Exploiting Code Mobility in Decentralized and Flexible Network Management

Mario Baldi, Silvano Gai, and Gian Pietro Picco

Dip. Automatica e Informatica, Politecnico di Torino
C.so Duca degli Abruzzi 24, 10129 Torino, Italy
[mbaldi | silvano | picco]@polito.it

Abstract. Network management is gaining increasing importance due to the pervasiveness of computer networks. Nevertheless, mainstream approaches to network management are presently limited by centralized management strategies and poor flexibility—a consequence of their rigid client-server architecture. In this paper we analyze how to overcome these problems by new design paradigms and technologies encompassing the capability to relocate dynamically the components of a distributed application. We evaluate the opportunities offered by this approach and provide feasibility considerations, also discussing a few interim architectural solutions adopted in our on-going implementation work.
Keywords: code mobility, mobile agents, network management.

1 Introduction

Computer networks are taking a pivotal role in present and future information systems. The increasing importance of computer networks has given rise to a high demand for adequate *network management*. The two main approaches proposed in the literature come from the Internet Engineering Task Force (IETF) and the International Organization for Standardization (ISO) and are characterized by centralization and a low degree of flexibility and reconfigurability. These problems are a consequence of the client-server (CS) paradigm adopted in the management architecture. *Management agents* co-located with network devices act like servers that communicate on request device data to a management station, where all the management computation actually takes place. This choice was aimed at keeping the agents as simple as possible, due to limitations on the resources available in the devices hosting them—a constraint being relaxed by technological achievements. Yet, it has been demonstrated [20] that this solution lacks scalability and generates congestion in the network area around the management station, thus heavily limiting the effectiveness of the approach.

The increased importance of computer networks has also stimulated research on large scale distributed systems. In this context, new approaches to application design and implementation have been developed that embody the notion of *code mobility*, i.e., the capability of moving dynamically the components of a distributed application among the nodes of a computer network [16]. The key idea behind code mobility is to provide a complementary solution (if not an alternative) to the traditional CS structure of distributed applications, thus enabling a better use of bandwidth resources and a higher degree of flexibility and reconfigurability. This idea benefits from the availability of new programming languages providing direct support for some form of code mobility. Nevertheless, the promises of this new research area are still to be verified by experimental evidence from the application in real world domains.

The synergy between research on network management and on code mobility is promising. Network management is looking for ways to overcome CS limitations and code mobility now provides technologies and design paradigms for this goal. On the other hand, network management provides a good test-bed for the mobile code approach. The goal of this work is to examine challenges and opportunities arising from the synergy of these two fields, investigating if and how code mobility can help in solving the problems of network management approaches. Application of code mobility to network management is not new [12,6], yet recent developments in code mobility [1,4] contributed ideas that subsume older approaches. In this paper, we propose neither a new full-fledged approach to network management nor precise modifications to current approaches. Our analysis and discussion is meant to provide suggestions for such activities and guidance for our on-going work on the implementation and assessment of a management application exploiting code mobility.

The paper is structured as follows. A brief overview on network management approaches is provided in Sect. 2, while Sect. 3 presents technologies and design paradigms based on code mobility. The core of the paper is Sect. 4, where benefits and drawbacks stemming from the application of code mobility to network management are evaluated together with some architectural solutions oriented to implementation. Finally, future directions for this work are discussed in Sect. 5.

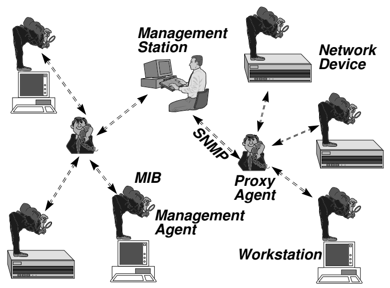
2 Network Management Approaches

In this section, we present an overview of the approaches to network management found in literature, discussing how they do (or do not) provide support for decentralization and flexibility.

