CHAPTER 3

MOBILE AGENTS AND THEIR APPLICATIONS ON NETWORK MANAGEMENT

3.1. INTRODUCTION

Code mobility is not a new concept. In the recent past, several mechanisms have been designed and implemented in order to move code among the nodes of a network (e.g. remote batch job submission [BOG73]). A more structured approach has been followed in distributed operating systems research. New approaches to application design have been developed embodying the notion of code mobility, i.e., *the capability of dynamically moving the components of a distributed application among the nodes of a computer network* [CAR97]. The key idea behind code mobility is to provide an alternative to the traditional Client/Server (CS) structure of distributed applications, thus enabling a better use of bandwidth resources and a higher degree of flexibility and reconfigurability. In particular, mobile code-based approaches give rise to significant network load savings through replacing remote CS network communication by local interactions (see Figure 3.1).

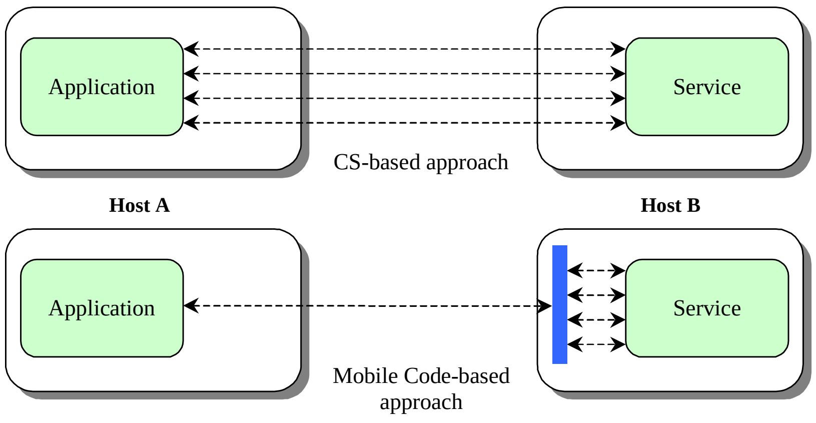

Figure 3.1. CS vs. Mobile Code-based approaches

Furthermore, embedding *migration* capabilities into the mobile code provides an extra level of autonomy and brings forth the concept of the *Mobile Agent* (MA). In general, a *software agent* can be defined as a computational entity, which acts on behalf of others, is autonomous, proactive and reactive, and exhibits capabilities to learn and co-operate [NWA96]. Software agents can be classified as *stationary* (static) or *mobile*. An MA is a software agent with the ability of autonomously moving form host to host.

Given the definition of mobile code, it is clear that this paradigm can provide the technology needed to achieve management distribution. Mobile code can be linked dynamically on network devices either proactively by the manager or reactively by the network device [FUG98]. This way, the management primitives embedded within mobile code become available on the device only when requested by management operations, thus consuming device resources only when this is really needed. Furthermore, MAs add a new dimension in distributed management due to their ability to autonomously migrate to different network devices and perform complex management tasks without the manager's intervention [PIC98].

As a result, mobile code and MAs in particular represent a hot trend in distributed management arena, reflected to numerous research activities [CHE98, BIE98b]. Similarly to other application domains, a lot of hype surrounds MAs in network management and has given rise to high expectations. Yet, until present, there has been very limited interest from industry to support MA-based management, and there are no clear signs of a near-term take-off of this technology. This is mainly due to several open issues still not sufficiently addressed by the agent community (e.g. security concerns), while it is still unclear whether the use of MAs in management applications may actually offer performance gain. This chapter first introduces the reader to mobile code and MA technologies, surveys research initiatives in MA-based management and attempts to identify suitable models that enable the effective use of this technology in the management domain.

