Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications

Mario Baldi
Dip. Automatica e Informatica
Politecnico di Torino
C.so Duca degli Abruzzi, 24
I-10129, Italy
+39-11-564-7067
`mbaldi@polito.it`

Gian Pietro Picco
Dip. Automatica e Informatica
Politecnico di Torino
C.so Duca degli Abruzzi, 24
I-10129, Italy
+39-11-564-7008
`picco@polito.it`

ABSTRACT
The question of whether technologies supporting mobile code are bringing significant benefits to the design and implementation of distributed applications is still an open one. Even more difficult is to identify precisely under which conditions a design exploiting mobile code is preferable over a traditional one. In this work, we present an in-depth evaluation of several mobile code design paradigms against the traditional client-server architecture, within the application domain of network management. The evaluation is centered around a quantitative model, which is used to determine precisely the conditions for the selection of a design paradigm minimizing the network traffic related to management.

KEYWORDS
Mobile code, network management, management by delegation, mobile agent, remote evaluation.

1 INTRODUCTION

Recently, a new approach to the development of distributed applications has become popular, fostered by the availability of a new generation of programming languages and systems which provide the capability to relocate application code across the nodes of a computer network. Among these systems, often called *Mobile Code Systems* (MCS), Java is surely the best known, to the point that many vendors of network equipment already announced on-chip Java support. It is still unclear how to exploit the new technologies, concepts, and paradigms enabled by these systems, and which are the advantages that can be potentially benefited [5]. Many researchers suggest that a major benefit provided by mobile code is the capability to reduce network communication by moving client's knowledge close to server's resources, thus accessing them locally. These arguments are often supported only by qualitative and superficial considerations, rather than by a careful analysis of the problem, possibly supported by quantitative evidence.

Previous work [2] presented a characterization of mobile code that abstracted from the details of the different technologies by deriving design paradigms that can be adopted for structuring distributed applications. Also, it suggested that the exploitation of a mobile code design is not an advantage per se, rather it must be evaluated as an alternative to traditional design in the context of the particular application being designed. This work builds on the aforementioned results, by showing an in-depth quantitative analysis of the tradeoffs among different architectures exploiting mobile code. This is done in the real application domain of network management. We present a quantitative model for traditional and mobile code design of network management functionalities, and we show how to determine the optimal design paradigm according to the model parameters.

The contribution of this work can be regarded under different perspectives. From a software engineering perspective, it constitutes an experience in evaluating design alternatives from a quantitative point of view. From the network management perspective, it provides managers with a formal tool to determine the best way to perform management operations. Finally, it provides formal arguments for the pro and cons of mobile code in a relevant application domain.

The paper is structured as follows. Section 2.1 provides some background information on network management and Section 2.2 motivates a mobile code approach in this context. Section 2.3 describes research on code mobility and its application to network management in particular. Section 3 develops a model of traffic which is used to compare the various paradigms. This model is then refined in Section 4 to take into account non-uniform networks. Finally, a case study is provided in Section 5, where the model is instantiated with parameters taken from a real implementation being developed at our university and the tradeoffs of the paradigms are examined for a specific management functionality. Some final remarks are provided in Section 6.

2 BACKGROUND

2.1 Network Management

Network management is split in two nearly separate worlds [7]: IETF management, which relies on the Simple Network Management Protocol (SNMP) and its derivatives, and ISO management, which relies on the Common Management Information Protocol (CMIP). Both protocols assume a centralized management architecture based on a client-server paradigm. *Management agents* co-located with network devices act like servers that communicate on request device data to a network management station (NMS). Device data are stored in a *management information base* (MIB)[1], a hierarchical base of information managed by each agent. In this setting, all the computation related to management, e.g., statistics, is demanded to the NMS. The operations available to the NMS for accessing a MIB are very low level. In SNMP, for instance, the NMS can only `get` and `set` atomic values in the MIB. This fine grained client-server interaction is often called *micro-management*, and leads to the generation of intense traffic and computational overload on the NMS. This centralized architecture is particularly inefficient during periods of heavy congestion, when management becomes important. In fact, the NMS increases its interactions with the devices and possibly uploads configuration changes, thus increasing congestion. In turn, congestion, as an exceptional status, is likely to trigger notifications to the NMS which worsen network overload. Due to this situation, access to devices in the congested area becomes difficult and slow.

Centralization is being addressed by IETF and ISO by adapting their management architecture, as with SNMPv2 [3]. However, experimentation showed that these new proposals do not provide yet the desired level of decentralization needed to cope with large networks. The interested reader can find an extensive survey of approaches to network management in [9].

2.2 Why Mobile Code in Network Management?

A first issue is the possibility to perform *semantic compression* of information. Suppose a table in the MIB has to be searched for a value matching some criteria. In a traditional approach, a search function can be executed by the NMS on the elements retrieved from the remote agent. In the worst case, all the elements in the table need to be transferred on the NMS. Placing the search function on the agent, rather than on the NMS, enables semantic compression. The table can be searched locally on the device, where it is stored, and only the final value is sent across the network. As a pleasant side-effect, a *higher abstraction level* is provided to the network manager, who can now think about the function as provided directly by the management agent, without being aware of its implementation in terms of elementary primitives provided by the management protocol. This idea can be pushed even further, giving *autonomy* to management agents. In traditional approaches, the only action agents can perform on their own is the generation of an *event* (or *alarm* in the OSI jargon), i.e. a signal which is sent autonomously to the NMS when a given condition is satisfied. Events are usually simple and predefined and the actual response to them is completely up to the NMS. In general, agents can be extended to check arbitrarily complex event and possibly react locally without requiring the intervention of the NMS.