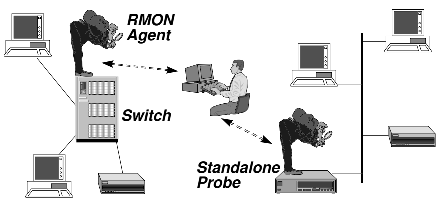
Mainstream Approaches The approach more widely applied comes from the IETF and is based on the *Simple Network Management Protocol* (SNMP) [2,3]. Of great relevance is also the one proposed by ISO for application within OSI networks, which is based on the *Common Management Information Protocol* (CMIP) [13]. The two approaches are very similar in the architecture they adopt though they differ slightly in the way they operate [10]. Both assume the presence of a *management station* (operated by the network manager) that interacts with *agents* running on network nodes, e.g., bridges, routers, and workstations. Agents are computational entities whose purpose is to provide a standardized interface for accessing information about the network device on which they reside. Each agent stores information in a local information base, named *management information base* (MIB) in the IETF approach, and *management information* *tree* (MIT) *database* in the ISO approach. The management station interacts with agents through a *management protocol* that specifies the packet format for a set of basic operations. Messages containing data and operation codes are transmitted using transport layer services.

Both SNMP and CMIP adopt a CS paradigm. The management station acts as a client that provides a user interface to the network manager and interacts with agents. Agents act as servers that manage remote access to their local information base. The management station fetches from agents small chunks of information (single data items or, in SNMPv2, a bulk of them), processes them, and requests further information or sends some data to agents by using very low-level commands. This fine grained CS interaction is called *micro management* and leads to the generation of high traffic and overload for the management station, that performs all the actual computation.

Drawbacks of Centralization in Network Management The IETF and ISO approaches are characterized by high centralization, since their architecture puts almost all the computational burden on the management station. Centralization has proven to seriously limit the scalability of network management [20]. As the dimension of the network grows, the management station has to communicate with a larger number of devices, and to store and process an ever increasing amount of data. This leads to the need for high cost hardware dedicated to the management station, to poor performance, or even to the impossibility to cope with the dimension of the network. In some cases, a separated network is employed to carry management data, but in most cases *in band management* is employed, that is, the very network that is being managed is used to carry management information. Consequently, the area of the network around the management station experiences heavy traffic due to the combination of messages sent around by the management station and those containing data from the devices. The worst shortcomings of the centralized in band approach show up during periods of heavy congestion, when management intervention is particularly important. In fact, during these periods: *i)* the management station increases its interactions with the devices and possibly downloads configuration changes, thus increasing congestion, *ii)* access to devices in the congested area becomes difficult and slow (sometimes even impossible), and *iii)* congestion, as an abnormal status, is likely to trigger notifications to the management station which worsen congestion. Similar problems affected routing table computation some years ago when it was centralized. A Routing Control Center (RCC) gathered information on network topology, calculated the routing table for each router in the network, and downloaded it into the proper device. The heavy traffic load in the area around the RCC and the difficulty of managing network areas far from the RCC led to the development of distributed routing, which is still being applied successfully. Analogously, in order to cope with problems that arise from centralization, the network management functionality must be decentralized, i.e., the complex diagnosing and information gathering activities must be moved from the management station station into the network. Both IETF and ISO have taken some steps in this direction by introducing some primitive form of decentralization in



(a) SNMPv2 proxy agents.



(b) RMON probes.

Fig. 1. Decentralization features in the IETF approach.

their approaches, as it will be described in the following section. In turn, the idea of decentralizing management functionality has been followed by the approach called management by delegation, that will be described later and will be taken even further by our solutions exploiting mobile code.

Decentralization Features Event notification is a first step in decentralization and aims at relieving the management station from some of its processing burden. SNMP agents are designed to notify upon the occurrence of a few significant events by issuing *traps*, i.e., messages sent without an explicit request from the management station. Since the IETF approach aims at keeping agents as simple as possible, they can notify only upon events that can be identified with a light computation. Typical examples are changes in the status of a link (from down to up, or vice-versa) or a device that has been reset. In turn, the ISO approach encompasses more complex agents, able to perform more processing on the data collected, thus increasing the number of significant situations (*events* or *alarms* in the OSI jargon) possibly dependent on several parameters that can be recognized and notified. Anyway, in both the approaches the agent is responsible only for the notification of an event and has no means of recovering from it. In other words, all the "management knowledge" resides on the management station.