The remainder of this chapter is organised as follows: Section 3.2 provides an overview of code mobility paradigms, with Section 3.3 listing their advantages when used in distributed applications. Section 3.4 highlights the differences between MAs and the other mobile code paradigms, suggesting application scenarios where the former can offer performance or flexibility benefits over he latter. Section 3.5 focuses on agent mobility describing aspects that affect the design of MA platforms. It also reviews mobile agent languages and standardisation approaches, classifies mobility schemes, briefly describes commercial Mobile Agent Platforms (MAP), lists application fields utilising MAs and discusses performance issues related to their practical use. Section 3.6 concentrates on MA-based management: it describes several MAPs tailored to Network & Systems Management (NSM), provides a survey of research activity on the field, discusses performance aspects related to these applications, and suggests ways to effectively utilise MAs in terms of organisation and mobility models to optimise the performance of management applications. Finally, Section 3.7 summarises the chapter.

3.2. CODE MOBILITY PARADIGMS

Mobile code paradigms encompass different technologies, all sharing a single idea: to enhance flexibility by dynamically transferring programs to distributed devices and have these programs executed by the devices. The program transfer and execution can be triggered by the device itself, or by an external entity. Fuggetta et al. [FUG98] made a detailed review of mobile code, where they clearly define the boundaries between technologies, paradigms and applications. In particular, they identified three different types of mobile code paradigms: (a) Remote Evaluation, (b) Code on Demand, and (c) Mobile Agents (see Table 3.1).

	Initiator (A)	Co-operator (B)
Client-Server	–	**Code** **Data** **Processor**
Remote Evaluation	Code	**Data** **Processor**
Code on Demand	**Data** **Processor**	Code
Mobile Agent	Code	**Data** **Processor**

Table 3.1. Design paradigms for mobility (the boldfaced typesetting indicates where the interaction takes place).

3.2.1. Remote Evaluation

In the Remote EValuation (REV) paradigm [STA90], a client (initiator) has the know-how necessary to perform a service but it lacks the required resources, which are located at a remote server (co-operator). Consequently, the client sends the service know-how to the remote site that executes the code using the resources available there. This is a form of *push*. The information transferred includes the agent code plus a set of parameters (arguments). After executing the operation, the remote site returns the results back to the initiator of the remote evaluation [ROT97].

Given the above definition of REV, it is clear that several standardisation and research approaches on distributed management, exemplified by the Management by Delegation paradigm (e.g. Script MIB, MbD, flexible agents, spreadsheet approach, etc), can be considered as direct application of REV on the NSM area, as they all share the idea of moving NSM functionality from the managers (initiators) to the management agents (co-operators).

3.2.2. Code On Demand

In the Code On Demand (COD) paradigm, the initiator *A* is already able to access the resources it needs, which are co-located within the same device. However, it lacks the information on how to process such resources. Thus, *A* interacts with its co-operator *B*, requesting the service know-how. A second interaction takes place when *B* delivers the know-how to *A*, which can subsequently execute it. This is a form of *pull* [FUG98].

A widely used technology based on COD are the Java *applets*: applets are programs written in the Java programming language; when a Java-enabled browser is used to view a web page that contains an applet, the applet's code is transferred to the end system and executed by the browser's Java Virtual Machine (JVM).

3.2.3. Mobile Agents

In the MA paradigm, the service know-how is owned by the client, but some of the required resources and data are located at a remote server. Hence, component *A* migrates to the server carrying the know-how and possibly some intermediate results. After its arrival, *A* completes the service using the resources available there [FUG98]. The MA paradigm differs from other mobile code paradigms on that the associated interactions involve the mobility of a computational component. In other words, while in REV and COD the focus is on the transfer of code between components, in the MA paradigm a whole computational component is moved to a remote site, along with its state, the code it needs, and some data required to perform the task. In that sense, MAs can be regarded as a 'superset' of REV/COD paradigms, as they can offer all the functionality provided by the latter, with the additional ability of autonomous migration.

3.3. MOBILE CODE - ADVANTAGES

Mobile code technologies represent a powerful programming paradigm, which is useful when designing distributed applications. The commonly agreed benefits of mobile code have been discussed in many research papers [HAR95, GRE97, BAL97, PIC98] and summarised in the following:

- *Enhanced Flexibility*. Clients typically access the resources hosted by a server through a set of services, whose interface is typically predefined and commonly agreed among the client and the server. Mobile code can be used to extend and update dynamically capabilities of applications, thereby enhancing systems flexibility.