In principle, the extension of a management agent can be achieved with traditional technology based on the client-server paradigm, like in several CORBA-based approaches; this solution presents some relevant drawbacks. First of all, one could argue that the reason why the primitives offered by management agents are so poor is that agents must be lightweight, since they can be embedded in network devices equipped with limited computational resources. In this setting, adding management primitives to agents is just not desirable. Even if adequate computational resources are available, another relevant issue is the frequency of invocation of primitives. A function may be used only every now and then, e.g., because its execution is needed only during periods of heavy congestion, or because its code may change slightly according to operating conditions. In these cases, hard-wiring the function into the agent just makes it bigger, wasting device resources without appreciable gain. In general, the dynamic nature of management operations demands for dynamic customizability of agent primitives.

Mobile code provides the technology needed to enhance network management with the degree of *flexibility* needed to cope with the problems above. Mobile code can be linked dynamically on network devices either proactively by the NMS or reactively by the network device. This way, the management primitives embedded in it become available on the device only when requested by management operations, thus consuming device resources only when this is really needed. Furthermore, the capability to embed in the mobile code the strategy for migration, provides an extra level of autonomy to management operations. Hence, a NMS can extend dynamically a management agent with mobile code, thus improving flexibility, and delegate to such code the capability to migrate autonomously on different network devices to perform complex management tasks without involving the NMS.

[1]MIB is actually the term used in SNMP only. CMIP uses the term *management information tree* (MIT) database, instead. Hereafter, we ignore the difference for the sake of simplicity.

2.3 State of the Art

Existing mobile code technology supports code mobility in different ways[2]. Code mobility can be described at a higher level of abstraction, without delving into the details of the technology. In this work, we focus on the description provided in [2]. Here, mobile code design paradigms are identified as the architectural styles that can be used to exploit code mobility in the design of a distributed application. Design paradigms are characterized by components and the interactions between them. A component may have a flow of control or it may be an architectural element representing passive data and physical devices. In the first case, it is called a *computational component*, otherwise it is called a *resource*. A particular kind of resource is a *code component*, which contains the code necessary for the execution of a given task. Components are hosted by *sites*, the architectural abstraction that provides support for component execution and access. A service needs access to resources, and its behavior (the know-how about the service) is described by a code component which is executed by some computational component in the architecture. In order for actual computation of the service to take place, these three elements must be co-located in the same site. Interactions between components rule the relocation of components needed for service execution. Three mobile code paradigms are identified using these abstractions, and compared with the traditional client-server paradigm.

In the *Client-Server* (CS) paradigm, a client component needs to execute a specified service, but lacks both the resources and know-how needed. Instead, a server component owns both code and resources, and can perform the service on behalf of the client. To do so, the client requests the service to the server, which ships back the results after execution. In the *Code On Demand* (COD) paradigm, the client component owns the resources needed by the service execution although it lacks the corresponding code component, available on the server. Hence, it requests the code to the server component and, after the code has been received, executes the service locally. In the *Remote EValuation* (REV) paradigm, the client component owns the know-how about the service, and lacks the corresponding resources, which are owned by the server component. Thus, the client sends a request to the server including the code component needed to perform the service. This is executed by the server, that exploits local access to the resources needed, and ships back the results. When the *Mobile Agent* (MA) paradigm is used, the client component knows how to perform the service, but lacks the resources. Unlike REV, the whole computational component is migrated to the site where the resources reside. There, the component performs service execution with local access to resources.

The approaches to network management that involve code mobility are usually grouped under the label *management by delegation*. Management by delegation has been conceived originally before the appearance of Internet-based MCSs, as a means to cope with the micro-management phenomenon of centralized management. The original proposal, further extended in [6], identifies an architecture for the dynamic uploading of management scripts on network devices using a combination of the REV and CS paradigms. In this architecture, management scripts are *delegated* to perform management operations on the device on behalf of the NMS. Clearly, this assumes the presence of a specialized run-time support on the network device, capable of executing mobile code. However, in management by delegation code migration is always triggered by the NMS, and there is no support for autonomous mobility of delegated scripts. The original idea has been interpreted in many ways by researchers leading to prototypes which are often based on relatively limited "mobile code" technology, like interpreted Perl scripts. Management by delegation is presently under standardization by IETF and ISO working groups [10]. As discussed in [1], the characterization of mobile code made earlier shows that management by delegation can benefit from the recent developments in code mobility.

3 MODEL OF MANAGEMENT TRAFFIC

In this section, we derive a model for the traffic generated by network management. It differentiates among the design paradigms described in Section 2.3 that can be exploited to implement a given functionality. The model is conceived to guide the manager during the selection of the right design choice for a management functionality. However, the model presented in this section is a general one, and has to be adapted to the management scenario where the manager operates. This scenario must take into account the actual management protocols in place, as well as the technology actually used to implement a given paradigm. An example of such a refinement is provided in Section 5.

In what follows, a measure of the dimension of the managed network is given in terms of the number N of managed devices. The complexity of the management task is taken into account by the number Q of queries performed on the MIB. The transmission overhead introduced by protocol encapsulation, and possibly traffic control or connection setup, is taken into account down to the network layer, as defined by the OSI model. In fact, the focus is on the traffic gener-

[2] A classification of mobile code technologies, design paradigms, and applications can be found in [5]. Moreover, [4] analyzes the impact of mobility on the design of a programming language.

ated on the network as a whole; the amount of data exchanged between network layer entities is independent of the lower layer technology, and thus invariant throughout the whole network. If a chunk of data of size X is to be transmitted at the application level, we represent the actual amount of data exchanged at the network layer as $X' = \alpha(X) + \beta(X)X$. In this expression, $\beta(X)$ may account, for example, for the overhead introduced by message encapsulation, while $\alpha(X)$ may account for the control information exchanged during the setup phase in a connection-oriented protocol. In the remainder of the section, however, we use the equivalent expression $X' = \eta(X)X$ where $\eta(X) = \frac{\alpha(X)}{X} + \beta(X)$, $\eta(X) > 1$. This analytical transformation does not change the meaning of the model; it just provides a more compact representation. $\eta(X)$ is called *overhead function* since accounts for the control information (*protocol overhead*) added to X by the network and the above layers. In the following, we write simply ηX in place of $\eta(X)X$, in order to simplify formulae.