According to the ISO approach, the management station can invoke actions on the objects stored in the MIT database, which is object-oriented. Execution of these actions, which can be fairly complex, reduces both the processing load on the management station and the management traffic. The IETF approach does not provide an explicit remote action invocation mechanism. Nevertheless, it can be implemented by forcing the agent to execute actions as a side effect of a change, operated by the management station, in predefined data items in the MIB. In other words, changes in the information base trigger actions performed remotely by the agent. SNMPv2 takes a step further towards decentralization by defining the *proxy agent* concept, i.e., a network management agent that acts as both a client and a server (Fig. 1a). On one side, a proxy agent sends commands to other agents and handles their responses; on the other side, it answers to commands received by a management station.

The IETF has proposed another approach, known as *Remote MONitoring* (RMON) [18], that introduces a higher degree of decentralization. RMON assumes the availability of network monitoring devices called *monitors* or *probes*. By monitoring packet traffic and analyzing the headers, probes provide information about links, connections among stations, traffic patterns, and status of network nodes. Hence, RMON can be regarded as a *traffic-oriented* approach because the status of the network is determined by direct inspection of the packets flowing in it, rather than inspection of the status of each device, like in the *device-oriented* approaches reviewed before. As shown in Fig. 1b, probe implementations can be stand alone devices dedicated to link monitoring or can be embedded into network devices. SNMP is used for communication between the management station(s) and the agent running on each probe. Some of the processing that in traditional approaches is performed by the powerful hardware of the management station, in RMON is performed by the agents on the probes *locally* and *independently* of the management station. A probe can detect failures, misbehaviors, and identify complex relevant events even when not in contact with the management station, which is likely to happen when the network is overloaded or in critical conditions—i.e., when network management is vital. The concept of *alarm* increases decentralization: the agent on the probe is configured to take periodic statistical samples of relevant parameters and, whenever a predefined threshold is crossed by any of them, to notify the management station. In addition, since the agent on the probe can perform semantic compression of data by pre-processing the information collected before sending it to the management station.

Management by Delegation Management by delegation [6] (MbD) represents a clear effort towards decentralization and increased flexibility of management functionality. The management architecture still includes a management protocol and agents, yet an *elastic process run-time support* is assumed on each device. Instead of exchanging bare CS messages, the management station can specify a task by packing into a program a set of commands to agents and sending it to the devices involved, thus *delegating* to them the actual execution of the task. Such execution is completely asynchronous, enabling the management station to perform other tasks in the meantime and introducing a higher degree of parallelism in the management architecture. In MbD, a portion of the functionality of the management station is actually decentralized and put directly onto the devices. Moreover, since the code fragments (which the authors claim can be written in any language) are not statically bound to devices, they can be changed and re-sent by the management station at any time. This enables more flexibility, because the management station can customize and enhance *dynamically* the services provided by the agents on the devices.

MbD can be considered a precursor of the ideas discussed here. Nevertheless, the paradigms described later cover different and broader aspects of mobility and can include MbD as a special case. Furthermore, they can be implemented by leveraging off of a number of languages providing abstraction expressly conceived for code mobility.

3 An Overview of Research on Code Mobility

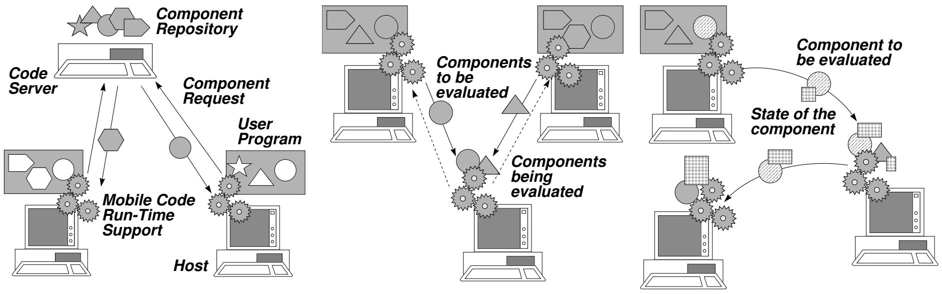
Interest in code mobility has been raised mainly by a new family of programming languages, usually referred to as *mobile code languages* (MCLs), that recently mushroomed out on the Internet, coming both from industry and academia. The idea behind these languages is to overcome the limitations of CS using new paradigms embodying a notion of code mobility. Presently, CS is used as a design paradigm for distributed applications in a myriad of application domains. In this paradigm, client and server are computational entities usually located on different hosts. Clients can access resources located on the server host by invoking remotely (e.g., through message passing or remote procedure call) services that comply with the interfaces provided by the server. The code describing how to perform the services (on the server) and how to request remote services (on the client) is bound statically to the host where it will be executed. Experience with the CS paradigm in large scale distributed systems has shown that enhancements are needed at least for the following concerns:

