Mobile Agents: An Introduction

Gian Pietro Picco
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
picco@elet.polimi.it

Abstract— **Mobile agents are enjoying a lot of popularity as a novel abstraction for structuring distributed applications. In this paper, we provide an introduction to the related research field by showing evidence of the benefits mobile agents can potentially achieve, illustrating the foundations of architectures and technologies for mobile agents, and discussing some of the open issues still hampering a wider acceptance of this paradigm.**

Keywords— **Mobile agent, mobile code, design paradigm.**

I. Introduction

Conventional distributed systems typically assume a static configuration of the environment where the distributed application executes. Communication among a set of hosts is enabled by physical links whose configuration is fixed and statically determined. Similarly, the various portions of the distributed applications that run on the nodes of the system are typically bound to such nodes for their whole life. Hence, the topology of the system, both at the physical and logical level, is essentially assumed as fixed.

This view is being challenged by technical developments that introduce a degree of *mobility* in the distributed system. Some forms of mobility are already evident to the general public, through the increasing pervasiveness of cellular phones, wireless LANs, and Internet mobile access. Wireless networking introduces a form of *physical mobility* by enabling untethered communication among hosts, even while they are moving within the physical space.

At the same time, a less evident but equally revolutionary form of mobility is reshaping the logical structure of distributed systems, by providing a fluid software fabric in which the components of an application can dynamically change their location. This form of *logical mobility*[1], often called *code mobility*, allows for the code and possibly also the state of an executing program to be migrated, in part or as a whole, at run-time. Proposals vary according to the granularity of the unit of mobility and the modality of relocation. Variants exist where the code and state of an executing program are always migrated together, and the decision about migration is taken autonomously by the program itself. This latter kind of roaming application component is usually termed a *mobile agent*.

The key conceptual contribution of mobile agents, and more in general of code mobility, is to raise the location where an application component is executed from the status of configuration or deployment detail to that of first-class element in the application design. This change of perspective has the potential for inspiring a new breed of concepts, models, and, ultimately, technologies that will shape the next generation of distributed computing.

[1]The reader interested in the relationship between physical and logical mobility, and related research issues, is redirected to [1].

The goal of this paper is to introduce the reader to the research field concerned with mobile agents. This goal is achieved by presenting the conceptual foundations, that have their grounds in logical mobility at large, and by reviewing the state of the art.

The paper is structured as follows. Section II presents the rationale for using mobile agents, and hints at why and when mobile agents are preferable over other solutions. Section III reviews the basic architectural paradigms for code mobility, including mobile agents. Section IV contains a critical discussion of the mobile agent technology currently available. Section V presents reflections on the present and the future of the research area. Finally, Section VI contains the concluding remarks.

II. The Case for Mobile Agents

A question that is often a subject of debate is what can be done with mobile agents that cannot be done with conventional technology or, in other terms, whether a "killer application" exists for mobile agents.

This question appears to be ill-defined. Since the inception of this research area, Harrison et al. [2] pointed out, later supported by other researchers, that any functionality that can be implemented with mobile agents can also be implemented with conventional technology. Thus, the question of whether a killer application exists is an irrelevant one. Mobile agents should be considered only as another tool in the arsenal of the designer of distributed applications.

It is true, however, that some application domains are more amenable than others to mobile agents or, in general, code mobility. Thus, the key point is to understand precisely *what* are the potential advantages and benefits the mobile approach may provide over conventional technologies, and *when*, i.e., under what conditions, these benefits can be effectively achieved.

In the following, we analyze a couple of success stories available in literature, and then draw some general considerations about the use of mobile agents.

A. Mobile Agents for Database Access

Papastavrou et al. [3] tackled the problem of enabling access to databases on the Web. A conventional solution exploits the Java Database Connectivity (JDBC) [4] API for enabling the client (a Web browser) to access and manipulate a relational DBMS. To use this API, a client must first download the JDBC driver, containing the interface

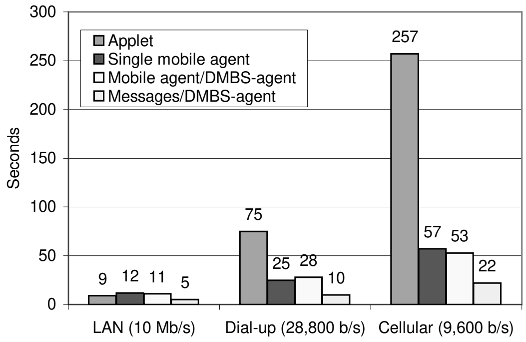

Fig. 1. Comparing the latency generated by short DBMS transactions on the Web. (Adapted from [3].)

to the remote DMBS. Such driver is typically downloaded dynamically as part of the Java applet that constitutes the application front-end. This effectively makes a Web browser an application-specific DBMS client, without any user intervention.

Nevertheless, there are a couple of problems with this approach. First, the setup phase, that involves the download and initialization of the JDBC driver, is typically a very resource-consuming and slow procedure. Second, the Web client is no longer lightweight (as Web interaction would require), rather it becomes more similar to a full-fledged, data-aware LAN client.

The authors propose the use of mobile agents to obviate to the aforementioned problems in the following way. Upon connection to the Web site, the client downloads the application front-end, but instead of downloading the JDBC driver, it receives a small program—the code of the mobile agent. When the user specifies a query through the front-end, this code is used to spawn a mobile agent that contains the query and is shipped to the target DBMS, where the JDBC driver is loaded and initialized for the agent. At this point, the mobile agent is co-located with the DBMS and can exploit *local* interaction to submit the query and collect the results, before returning to the client. This solution is likely to improve bandwidth usage and the latency perceived by the user, by avoiding the driver download and by exploiting local interaction with the DBMS.