3.1 Model of the Overall Traffic

We derive here a model for the traffic generated by a management task which involves retrieving a set of data from managed devices. In devising this model, we assume that the same management operations are executed on each device. This is not necessarily the case for every management task, but allows for a simpler notation without compromising the generality and significance of what can be inferred from the model.

3.1.1 Client-server With the traditional client-server paradigm, the NMS requests an operation to the management agent by sending a request message to it. We assume that such information has size I_q. In order to perform the management task, Q request messages must be sent to each of the N managed devices. Device n answers the q^{th} request with a reply whose size is R_{qn}. The overall traffic is then:

$$T_{CS} = \sum_{n=1}^{N}\sum_{q=1}^{Q}(\eta_{CS} I_q + \tilde{\eta}_{CS} R_{qn}) \qquad (1)$$

where we differentiate between the overhead function used to send message requests (η_{CS}) and the one used for replies ($\tilde{\eta}_{CS}$). Multicast communication is sometimes used to send the same management request to multiple devices. In our model we do not consider this case as it is not very common. Moreover, we assume that the queries contained in each request message are dependent on the previous reply, i.e. it is not possible to aggregate request messages.

3.1.2 Remote Evaluation If the REV paradigm is exploited, the Q requests are embedded in a code fragment of size C_{REV} sent on managed device n. Remote evaluation of the code produces the Q results R_{qn} which are sent back collectively to the NMS. This pairwise interaction has to take place for each of the N managed nodes. The overall traffic generated is given by

$$T_{REV} = \sum_{n=1}^{N}(\eta_{REV} C_{REV} + \tilde{\eta}_{REV} \sum_{q=1}^{Q} R_{qn}) \qquad (2)$$

3.1.3 Mobile Agent The NMS unleashes a mobile component that visits each of the N nodes and collects information locally. When modeling such a component, we model the code and the portion of the state needed for its execution (C_{MA}) as separate from the portion of the state relevant to the application. The latter grows as long as this agent travels from node to node. In fact, if we denote with $S_{MA,n}$ the size of the state of the agent during the trip towards node n, then

$$S_{MA,n} = \begin{cases} 0 & \text{if } n = 1 \\ \sum_{m=1}^{n-1}\sum_{q=1}^{Q} R_{qm} & \text{if } n > 1 \end{cases}$$

The first expression accounts for the fact that the mobile agent has not yet collected any information when traveling from the NMS to the first node. The second one accounts for the fact that, when traveling to node n, the agent already carries all the replies collected on the previous $n-1$ nodes. After information on the last node has been collected, the mobile agent sends back to the NMS all the results collected. An alternative design would return the component back to the NMS. Although sometimes viable, we chose the first solution for uniformity with the REV case. The overall traffic generated is given by

$$T_{MA} = \sum_{n=1}^{N}\eta_{MA}(C_{MA} + S_{MA,n}) + \tilde{\eta}_{MA}\sum_{n=1}^{N}\sum_{q=1}^{Q} R_{qn} \qquad (3)$$

3.1.4 Code On Demand With the COD paradigm, management agents can be augmented dynamically with code implementing primitives at a higher level of abstraction, thus providing agents with the capability to perform the Q operations locally, rather than across the network. The protocol we assume in this model is the following. As in the CS paradigm, the NMS requests an operation by sending a message that contains the operation signature, I_{COD}. If the operation has already been installed on the managed node, a reply is sent which contains the result of the Q queries, like in a REV implementation. On the other hand, if the code for the operation has not been installed yet, the agent replies with a message (that we assume of size I_{fetch}) requesting the dynamic download. The code, of size C_{COD}, is transferred and linked on the agent device, where it becomes available for future invocations, and the corresponding operation is performed. Consequently, the expression of traffic at equilibrium is:

$$T_{COD,stable} = \sum_{n=1}^{N}(\eta_{COD} I_{COD} + \tilde{\eta}_{COD}\sum_{q=1}^{Q} R_{qn}) \qquad (4)$$

By converse, during the "setup" phase, there is an overhead represented by the message sent by the agent which requests the download, plus the actual code transfer: $T_{COD} = T_{COD,stable} + T_{COD,setup}$, where[3]

$$T_{COD,setup} = \sum_{n=1}^{N} (\eta_{COD} I_{fetch} + \eta_{COD} C_{COD}) \quad (5)$$

3.1.5 Evaluation We elaborate on the expressions given earlier in order to find out the tradeoffs that must be considered in order to minimize the overall traffic. To this end, we compare the equations determined for the various paradigms.

For instance, the use of a REV paradigm is an improvement over traditional centralized management only if $T_{CS} \geq T_{REV}$. After elaboration of (1) and (2):

$$N \sum_{q=1}^{Q} \eta_{CS} I_q + \sum_{n=1}^{N} \sum_{q=1}^{Q} \tilde{\eta}_{CS} R_{qn} \geq N \eta_{REV} C_{REV} + \sum_{n=1}^{N} \tilde{\eta}_{REV} \sum_{q=1}^{Q} R_{qn}.$$

In order to simplify notation and have a better insight into the meaning of the formulae, we introduce some assumptions. Instead of differentiating the contribution of each request I_q and reply R_{qn} in (1), we consider the average values $\bar{I}$and $\bar{R}$. Consequently, the formula can be rewritten as:

$$NQ\eta_{CS}\bar{I} + NQ\tilde{\eta}_{CS}\bar{R} \geq N(\eta_{REV} C_{REV} + \tilde{\eta}_{REV} Q\bar{R})$$

It is likely[4] that $Q\tilde{\eta}_{CS}\bar{R} \geq \tilde{\eta}_{REV} Q\bar{R}$ since usually a fixed overhead is associated to each packet and thus, the longer the message being segmented, the smaller the relative overhead. Hence, if more results $\bar{R}$ can be transmitted together in a single message, the overhead is likely to be smaller, although depending on the protocols used to implement communications in CS and REV. We call the difference in the overhead introduced to send the results of the queries $\Delta O_{CS,REV} \geq 0$; REV is more convenient than CS if

$$\eta_{REV} C_{REV} \leq \Delta O_{CS,REV} + Q\eta_{CS}\bar{I}, \quad (6)$$

that is, if the size of the message containing the code to be evaluated remotely is smaller than the overall size of the message requests needed in a CS paradigm plus the difference in overhead introduced when transmitting the result. Clearly, REV is convenient when the number of instructions Q needed to perform a query is high and C_{REV} effectively compacts the representation of the local interactions I_q within the code, e.g., using loop control structures.