Server flexibility In CS applications, the set of services offered by the server is fixed, defined *a priori* by the application designer, and accessible through interfaces *defined statically*. The services provided or the particular interface may not be suitable for different unforeseen user needs. In addition, evolution of the application usually involves high-cost activities and may limit availability. For instance, in the network management approaches described so far, if the protocol, the information base structure, or the functionality of the agent are changed, the agent have to be extended and rebuilt. In other words, there is no way to dynamically extend the capabilities of the server side, in terms of code describing its actual behavior.

Bandwidth use Particular tasks involving continuous and intensive CS interaction through the network may result in bandwidth waste. For instance, monitoring stock information maintained by a server may require continuous polling by the client. As we described before, this is a major problem of micro management in SNMP where the management station polls continuously the agent, thus wasting bandwidth and, even worse, increasing network overloading.

Mobile code paradigms do not bind statically the code performing services to one or more hosts, rather they allow *migration* of code describing a service (and possibly the associated state of execution) to a different host, enabling the design of distributed *mobile code applications* (MCAs). At least three design paradigms are conceivable [1] to support this idea, namely:

Code on Demand (COD) The code that describes the behavior of a component of an MCA can change over time. A component running on a given host can download and link on-the-fly the code to perform a given task from a different (remote) component that acts as a "code server" (Fig. 2a). Hence, in a COD paradigm, the client owns the resources needed to perform a service, but lacks part of the code needed to perform it.



(a) Code on Demand. (b) Remote Evaluation. (c) Mobile Agent.

Fig. 2. Mobile code design paradigms.

Remote Evaluation (REV) In this paradigm, inspired by the pioneering work described in [8], any component of an MCA can invoke services provided by other components distributed among the hosts, like in a CS paradigm. The clients, in addition to the name of the service requested and the input parameters, send also the code describing the service to be executed. Upon execution on the server, this code will be allowed to access resources co-located with it (Fig. 2b). Hence, the clients own the code needed to perform a service, while the servers own the resources and offer a unique service constituted by the execution of client code.

Mobile Agent (MA) The term "agent" has presently many different meanings according to the research domains where it is used. In the scope of this work, a mobile agent is basically an *execution unit* (EU), like a UNIX process or a thread in a multi-threaded environment, that, while in execution at a given network node, is able to *migrate autonomously* to a different node and there resume execution seamlessly, as depicted in Fig. 2c. Hence, the agents in a MA paradigm own the code to perform a service, but do not own the resources needed to accomplish it. They are allowed to migrate their whole virtual machine on a different host (at least from a conceptual point of view), where the needed resources are located.

The paradigms above have been derived from the analysis of current MCLs, which provide for code mobility under two different forms, as discussed in [4]. If *strong mobility* is supported (by a *strong MCL*), the EUs coded in the MCL can move their code *and execution state* to a different host. The execution state contains the information related to the state of the EU, e.g., the instruction pointer. Migration is transparent, i.e., the EU resumes execution on the new host after the instruction that triggered migration. If *weak mobility* is provided (by a *weak MCL*), the EUs on a host can be bound dynamically to code coming from a different host. In currently available MCLs this encompasses two cases. Either the EU dynamically links code downloaded from the network, or the EU receives its code from another EU. In the latter case, two more options are possible. Either the EU in the destination site is created from scratch to run the incoming code, or a pre-existing EU links the incoming code dynamically and executes it.

