SUPPORTING DISCONNECTED COMPUTING IN MOBILE AGENT SYSTEMS

PAULO J. MARQUES, PAULO S. SANTOS, LUÍS M. SILVA, JOÃO G. SILVA
CISUC, University of Coimbra, Portugal

pmarques@dei.uc.pt

ABSTRACT

Mobile agents are typically referred as a very adequate paradigm for using in mobile and disconnected computing environments. They are autonomous, have a strong sense of location and are able to operate independently of a central server. Nevertheless, although this is true, for developing a mobile agent system that is capable of operating in a disconnected computing environment, much more is needed: persistence stores where the agents can be safely stored; agent reactivation mechanisms, which allow agents to migrate when a network connection is detected; policies and notification mechanisms for dealing with changing locations; among other things. In this paper, we present the mechanisms developed for the M&M mobile agent framework in order to address mobile and disconnected computing environments. These mechanisms can be readily adapted to any other platform that needs to deal with disconnected computing.

Keyworks: Distributed agents, mobile agents, mobile computing, disconnected computing

1. INTRODUCTION

Over the last few years there has been a huge proliferation in mobile devices: laptops, palmtops, cell phones, and others. At the same time, all these devices are in a form or another capable of connecting to the Internet and of accessing and processing information. Much research is being made on how to program these devices so that they are able to interact with the network in a transparent way. For instance, mobile IP [1] envisions a way of allowing devices to communicate with the Internet independently of their IP, which can change because devices change location and attach themselves to different sub-networks.

Although all the current research and all developed methods, one central problem is the fact that the basic interaction metaphor being used in disconnected computing is exactly the same of the one used for fully connected devices: client/server interactions. And, as it is well known, that does not work very well in disconnected environments. Disconnected computing is much more than simple mobile computing. Whereas in mobile computing it is assumed that the devices change locations, in disconnected computing, the principle is that devices change locations and are only connected to the network once in a while. At the same time, while they are disconnected, they should be able to operate seamlessly.

Trying to use the client/server programming model for disconnected computing is a fundamental problem. Client/server assumes that the server is always these so that the invocations can be made. This is not true. While approaches like queued RPCs ease the problem, they do not solve it. A queued RPC may prevent an invocation from failing but the fact is that the invocation does not take place. And, as the programming model given to the developed does not change, the situation is troublesome. What is needed is a programming model that takes into account that the program running on the device is operating in a disconnected computing environment, acknowledges the fact that the location of the client program changes dynamically, and does not necessarily rely on a server for operation. Mobile agents provide this kind of programming model.

From the point of view of a developer, while developing a mobile agent, he is implementing an object that is capable of moving itself in an autonomous way. It is an active object with its own thread of execution, which does not depend on an external flow of execution to migrate between systems. At the same time, this object has a strong sense of location – it knows where it is and can be programmed to act accordingly; can probe the network, sensing if is it available or not; and is capable on interacting with local and external sources. As a metaphor for distributed computing in disconnected mobile environments, it is much more appropriated than client/server development. It does not mask the environment; it acknowledges it and gives the

programmer proper mechanisms for dealing with the problems involved.

While in theory mobile agent programming seams to leverage a much more appropriate programming model for disconnected computing, in real systems much more is needed than a simple generic mobile agent platform. Most of currently available platforms [2] were though for fully connected environments. In most cases, these platforms will not provide the kind of support needed for deploying agents in partially connected environments. For instance, many platforms depend on a central server which controls (and communicates) with all running agencies (i.e. agent servers). Obviously, this does not work if there isn't a network connection between the controlling server and the agencies. Other problem is that many platforms do not provide a way to store and reactivate agents transparently from persistent storage. This has serious implications. Just consider an agent that is running on a disconnected device. The agent may discover that there isn't a network connection available. But, what action should it take? Sleep for a while and then try again? That seams wasteful, and if the user shuts down the machine, the agent is lost. Terminate itself? Then, the information or action that it should perform is lost. As we can see, proper mechanisms for agent storage and reactivation are needed.