- *Exploitation of increased resources availability*: Managed devices resources are characterised by continuously increasing availability in terms of processing power, disk and memory capacity. Mobile code takes advantage of that feature to achieve processing load distribution. In NSM field, this advantage can be exploited to perform collection and filtering of management data locally, in a distributed fashion. That way, expensive platforms (managers) dedicated to issuing management requests, collecting, analysing and presenting data are not any longer necessary.
- *Reduction of network traffic*: The transfer of mobile code to the source of data creates less traffic than transferring the data, as mobile code can perform *semantic compression* of data, delivering pre-processed, high-level information.
- *Asynchronous interaction*: Once downloaded, mobile code can perform distributed tasks, even if the delegating entity does not remain active.
- *Interaction with real-time systems*: Installing mobile code close to a real-time system may prevent delays caused by network congestion. In NSM, this problem arises when a number of successive interrelated Management Information Base (MIB) values need to be retrieved to present a snapshot of the system's state, e.g. on Simple Network Management Protocol (SNMP) table retrievals [PIC01].
- *Support for heterogeneous environments*: Mobile code is separated from the hosts by an environment able to receive and instantiate the received code. If the framework is in place, mobile code can target any system, especially when the framework is implemented by a platform-independent language, e.g. Java. The cost of running a JVM on a device is decreasing. Java chips will probably dominate in the future, but the underlying technology is also evolving in the direction of ever-smaller footprints (e.g., *picoJava*[1]).

3.4. MAs vs. REV and COD

Since all the mobile code design paradigms allow the dynamic relocation of the components of a distributed application, a legitimate question to ask is whether one should choose the purest MA paradigm or just an approach exploiting COD or REV. Several researchers have tackled the problem of finding out what are the potential assets of MAs (see for instance [HAR95, FUG98, LAN99]). According to [MIL99], the application domains in which MAs have potential deployment are: (a) data-intensive applications where the data are remotely

[1] The picoJava [picoJava] core is a small, flexible microprocessor core that directly executes Java bytecode instructions; picoJava can be used to run applications in small electronic appliances such as organisers, pagers, and cell phones.

located; (b) extensible servers; (c) applications in which agents are launched by an appliance (e.g. a cellular phone) to a remote server, where the user does not necessarily have to stay connected waiting for the MA's return. However, as it has been shown in the preceding section, the use of REV/COD suffices in the first two application domains.

Certainly, every MA-based application could be alternatively designed using existing established technologies [CHE95]. As with every design choice, the answer is given by application requirements and engineering tradeoffs. Detailed comparative analysis of the three mobile code paradigms can be found in [PUL99], with [BAL98] focusing on their quantitative evaluation, using NSM applications as a case study. In addition, [MAG96, BIE98b, CHE98] discuss the potential of MAs in management applications and [HAY99] explores their assets in the wider field of telecommunications. In this section, we identify some aspects related to distributed applications design where MAs can offer more efficient or flexible solutions compared to REV/COD, with special focus given on distributed NSM applications. Given that the distributed NSM approaches reviewed in Sections 2.9 & 2.10 apply the REV paradigm concepts in the NSM area (see Section 3.2.1), the advantages of MAs over REV/COD are, in effect, valid also when comparing MA-based solutions against existing distributed management approaches.

- *New programming paradigm*: MAs provide a subjective advantage because of the metaphor they embody. Agents that are able to dynamically and autonomously relocate themselves according to the application needs may provide, for certain applications, the building block of a uniform and elegant design where every active component is able to spontaneously relocate itself. This characteristic of MAs is probably at the core of their popularity, and provides also a link to other disciplines, like artificial intelligence, that brought this concept to the extreme by proposing *agent-oriented programming* [JEN00] as a new way to create distributed applications.
- *Decreased dependency on a master process*: Approaches based on REV or COD have increased dependency on a master process (delegator) that communicates with one or more mobile code components for controlling and co-ordinating their tasks. Conversely, MAs carry out these tasks in a programmable and autonomous fashion that obviates the need for co-operation with the master process [KAW00]. In this sense, the interaction between the MA and the delegator is limited to the stages of transmission and return of data [PUL99].
- *Space savings*: Resource usage is limited, because an MA resides only on one node at a time. In contrast, static extensible servers require duplication of functionality at every location. MAs carry the functionality with them, so it does not have to be duplicated [BIE98b].