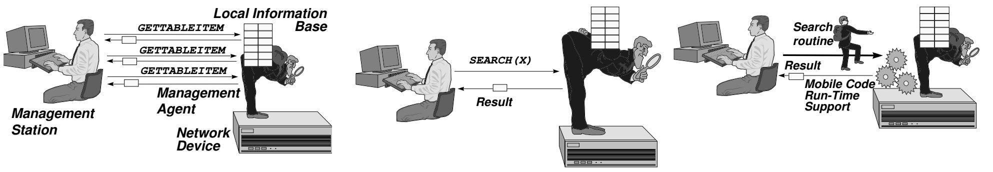
Available MCLs provide direct support for at least one of the above design paradigms, though the latter are at a higher abstraction level and do not necessarily rely on the availability of MCLs for their eventual implementation. For instance, strong MCLs support naturally the MA paradigm. In Telescript [19], a strong MCL, a special thread called *agent* can migrate to a different node by executing a `go` operation that suspends the execution of the thread, serializes it for transmission and sends it to the destination node, where it is resumed. In Agent Tcl [7], another strong MCL, the Tcl interpreter is extended to provide an additional `jump` command, whose effect is analogous to the Telescript `go`, except that it freezes the whole UNIX process containing the interpreter rather than a single thread. In turn, Java [14] is an example of a weak MCL that supports directly the COD paradigm. The Java class loader, responsible for resolving class names at run-time, can be modified by the programmer to retrieve the corresponding code from a remote host. Finally, TACOMA [9], Facile [11], M0 [17], and Mole [5] are weak MCLs that support the REV paradigm. In these MCLs, a procedure or function can be sent to a remote host, at most with the portion of the global environment needed for their remote execution.

4 Network Management and Code Mobility Put Together

In this section we discuss how code mobility can be successfully exploited for the provision of decentralized and flexible solutions for network management.

Code on Demand: Flexibility In the IETF approach the MIB structure is implemented by the code of the management agent and cannot be changed at run-time. This lack of flexibility represents a limit, in particular for event notification, because the definition of significant events is hard-wired in the MIB structure and customized events cannot be defined. On the contrary, the ISO approach allows the MIT to be changed dynamically, thus providing higher flexibility at the expense of an increased complexity in the management agents.

Code on Demand can increase flexibility while keeping agents simple and small, thus combining the advantages of IETF and ISO approaches. In this setting, an agent becomes an extensible component of an MCA. Whenever the structure of the local information base needs to be changed, e.g., to modify the criteria to detect an event, the corresponding portion of the agent code can be invalidated and replaced automatically with a new one retrieved from a code server. Hence, the agent can be "patched" without any need to rebuild and reinstall it. Even more important, the update need not to be done manually by the network administrator for each agent on each network device, rather it can be made once on the code server and propagated *on demand* on every node. Consequently, this approach can be exploited to perform *i) perfective* maintenance as in this case, *ii) adaptive* maintenance, i.e., to upgrade the agent code as a consequence of a change in the underlying software platform, e.g., a change in the operating system version. As an aside, it can be noted that a COD paradigm



(a) Client-server. (b) Client-server. (c) Mobile code.

Fig. 3. Three different ways to perform table lookup.

is likely to be of great help also in network domains other than network management. For example, a new protocol module could be added to a router in a fairly automated way by downloading it on demand from a central repository.

Remote Evaluation: Bandwidth Saving and Processing Distribution In mainstream network management approaches, data are moved from agents to the management station and there processed. Afterwards, some data are possibly sent back to agents. Again, this aims at keeping agents small, without overloading them with the code implementing the operations the management station is in charge of. As an example, let us consider the case in which the management station has to search for a single value in a table, which is commonly used to store information inside devices. With mainstream approaches this can be done in one of the following ways:

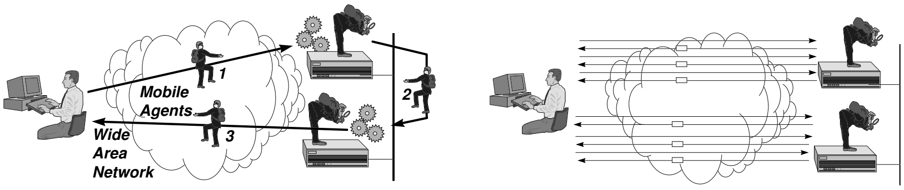
1. The whole table is transferred into the management station, where its rows are searched for the value (Fig. 3a). Hence, the larger the table the more bandwidth is wasted and the more computational overload is generated on the management station.
2. The searching routine can be implemented into the agent code (Fig. 3b) and activated either by a remote action invocation (ISO approach) or as a side effect triggered by a change of some value in the MIB (IETF approach). Again, the agent tends to become big and complex as a larger number of routines are implemented. Even worse, the payoff of this increased size may be very little if many of these routines are executed only now and then.