The fine point to retain from this discussion is that if an agent platform is going to be used to work in disconnected computing environments, generic support for mobile agents is not enough. A lot more is needed.

In this paper, we discuss the requirements that an agent platform must support for being capable of operating in disconnected computing environments. These requirements are discussed in terms of the lessons learned while extending the M&M mobile agent framework [3] for dealing with this kind of environments.

The paper is organized as follows. Section 2 discusses related work in the area; Section 3 describes the guidelines that an agent platform must follow for properly supporting disconnected computing environments; Section 4 presents the model and implementation work done on the M&M project. Finally, Section 5 presents the conclusions of the paper.

2. RELATED WORK

Agent TCL[1] [4] represents early work on supporting mobile agents in disconnected computing environments. In Agent TCL, the support for disconnected computing is implemented through a "docking system". There are two types of agent platforms: those which have an associated dock and those which do not. Every platform which operates on a mobile device has a master dock associated. The dock represents the device on the fully connected network. When an agent tries to migrate to a mobile device that is not connected, it is sent to the device's master dock. There, it is stored in disk. When the mobile device is again connected to a network, it notifies its master dock of its current IP. The agent can then be transferred to the mobile device.

Although the general model used in Agent TCL is quite powerful and adequate, a closer inspection of the approach reveals some shortcomings. The main shortcoming has to do with naming (the identification of mobile devices and their docks is based on the hostnames associated to the devices, which does not work when devices connect through ISPs) and programming model (for the system to work properly, the logic of migrating the agents is not transparent). The work discussed on this paper is based on the approach implemented in Agent TCL, but extended to be more general and transparent to the programmer. M&Ms also supports different disconnected computing models.

Magenta [5] is another system that supports disconnected computing. In Magenta there are two types of identities: place (*lieu*) and agent. A *lieu* is a location where an agent can reside. In Magenta, every *lieu* knows and communicates with every other *lieu*. When an agent travels its itinerary, if it cannot migrate to a certain *lieu*, it goes to the next *lieu*. It then notifies that *lieu* of the failed migration, which is propagated to every *lieu* in the system. Agents that need to migrate to mobile devices that are not available wait in the final *lieu* of wired network. When the *lieu* that was unavailable comes back to life, this information is also propagated to every other *lieu* in the system. The agents that have failed to travel to that *lieu* can then migrate there. Although this is an interesting solution, we believe it is not very scalable (too many updates being propagated through the network), it is not very transparent, nor its semantics clear (the agents have a fixed itinerary, but its order can be changed due to failed migrations).

The OnTheMove project [6] main objective is to create an Application Programmer Interface for use in multimedia mobile applications. The OnTheMove framework combines messages queues and mobiles agents and provides an extensive framework that includes directory services, events, security, replica management, quality of service, and other features. While it uses mobile agents, the main concern of the project seams to be adaptation to wireless networks and multimedia applications.

Finally, Concordia [7], SOMA [8] and MAP [9] are generic mobile agent toolkits that support disconnected computing. While for Concordia, besides the fact that it supports queuing of agents, there isn't much information available, the approach used in SOMA is similar to the one of Agent TCL. MAP uses the concept of a lookup server.

[1] Agent TCL is now called D'Agents.

3. SUPPORTING DISCONNECTED COMPUTING ENVIRONMENTS

For a mobile agent platform to properly support disconnected computing, several features are desirable. We will now briefly discuss some of the problems and issues that must be addressed. It should be noted that the list is by no means exhaustive. It just points some important points to take into account. Other problems, which are not discussed, include security, fault tolerance, agent tracking and inter-agent communication, among others.

Platforms should be independent of central servers

Currently, many mobile agent platforms depend directly of the availability of a master platform. This master is responsible for things like agent tracking, agent monitoring, acting as a code repository for the agents that are migrating (i.e. in many systems, while the state of the agent migrates between nodes, the code is fetched from a central server), authentication, and so on.

