates to the negotiating agents of E and F. F could not reschedule any of his previous engagements, but E is able to reschedule one of his previous engagements.
5. The agent finally sends a notification with the meeting details to A, B, C, D, E because the quorum of 4 participants is reached, although F could not participate.

Expected Benefits — The five chief benefits we expect are briefly discussed below.

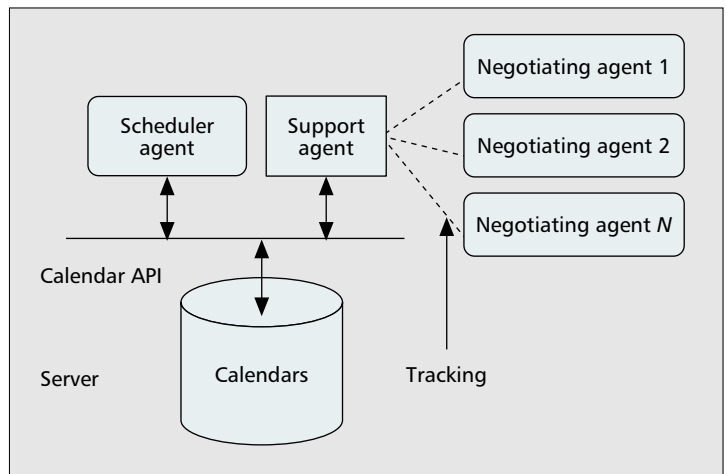
- *Performance*: As already stated, we expect our mobile-agent-based multiparty event scheduler to outperform its client/server counterpart, even if the latter is optimized.
- *Easy customization*: Several mobile-agent-based schedulers with slightly different functionality can connect to the same calendar server.
- *Device independence*: We expect to launch our application from any device, even if the device does not have sufficient memory capacity to store the calendars of all the targeted participants. A mobile-agent-based multiparty event scheduler can be launched from almost any device, especially since some of today's mobile agent platforms (e.g., Grasshopper [6]) run on small footprint devices.
- *Enhanced privacy*: The only information that should be given to an event organizer is the date(s) when all participants are available (if any). Existing tools have the weakness of making available to events' organizers the free/busy information contained in participants' agendas. Mobile agents can intermediate between organizers and participants' agendas.
- *Automation*: The scheduling of an event is a multistep process, as shown by the scenario we presented. All the phases should be automated, including the negotiations inherent to rescheduling. As shown by the scenario, this can be achieved by using agents that negotiate on behalf of end users, in addition to the mobile-agent-based scheduler.

Performance and easy customization benefits are the most critical benefits. A client/server approach cannot provide these two benefits at the same time. If, for instance, the schedulers are preinstalled on the calendar server for performance reasons, customization will be difficult because all the customized versions of the scheduler will need to be preinstalled.

Device independence, enhanced privacy, and automation can somehow be achieved with stationary approaches. Device independence, for instance, can be achieved by preinstalling the scheduling application on the mail server or on another server that can communicate with the mail server. Clients with memory constraints can send a request to the preinstalled application and get back the final results.

A stationary approach worth mentioning is the intelligent scheduler. In the recent past, the distributed artificial intelligence community has attempted to model and automate the scheduling of multiparty events, as evidenced by [14, 15]. Every potential participant has his/her calendar on his/her desktop.

Group meetings are scheduled with distributed multitask



■ Figure 2. High-level architecture.

processes. In most cases, heuristic strategies are used [16]. These attempts automate the process and enhance privacy. However, they tie the end user to his/her desktop. Device independence is not achieved, putting aside the large amount of data transferred over the network during the scheduling process.

The Elaborate Case Study: Prototypes

The mobile-agent-based prototype and client/server counterparts are successively presented.

Mobile-Agent-Based Multiparty Event Scheduler

The high-level architecture is presented first and the actual implementation second.