Using a REV paradigm, the search routine could be sent to the device on which the table is stored and executed there, as shown in Fig. 3c. Hence, the table could be searched locally and only the result should be sent back to the management station. This solution saves bandwidth, since only the result is shipped back, and distributes the computational load among the agents, rather than concentrating it on the management station. Another benefit that can be achieved by REV is to raise the level of abstraction for the interaction between the management station and the agents. As mentioned in Sect. 2, the operations available to the management station are usually very low level, particularly in the IETF approach, where the management station can only get and set values on the agents' MIB. Using REV, sequences of primitive operations that do not require a direct processing by the management station (e.g., to retrieve a sequence of values) can be grouped in higher-level operations and sent to the agent, where they are executed independently of the management station. One could envision a scenario where the manager builds her own management procedures upon lower-level operations, stores them on the management station and invokes their remote evaluation on the appropriate device, when needed. This brings a nice side-effect of modularity into the management architecture, and provides better performance by reducing the number of messages exchanged between the agent and the management station, thus limiting the load in the area around it. Management by delegation [6] which is subsumed in our paradigms, highlighted as a shortcoming the explicit invocation of remote execution on each node. We agree with this observation as we do not propose to exploit REV for the decentralization of every management task. The REV paradigm, as well as the other design paradigms proposed here, is not a silver bullet and its use has to be assessed case by case according to a set of parameters, specific to the functionality that has to be implemented. For example, in the searching example above, a REV paradigm is justified—from the point of view of the traffic generated on the network—only if the table to be searched is big and the search routine code (as it is likely) is short. Under these assumptions the bandwidth needed for management is reduced, otherwise CS will probably perform just as well.

Mobile Agent: Autonomous and Mobile Management Functionality The capability to send the same code consecutively to many nodes maintaining its state from one hop to another can be of great help in enhancing the decentralization of the management functionality over the managed network nodes. The management station, instead of querying each node to gather the information needed to perform some operation, could inject a mobile agent into the network. The mobile agent should have the goal of visiting all the nodes which have the needed information and processing it locally (thus performing semantic compression of data), retaining the state of its computation whenever it moves to a different node. This could be beneficial in:

- Increasing flexibility, since access to resources on network nodes is obtained through the mobile agents, whose behavior can be determined by the management station on a customized basis, like in the REV paradigm.
- Reducing traffic around the management station, because each query to the network device is performed locally by the mobile agent without involving a message on the net. Moreover, once in place the agent can perform its processing independently of the connectivity between the management station and the device—a useful feature for dealing with unreliable or lossy links.
- Distributing processing load, because operations usually performed by the management station are executed on the devices by the mobile agent.

The management station can leave even more freedom to the agent, omitting the nodes on which the agent can access the needed information and allowing the agent to determine them according to its current state. Nevertheless, as in the case of REV, the various options provided by code mobility must be assessed



(a) Mobile Agent. (b) Client-server.

Fig. 4. Interaction between management station and devices.

carefully before implementation. For example, if the agent must collect information about a set of nodes, its state (containing such information) is likely to grow as the agent travels to various nodes. Furthermore, if the agent is free to determine its own destination, its code is likely to be quite complex and produce a high overhead, since it must be carried at each node. Hence, in these conditions and when a great number of nodes are involved, MA may not be the right approach—at least from a traffic overhead perspective. Of course, communication overhead is not the only criteria involved in the decision; the topology and physical characteristics of the network fragment under responsibility of a mobile agent, the particular management task to be performed, and the requirements for flexibility and reconfigurability are likely to be driving factors as well. For example, consider a scenario where some of the managed devices reside in a LAN, on which bandwidth is plentiful and cheap. The management station is far from the managed devices and reachable only through a (large) number of expensive (slow) links. In this case, mobile agents bring a considerable benefit because they provide the capability of collecting information about the set of network devices in an autonomous fashion, without being connected to the management station (Fig. 4a). Even though their movement from device to device requires a lot of bandwidth, it takes place in the local area (step 2 in Fig. 4a) where bandwidth is not a concern. In the traditional approaches this operation would require many data exchanges on slow links between the management station and the devices (Fig. 4b) with higher latency and risk of partial failure.