We can apply the same reasoning and assumptions in order to determine which is the most convenient paradigm between REV and MA. It can be seen that a MA implementation always generates more traffic than a REV one. In fact, substituting and elaborating (2) and (3) in $T_{MA} \geq T_{REV}$ within the assumptions above, we obtain:

$$N\eta_{MA} C_{MA} + \tilde{\eta}_{MA} QN\bar{R} + \sum_{n=1}^{N} \eta_{MA} Q(n-1)\bar{R} \geq N(\eta_{REV} C_{REV} + \tilde{\eta}_{REV} Q\bar{R}).$$

A first simplification is to assume

$$N\tilde{\eta}_{REV} Q\bar{R} \simeq \tilde{\eta}_{MA} QN\bar{R}. \quad (7)$$

This assumption holds when Q is sufficiently large and $\tilde{\eta}_{REV} \simeq \tilde{\eta}_{MA}$: in this case, more results are packed together for both, rather than being sent individually like with CS, and the difference in overhead is likely to be negligible. A more gross simplification is to consider $C_{REV} \simeq C_{MA}$. Usually is $C_{REV} < C_{MA}$, since C_{MA} contains the execution state as well as the code determining the next migration. If we apply this oversimplification the equation above becomes

$$\sum_{n=1}^{N} \eta_{MA} Q(n-1)\bar{R} \geq 0,$$

that is, REV is always more convenient than MA, because the latter must carry the state which is growing at every hop. As we discuss in Section 3.3, the possibility of performing semantic compression introduces a different evaluation criteria.

The application of the COD paradigm depends on the frequency of invocation which has not been considered yet. So far, we have given the expression of the traffic generated for a single execution of a management task. However, in general it may be interesting to consider how this varies over U different invocations. For the other paradigms, this additional parameter does not affect the expression of the traffic. The traffic generated during U executions of an implementation with a paradigm $p \in \{\text{CS}, \text{REV}, \text{MA}\}$, $T_p(U) = UT_p$. The expression of $T_{COD}(U)$ when COD is used is

$$T_{COD}(U) = T_{COD,setup} + UT_{COD,stable}$$

Calculation of $T_{REV}(U) \geq T_{COD}(U)$ under the likely assumptions that $\bar{I} \simeq I_{COD} \simeq I_{fetch}$ and $\tilde{\eta}_{REV} Q\bar{R} \simeq \tilde{\eta}_{COD} Q\bar{R}$ yields

$$\eta_{REV} C_{REV} \geq \frac{U+1}{U} \eta_{COD} \bar{I} + \frac{1}{U} \eta_{COD} C_{COD}$$

Clearly, if U is large, i.e. the primitive is invoked many times before being upgraded or discarded, the disequation above can be approximated by

$$\eta_{REV} C_{REV} \geq \eta_{COD} \bar{I}$$

which is always satisfied, the threshold being a REV code composed by a single instruction. If, by converse, U is small, all the terms must be considered. However, we can assume that, although shipped with a different paradigm, the code describing the management function at hand is the same, i.e. $C_{REV} = C_{COD} = C$, and that we are comparing implementations of different paradigms with the same technology, i.e. $\eta_{REV} = \eta_{COD} = \eta$. Under

[3]We do not model the intermediate situation where some of the nodes already contain the code, since we do not consider it relevant to the goals of this paper and can be easily derived from the result presented.

[4]It must be reminded that we use ηX as a shortcut for $\eta(X)X$.

these assumptions, we obtain

$$\eta C \geq \frac{U+1}{U-1}\eta \bar{I}$$

which confirms the intuition that if a function is used at least two times in a row, caching its code saves bandwidth. Finally, comparing $T_{CS} \geq T_{COD}$ under the assumption $\bar{I} \simeq I_{COD} \simeq I_{fetch}$, we obtain that

$$U \geq \frac{\eta_{COD}\bar{I} + \eta_{COD}C_{COD}}{Q\eta_{CS}\bar{I} - \eta_{COD}\bar{I} + \Delta O_{CS,COD}} \qquad (8)$$

where $\Delta O_{CS,COD}$ is the difference in the overhead introduced by the two implementations to send back the results.

3.2 Model of the Traffic around the NMS

As we pointed out in Section 2.1, the NMS is likely to represent a bottleneck of the management system in a centralized approach, due to the micro-management phenomenon. Thus, although the measure of the overall traffic generated is surely relevant, it is important to compare the performance of the different paradigms also in terms of the traffic generated from and to the NMS.

The measure of the overall traffic generated by management operations and the measure of the management traffic flowing through the NMS coincide for all the paradigms considered, except for the MA paradigm. In fact, in the other paradigms, the NMS is always interacting directly with all the devices being managed. Instead, when the MA paradigm is used, there is traffic through the NMS only when the mobile agent is injected into the network and when the agent reports back the collected results, that is

$$T_{MA,Mgm} = \eta_{MA}C_{MA} + \tilde{\eta}_{MA}\sum_{n=1}^{N}\sum_{q=1}^{Q} R_{qn}. \qquad (9)$$

The remainder of the traffic is generated without involving the NMS. It is useful to compare $T_{REV} \geq T_{MA,Mgm}$. If we consider $\bar{R}$ instead of the single contributions, and apply assumption (7), then

$$\frac{\eta_{MA}C_{MA}}{\eta_{REV}C_{REV}} \leq N$$

holds, i.e. MA is convenient with respect to REV if the ratio between the traffic generated when unleashing the mobile agent and the one to move the code to be evaluated remotely is smaller than the number of nodes being managed. Notably, for a single node REV is going to be more convenient, as usually $C_{MA} > C_{REV}$. Here, the tradeoff is between the size of the code of the mobile agent and its autonomy as far as mobility is concerned.