High Level Architecture — The prototype design relies on four chief assumptions:

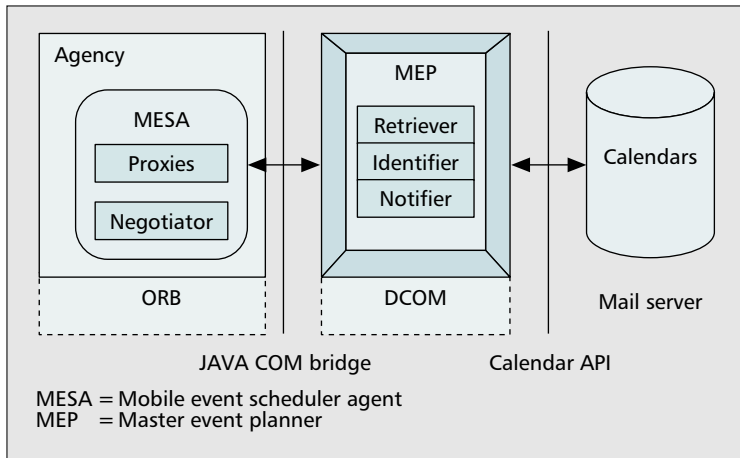
- The calendar server provides an application programmer interface (API) for accessing the information contained in the calendars. No assumption is made on the nature of the API.
- The calendar server is equipped with a platform to welcome and execute mobile agents.
- The calendar server hosts a scheduling support application. It is this application that reschedules a previously arranged meeting when required.
- The application is coded in Java since Java is the most widely used language for programming mobile agent applications.

The high-level architecture is depicted in Fig. 2. The main components are the scheduler agent, support agent, and negotiating agents. The support agent is fixed, keeps track of the negotiating agents that may be mobile, and mediates between the scheduler agent and the negotiating agents for rescheduling events. It also does the actual rescheduling.

Mobile negotiating agents can migrate to the server and stay there during the whole negotiation process. Although it could have been interesting to evaluate the performance of these agents against fixed negotiating agents, we have excluded this from the scope of this article in order to focus on the information retrieval aspects.

In the rest of this article we focus on the scheduler agent since our interest lies more in evaluating the performance benefit of a mobile agent for information retrieval than in designing a full-fledged multiparty event scheduler. The components of the scheduler agent are as follows:

- The user interface: It interacts with the event organizer to get the list of participants and the e-mail addresses. It also interacts with each participant to get the authorization to access the electronic agendas.
- The information retriever: It interacts with the calendar



■ Figure 3. Software architecture.

server to retrieve the information required to identify the dates where all targeted participants are available.

- The date identifier: It uses the information to derive possible dates and time slots.
- The date negotiator: It interacts with the support agent to negotiate rescheduling.
- The notification sender: It sends the schedule to the participants.

Implementation — The implementation has been limited to the scheduler agent. The main modules of the software architecture are depicted by Fig. 3:

- The mobile event scheduler agent (MESA): It includes the user interface, a proxy information retriever, a proxy date identifier, and a proxy notification sender. Proxies are used because the API that allows retrieval of the information is not necessarily in Java, and the information retrieved is not necessarily easily processed by Java code.
- The master event planner (MEP): It resides on the calendar server. It contains the information retriever and date identifier. The information retriever is programmed in a language in which the APIs supported by the server can easily be accessed. The date identifier is also programmed in the same language. This allows easy processing of the information retrieved from the calendars.
- The bridge allows mapping between the proxies and the real entities. It should be noted that there will no need for proxies if the API offered by the calendar server can easily be accessed from a Java program. In that case there will also be no need for an MEP residing on the server. The software architecture will be equivalent to the high-level architecture depicted in Fig. 2.

Prototype — The prototype is built in the MS Outlook calendar environment [17] and relies on the mail server of Ericsson Research Canada (Montreal). All the software components described above have been implemented. The proxy parts of the MESA are implemented as a fixed event scheduler agent (FESA). It is necessary to build a bridge and an MEP, since the calendar APIs are in Visual Basic [18] and cannot easily be accessed from a Java program.

The bridge is JIntegra-based [19]. JIntegra allows communications between the Java world and the MS Visual Basic world. It is used to generate the MEP Java proxy interfaces. It acts as a bidirectional JAVA COM bridge for communicating between the MESA and MEP real entities. Grasshopper 2.1 is used as a mobile agent platform, but the mail server itself is not equipped with Grasshopper for security reasons.