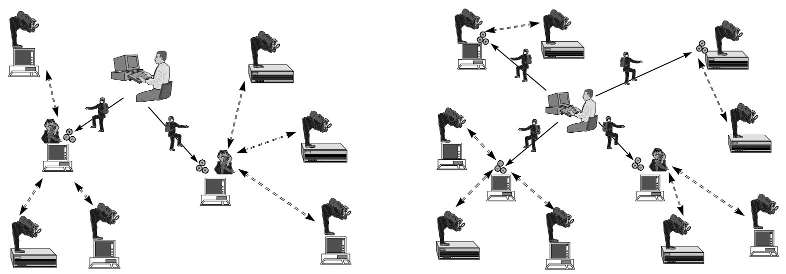
The mainstream approaches to network management presented in Sect. 2 exploit transport layer protocols to carry management messages. This allows the management station to communicate transparently with agents all over the network but, on the other hand, it makes management completely impossible when network level routing is disrupted. Mobile agents can help in overcoming this problem. If the management station does not have network level connectivity with a node to be managed through an agent, it can provide its agent with a route calculated from historical routing information and send it to the first hop on the route. Whenever the agent resumes execution on an intermediate hop, it tries to reach one of the next hops towards the target node using its internal route, until it reaches the target and performs its management task.

Finally, mobile agents can also be effective in detecting and bypassing routing loops. If the management station has clues to the presence of a loop, it can send a mobile agent to the first router towards the destination. This agent uses the information in its local routing table to move closer to the destination, while keeping track of the routers it is traversing. The presence of a routing loop is detected whenever the agent finds itself on a router already visited. At this point the mobile agent can try to: *i)* bypass the routing loop, trying to reach the destination through an interface chosen randomly, or *ii)* resolve the routing loop, e.g., disabling the interface on the loop and monitoring the reaction of the routing protocol. In principle, the management station could detect the routing loop on its own by reading the routing tables with the management protocol. Nevertheless, due to the presence of routing problems, it is likely that the management station does not have connectivity to some of the routers, and thus the partial or out-of-date information is not sufficient to identify the loop.

Feasibility Considerations and Interim Solutions The lack of network devices supporting direct execution of MCLs, which require a dedicated and often heavy run-time support, is today a major obstacle to the exploitation of code mobility in network management. Nevertheless, the opportunities offered by code mobility within the networking domain are becoming recognized by an increasing number of researchers and organizations, and the first efforts to embed code mobility at the network device level are beginning to appear [15]. In parallel, network device manufacturers are placing more and more computational power devoted to network management into their devices, thus enabling mobile code support to take advantage of existing hardware. For instance, the switches of some manufacturers are already equipped with a processor devoted to packet analysis and other functionalities required by RMON. Moreover, the problem of having a common format to ensure interoperability among the devices of different vendors, possibly using different languages, will exist. This problem is already evident in the mobile code arena, where the adoption of translation strategies based on an intermediate language has boosted portability but, on another hand, requires agreement on such a language in order to achieve full interoperability among different mobile code environments.

However, some interim solutions can be devised which, although less effective, are easier to implement and do not require support for code mobility on every node. These interim solutions could be a test-bed for code mobility in network management, in order to assess its effectiveness and possibly raise interest among the equipment manufacturers. Furthermore, they could be useful to manage legacy systems without built-in support for mobile code. In the remainder of this section we sketch briefly three architectures which can be used separately or in conjunction for this goals. A scenario where all the interim solutions are applied to the same network is shown in Fig. 5b.

1. Code mobility can be exploited to interact with SNMPv2 proxy agents running on workstations equipped with mobile code run-time support. In the interaction with proxy agents the management station gets the advantages previously described for the interaction with management agents. For example, management functions can be evaluated on the proxy agents' workstations (Fig. 5a) with a REV or MA paradigm, or the functionality of the proxy agent can be updated using a COD paradigm.



(a) Mobile code and proxy agents. (b) Mixed interim solutions.

Fig. 5. Interim solutions.