Two enhancements to this idea are also considered in [3]. The first one stems from the observation that a JDBC driver must be loaded and initialized at the DBMS server for each mobile agent received. To optimize execution for a stream of mobile agents belonging to the same application,

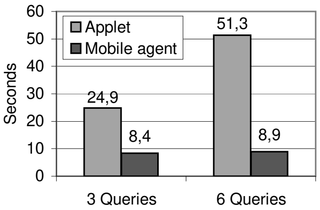

Fig. 2. Comparing the latency generated by Web queries executed in parallel on multiple nodes. A fixed 10Mb/s network is considered. (Adapted from [3].)

the authors propose the use of another mobile agent, that is spawned as soon as the client submits the first query. This second agent, called a DBMS-agent, is dispatched to the DBMS server where, after its JDBC driver is loaded, it remains "parked" for the duration of the application. The DBMS-agent acts as a proxy for the mobile agents carrying the client queries. These agents do not need to load JDBC drivers; instead, they act as messengers, by delegating the invocation of the query to the already JDBC-enabled DBMS-agent, from which they collect the results that are brought back to the client. Clearly, this solution speeds up the execution of queries after the first one, by reducing the time spent on the DBMS to initialize the agents for JDBC.

Finally, a second variant further refines the previous one by using messages, rather than mobile agents, to carry the query from the client front-end to the DBMS-agent parked at the DBMS, and for returning the results.

A comparison between the conventional JDBC solution and the three variants described that exploit mobile agents is allowed by the experimental results provided in [3]. The improvements brought by mobile agents are remarkable. Figure 1 shows the latency perceived by the client for a short transaction made of three queries. The combination of a DBMS-agent and messages leads to transactions that are about two times faster on a fixed Ethernet network, and up to ten times faster on a GSM cellular link.

Similarly remarkable results hold for access to multiple DBMS, as shown in Figure 2. In this case, the ability of mobile agents to be dispatched concurrently towards different nodes, and back to the source, determines an improvement over a sequence of JDBC queries performed by the client.

B. *Mobile Agents for Network Management*

Baldi and Picco [5] evaluated whether and how mobile agents, together with other forms of logical mobility, can be useful in the field of network management. Mainstream network management architectures, like the Simple Network Management Protocol (SNMP) [6], are still largely centralized. A network management station (NMS) operated by the manager, polls data from the network devices, and subsequently performs all the computation. To make matters worse, the primitives available to poll data are extremely fine-grained; thus, a high number of interactions retrieving low-level data are needed in order to reconstruct the desired high-level information. Clearly, this centralized architecture poses a tough challenge to scalability. In particular, it tends to cause congestion in the NMS neighborhood, thus paradoxically complicating the task of managing the network.

In their work, the authors build an analytical model of a generic network management task, and compare the traffic generated by plain SNMP against the one generated by mobile agents. In their solution, the set of SNMP requests is encapsulated in a program that is able to migrate autonomously from one managed device to another and gather the relevant results, that are returned to the NMS with a single message.

The results are interesting, in that they show clearly how mobile agents can be beneficial or detrimental depending on the application requirements and design goals. For instance, the idea of a mobile agent that simply visits each node and collects the relevant data does not seem to be a good one, as shown[2] by Figure 3 from the point of view of overall traffic reduction. If all the data gathered must be returned to the NMS, the agent must carry at each hop the whole amount of data collected at that point. Thus, the overall traffic generated by the mobile agent solution increases much more rapidly than the one generated by the client-server solution exploited by SNMP.

On the other hand, the ability to relocate a computation directly on the data source opens up the possibility to perform *semantic compression*, i.e., to reduce the amount of information being transmitted by filtering it at the source, based on its content. Thus, for instance, it is possible to find out what is the most loaded interface on a router by performing the necessary SNMP queries while co-located with it, rather than by transmitting all the necessary data back to the NMS. This particular form of semantic compression local to a device, may not be sufficient to overcome the state growth problem depicted above, as shown in Figure 3. Mobile agents able to roam multiple devices may provide a form of *global* semantic compression, where the filtering takes place not only locally to a device, but also across multiple devices. Thus, for instance, the most loaded interface in the network could be computed by retaining, at each hop, only the current maximum value, hence eliminating the state growth. This last possibility may provide a significant reduction in the overall management traffic (up to 30% for the data set shown in Figure 3).

Nevertheless, as mentioned before, the overall management traffic is only of marginal importance. To overcome the limitations of the centralized SNMP architecture, the main priority is to reduce the traffic around the NMS—a task mobile agents turn out to be extremely effective at. While SNMP requires a pairwise interaction between the NMS and each managed device, mobile agents once unleashed can visit the devices autonomously, without requiring any communication with the NMS until all the results have been collected. Thus, no matter how many devices are visited by the mobile agent, the NMS is involved only in the initial dispatching of the agent and in the final collection of results, while the remaining traffic is steered by the mobile agent away from the NMS. This fact alone leads to a significant traffic reduction even when no semantic compression is performed, as shown in Figure 3.

Clearly, mobile agents unveil their full potential when the benefit of reducing the traffic around the NMS is combined with the possibility of performing semantic compression. In this case the improvement over the plain SNMP client-server scheme is outstanding, as witnessed by Figure 3. The

[2]For the sake of presentation, the chart relies on some simplifying assumptions. Requests and replies are assumed to have all the same size (50 and 100 bytes, respectively), the size of the migrated code is assumed to be always 2 Kbytes, and the number of queries per node is assumed to be 20. See [5] for a discussion of how the traffic expressions are determined.