The MEP is installed on a separate and dedicated machine equipped with Grasshopper. It is a 400/100 MHz Intel Pentium II running Windows NT 4.0. The Outlook calendar API is used by the MEP to access the information contained in the dummy calendars referenced on the dedicated machine as a set of personal folder files, although the mail server is not equipped with Grasshopper.

Three small footprint devices running Windows CE 2.1.1 are part of the experiment: a NEC Mobile Pro 780 with a 168 MHz processor, and HP Jornada 680 and 690 models, both with a 133 MHz processor. These handheld devices are equipped with Personal Java 1.0 [20] and Grasshopper 2.1. The porting on PJava has been fairly easy. Three 400 MHz Pentium II laptops, running Windows NT 4.0, are also part of the experiment.

Client Server Counterparts

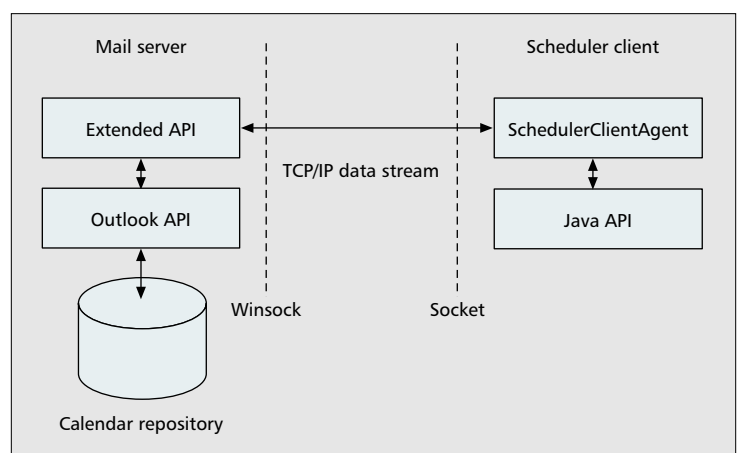
Two client server counterparts were built for performance evaluation purpose. The first is termed *plain* client server and relies on the same principles as the client/server applications built for performance evaluation purposes in the studies mentioned in the overview. The second is termed *optimized* client server because it optimizes the network load. This section describes what the two applications encompass and discusses how the optimized client server application is implemented.

Scope — The scheduling of a multiparty event is a multistep process. However, some of the steps are not pertinent to performance evaluation.

- In both mobile agent and client/server approaches, the notifications are sent using the same means, meaning e-mail or the MS Outlook meeting planner.
- As for our current mobile agent prototype, we have not put much effort in the rescheduling process.

We have restricted the scope to the steps that are meaningful for performance evaluation:

- Information retrieval: The information is retrieved online from the mail server, using remote procedure call (RPC), rather than locally on the MEP node as in the mobile agent approach. Performance in terms of network load will be significantly different.
- Date identification: In the plain client server approach, all the information needed to identify the date is retrieved and stored in collection items before the beginning of the identification process. In the optimized client server approach,



■ Figure 4. Optimized client server implementation.

the information is retrieved chunk by chunk, and the date identification algorithm is applied to a chunk at a time. Performance will be affected.

Optimizing the Client Server Application — Let us assume we have N targeted participants and M possible days. The algorithm goes as follows, if we use a daily calendar as granularity:

- Download at a time the daily diary for one participant.
- In the best case, the date and time slot suitable for everybody are identified after having downloaded the calendar of one day for the N participants.
- In the worst case, all the N calendars will be downloaded for the M days.

Even though we have chosen a daily calendar as granularity, other granularities (e.g., weekly or monthly calendars) could have been chosen, and the algorithm would basically remain the same.