A comparison with COD at the equilibrium point, within the assumptions enumerated before, yields $T_{COD,stable} \geq T_{MA,Mgm}$, that is,

$$\frac{\eta_{MA}C_{MA}}{\eta_{COD}\bar{I}} \leq N.$$

Hence, the MA paradigm is more convenient than COD only if the size of the code and execution state of the mobile agent are sufficiently small. If we assume that the size of a single instruction in the code of the mobile agent is $\bar{I}$, its code must contain a number of instructions smaller than the number of nodes being managed.

As for CS, always within the assumptions above, $T_{CS} \geq T_{MA,Mgm}$ if

$$\Delta O_{CS,MA} \geq \eta_{MA}C_{MA} - QN\eta_{CS}\bar{I},$$

that is, if the difference of the protocol overhead generated to send the results in the two cases is greater than the difference between the size of the code of the mobile agent and the overall size of request message. Note that, if $QN \gg 1$, the right hand side is likely to be negative, thus always verifying the inequality.

3.3 Semantic Compression

As we mentioned in Section 2.2, in many management tasks the size of the results sent back to the NMS can be reduced by semantic compression.

The CS paradigm exploited by traditional approaches, where the primitives offered by the agent are often low level and anyway fixed, does not allow semantic compression to be performed whenever possible. A COD paradigm, on the other hand, may enable semantic compression by installing a proper functionality on the management agent. In this case,

$$T'_{COD,stable} = \sum_{n=1}^{N}(\eta_{COD}I_{COD} + \tilde{\eta}_{COD}R_n)$$

where we assumed that semantic compression of q data values R_{qn} yields a single value R_n. Clearly, $T_{COD,setup}$ is unchanged. The REV and MA paradigms can achieve semantic compression by executing remotely a routine that performs the compression, like the search routine described in Section 2.2. This generates a traffic

$$\begin{aligned} T'_{REV} &= \textstyle\sum_{n=1}^{N}(\eta_{REV}C_{REV} + \tilde{\eta}_{REV}R_n) \\ T'_{MA} &= \textstyle\sum_{n=1}^{N}[\eta_{MA}(C_{MA} + S'_{MA,n}) + \tilde{\eta}_{MA}R_n] \end{aligned}$$

and $T'_{REV} < T'_{MA}$, since in the expression of the traffic generated by MA there is still a term $S'_{MA,n}$ which grows linearly with the number of nodes visited and, in addition, usually $C_{REV} < C_{MA}$.

If we consider only the management traffic involving the NMS, like in the previous section, we obtain that

$$T'_{MA,Mgr} = \eta_{MA}C_{MA} + \tilde{\eta}_{MA}\sum_{n=1}^{N} R_n \qquad (10)$$

To evaluate the improvement introduced by semantic compression, we compute the difference $\Delta T'_p$, $p \in \{\text{REV}, \text{MA}, \text{COD}\}$, between the traffic generated with the traditional CS paradigm and the traffic generated in the situations above. In doing this, we apply the usual approximation of considering average sizes $\bar{I}$ and $\bar{R}$ for instructions and results, respectively. For the sake of simplicity, we consider a unique overhead function η across all the paradigms, without differentiating between requests and replies. For mobile code paradigms, this is not a big approximation if we restrict the evaluation to implementations that use the same application level protocol for transferring both the code and the results. The real approximation is introduced in considering as equivalent the overhead introduced by a particular MCS (e.g, Java Aglets) and the overhead introduced by a traditional CS-based management protocol (e.g., SNMP). Under this assumptions, however, we obtain:

$$\begin{aligned}\Delta T'_{COD,stable} &= N(Q-1)(\eta\bar{I}+\eta\bar{R})\\ \Delta T'_{REV} &= NQ\eta\bar{I}+N(Q-1)\eta\bar{R}-\eta C_{REV}\\ \Delta T'_{MA,Mgm} &= NQ\eta\bar{I}+N(Q-1)\eta\bar{R}+\Delta O_{CS,MA}-\eta C_{MA}\end{aligned}$$

After some trivial manipulations of the formulae above, we can express the above as functions of $\Delta T'_{COD,stable}$:

$$\begin{aligned}\Delta T'_{REV} &= \Delta T'_{COD,stable}+N(\eta\bar{I}-\eta C_{REV})\\ \Delta T'_{MA,Mgm} &= \Delta T'_{COD,stable}+N\eta\bar{I}+\Delta O_{CS,MA}-\eta C_{MA}\end{aligned}$$

These formulae show that REV is never better than COD when the management task is repeated a number of times large enough to neglect the COD setup traffic. This is not true only if $\eta C_{REV} \leq \eta\bar{I}$, which is clearly unlikely, as it means that the message containing the code being evaluated remotely is smaller than a simple request message. The traffic around the NMS is reduced with respect to COD if

$$\eta C_{MA} \leq N\eta\bar{I}+\Delta O_{CS,MA},$$

which highlights that, if the code of the mobile agent is kept sufficiently compact, the gain in traffic grows linearly with the number of nodes.