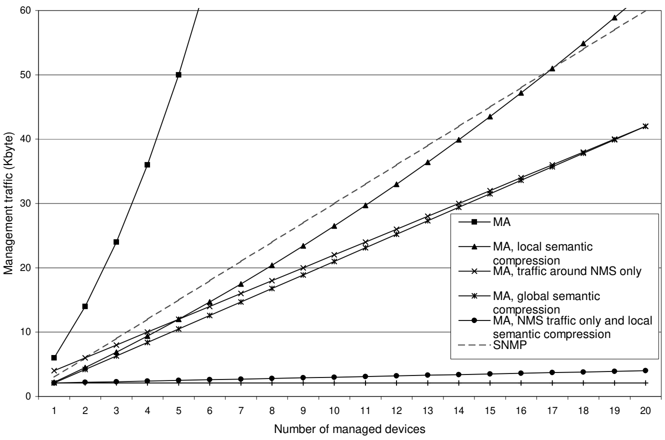

Fig. 3. Plain SNMP vs. mobile agents.

traffic generated around the NMS not only becomes negligible with respect to the one generated by SNMP (2.1 Kbytes instead of 60 Kbytes, for a traffic reduction of over 95%), but also constant with respect to the number of nodes.

C. Commonly Agreed Benefits of Mobile Agents

The applications discussed thus far highlight many of the benefits of mobile agents. Several researchers have tackled the problem of finding out what are the potential assets of mobile agents (see for instance [2], [7], [8]). By and large, mobile agents, and more generally code mobility, are acknowledged to bring two main advantages over conventional, client-server based technologies:

Enhanced Flexibility. Clients typically access the resources hosted by a server through a predefined set of services, whose interface is predefined and commonly agreed among the client and the server. It is usually complex, if at all possible, to change dynamically this interface to encompass new services, or to adapt the existing ones to previously unforeseen needs. Mobile agents can be used to update dynamically the interface on the client and/or server side. For instance, this technique is used by the application discussed in Section II-A to change dynamically the interface of the DBMS server, and allow the client to interact with it without requiring JDBC.

Reduced Bandwidth Consumption. The ability to migrate a client program to achieve co-location with the resources it must access on the server reduces the need for remote communication and thus may enable, under some conditions, a more efficient use of the communication link. In particular, *semantic compression* may often be achieved, as discussed in Section II-B.

Moreover, several other advantages have been identified, including for instance:

Improved Fault Tolerance. In conventional systems, a high-level interaction between a client and a server, e.g., a commercial transaction or a complex network management task, unfolds as a series of pairwise low-level interactions under the form of request and replies. During these interactions, the state of the overall computation is distributed. This fact heavily complicates the task of recovering from a fault, due to the distributed consensus problem. Instead, agents embedding the code describing the whole high-level interaction can migrate on the server. Thus, the state of the interaction remains entirely local, and faults can be dealt with easily, e.g., using checkpoints.

Support for Disconnected Operations. Mobile agents can carry out their tasks autonomously and independently of the application that dispatched them. This capability, that is at the core of many of the advantages mobile agents provide, is particularly useful in scenarios characterized by physical mobility, where the constraints posed by terminals and communication links often force the user to disconnect from the network, e.g., to save battery power. In these scenarios, a mobile agent can be unleashed into the network to perform some task on behalf of a user, who meanwhile is totally disconnected. Results can be eventually gathered by the user upon reconnection.

Protocol Encapsulation. In conventional systems, data is typically a passive element that gets processed by other active components in the system. Thus, for instance, network packets contain data that is processed at intermediate nodes and then forwarded to destination; similarly, documents in a workflow process are exchanged among the process actors, that manipulate them according to organization-wide procedures. Mobile agents and code mobility may change this view dramatically, by allowing a piece of data to travel within the system together with the application logic needed to interpret and manipulate it. Thus, a packet could flow in the network carrying along with it its own routing routines, as proposed in some active network schemes [9]; a document could contain code representing automated business procedures that would get invoked at various step during the workflow [10]. Clearly, this possibility greatly improves the flexibility of the system, simplifying the deployment of different, co-existing policies for using data.

After this brief discussion, it is evident that mobile agents hold the potential for providing relevant benefits. The two case studies presented made also clear, however, that mobile agents can be exploited in different ways, and that the benefits obtained depend heavily on the particular design alternative chosen. In the next section, we elaborate on this latter theme by putting mobile agents in the more general context of design paradigms for code mobility.

III. Mobile Agents as a New Design Paradigm

Most of research about mobile agents spurred from and focused on technologies that enable a unit of execution to migrate at run-time to a different host. Nevertheless, the key contribution of the field is not technology. Exploiting mobile agents, and in general logical mobility, is not strictly tied to the use of technology expressly developed to support mobility, much like exploiting object-orientation is not limited to using an object-oriented programming language. Instead, the crucial contribution put forth by logical mobility is to foster a new *design style* for distributed applications, where the location of components is not configured once and for all, but becomes instead a first-class element under the control of the designer.

Nevertheless, as evidenced by the various alternatives considered in the previous section, the basic idea of mobile agent can be exploited in several variants and, moreover, different terminologies are popular among researchers. Thus, for instance, Papastavrou et al. term their solution exploiting a mobile agent "parked" at the Web server an instantiation of the *client/agent/server* paradigm [3]. Others would refer to the very same solution as an example of a *single-hop* mobile agent, as opposed to the common notion of a *multi-hop* mobile agent that roams several nodes in sequence.

The scene is complicated even further by the fact that mobile agents represent only a subset of the larger space of alternatives offered by logical mobility. Alternatives are possible where migration does not involve the whole unit of execution representing an agent, but only some of its constituents, typically the code. Although these forms of mobile code are often improperly considered mobile agents as well, the design style they foster is profoundly different, both in terms of the abstractions they represent and of the impact they have on the overall efficiency of the resulting implementation.

In this section, we hint exactly at these two aspects. In the first part, we present one of the few characterizations of the design paradigms for code mobility available in literature. In the second part, we briefly discuss how forms of code mobility that are less powerful and expressive than mobile agents may actually be a better solution in some cases. Finally, we provide some general remarks about design issues related with mobile agents.