Two hurdles have to be overcome in order to implement the algorithm using the MS Outlook API:

- Although the granularity of the API caters to the retrieval of daily calendars, the penalty in terms of response time is rather high. The *Restrict()* and *Find()* methods provided by the API use path expressions (e.g., *oneCalendar.Items.Find(filter)*) to access remote objects. The calendar object needs to first interact with its collection items before proceeding to find the subset of items that matches the filter criteria.
- The API does not permit connection to the calendars of more than one participant at the same time. This implies that it is required to reconnect to the calendar of each participant, whenever another daily calendar needs to be retrieved. Unfortunately, connecting to a calendar costs around 17 kbytes in our experience. This induces a rather high penalty in terms of network load.

To overcome these hurdles, we have extended the MS Outlook APIs, as shown in Fig. 4. The extended API is implemented in Visual Basic and mapped onto the MS Outlook API. Some of the methods provided by the extended APIs are listed below for illustration purposes:

- *Load_Calendars()*: Allow to load all involved calendars' collection items in memory.
- *Find_Participant_Daily_Diary()*: Find the subset of items with respect to a daily diary with a desired filter.
- *Send_Diary()*: Send the requested diary to the client application.

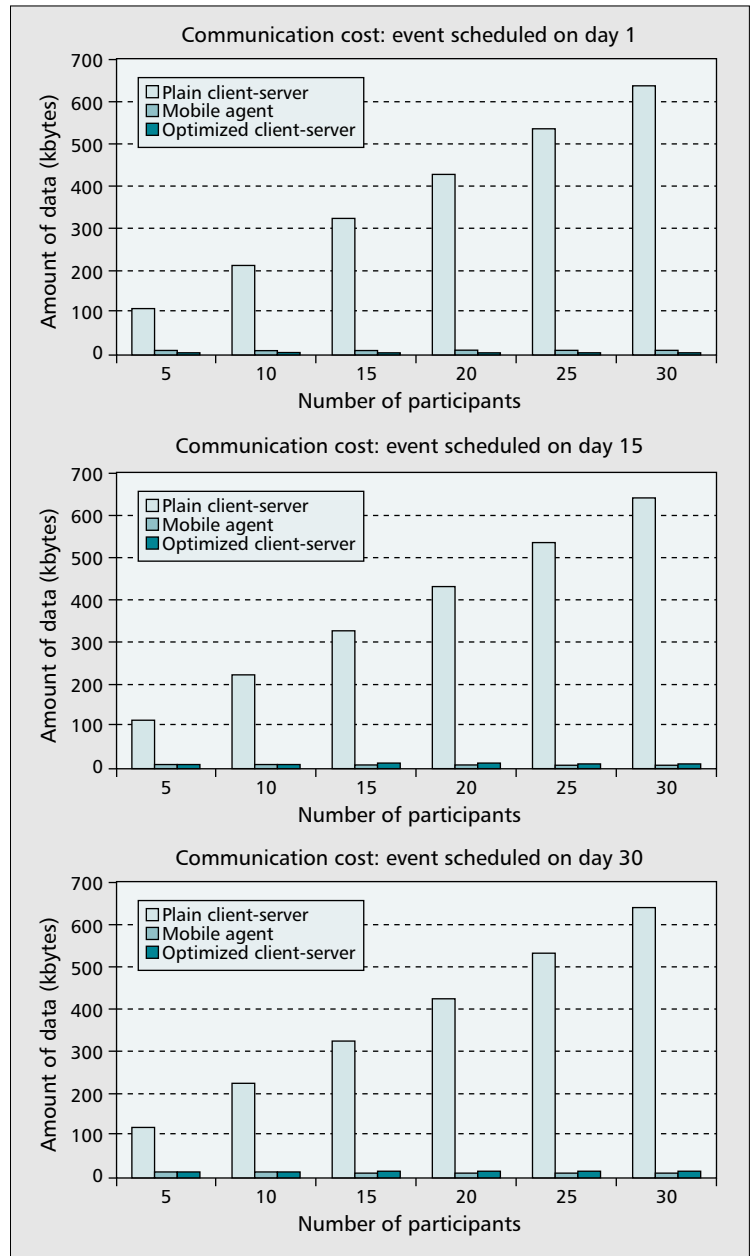
The Elaborate Case Study: Evaluation Against Client/Server

The measurements were performed at night because we did not have a dedicated and isolated network. Performing measurements at night allowed us to minimize the clogging and adverse factors. Our testbed workstations are connected to a 100 Mb/s Ethernet LAN segment. This section starts by introducing the metrics and measurement data. It then analyzes the data.