The MA paradigm enables also a *global* form of semantic compression across all the devices, while the other paradigms enable only a *local* semantic compression on each device. For example, a mobile agent may travel across a set of nodes looking for the most loaded network interface. In doing this, only one data value needs to be carried from hop to hop—the state of the agent does not increase. In other words, if we assume that such data value has a fixed size $\bar{R}$, then $S_{MA,n} = \bar{R}$, and

$$T''_{MA} = N(\eta_{MA}C_{MA}+\tilde{\eta}_{MA}\bar{R}).$$

Again, MA is not more advantageous than the COD paradigm, at least as far as overall traffic is concerned. The calculation of $\Delta T''_{MA} \geq \Delta T'_{COD}$, assuming that the overhead function is the same, shows that this condition is met only if $\eta C_{MA} \leq \eta\bar{I}$ which, as we discussed earlier for local compression with REV, is practically never met. The real advantage of global compression, however, shows up when considering traffic involving the NMS. In this case, the traffic is simply

$$T''_{MA,Mgm} = \eta_{MA}C_{MA}+\tilde{\eta}_{MA}\bar{R} \qquad (11)$$

and calculation of $\Delta T''_{MA} \geq \Delta T'_{COD}$ shows that MA is better than COD if

$$\eta C_{MA} \leq N(\eta\bar{I}+\eta\bar{R})-\eta\bar{R}$$

The first addendum actually represents a whole CS interaction: basically, the message containing the code of the mobile agent must be smaller than N pairs of request and reply messages.

4 MODEL OF MANAGEMENT COSTS

The model devised so far is meaningful only for a uniform network; for example it applies to the management of a single LAN. However, corporate intranets and extranets spanning across long-haul links with different characteristics, require an explicit modeling of communication costs. A natural way to relate network traffic with the communication cost associated to the links is to assign to each link a weighting cost coefficient $0 \leq \lambda_l \leq 1$. The value of the cost coefficient has to be determined by the manager according to the notion of cost associated to the link, and may be actually a combination of several factors. For instance, a high cost may be due to high latency or low-bandwidth on the link, or to the fact that a link connected to the NMS should be kept as unloaded as possible, or to security considerations.

The formulae derived so far are still valid in the case of management of a single LAN (Fig. 1.a). We approximate the LAN as a shared media or as a mesh of identical cost paths between any pair of nodes. The same coefficient λ is associated to each link, and the overall cost of performing a given management task is $K = \lambda T$, where T can be each of the expressions derived in the previous section.

A common case that involves differentiation of costs is management of a remote LAN, shown in Fig. 1.b. In this case, the managed devices are placed in a high-speed network whose links are characterized by the same coefficient λ. The LAN has an entry point, e.g., a router, to which the NMS is connected through a link[5] with different, and most likely higher, coefficient λ_0. Thus, the cost of reaching any device from the NMS is weighted by a coefficient $(\lambda_0 + \lambda)$. In this situation, the cost for any management task performed using a paradigm $p \in \{\mathrm{CS}, \mathrm{REV}, \mathrm{COD}\}$ is

$$K_p = (\lambda_0 + \lambda)T_p.$$

On the other hand, for the MA paradigm the expression of the cost is actually a generalization of the traffic $T_{Ma,Mgm}$ computed in the previous section:

$$K_{MA} = \lambda_0 T_{MA,Mgm} + \lambda T_{MA}$$

In fact, traffic across the link connecting the NMS to,

[5] Here, the term link must be considered in a broad sense. The NMS can be actually connected to the managed LAN through a sequence of physical links and nodes. In this case λ_0 is the weight associated to the whole path from the NMS to the managed LAN.

say, the router of the LAN is concerned only with the first trip of the agent and the final delivery of aggregated results, and thus has the same expression derived earlier. Traffic within the managed LAN is generated since the mobile agent is injected from the ingress router into the managed LAN. If we imagine to place a virtual NMS on the ingress router, we can easily see that the traffic generated within the LAN has the same expression derived for the general case of the MA paradigm. The traffic $T_{MA,mgm}$ computed earlier is a special case of the formula above, where $\lambda_0 = 1$ and $\lambda = 0$. A more general case is management of an internetwork, i.e. a collection of LANs interconnected through internetworking devices—usually routers—which enables communication among stations on different LANs. In this setting, the NMS is connected through different links to each of the remote LANs (Fig. 1.c). In our model, the link connecting the NMS to the l^{th} of the L LANs is assigned a cost coefficient λ_{0l}; the link connecting the LANs l and m is assigned a coefficient λ_{lm}; each link within LAN l is assigned a coefficient λ_l. In this respect, remote management of a LAN can be seen as a special case of internetwork management where only one LAN must be managed. For the sake of simplicity, we assume that the cost coefficient is constant with respect to the direction of communication over the link, i.e. $\lambda_{lm} = \lambda_{ml}$. Under these assumptions, the overall cost for the CS paradigm is

$$K_{CS} = \sum_{l=1}^{L} \sum_{n=1}^{N} \sum_{q=1}^{Q} [(\lambda_{0l} + \lambda_l)(\eta_{CS} I_q + \tilde{\eta}_{CS} R_{lnq})]$$

where we assumed, in order to avoid cluttering the expression, that each LAN contains the same number N of managed nodes. The expressions for the cost of the REV, COD, and MA paradigms are derived similarly.

5 A CASE STUDY

In this section we illustrate the use of the model by focusing on the implementation of a specific management task: collecting information about the load level

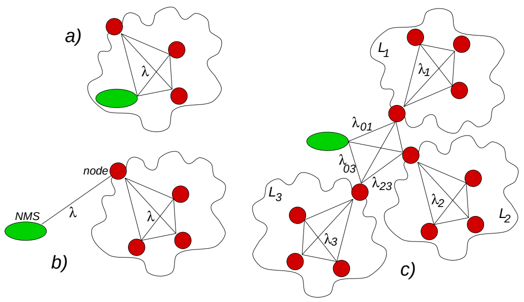

Figure 1: Different management configurations: a) local management of a single LAN, b) remote management of a LAN, and c) management of an internetwork.

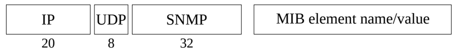

Figure 2: Encapsulation of SNMP requests and replies.

of network interfaces. The goal is to show how to refine the model to cope with the details of the technologies used to implement design paradigms and how to use the model to actually determine the best design choice.