A. Design Paradigms for Code Mobility

The characterization of architectural paradigms we report here, originally proposed in [11], focuses on the different ways it is possible to perform a service, that would be otherwise performed with a client-server paradigm, by moving around the three fundamental elements of such service. The elements are the know-how about the service (i.e., its code), the resources that are needed to carry it out (e.g., a file, a DBMS, an object), and the active component in the system that is in charge of performing it. Clearly, a service can be effectively performed only when these three capabilities are co-located. In a pure client-server paradigm, these capabilities are permanently co-located by design on the server node, and are exploited remotely by the client. Instead, the following mobile code paradigms provide different strategies for relocating these capabilities at run-time.

In the *Code on Demand* (COD) paradigm, one of the two components (e.g., the client) lacks the know-how about how to perform the service, although it owns the necessary resources. The corresponding code is then downloaded from a remote server acting as a code repository, and subsequently executed. This paradigm provides enhanced flexibility by allowing the server to change dynamically the behavior of the client (or vice versa). For instance, this is the scheme typically employed by Web applets.

In the *Remote Evaluation* (REV) paradigm, the client owns the know-how about the service, but lacks the resources necessary to its execution, which are owned by the server component. As proposed in the pioneering work described in [12], a sort of enhanced client-server interaction takes place, where the client includes in the request to the server also the code required to perform the service. After this code is received and its execution has started on the server, the interaction proceeds as in the client-server paradigm: the code received is able to access the resources now co-located with it, and eventually send the results back to the client. This design solution underlies well-known systems like remote shells and SQL servers.

Finally, in the *Mobile Agent* (MA) paradigm the client knows how to perform the service but lacks part of the resources, which are owned by the server. The client then autonomously migrates to become co-located with the server, and thus to perform the service by exploiting local access to resources. This definition encompasses the common definition of a multi-hop mobile agent.

It is worth noting how COD and REV actually stress the notion of *mobile code* as opposed to mobile agents. For instance, the design solution presented in [3] and described in Section II-A, that exploits a mobile agent that remains parked at the DBMS to act as a proxy for further queries, could appear at first as an instantiation of the COD paradigm. Similarly, the scheme exploited in the same application to deal with multiple DBMSes resembles closely the REV paradigm, although in that application the whole agent is always returned to the client.

Instead, these solutions must be regarded as instantiations of the mobile agent paradigm, because a whole active component is being relocated. Instead, both COD and REV allow relocation of just the code portion of this component. This difference, that at first could appear as irrelevant or pedantic, may change dramatically the performance of the resulting design, as described in the following.

B. Mobile Code or Mobile Agents?

Since all the aforementioned design paradigms allow the dynamic relocation of the components of a distributed application, a legitimate question to ask is whether one should choose the purest mobile agent paradigm, or just a mobile code design, exploiting code on demand or remote evaluation. As with every design choice, the answer is given by application requirements and engineering tradeoffs.

Mobile agents provide a relevant asset because of the metaphor they embody. Agents that are able to dynamically and autonomously relocate themselves according to the application needs may provide, for certain applications, the building block of a uniform and elegant design where every active component is able to spontaneously relocate itself. This characteristic of the paradigm is probably at the core of its success, and provides also the link to other disciplines, like artificial intelligence, that brought this concept to the extreme by proposing agent-oriented programming [13] as a new way to create distributed applications.

On the other hand, a naive use of mobile agents may lead quickly to a highly inefficient design. Both case studies in Section II already evidenced this possibility. In the DBMS application, the alternative where a mobile agent is employed to carry the query from the client to the DBMS-agent is always the worst of mobile agent ones. Clearly, in that case the cost of creating a mobile agent for each request just to carry a message is overkill; message passing is much more effective in this case. A similar argument holds for the network management application of Section II-B, where the use of a mobile agent that roams the network and collects information—indeed a very popular piece of the mobile agent folklore—may actually lead to a design that actually performs worse than the conventional one.

This latter remark, however, does not necessarily hold for the other mobile code paradigms. Figure 4 is adapted from [5], where the authors considered, besides mobile agents, also the aforementioned mobile code paradigms. As the leftmost chart shows, COD allows a traffic reduction of about 30%, even in the case where no semantic compression is possible, i.e., when mobile agents would perform really poorly. The reason lies in the fact that COD allows to pay the cost for migrating code only once, when the code fragment gets installed at destination. From that point on, communication takes place like in a normal client-server interaction. Thus, the higher is the frequency of invocation of the management task, the bigger is the gain COD can provide[3]. Instead, the chart in the center of Figure 4 shows that also REV may be more convenient than client-server even without semantic compression, if the number of queries that must be performed locally on the device is high enough. Thus, mobile code may succeed where mobile agents are failing.

Moreover, under some conditions mobile code may be more convenient even in the scenarios where mobile agents would seem always advisable. For instance, the rightmost chart in Figure 4 compares the performance of code on demand against mobile agents when global semantic compression is enabled and only the traffic generated around the NMS is considered—optimal conditions for using mobile agents. Still, if the network of managed devices is small enough, mobile agents might lead to worse performance.

C. Some Reflections

Mobile agents are a powerful abstraction. Nevertheless, expressive power always comes as a tradeoff for efficiency; relocating a whole executing unit is more demanding than relocating only one of its constituents. The considerations expressed thus far reveal that mobile agents are only one extreme of the spectrum of alternatives that, starting from the static client-server paradigm provide the ability to reconfigure dynamically the system through code mobility. Thus, if the motivation for using mobile agents is to optimize system performance, e.g., in terms of traffic or latency, attention should be paid to the choice between mobile agents and mobile code.