Metrics and Measurement Data

We have used two metrics: network load and response time. The network load measures the amount of data transported over the network during the scheduling process. The response time measures the duration of the process. These two metrics are influenced by three main factors:

- The number N of targeted participants
- The order I of the day where everybody is available with $1 \leq I \leq M$



■ Figure 5. Network load for the best case scenario ($I = 1$ st day; b) network load for the average case scenario ($I = M/2 = 15$ th day); c) network load for the worst case scenario ($I = M = 30$ th day).

- The density of the calendar of each involved participant
We have contrasted the mobile agent application, plain client/server, and optimized client server approaches assuming: $5 \leq N \leq 30$ and $M = 30$.

We have randomly populated the dummy calendars with none, one or two appointments of one hour scheduled during working hours (the appointment sample contains the subject “Busy time for test” and lasts 1 h). This was kept steady throughout the experiment.

The event duration was varied from 1 to 8 h since the user preferences were not too restrictive. These predefined preferences can be summarized as follows: “accept any event request coming from my mobile agent close user group, accept meeting for workdays (i.e., from Monday to Friday), accept events for regular working hours only (i.e., between 9:00 a.m. and 5:00 p.m.)”

- The network load is plotted as a function of the number of

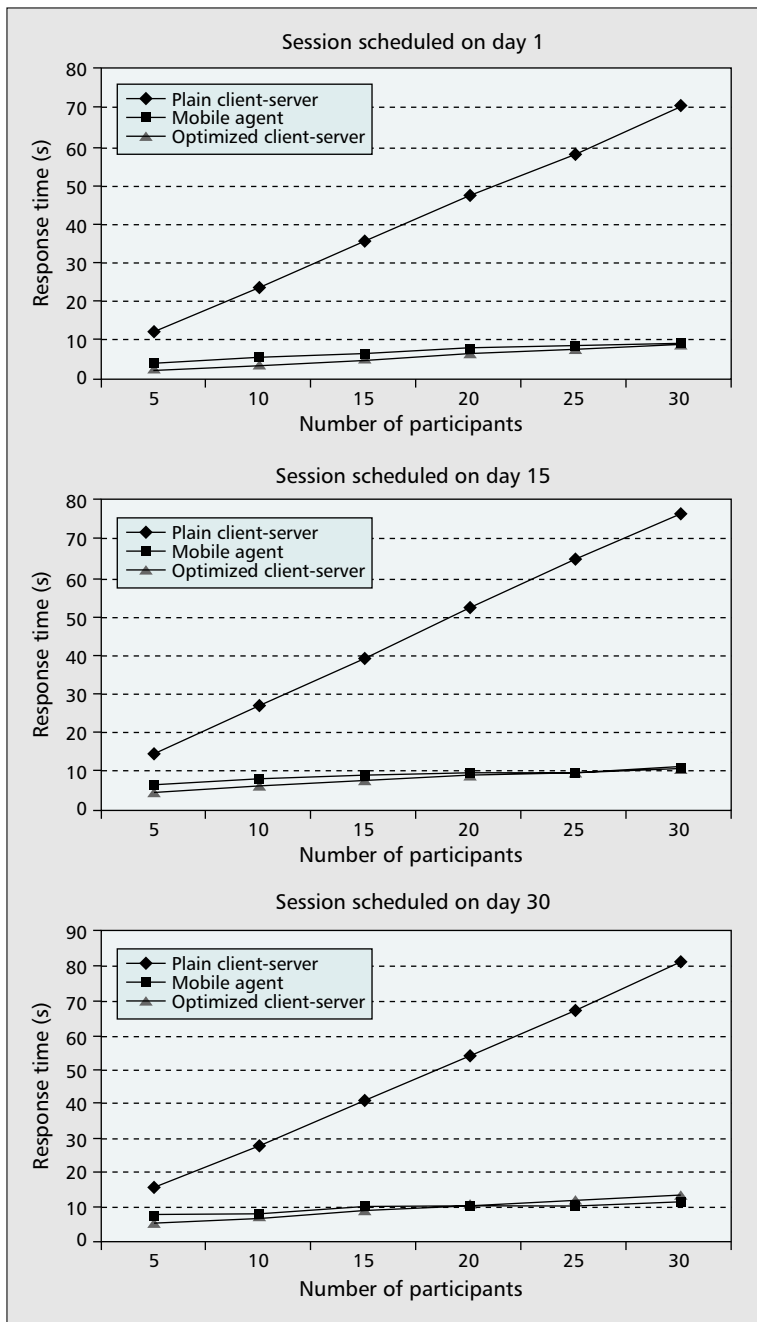


Figure 6. a) Response time for the best case scenario ($I = 1$ st day); b) response time for the average case scenario ($I = M/2 = 15$ th day); and c) response time for the worst case scenario ($I = M = 30$ th day).

targeted participants. Measurements are shown for the best case scenario (Fig. 5a, everybody is available on the first candidate day), the average case scenario (Fig. 5b, everybody is available on the 15th day), and the worst case scenario (Fig. 5c, everybody is available on the last candidate day).

- The response time is plotted as a function of the number of targeted participants. As for the network load, measurements are shown for the best case scenario (Fig. 6a), the average case scenario (Fig. 6b), and the worst case scenario (Fig. 6c).

Network Load Data Analysis

For the mobile-agent-based application, the load is constant, as shown by the three figures. It is roughly equal to the sum of the MESA size (8551 bytes) and FESA size (3658 bytes). The host at the destination downloads the FESA when it is

needed, and the data transported by the mobile agent (e.g., e-mail and IP addresses) is insignificant.

For the client-server-based applications, the load is linearly proportional to the number of participants, but the slope is much less steep in the optimized client server case. In fact, the load induced by the plain client server is independent of the order I , because the client always downloads all the agendas, and the agendas have roughly the same size.