We used SNMP as an implementation of the CS paradigm and Java Aglets [8] to implement the mobile code paradigms, because of their diffusion and level of support. For each implementation, the traffic generated has been measured and the overhead of each protocol layer down to the IP one has been isolated[6]. Measurements provided the information needed to calculate the value of the parameters of the traffic formulae presented in earlier sections.

SNMP requests and replies are encapsulated into UDP messages (Fig. 2). In our case study, requests and replies are short and are always carried within a single UDP message[7]—no fragmentation takes place. Hence, as described in Section 3, we can express the overhead function $\eta_{CS}(X) = \tilde{\eta}_{CS}(X) = \frac{\alpha_{CS}(X)}{X} + \beta_{CS}(X)$ by assuming $\alpha_{CS} = 60$ and $\beta_{CS} = 1$, independently of the values of I_q and R_{qn}. In Java Aglets, invocation of the `dispatch` method on an aglet triggers the migration of the classes describing its code and of the value of its attributes. No execution state is retained across migration. In order to implement the MA paradigm, we added explicitly extra information to the aglet's state to obtain control of the execution flow after migration. Java Aglets communication facilities are built upon the Agent Transfer Protocol (ATP) which exploits the services offered by TCP. Upon migration, the code and state of an aglet are prepended by an ATP header of variable length. Reliable transfer is ensured by an acknowledgment ATP message sent by the receiver. To transmit the result to the NMS we used the message passing facility of Java Aglets, which is built upon ATP as well.

Sending an ATP message requires a TCP connection to be setup and subsequently torn down, i.e. 5 TCP messages[8]. As shown in Fig. 3, an ATP message is possibly segmented into TCP messages, whose maximum payload is 1460 bytes in our implementation. Each TCP message is prepended by a 20 byte TCP header and en-

[6] The measurements have been performed over an Ethernet LAN which introduces an additional 26 byte overhead on each packet. Moreover, the overall traffic generated on the LAN is slightly larger than the aforementioned because Ethernet packets have a minimum size of 72 bytes.

[7] Some SNMP implementations may segment messages also in this case; we do not consider this here.

[8] The equations derived in this section assume that no message is lost in the network.

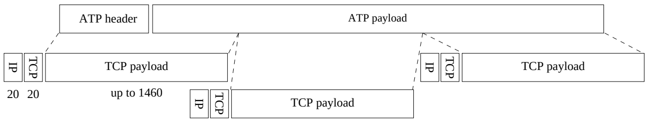

Figure 3: Segmentation of the ATP `dispatch`.

capsulated into an IP packet that introduces a 20 byte header. A TCP message must be acknowledged by the receiver either explicitly or implicitly. We assume explicit acknowledgment and no traffic in the reverse direction of the TCP connection transporting the code, i.e. acknowledgment piggybacking is not used. The traffic generated by an ATP message is given by

$$\alpha_{TCP} + \beta_{TCP}(H_{ATP} + C) + \beta_{TCP}\alpha_{ATP}$$

where α_{TCP} is the traffic generated for setting up and tearing down the TCP connection (200 bytes), α_{ATP} is the ATP acknowledgment message size (120 bytes), H_{ATP} is the ATP header size, C is the size of the ATP message payload—code and state in the case of aglet migration. β_{TCP} is the overhead introduced by the transmission and acknowledgment of TCP messages:

$$\beta_{TCP}(X) = \frac{2H_{TCP/IP}}{X}\left\lceil \frac{X}{p_{TCP}} \right\rceil + 1$$

where $H_{TCP/IP}$ is the size of the TCP and IP headers (40 bytes) present in data and acknowledgment TCP messages, and p_{TCP} is the size of the maximum TCP payload (1460 bytes). In our case study, since the same MCS is exploited for the implementation of all the three mobile code paradigms, and replies are sent using the same ATP facilities used for code migration, the overhead function $\eta_p = \tilde{\eta}_p = \eta$ is independent of the paradigm:

$$\begin{aligned} \alpha(X) = & \ \alpha_{TCP} + \left(\frac{2H_{TCP/IP}}{H_{ATP}+X}\left\lceil \frac{H_{ATP}+X}{p_{TCP}} \right\rceil + 1\right) H_{ATP} \\ & + \left(\frac{2H_{TCP/IP}}{\alpha_{ATP}} + 1\right)\alpha_{ATP} \\ \beta(X) = & \ \frac{2H_{TCP/IP}}{H_{ATP}+X}\left\lceil \frac{H_{ATP}+X}{p_{TCP}} \right\rceil + 1 \end{aligned} \tag{12}$$

At this point, the model has been refined to take into account the details of the technology selected for the implementation. We can now evaluate the different architectures. However, this needs some information about the topology of the network and the characteristics of the task to be performed. We assume that the managed network is composed of $N = 50$ nodes, and we temporarily assume a uniform network. In our CS implementation, the load level of an interface can be obtained through an SNMP query which retrieves 5 MIB variables; assuming that each device has 30 interfaces, $Q = 30$. From our data, each SNMP request and reply is 48 and 66 byte long, respectively. Moreover, we assume that each query is sent in a separate request message. Substituting these values and $\eta_{CS} = 2.01$ in (1), the traffic generated by the SNMP implementation of the management task is $T_{CS} = 335.6$ Kbytes.

The same management task can be implemented with the REV paradigm. The Java bytecode performing the Q queries on each node is $C_{REV} = 5.6$ Kbytes and is sent on each node prepended by a header $H_{ATP} = 120$ bytes, and augmented by the overhead determined by (12), which yields $\eta(C_{REV}) = 1.32$ and, under similar conditions, $\eta(Q\bar{R}) = 2.91$. According to (2) we obtain that $T_{REV} = 437.1$ Kbytes. Thus, as far as global generated traffic is concerned, it is more convenient to implement the management task according to the CS paradigm. This can be inferred using (6) obtaining that REV is convenient over CS if $Q > 87$. However, this is not a general result and depends on the particular technology chosen. In particular, using ATP rather than directly TCP connections to send replies introduces a huge, unnecessary overhead. Furthermore, Java Aglets are probably overshooting for the implementation of a REV paradigm. For this task, scripting languages are probably much more efficient as witnessed by earlier work on management by delegation, because they provide a more compact code representation. Ongoing experimental work will provide better insight on these technological issues.