State of the art research still does not provide clear guide-

[3]The charts in Figure 4 assume twenty invocations.

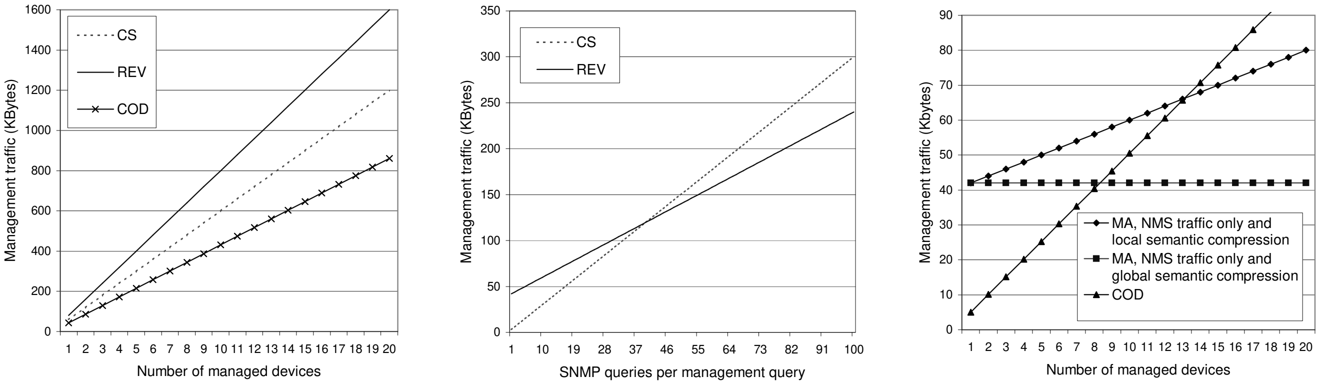

Fig. 4. Mobile code for network management.

lines about when to use a design paradigm or the other, and studies like those reported in this paper are extremely rare. Moreover, there might be other and better paradigms besides those presented here, or new and more effective ways to combine them among themselves and with conventional paradigms like client-server.

A new viewpoint on these issues may be contributed by researchers investigating theories of mobility [14]. Research has concentrated thus far on formal models where the unit of execution coincides with the unit of mobility, with few exceptions (e.g., [15]). Nevertheless, the fact that these investigations are set in an abstract environment, free of the idiosyncrasies of real implementations, is likely to help unveil some of the fundamental characteristics of logical mobility at large. While this is likely to improve our understanding of the general issues related with the fundamental design styles concerned with mobility, this is also hopefully going to shape the next generation of the related technology, by shedding some light on the fundamental constructs that are needed to deal with mobility and their semantics.

IV. Mobile Agent Technology

Technology has traditionally been the main focus of research on mobile agents. As a matter of fact, the very term "mobile agent" was made popular by the Telescript language [16], developed by General Magic in 1994. The emphasis on technology is witnessed, among the other things, by the large number of systems contained in the Mobile Agent List [17], maintained at the University of Stuttgart, Germany, that provides an approximate census of the mobile agent systems currently available.

In this section we give an overview of the fundamental characteristics concerned with migration, and discuss other relevant issues. The presentation is necessarily concise. Surveys of mobile agent technology can be found in literature (e.g., [7], [18], [19], [20]) that may help the reader in comparing available systems.

A. Core Support for Mobility

Mobile agent systems typically identify the agent with a unit of execution belonging to the lower layers of the virtual machine, e.g., a thread or a process. A unit of execution[4] is constituted by the code governing its behavior, by the data associated with it and necessary to its computation, and by its execution state, e.g., program counter, and call stack. Mobile agent systems allow migration of the whole unit or a part thereof, i.e., one or more of the three constituents mentioned above. The most relevant differences among existing systems lie exactly in what they allow to move, and how it is actually moved.

A first distinction can be drawn based on whether the execution state is migrated along with the executing unit or not. Systems providing the former option are said to support *strong mobility*, as opposed to systems that discard the execution state across migration, and are hence said to provide only *weak mobility*. In this latter kind of systems, if the application requires the ability to retain the thread of control, extra programming is required in order to save manually the execution state. Instead, in systems supporting strong mobility, migration is completely transparent to the migrated program, which resumes execution right after the migration instruction. This has the double advantage of reducing the programming effort of using migration to the invocation of a single operation, and of requiring a smaller code size of the migrated code.

Despite these advantages, most of the mobile agent systems support only weak mobility. The reason lies in the fact that the vast majority of them are built on top of the Java Virtual Machine (JVM), which provides mechanisms sufficient to implement weak mobility (namely the ability to program the class loader) but insufficient to deal with the execution state. This is perceived by the community as one of the main drawbacks of Java as the implementation platform, and research is investigating alternative solution based either on a pre-compilation step (e.g., [21]) or on modifications of the JVM (e.g., [22]). Systems that are not based on Java do not suffer of this limitation and often provide strong mobility, like in Telescript, Tacoma [23], Ara [24], and D'Agents [25].

Another dimension to understand the mechanisms supporting mobility is constituted by the strategies employed to relocate the code constituting the executing unit. Al-

[4] We follow the classification proposed in [7], where the reader may find a more detailed description of the issues briefly mentioned here.

though a number of strategies are potentially meaningful and useful, the use of Java as an implementation language has often biased the designers of mobile agent systems towards mechanisms that are directly inspired by the Java class loader, and its use within Web browsers to support applet downloading. In this scheme, only the agent's root class is migrated along with the agent; after migration, additional classes needed for execution of the agent are downloaded dynamically from the agent source host, or from some other code repository. This mechanism, adopted by many of the Java-based systems, notably Mole [26] and Aglets [27], relies on the assumption that the code repository is always available, thus implicitly neglecting one of the main advantages of mobile agents, i.e., the ability to support disconnected operations. On the other hand, other systems, e.g., D'Agents, always ship the whole code base together with the agent, thus in many cases sending also classes that are used infrequently. The μCode [28] system supports both strategies, and in addition allows the programmer to compute dynamically the agent's class closure and subsequently choose which classes should be migrated and which not, thus achieving maximum flexibility.