The very low load induced by the optimized client server can be explained by the fact that we have managed to transport over the network only what is strictly necessary, thanks to our extended APIs. Logging onto a server calendar from a remote client, for instance, costs around 17 kbytes according to our experience. This logging is done locally with our extended APIs and does not induce any extra load on the network.

From the comparison standpoint, it can be concluded that the mobile-agent-based multiparty event scheduler outperforms by far its plain client-server-based counterpart in all scenarios. It also outperforms on average its optimized client server counterpart after a threshold is reached. As shown by Fig. 6b., this threshold is equal to the average number of participants (e.g., $N/2 = 15$), when everybody is available the day that corresponds to the middle of the sorted list ($I = M/2 = 15$ th day). It is important to note that although the optimized client/server does bring significant performance gains, easy customization remains out of its reach.

Response Time Data Analysis

As shown by Figs. 6a, 6b, and 6c, the response time is linearly proportional to the number of participants. However, the slope is much less steep in the mobile agent and optimized client server cases. The mobile-agent-based multiparty event scheduler outperforms by far its plain client-server-based counterpart in all scenarios.

This comes as no surprise since it is in line with the results obtained in the past for other applications [11, 13]. However, even in the average case, the threshold after which the mobile-agent-based approach starts outperforming the optimized client server approach is rather close to the maximum number of participants we have considered in the experiment. This is equally due to the performance gains brought by our extended API. However, as already stated, easy customization remains out of the optimized client/server's reach.

Summary, Lessons Learned, and Items for Future Work

This article provides a terse overview of mobile agents and their use for information retrieval. The overview is extended by an elaborate study of the information retrieval aspects of multiparty event scheduling. The overview introduces mobile agents, and reviews their use for information retrieval. The review indicates that quantitative studies are scarce. It also reveals that the key pitfall of the few existing empirical studies is the granularity used by the client/server counterpart to retrieve data. This granularity is too coarse and biases the comparison in favor of mobile agents.

A key objective assigned to the case study is to avoid this

pitfall. In the case study we motivate the multiparty event scheduling problem, present the benefits expected from the use of mobile agents, and mention the related work done in distributed artificial intelligence. The three prototypes we built were described. The first is the mobile-agent-based multiparty event scheduler, the second the plain client/server counterpart, and the third the optimized client server counterpart.