- *Easier update of decentralised tasks*: REV and COD paradigms are not suitable in cases that mobile code needs to be frequently updated, since every update involves broadcasting the updated code to all devices, resulting in excess usage of network resources. In the MA paradigm, such updates are easier and more efficient as the MA code is updated at a central location (code repository) and does not involve network interactions.
- *Traffic around the manager station*: A main priority in NSM is to reduce the traffic around the manager platform [BAL98]. REV and COD imply a pairwise interaction between the manager and each managed device, where mobile code has been installed [PIC01]. Should the manager-managed systems communication is relatively frequent or the size of the managed network is fairly large, the associated traffic will affect the manager's network neighbourhood, although it is very much reduced in comparison with SNMP traffic. On the other hand, MAs once unleashed can visit the devices autonomously, without requiring any communication with the manager until all the results have been collected. Thus, no matter how many devices are visited by the MA, the manager is involved only in the initial dispatching of the agent and in the final collection of results, while the remaining traffic is steered by the MA away from the manager.
- *Management of remote subnets*: Considering the management of remote subnets, connected to the manager site through low-bandwidth links, when using REV or COD the traffic over interconnecting links increases with the number of devices residing in the remote subnets and the frequency of communication required with the manager station, due to the pairwise interaction between the manager and each managed NE. MAs offer a more efficient solution as they need to traverse the interconnecting links only twice (to visit the remote subnets and return the results), regardless of the number of visited devices. Even if the state of the MA increases during this operation, bandwidth is assumed to be 'cheaper' within the LAN than on the low-bandwidth link.
- *Short-term distributed tasks*: When distributed tasks are intended to run over a set of devices for a relatively short period, it is more efficient to use a MA-based approach, where a MA object sequentially visits the devices rather than broadcasting mobile code and obtain the results from every NE. That also reduces the deployment time, especially when the management of multiple NEs is involved (only one MA is issued by the manager) [BOH00b].
- *Reduced deployment cost and delay*: MAs can be used to dynamically increase availability of certain services. For example, the density of fault detecting or repairing agents can be increased upon detecting malfunctions, through creating *clones* of existing MAs and dispatching them to areas of concern. Hence, the traffic and latency involved in

the deployment procedure can be significantly reduced, as it is carried out autonomously and not initiated from a central location [LIO99].

- *Local vs. global semantic compression of data*: By visiting a number of devices, MAs can also extend the concept of semantic compression enabled by REV and COD. These paradigms can provide only a form of compression that is limited to a single device, and local to it. The MA paradigm, in contrast, enables global semantic compression of data across all the network devices visited.
- *Device vs. Domain-level view*: In parallel to the previous argument, approaches based on REV or COD imply a local view of the device where the distributed code statically resides. Should domain-level information incorporating data collected from a set of devices is required, data correlation process should be performed at the manager station [LOP00]. Alternatively, a collaboration scheme between static agents could be applied in order to exchange and correlate information [MOU98a], however that would imply a more complex system design. MAs offer a more simple solution to this problem as they can perform data correlation through sequentially visiting the entire set of devices.

Some of the aforementioned arguments are also valid when comparing the MA paradigm against static approaches based on distributed objects, with the additional advantage of enhanced flexibility and reconfigurability of the former over the latter, as MAs allow the dynamically augment management services at runtime, which is not possible when these services are realised through rigidly configured static objects.