Finally, the third dimension is constituted by the data the mobile agent may carry along during migration. The unit of execution running at the source is likely to contain bindings to resources (e.g., objects, files, other units) that are shared with other units on that host. To allow mobility of the executing unit requires both a mechanism and a policy to determine how these bindings are handled upon migration. As discussed in [7], a number of strategies are possible. Essentially, the binding to a resource can be severed, retained, or re-established with a different but compatible resource. When the binding is retained, two alternatives are possible: either the resource is migrated along with the agent, or the binding is stretched across the network by creating a *network reference* from the new host of the agent. When the binding is instead re-established to a new, compatible resource, such resource is typically constituted either by a copy of the original one, or by a stationary resource having the same type (e.g., a printer).

Once more, the use of Java seems to have hampered creativity, favoring solutions that are easily implementable with the language features. The typical strategy for Java-based systems is simply to rely on the serialization mechanism provided by Java, that allows to copy the full object closure of an agent, while enabling the programmer to tag those fields that should not be serialized. Thus, data management upon migration is completely left to the programmer. Exceptions are the FarGo system [29], which introduces an elegant model that aims at building a distributed application out of (possibly mobile) components linked by the aforementioned binding types, and the μCode system, for which a package implementing the binding mechanism described in [7] has been implemented. Among the non-Java systems, once again the Telescript language already had a sophisticated mechanism that ruled migration and security through a single notion of ownership.

B. Other Relevant (and Open) Issues

The research challenge involved in providing support for mobility does not end with the design of core mechanisms and constructs that enable the relocation of software components. Mobility introduces several other challenges, that we are going to review briefly in this section. Among these, security is probably the one that attracted most research efforts, and also raised several concerns about practical uses of mobile agents. It is not uncommon to hear people make analogies with viruses when learning about code mobility.

To be fair, it must be said that the importance of security is sometimes overestimated. A significant example is provided precisely by the network management application domain discussed earlier, where the current security practice in the SNMP world is to protect accesses to the device data through a password that is sent on the wire unencrypted. Clearly, in this domain the use of a mobile agent system that does not provide security does not represent a step backward, and yet it could provide the remarkable benefits discussed in Section II-B.

In the applications for which security is indeed an issue, two specular problems capture the attention of designers. The first problem is how to protect a machine that is executing foreign, untrusted mobile code. Most of the issues raised by this problem can be successfully tackled by reusing or adapting well-known solutions already developed, e.g., for Internet security. A notable exception to this statement is constituted by resource control, at least for Java-based systems. A Java thread, the executing unit that typically embodies a mobile agent, can run indefinitely and consume all the resources of the associated JVM. Providing resource allocation in Java is an open research issue, that is being targeted both inside (e.g., [22]) and outside (e.g., [30]) the mobile agent community. On the other hand, many non-Java based systems, beginning with Telescript, provide quite sophisticated resource control mechanisms.

Nevertheless, the true conceptual challenge posed by mobile agents to security is in the dual problem, i.e., protecting the migrating code from the host that is executing it. This problem is specifically introduced by migration, and is complicated by the fact that the very same parts of the agent that must be protected, i.e., its code and data, must at the same time be disclosed to the host in order to enable execution. Many scenarios that involve the use of mobile agents for commercial transactions, e.g., those related to e-commerce, are severely complicated by this security concern. A solution to this problem has been initially claimed impossible, until a few researchers showed that is at least possible to prevent or detect tampering (see for instance some of the papers in [31]).

Mobile agents introduce other new challenges, particularly as far as communication is concerned. In conventional distributed applications, the parties involved in communication are stationary, and thus communication channels can be easily established and maintained. Instead, mobile agents define a scenario where the communication endpoints may be in continuous movement, and whose current location may be difficult, if not practically impossible, to track. Thus, as pointed out in [32], the problem of ensuring reliable communication among mobile agents is not a matter of ensuring fault tolerance, like in distributed systems, but is posed by the sheer presence of mobility even under the assumption of a fault-free network. Typical mechanisms, like broadcasting or forwarding, may "miss" an agent that is crossing the same link in the opposing direction, or may chase the mobile agent forever.

Most of existing systems ignore or circumvent this issue, and typically provide either conventional mechanisms (e.g., remote procedure calls) without any adaptation for mobility, or restrict communication in some way, e.g., by limiting it to co-located agents or by constraining agent movement. Indeed, research is focusing more on defining new primitives for inter-agent communication than on analyzing the related guarantees. The Telescript language has inspired a number of systems where a *meet* primitive belongs to the default interface of agents, and can be invoked by other agents in order to establish a sort of rendezvous. Some systems provide communication among group of agents, either by exploiting an event dispatching mechanism limited to the scope of a host (e.g., Aglets [27]), through dedicated primitives (e.g., Mole [33]). or by adapting a coordination perspective that exploits tuple spaces, (e.g., Lime [34] and MARS [35]).

Nevertheless, reliability, and in particular fault tolerance, is an important issue in mobile agent systems, and a one that impacts facets other than inter-agent communication. We discussed in Section II how the ability to constrain the scope of interactions to a local environment is a feature of mobile agents that may enhance the fault tolerance of the overall distributed system. Nevertheless, this is true only if agent migration is itself fault tolerant, and if proper mechanisms for local recovery are in place.