If the management task is repeated $U = 20$ times, the traffic generated by a COD implementation is $T_{COD} =$ 2.6 Mbytes, as given by (4) and (5) where $C_{COD} = 5.1$ Kbytes and $\eta(\bar{I}) = 58.3$. In this case the COD design is more convenient than the CS one, which generates 6.6 Mbytes. Using (8) it is possible to foresee that the COD approach is convenient if the management task is supposed to be performed at least $U = 4$ times.

As we discussed earlier, an MA design reduces the traffic involving the NMS. In fact, according to (9) $T_{MA,Mgm} =$ 34.3 Kbytes, with a code size for the aglet of 6.6 Kbytes. This can be further improved leveraging on semantic compression. We consider now a different management task: collecting load on the most loaded interface on each device. This task enables local semantic compression and, according to (10), $T'_{MA,Mgm} = 10$ Kbytes, being $\eta(N\bar{R}) = 2.9$. Moreover, if global compression is possible, for instance when the most loaded interface in the whole network has to be searched for, the traffic is reduced to $T''_{MA,Mgm} = 9.6$ Kbytes, as derived from (11) with $\eta(\bar{R}) = 65.9$. Again, in this case the difference in the traffic generated with local and global compression is small because the fixed overhead component in the transmission of the results to the NMS is large, due to the particular technology chosen.

As an example of application of our model of costs, let us consider that the most loaded interface has to be found in an internetwork of $L = 15$ LANs, each containing $N = 5$ nodes. A reasonable value for the cost associated to each LAN is $\lambda_{LAN} = 0.01$, with a cost associated to WAN links of $\lambda_{WAN} = 1$. Under these assumptions, the cost of performing the management task using SNMP is given by $K_{SNMP} = 508.4$ Kbytes, while the cost with an implementation based on the MA paradigm is $K_{MA} = 130.3$ Kbytes. The movement of the agent among the managed devices generates a considerable amount of traffic, as we described earlier; nevertheless, these movements are mostly inside high bandwidth LANs and only a small percentage of them involve wide area links and the NMS. This confirms that the MA paradigm can be cost effective for the management of internetworks.

6 CONCLUSIONS

We discovered that, in the selected application domain and with the particular goal of optimizing network traffic, the design tradeoffs depend on the characteristics of the network being managed (costs, number of nodes, protocols) and of the management task (possibility of local/global semantic compression, expected frequency of invocation, complexity of the task, dimension of replies). The characteristics of the technology actually used for the implementation also affect these tradeoffs according to the management protocols (overhead) and the MCS (expressiveness of the language, formats used for transfer, overhead) used. Hence, determining when to use a mobile code design paradigm in place of a traditional client-server architecture requires *i)* a model of the management functionality to be implemented, together with information about the managed network; *ii)* a precise quantitative characterization of the management protocols and the MCS to be used for the implementation. This work is a partial answer to the first point. In fact, it provides a general framework that is conceived for being tailored and customized by the manager according to the peculiarities of her needs, as exemplified in Section 5. Quantitative information about the performance and the overhead introduced by MCSs is still missing in literature. However, this measurement and modeling activity is done once and for all, and can guide a highly effective design of management tasks, as shown throughout the paper.

Our long term work on the theme of this paper envisions a scenario where the manager is provided with a software platform which guides the "management life cycle" as describe above. Management operations could be specified by "packing" sets of management instructions into reusable library components—possibly reusing off-the-shelf components. The same complex functions could be embedded in different components implementing different design paradigms. This platform would provide tools to derive customized models of the managed networks from general models as we exemplified, and to visualize and simulate the performance of the management operations according to different design and implementation choices. Ongoing work is headed towards the realization of such a platform, the refinement of our model, and a quantitative characterization of existing MCS.

ACKNOWLEDGEMENTS
We wish to thank Jean-Philippe Martin-Flatin, Carlo Ghezzi, Fulvio Risso, and Emiliano Graglia for their insightful comments on early drafts of this work.

REFERENCES

[1] M. Baldi, S. Gai, and G. P. Picco. Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Mobile Agents*, volume 1219 of *LNCS*, pages 13–26. Springer, Apr. 1997.

[2] 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*, pages 22–32. ACM Press, 1997.

[3] J. Case et al. Structure of Management Information for version 2 of the Simple Network Management Protocol. RFC 1902, Jan. 1996.

[4] G. Cugola, C. Ghezzi, G. P. Picco, and G. Vigna. Analyzing Mobile Code Languages. In *Mobile Object Systems*, volume 1222 of *LNCS*, pages 93–111. Springer, Apr. 1997.

[5] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. Technical report, Politecnico di Milano, Italy, July 1997. Submitted.

[6] G. Goldszmidt and Y. Yemini. Distributed Management by Delegation. In *Proc. of the 15th Int. Conf. on Distributed Computing*, June 1995.

[7] K. Jones. Internet's SNMP and ISO's CMIP Protocols for Network Management. *Int. J. of Network Management*, pages 130–137, Sept. 1994.

[8] D. Lange. Java Aglet Application Programming Interface (J-AAPI). IBM White Paper, Feb. 1997.

[9] J.-P. Martin-Flatin and S. Znaty. A Simple Typology of Distributed Network Management Paradigms. In *Proc. of the 8th IFIP/IEEE Int. Workshop on Distributed Systems: Operations & Management (DSOM'97)*, Oct. 1997.

[10] J. Schönwälder. Network Management by Delegation From Research Prototypes Towards Standards. In *Proc. of the 8th Joint European Networking Conf.*, May 1997.