2. Mobile code could be sent to workstations with proper support which are installed in the managed area. As shown on the left in Fig. 5b, this code should interact with management agents on the nodes in the area around such workstations by using a conventional management protocol, e.g., SNMP or CMIP. This solution is not as beneficial as mobile code executing directly on network devices. Nevertheless, it enables local exchange of management protocol messages (e.g., useful on LANs with large capacity), having mobile code traveling from the management station to the managed areas. This solution is being implemented in a prototype study at our university.
3. When network devices will be equipped with mobile code run-time support (upper right part of Fig. 5b), this will be exploited in place of the workstations above. Mobile code will interact not only with the devices hosting the run-time support, but also with legacy ones installed in their neighborhoods, using conventional management protocols.

5 Conclusions and Future Work

In this work we investigated how new developments in design paradigms and technologies based on code mobility can be used to provide decentralized and flexible network management solutions, and discussed the benefits expected by their application. Nevertheless, we believe that network management would benefit from a combination of paradigms whose convenient use must be assessed case by case according to the functionality to be implemented, taking into account the CS paradigm as well. Finally, we discussed some of the obstacles to immediate applicability of these ideas on a large scale. Our work is at an initial stage, and we are currently investigating different research threads, including:

- *Development of a quantitative model* to identify which parameters of a network management application must be taken into account in order to select the most convenient approach, along the lines depicted in [1].
- *Prototyping with mobile code technology*. We are presently investigating the use of code mobility paradigms with one of the interim solutions sketched earlier using Agent Tcl [7] as our implementation language.
- *Probing of quantitative data* to demonstrate the effectiveness of the approach with respect to generated traffic.

Acknowledgments The taxonomy for code mobility discussed here is joint work of one of the authors with G. Vigna and other people at Politecnico di Milano.

References

1. A. Carzaniga, G.P. Picco, and G. Vigna. Designing Distributed Applications with a Mobile Code Paradigm. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, 1997. To appear.
2. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of Management Information for version 2 of the Simple Network Management Protocol. RFC 1902, January 1996.
3. J. D. Case, M. Fedor, M. L. Schoffstall, and C. Davin. Simple Network Management Protocol. RFC 1157, May 1990.
4. G. Cugola, C. Ghezzi, G.P. Picco, and G. Vigna. Analyzing Mobile Code Languages. In [16].
5. M. Straßer, J. Baumann, and F. Hohl. MOLE—A Java Based Mobile Agent System. In [16].
6. G. Goldszmidt and Y. Yemini. Distributed Management by Delegation. In *15th International Conference on Distributed Computing*, June 1995.
7. R.S. Gray. Agent Tcl: A transportable agent system. In *Proceedings of the CIKM'95 Workshop on Intelligent Information Agents*.
8. J. W. Stamos and D. K. Gifford. Remote Evaluation. *ACM Transactions on Programming Languages and Systems*, 12(4):537–565, October 1990.
9. D. Johansen, R. van Renesse, and F.B. Schneider. An Introduction to the TACOMA Distributed System - Version 1.0. Technical Report 95-23, University of Tromsø and Cornell University, June 1995.
10. K. Jones. Internet's SNMP and ISO's CMIP Protocols for Network Management. *International Journal of Network Management*, pages 130–137, September 1994.
11. F.C. Knabe. Language Support for Mobile Agents. Technical Report ECRC-95-36, European Computer-Industry Research Centre, Germany, December 1995.
12. T. Magedanz, K. Rothermel, and S. Krause. Intelligent Agents: An Emerging Technology for Next Generation Telecommunications? In *INFOCOM'96*, San Francisco, CA, USA, March 1996.
13. OSI. ISO 9595 Information Technology, Open System Interconnection, Common Management Information Protocol Specification, 1991.
14. Sun Microsystems. *The Java Language Specification*, October 1995.
15. D.L. Tennenhouse and D.J. Wetherall. Towards an Active Network Architecture. *ACM SIGCOMM Computer Communication Review*, 26(2), April 1996.
16. C. Tschudin and J. Vitek, editors. *Mobile Object Systems*, Lecture Notes on Computer Science. Springer-Verlag, 1997. To appear.
17. C. F. Tschudin. *An Introduction to the* `MO` *Messenger Language*. University of Geneva, Switzerland, 1994.
18. S. Waldbusser. Remote Network Monitoring Management Information Base. RFC 1757, February 1995.
19. J.E. White. Mobile Agents. In *Software Agents*. MIT Press, 1996.
20. Y. Yemini. The OSI Network Management Model. *IEEE Communications Magazine*, pages 20–29, May 1993.