The problem is that although mobile agents seam to be a completely decentralized programming paradigm, in practice, many desirable features are more easily implemented by having one or more central servers. The agents themselves may be decentralized, but typically the supporting infrastructure is not.

If an agent platform is to be used in disconnected computing environments, the infrastructure of the platform must be much more peer-to-peer than it is in most of current systems. It is certain that this brings some problems in areas like agent tracking and monitoring or inter-agent communication. Nevertheless, if a device is disconnected and has agents in it, the infrastructure must be prepared to deal with it, and make that information explicit in the distributed system.

Agent storage and reactivation

If an agent is in a mobile device and tries to migrate, it may found that there isn't a network connection available or that the target server is not present. Either way, in generic platforms, the problem is reported to the agent, which is responsible for deciding what to do. While this is appropriate in the case where there is a genuine network problem or a problem with the target host, in the cases where the device was voluntarily disconnected from the network and is operating on its own, it is not. In that case, the agent should be transparently queued until a network connection is available, at which time it should be migrated.

The problem here is not the fact of being the agent taking the decision. In fact, this is a desirable feature most of times. The problem is that, if a developer is programming a mobile agent for working in disconnected computing, the queuing operation should be transparent to him. The programmer should not be forced to hand code the logic necessary to passivate an agent each time it cannot migrate due to the lack of a network connection. The programmer should be given the option to hand code that kind of situations, but that should not be the default case.

At the same time, many agent platforms lack a persistent storage facility with the necessary features for disconnected computing. The persistent storage facility should be able to intercept a migration that is bound to fail due to the fact that a mobile device is disconnected from the network, and store the involved agent. It should also be able to autonomously detect the presence of a network connection and transfer the agent, as if the original migration was taking place. Many platforms lack such kind of persistent storage, and other platforms which support similar facilities, force the programmer to hand code the logic for agent storage and reactivation in the agent. The most inadequate platforms are those which don't even support agent storage and, when used in this type of environments, force the programmer to put the agent to sleep for a while, and try back from time to time. Obviously, if the host is disconnected; the agent platform terminated; or event if it simply crashes, the agents are lost. An agent storage protects agents against this kind of failures [4,7].

Programming model

As it was discussed in the previous points, the programming model given to the developer should clearly address the fact that agents are being used in disconnected computing environments. This means that the programmer should not be forced to hand code logic for dealing with that fact simply because the necessary features were not included in the basic infrastructure. Nevertheless, the programming model should give the developer the ability to hand code some of those behaviors. For instance, although in the default case the agents should be transparently stored and queued for transmission when a network connection is not available, in some cases, the developer may which to customize that behavior. For instance, a personal information manager agent that accompanies a user on his laptop, having detected that there isn't a network connection available, so that it can migrate to the desktop computer of the user, may ask the user to establish a connection, or even offer to do it for him.

Another important point is that if a system allows an agent to be routed through several servers and eventually be reactivated on a different server from the normally expected by its itinerary, then there should be a clear way to immediately notify the agent on reactivation that it is not on the location that it was expecting to be. This is a very important point, since it can be the source of many bugs, because an agent's logic may implicitly be counting on a certain itinerary.

This list does not enumerate all the problems, but should alert to the fact that simply having mobile agents and a mobile agent platform does not mean that it can be readily used for disconnected computing. Some important features

must be built into it. This is a point that normally is not mentioned on the literature while discussing mobile agents and that is of vital importance.

4. DISCONNECTED CUMPUTING IN THE M&M AGENT FRAMEWORK

4.1 OVERVIEW OF M&M

Over the last few years, we have been working on a mobile agent framework called M&M. The most distinctive characteristic of M&M is that there are no agent platforms. Instead, agents arrive and leave from the applications they are part of, not from agent platforms. The applications become agent-enabled by incorporating well-defined binary software components [10] into their code. These components give the applications the capability of sending, receiving and interacting with mobile agents. The applications themselves are developed using the current industry best-practice software methods and become agent-enabled by integrating the mobility components. The main benefits from this approach are:

- It is not necessary to design the whole application around agents. Agents are sent back to middleware, in pair with other distributed programming technologies.
- Security is integrated with the application security framework, rather than being completely generic.
- Agents interact directly with the applications from the inside. This eliminates the need to setup interface agents and configure/manage their security policies.
- There is no agent platform to install and maintain. Although there are still distributed applications to install and manage, this is much simpler than managing a separate infrastructure shared by a large number of distributed applications with different policies and requirements.
- The end-user sees applications, not agents. In this way, the acceptance of applications that use mobile agents is increased since what the end user sees is the added value functionality, not the agents.
- It is simple to program. The programmer only needs to visually drag-and-drop the necessary components from a component palette, and to configure their properties and interconnections.

The M&M framework was implemented using the JavaBeans component framework, and is centered on the so-called *Mobility Component*. This component provides the basic support for agent migration and management, and an extensibility mechanism that allows other components to connect to it. These other components may implement functionalities like different inter-agent communication mechanisms, security, persistence and others.

One very important aspect of the extensibility mechanism is that it is based on an event model – `AgentLifecycleEvents`. After a high-level service component has registered with the Mobility Component, it is notified whenever some state transition occurs in an agent. For instance, when an agent arrives, there is an `onAgentArrivalEvent`. Every listener is able of examine the agent, interact with it, and can veto the corresponding event. As an example, consider the security component. On being notified that an agent is arriving, it can verify its credentials and examine its state. If it finds the agent not to be trusted, it can veto the event, prohibiting the agent from arriving.

The complete discussion of the M&M project and its framework is beyond the scope of this paper. Interested readers can refer to the project documentation and papers, available at *http://mm.dei.uc.pt*.

4.2 GENERAL CONSIDERATIONS

The disconnected computing support implemented in M&M follows the guidelines discussed on Section 3. First, in M&M there isn't the notion of a central server. Each application that is agent-enabled with M&M components is capable of operating in isolation, and communication between applications is done on a truly peer-to-peer fashion. Although for some services, like inter-agent communication and agent tracking, there is the notion of a central server (which can be replicated), these components are not required for general operation and are prepared to deal with not being able to contact or being contacted by a certain application. (The approaches vary from registering that the agents/applications are in an unknown state; to keep trying to contact the applications; to report an immediate failure on the delivery of a message; and to perform queuing of messages. It all depends on the component.)

The M&M framework was also extended with a persistence storage component that allows agents to be queued, reactivated and transferred when certain events happen, such as a network connection becoming available.

Finally, because the programming model of M&M's agents is based on events, it was quite easy to make sure that agents are properly notified if they became to execute in a place where they might not expect to be executing. This is so because in the disconnected computing support implemented in M&M, we allow the agents to be rerouted to different servers, if their owners decide so. In terms of programming model, all the queuing, storage and retransmission of agents takes place transparently. Nevertheless, the programmer has the ability to customize the default behavior by intercepting the relevant events that each component propagates, and implementing its own code.

We will now discuss the disconnected computing model implemented in M&M.

4.3 DISCONNECTED COMPUTING IN M&M

One of the simplest models available in M&M for disconnected computing is depicted in Figure 1.

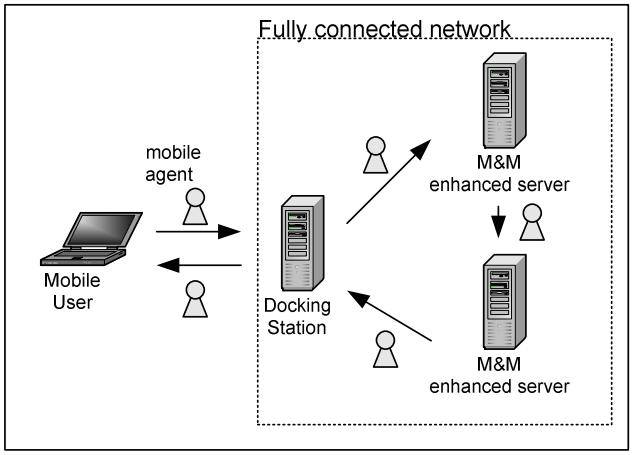

Figure 1 – Simple support for disconnected computing

There are some servers on the fully connected network called docking stations. Each mobile device can be registered in one or more docking stations[1]. A docking station is responsible for storing agents while the devices are not connected, and sending them to the devices when they become available.