Like for communication, however, fault tolerance is often neglected by mobile agent systems. Very few systems provide mechanisms to ensure fault tolerant migration, and existing ones (e.g., [36], [37]) exploit heavyweight transactional mechanisms that seriously compromise the whole idea of mobile agent. On the other hand, many systems provide some form of persistence for mobile agents, providing automatic or manual support for checkpointing.

V. Discussion and Open Issues

Mobile agent enthusiasts are often wondering why this concept has not made it yet into the mainstream of distributed systems. The blame is typically placed on the lack of proper security mechanisms, especially to protect agents from malicious hosts. However, as we already pointed out, this observation may hold for some specific application domains, or for specific technologies, but cannot be considered to hamper the whole research field. We believe the reason lies elsewhere.

A first reason is the way today's platforms convey the mobile agent concept. Thus far, researchers have focused too much on the technicalities of building a mobile agent system, often losing the global picture of *why* that system is being built. Too often, the systems contain features that totally hamper achievement of some of the main benefits of mobile agents. A macroscopic example is, for instance, the aforementioned design choice of migrating code solely by on demand dynamic download, which prevents *by design* the capability to provide support for disconnected operations, and heavily limits the overall autonomy of agents.

A related issue is the role of the Java language. It is evident even from this concise introduction that Java constitutes a double-edged sword for the field of mobile agents. On one hand, it provides a lot of features that simplify the life of the programmer of mobile agents. On the other hand, this very fact determines a strong bias in the design alternatives that are explored by designers, thus discouraging them from exploring avenues of research that are less tied to Java's idiosyncrasies—and possibly more fruitful.

Moreover, most of the current systems provide only the mobile agent abstraction, which considers the unit of migration coincident with the unit of execution. Nevertheless, in this paper we argued and provided evidence that mobile agents are not always the best abstraction for a design involving mobility, and the use of primitives that migrate code or state separately from the unit of execution may actually provide a better design.

Finally, the majority of current systems appears to be designed with the goal to substitute, rather than complement, existing middleware for distributed systems. The result is that they usually come packaged as stand-alone, monolithic software, that is typically difficult to tailor to specific needs. This clashes with the argument that mobile agents are only one among the many solutions that can be exploited by a designer, even in the context of the same distributed application.

These reasons are probably a consequence of the relative immaturity of the field. Mobile agent systems simply need to move out of the prototype phase towards a more rationale design, whose goals and requirements are clearly stated by taking into account the need to maximize the advantages of mobile agents and to foster a smooth integration with mainstream techniques.

However, there is probably another reason that is currently reducing the acceptance of mobile agent, whose nature is less technological. Mobile agents will be adopted only when a sufficient body of literature will provide incontrovertible evidence about when and how much they are useful. Unfortunately, case studies like those we presented in Section II, that not only analyze thoroughly the implications of using mobile agents in a real application domain, but also do it on a quantitative and experimental basis, are still very rare. The mobile agent research community must put more effort into validating its own outcomes, in order to gain credibility outside.

VI. Conclusions

In this paper, we provided a brief introduction to the research field concerned with mobile agents, touching also on the more general topic of code mobility.

Mobile agents and mobile code are destined to influence research in distributed systems for the years to come, by setting the premises for a new wave of architectures that exhibit a high degree of reconfigurability and adaptability.

Thus far, technology has been instrumental in disseminating new design paradigms where application components are not permanently bound to the hosts where they execute. Further research is needed to consolidate the conceptual foundations of this approach, as well as to assess precisely the tradeoffs that will determine its applicability.

VII. Acknowledgements

The author wishes to thank Amy Murphy and Antonio Corradi for their precious comments on early drafts of this paper.

References

[1] G.-C. Roman, G.P. Picco, and A.L. Murphy, "Software Engineering for Mobility: A Roadmap," in *The Future of Software Engineering*, A. Finkelstein, Ed., pp. 241–258. ACM Press, 2000.

[2] C.G. Harrison, D.M. Chess, and A. Kershenbaum, "Mobile Agents: Are they a good idea?," in *Mobile Object Systems: Towards the Programmable Internet*, J. Vitek and C. Tschudin, Eds., vol. 1222 of *LNCS*, pp. 25–47. Springer, Apr. 1997, Also available as IBM Technical Report.

[3] S. Papastavrou, G. Samaras, and E. Pitoura, "Mobile Agents for WWW Distributed Database Access," in *Proc. of the 15th Int. Conf. on Data Engineering (ICDE99)*, 1999.

[4] Sun Microsystems, "JDBC Technology," Available at `http://java.sun.com/products/jdbc`.

[5] M. Baldi and G.P. Picco, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications," in *Proc. of the 20th Int. Conf. on Software Engineering*, R. Kemmerer, Ed. Apr. 1998, pp. 146–155, IEEE CS Press.

[6] J. Case, M. Fedor, M. Schoffstall, and C. Davin, "Simple Network Management Protocol," RFC 1157, May 1990.

[7] A. Fuggetta, G.P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Trans. on Software Engineering*, vol. 24, no. 5, 1998.

[8] D.B. Lange and M. Oshima, "Seven good reasons for mobile agents ," *Communications of the ACM*, vol. 42, no. 3, 1999.

[9] D.L. Tennenhouse et al., "A Survey of Active Network Research," *IEEE Communications*, vol. 35, no. 1, Jan. 1997.

[10] G. Cugola and C. Ghezzi, "Design and Implementation of PROSYT: a distributed Process Support System," in *Proc. of the 8th Int. Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*, Stanford, CA, USA, June 1999.

[11] A. Carzaniga, G.P. Picco, and G. Vigna, "Designing Distributed Applications with Mobile Code Paradigms," in *Proc. of the 19th Int. Conf. on Software Engineering (ICSE'97)*, R. Taylor, Ed. 1997, pp. 22–32, ACM Press.