The optimized client/server did avoid the pitfall. The measurements made using network load and response time as yardsticks indicate that the optimized client server significantly outperforms the plain client/server. They also indicate that the mobile agent approach outperforms both plain client/server and optimized client/server. However, the gap between the optimized client/server and the mobile agent approach was not very significant in our experimental setting.

In our case study, we dispatch a mobile agent to a server in order to retrieve and process information locally, instead of first downloading the information to a client, then processing it. Our results are therefore transposable to most applications that rely on the same principle, namely information retrieval applications. This makes the lessons we have learned applicable to information retrieval applications in general.

The first lesson is that it may not be necessary to resort to mobile agents to improve the performance of information retrieval applications. Cleverly designed client/server applications may suffice. However, this design may go further than application design and may require extensions to the APIs offered by the server, as the case study shows. Furthermore, easy customization, which is easily achieved with mobile agents, will remain out of the reach of these cleverly designed client/server applications.

The second lesson is that mobile-agent-based applications can still beat these cleverly designed client/server applications. From the performance standpoint, we can conclude that mobile agents are suitable for information retrieval when performance is critical and the gains offered by the optimally designed client/server do not suffice. This will certainly be the case in the future when wireless and small footprint devices become ubiquitous. The mobile-agent-based scheduler and optimized client server scheduler we have built are now almost neck to neck when it comes to response time.

One way to reduce response time in an environment with several agenda servers is to dispatch several mobile agents in the network. However, the penalty in terms of network load needs to be minimized. This penalty depends on how many agents are dispatched in the network and how many servers each agent visits. Scheduling multiparty events with several mobile agents raises many interesting issues we would like to tackle in the future.

The first issue is the algorithm. It needs to be distributed. The second is the coordination model. Today's coordination models were recently surveyed [21]. It will be interesting to evaluate their suitability for the problem at hand. This suitability will depend, of course, on the distributed algorithm we would have designed. We will also go beyond multiparty event scheduling and build other case studies. Our ultimate goal is to use several of these applications as test cases for investigating the properties of information retrieval systems where several cooperating mobile agents are dispatched in the network.

References

- [1] D. Chess *et al.*, "Mobile Agents: Are They a Good Idea?" IBM Res. Rep. RC 19887 (88465), 1994.
- [2] A. Karmouch and V. A. Pham, "Mobile Software Agents: An Overview," *IEEE Commun. Mag.*, vol. 36, no.7, July 1998.