Whenever a device wants to use disconnected computing, or an agent whishes to make use of those facilities, it sets a flag on the header of its migration protocol. When the agent passes through the docking station, which serves as its point of contact with the fully connected network, the docking station detects this flag and knows that it is responsible for queuing and sending the agent to the particular mobile device from where it is originating. The programmer has the ability to manually specify a different docking station to be used. Even if an agent wants to use disconnected computing facilities, but it does not pass first through a docking station, a docking station can be made responsible for the agent, as long as the mobile device manually registers with a certain docking station, and the agent passes through it on its way home.

Note that a docking station can be responsible for many agents and for many mobile devices. Each docking station must maintain a mapping between the identification of a mobile device and its mobile agents that require disconnected operation. Also note that the identification of mobile devices is not based on their hostname, but on a unique identifier generated for each machine. Finally, if a agent tries to migrate from a device that is disconnected from the network, queuing can also occur on the device side, as long as, the persistent storage component was included on the client application.

Whenever a mobile device becomes connected, it notifies the docking stations where it is registered that the device is online and of its current location (IP address and port). An authentication mechanism is used so that malicious users do not try to impersonate as a certain mobile device.

Every agent that is stored on those docking stations is then transferred to the mobile device. Whenever a mobile device goes offline, or if a docking station is unable to contact it after a certain time, the mobile device is considered to be disconnected. Any agents that pass through the docking station and try to migrate home are stored until the mobile device becomes online. The M&M API also allows the usage of leases so that agents are not stored in docking stations forever.

While the agents are roaming the fully connected network, they have no restrictions. The only requirement for the system to work properly is that agents migrate to a docking station before they try to migrate to a mobile device.

M&M supports some other models of disconnected computing. One other interesting mode of operation allows the clients (mobile devices) to change docking stations and to use several docking stations simultaneously.

Many times, for performance reasons and others, like changing ISPs or traveling to a distant country, it is useful to be able to change the docking station that is going to be used by the mobile agents. If a mobile device wants to change its docking station, it starts by registering itself with the new docking station and by giving it indication that it should be able to queue and route agents belonging to it. After that, it notifies all the other docking stations where it is registered that its agents should be sent to the new docking station.

All the agents that are persistently stored on those docks are then forward to the new dock. Also, any agents that are rooming the network and arrive at an old dock are forward to the new dock. Since it is possible that cycles occur, if a cycle is detected due to successive transfers between docks an action is taken. Currently, three actions are supported: a) reawake the agent, notify it of its current location and that it is being routed in cycles; b) terminate the agent; c) store the agent and wait to be contacted by the mobile device.

Notice that although the mobile devices notify existing docks that all agents should be forwarded to a new dock, the existing device is still registered with those docks. The mobile device is still responsible for deregistering from each of the docks whenever their services are no longer required. Alternatively, it is possible to specify a maximum lease time for which a dock maintains the information concerning a certain mobile device.

It should also be noted that in every case were an agent is reawakened, even in a mobile device, it is notify of its

[1] Registration on a docking station is only allowed to the users and devices with proper permissions, being those integrated in the security framework of M&M.

current true location (mobile device real name or host name). This is especially important since mobile devices tend to change their name and IP dynamically and it is possible that reroutes occur that are not specified in the logic of the agent nor in its itinerary.

4.4 SAMPLE APPLICATIONS

For demonstrating the concepts discussed, two applications have been implemented. One is a revision control system that is based on mobile agents. Agents carry a briefcase which can contain committed and uncommitted documents. Agents resynchronize on connection.