[12] J.W. Stamos and D.K. Gifford, "Remote Evaluation," *ACM Trans. on Programming Languages and Systems*, vol. 12, no. 4, pp. 537–565, Oct. 1990.

[13] N.R. Jennings, "On Agent-Based Software Engineering," *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, 2000.

[14] G. Di Marzo Serugendo, M. Muhugusa, and C. Tschudin, "A Survey of Theories for Mobile Agents," *WWW Journal*, vol. 1, no. 1, 1998.

[15] C. Mascolo, G.P. Picco, and G.-C. Roman, "A Fine-Grained Model for Code Mobility," in *Proc. of the 7th European Software Engineering Conf. held jointly with the 7th ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE'99)*, O. Nierstrasz and M. Lemoine, Eds., Toulouse (France), Sept. 1999, vol. 1687 of *LNCS*, pp. 39–56.

[16] J.E. White, "Telescript Technology: Mobile Agents," in *Software Agents*, J. Bradshaw, Ed. AAAI Press/MIT Press, 1996.

[17] F. Hohl, "Mobile Agent List," `http://mole.informatik.uni-stuttgart.de/mal/mal.html`.

[18] N.M. Karnik and A. Tripathi, "Design Issues in Mobile-Agent Programming Systems," *IEEE Concurrency*, 1998.

[19] J. Kiniry and D. Zimmerman, "A Hands-On Look at Java Mobile Agents," *IEEE Internet Computing*, vol. 1, no. 4, 1997.

[20] T. Thorn, "Programming Languages for Mobile Code," *ACM Computing Surveys*, vol. 29, no. 3, pp. 213–239, Sept. 1997.

[21] T. Sakamoto and T. Sekiguchi and A. Yonezawa, "Bytecode Transformation for Portable Thread Migration in Java," In Kotz and Mattern [38].

[22] N. Suri et al., "Strong Mobility and Fine-Grained Resource Control in NOMADS," In Kotz and Mattern [38], pp. 2–15.

[23] D. Johansen, F.B. Schneider, and R. van Renesse, "Operating system support for mobile agents," in *Mobility, Mobile Agents, and Process Migration*, D. Milojicic et al., Eds. Addison-Wesley, 1998.

[24] H. Peine and T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents," in *Mobile Agents: 1st Int. Workshop MA '97*, K. Rothermel and R. Popescu-Zeletin, Eds. Apr. 1997, vol. 1219 of *LNCS*, pp. 50–61, Springer.

[25] R.S. Gray, G. Cybenko, D. Kotz, R.A. Peterson, and D. Rus, "D'Agents: Applications and Performance of a Mobile-Agent System," *Software: Practice and Experience*, 2001, to appear.

[26] M. Straßer, J. Baumann, and F. Hohl, "Mole—A Java Based Mobile Agent System," in *Special Issues in Object-Oriented Programming: Workshop Reader of the 10th European Conf. on Object-Oriented Programming ECOOP'96*, M. Mühlaüser, Ed. July 1996, pp. 327–334, dpunkt.

[27] D.B. Lange and M. Oshima, Eds., *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998.

[28] G.P. Picco, "μCode: A Lightweight and Flexible Mobile Code Toolkit," In Rothermel and Hohl [39], pp. 160–171.

[29] O. Holder, I. Ben-Shaul, and H. Gazit, "Dynamic Layout of Distributed Applications in FarGo," in *Proc. of the 21st Int. Conf. on Software Engineering (ICSE'99)*, D. Garlan and J. Kramer, Eds., Los Angeles, May 1999, pp. 163–173, ACM Press.

[30] G. Czajkowski and T. von Eicken, "JRes: A Resource Accounting Interface for Java," in *ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Vancouver, Canada, Oct. 1998, pp. 21–35.

[31] G. Vigna, Ed., *Mobile Agents and Security*, vol. 1419 of *LNCS*, Springer, 1998.

[32] A.L. Murphy and G.P. Picco, "Reliable Communication for Highly Mobile Agents," in *Proc. of the 1st Int. Symp. on Agent Systems and Applications held jointly with the 3rd Int. Symp. on Mobile Agents (ASA/MA'99)*, D.S. Milojicic, Ed., Palm Springs, Oct. 1999, pp. 141–150, IEEE Computer Society, To appear also in *J. of Autonomous Agents and Multi-Agent Systems*.

[33] J. Baumann et al., "Communication Concepts for Mobile Agent Systems," In Rothermel and Hohl [39], pp. 123–135.

[34] G.P. Picco, A.L. Murphy, and G.-C. Roman, "Lime: Linda Meets Mobility," in *Proc. of the 21st Int. Conf. on Software Engineering*, D. Garlan, Ed., May 1999, pp. 368–377.

[35] G. Cabri, L. Leonardi, and F. Zambonelli, "Reactive Tuple Spaces for Mobile Agent Coordination," In Rothermel and Hohl [39], pp. 237–248.

[36] M. Straßer K. Rothermel, "A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents," in *Proc. of the 17th IEEE Symp. on Reliable Distributed Systems (SRDS'98)*, Los Alamitos, CA, USA, 1998, pp. 100–108.

[37] F.M. Assis Silva and R. Popescu-Zeletin, "An Approach for Providing Mobile Agent Fault Tolerance," In Rothermel and Hohl [39], pp. 14–25.

[38] D. Kotz and F. Mattern, Eds., *Agent Systems, Mobile Agents, and Applications—2nd Int. Symp. on Agent Systems and Applications and 4th Int. Symp. on Mobile Agents, ASA/MA 2000*, vol. 1882 of *LNCS*, Zurich, Sept. 2000. Springer.

[39] K. Rothermel and F. Hohl, Eds., *Proc. of Mobile Agents: 2nd Int. Workshop MA'98*, vol. 1477 of *LNCS*, Stuttgart, Sept. 1998. Springer.