- [3] M.K. Perdikeas *et al.*, "Mobile Agents Standards and Available Platforms," *Comp. Net.*, vol. 31, no 19, Aug. 1999, pp. 1999–2016.
- [4] D. Kotz and R. Gray, "Mobile Agents and the Future of Internet," *ACM Op. Sys. Rev.*, Aug. 1999, pp. 7–13.
- [5] IBM Aglets, <http://www.trl.ibm.com/aglets>
- [6] IKV++ Grasshopper, <http://www.ikv.de/products/grasshopper>
- [7] Objectspace Voyager, <http://www.objectspace.com/>
- [8] OMG TC doc. orbos/97-10-05, "Mobile Agent System Interoperability Facilities Specifications," <http://www.omg.org>
- [9] B. Brewington *et al.*, "Mobile Agents in Distributed Information Retrieval," *Intelligent Information Agents*, Matthias Klusch, Ed., Springer-Verlag, 1999, Ch. 15, pp. 355–95.
- [10] K. D. Smith and R. B. Paranjape, "Mobile Web Agents for Telemedicine," *MATA'99, Mobile Agents Telecommun. App.*, Karmouch and Impey, Eds., pp. 405–17.
- [11] D. Johansen, "Mobile Agent Applicability," *Proc. MA '98: 2nd Int'l. Wksp. Mobile Agents*, Lect. Notes in Comp. Sci., no. 1477, Springer-Verlag, 1998, pp. 80–98.
- [12] R. Jain, F. Anjum, and A. Umar, "A Comparison of Mobile Agent and Client-server Paradigms for Information Retrieval Tasks in Virtual Enterprises, Research Challenges," *Proc. Acad./Ind. Working Conf.*, 2000, pp. 209–13.
- [13] T. Kawamura *et al.*, "Quantitative Evaluation of Pairwise Interactions Between Agents," *Joint Symp. ASA/MA 2000*, pp. 192–205.
- [14] S. Sen, "Developing an Automated Distributed Meeting Scheduler," *IEEE Expert*, vol. 12, no. 4, July/Aug. 1997, pp. 41–45.
- [15] S. Sen and E. H. Durfee, "A Formal Study of Distributed Meeting Scheduling," *Group Decision and Negotiation*, vol. 7, 1998, pp. 265–89.
- [16] A. Cesta and D. D'Aloisi, "Mixed-Initiative Issues in an Agent-Based Meeting Scheduler," *Int'l. Journal User Modeling and User-Adapted Interaction*, vol. 9 1/2, Apr. 1999, pp. 45–78.
- [17] Microsoft Outlook: <http://www.microsoft.com/office/outlook>
- [18] D. Gifford, *Outlook 2000 VBA, Programmer's Reference*, Wrox, 1999.
- [19] JintegraTM Java COM bridge: <http://www.linar.com/jintegra/doc>
- [20] Personal Java Spec., v. 1.1.3 : <http://java.sun.com/products/personaljava>
- [21] G. Cabri, L. Leonardi, and F. Zambonelli, "Agents for Information Retrieval: Issues of Mobility and Coordination," *J. Sys. Arch.*, 2000, pp. 1419–33.

Additional Reading

- [1] R. Jain and F. Anjum, "Mobile Agents for Personalized Information Retrieval: When Are They a Good Idea?," *2000 IEEE Wireless Commun. and Net. Conf. (WCNC)*, vol. 1, 2000, pp. 242–45.

Biographies

ROCH H. GLITHO [SM] (Roch.Glitho@ericsson.ca) holds M.Sc. degrees in business economics (University of Grenoble, France), pure mathematics (University of Geneva, Switzerland), and computer science (University of Geneva). He is currently pursuing a Ph.D. degree at the Royal Institute of Technology, Stockholm, Sweden. He works as an expert for the research branch of Ericsson. He is based in Montreal, Canada, where he leads research activities in service engineering. He moved to Canada in 1993 after having worked three years for Ericsson Telecom in Stockholm, Sweden. Prior to that he worked five years for a computer manufacturer in Oslo, Norway. His research interests include service engineering, mobile agents, Internet telephony, and network management. He is Senior Technical Operations Editor of *IEEE Communications Magazine* and a past Editor-in-Chief of *IEEE Communications Surveys & Tutorials*. He also serves as technical editor for the *Journal of Network and Systems Management*.

EDGAR OLOUGOUNA (edgar.olougouna@polymtl.ca) holds B.Eng. and M.A.Sc. degrees in electrical engineering (École Polytechnique, Montreal, Canada). He works as research assistant at the Mobile Computing and Networking Research Laboratory (LARIM), École Polytechnique. He has been an intern at Ericsson Research Canada for one year. His research interests include mobile agents, service engineering, Internet telephony, and security in electronic transactions for next-generations mobile systems.

SAMUEL PIERRE [SM] (samuel.pierre@polymtl.ca) is currently a professor of computer engineering at École Polytechnique de Montréal where he is director of LARIM. He is the author of three books, co-author of two books and five book chapters, as well as over 175 other technical publications including journal and proceedings papers. He received the best paper award of the Ninth International Workshop in Expert Systems and Their Applications, held in France in 1989. In 1994, one of these co-authored books, *Télécommunications et Transmission de données* (Eyrolles, 1992), received special mention from *Telecoms Magazine* (France). His research interests include wireline and wireless networks, mobile computing, performance evaluation, artificial intelligence, and telelearning. He is a member of ACM. He is an associate editor of *IEEE Communications Letters*, and he serves on the editorial board of *Telematics and Informatics* published by Elsevier Science.