Another developed application was a Personal Information Manager (PIM). This PIM carries information concerning its owner, like his list of contacts, calendar, email accounts and others. The PIM is always present on the same computer where his owner is, and is able of migrating between computers. The disconnected computing support is fundamental for when the agent is running on a laptop and is ordered to migrate home and the other way around. A screenshot of the PIM is shown in Figure 2.

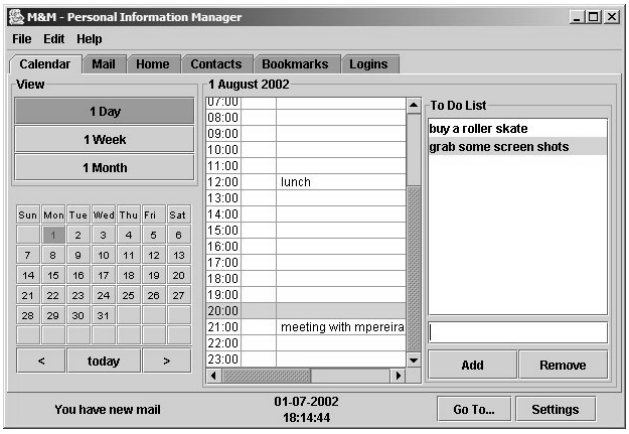

Figure 2 – Screenshot of M&M PIM that uses the disconnected computing features of the framework

5. CONCLUSION

In this paper a twofold discussion has been presented. First, we addressed the minimal requirements that mobile agent platforms should support for correctly enabling disconnected operation, and why those requirements are stricter than what is commonly found on off-the-shelf agent platforms. The main points are: basic independence from central servers; proper persistence storage and reactivation mechanisms; and transparency in terms of the programming model given to developers.

In the second part of the paper we have presented the M&M framework and described two of the disconnected computing models provided by the framework. These models follow the guidelines previously discussed.

Although the paper discusses the particular implementation done on M&M, the discussion is general and the solutions and observations made can be applied while designing any mobile agent platform that supports disconnected computing.

ACKNOWLEDGEMENTS

This investigation was partially supported by FCT, through the M&M project (project reference POSI/33596/CHS1999) and CISUC (R&D Unit 326/97).

REFERENCES

[1] J. Solomon, “Mobile IP the Internet Unplugged”, Prentice Hall, ISBN: 0138562466, 1st ed., 1998

[2] The Mobility List, available at: *http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html*

[3] P. Marques, L. Silva and J. G. Silva, “M&M’s: Building Binary Software Components for Supporting Mobile-Agent Enabled Applications”, in *Journal of Autonomous Agents and Multi-Agents Systems*, vol. 5(1), Kluwer Academic Publishers, Netherlands, 2002

[4] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla and G. Cybenko, “AGENT TCL: Targeting the Needs of Mobile Computers,” in *IEEE Internet Computing*, vol. 1(4), 1997

[5] A. Sahai, C. Morin, “Mobile Agents for Enabling Mobile User Aware Applications”, in *Proc. Autonomous Agents 98*, Minneapolis, USA, 1998

[6] E. Kovacs, K. Rohrle and M. Reich, “Mobile Agents OnTheMove - Integrating an Agent System into the Mobile Middleware,” presented at *ACTS Mobile Summit 1998*, Rhodos, Greece, 1998

[7] T. Wash, N. Paciorek, and D. Wong, “Security and Reliability in Concordia,” in *Proc. of the 31st Annual Hawai International Conference on System Sciences*, Hawai, USA, 1998

[8] P. Bellavista, A. Corradi and C. Stefanelli, “A Mobile Agent Infrastructure for the Mobility Support”, in *Proc. of the 2000 ACM Symposium on Applied Computing (SAC’2000)*, Como, Italy, 2000

[9] O. Tomarchio, L. Vita and A. Puliafito, “Nomadic users' support in the MAP agent platform”, in *Proc. of the 2nd International Workshop on Mobile Agents for Telecommunication Applications (MATA’2000)*, Heidelberg, Germany, 2000

[10] Sun Microsystems Inc, “JavaBens Specification 1.01”, available at: *http://www.javasoft.com/beans*