Conversely, in many cases a naive use of MAs may lead quickly to a highly inefficient design. In network monitoring applications for instance, the use of a MA that roams the network and collects information may actually lead to a design that performs worse than the conventional one, especially when large amounts of data are accumulated within the MA state at each host [BAL98]. Yet, this problem can be partially addressed by launching MAs able to perform semantic compression of data, thereby keeping their size practically constant. Increased security concerns is another argument that can be used against MA paradigm, as MAs typically visit a large set of potentially malicious hosts and face the risk of tampering. Concluding, MA technology cannot be considered as panacea in distributed applications design; if the motivation for using MAs is to optimise system performance, e.g., in terms of traffic or latency, attention should be paid to the choice between MAs and alternative mobile code technologies. In certain cases, a synergy of the two approaches may be preferable. In addition, there are several open issues related with MAs that must be addressed by MA community to help the wide spread of this technology:

- *Reducing migration overhead*: the MA code should be as lightweight as possible [PIC01];

- *Migration delays*: MAs have been criticised for being associated with migration delays of the order of seconds or even tens of seconds, depending on the agent configuration and functionality [KNI99, BOH00a];
- *Security*: Network devices should be protected from malicious agents (and agents from malicious machines) [VIG98];
- *Fault tolerance*: agents should be able to survive network and machine failures [NWA96];
- *Performance issues*: what would be the effect of having hundreds, thousands or millions of MAs roaming on a network? [NWA96]
- *MAs composition*: it is essential to ease and automate the composition of service-specialised MAs even by novice users, with no programming experience [MIL99].

3.5. Agent Mobility

The field of MAs and mobile code has lately become a hot research topic covered by many networking and software engineering conferences. As it has been shown throughout the preceding sections, MA technology represents a promising programming paradigm, which can enhance the flexibility and scalability of contemporary NMSs. However, its merits and weaknesses should be carefully evaluated to ensure its effective use in NSM applications. This section focuses on aspects related to agent mobility and MA platforms.

3.5.1. Elements of a Mobile Agent Platform

A basic component of a Mobile Agent Platform (MAP) is the MA Server (MAS), equivalent to an ORB in CORBA, that runs on each host where MAs can execute. The main purpose of the MAS is to provide an efficient execution environment able to receive, instantiate and dispatch agents, serve as an interface between incoming MAs and the underlying system resources and offer a set of services required by the MAs to perform their distributed tasks.

Focusing on MAs, according to the definition given in Section 1.1, they comprise three parts:

- the *code* part which defines the MAs' functionality;
- the *data* part (*persistent state*), including the values of the variables declared within the MA class;
- the *execution thread* (with an execution stack).

MAs state is dynamically updated as a result of their visits and interaction with distributed servers where information is collected. MAPs that provide *strong mobility* (see Section 3.5.5.1) enable the transfer of all three parts on every MA migration. Most platforms involve the transfer of only the code and state information (*weak mobility*).

An agent migration may be initiated either from the MA itself or the hosting MAS server by invoking a `move` primitive, which allows an MA to move to the next server included into its itinerary[2], through an agent transfer protocol (ATP). At the time that the `move` method is called, the MA's state is saved and transferred through the network. At the destination site, the MA state is recovered and the agent instantiated, typically provided with its own thread of execution. ATPs are used to transfer agents between MASs and can be based on several protocols such as sockets, HTTP, Java RMI, etc. In addition, MAPs also provide agent development and deployment facilities, defined in APIs. A set of classes and interfaces are supplied and should be integrated in the agent code in order to enable mobility.

Besides these basic functions, a MAP may include additional services and facilities. *Fault tolerance* features insure that agents are reliably transferred and are not lost due to a system or network failure. Many available MAPs include *directory* and *location* services [LAZ98]. These services allow the agents to be aware of the existence of other agents and to track their locations. Another important service is *security*, which deals with both the protection of hosts from malicious MAs, and the protection of MAs against malicious hosts. Security mechanisms have to ensure authentication, integrity, confidentiality and access control [FAR96]. Generally, MAPs also provide primitives allowing MAs to *communicate* with each other and with the servers on the visited machines. Inter-agent communication can be enabled by standardised Agent Communication Languages (ACL), such as the Knowledge Query and Manipulation Language (KQML) [KQML] and the FIPA ACL [ACL], developed by the Foundation for Intelligent Physical Agents. These languages have been designed mainly for *intelligent agents*[3]. In addition, a number of proprietary communication models have also been reported in the literature (a detailed description of these models may be found in [INCO]), including direct communication, blackboard, mailbox, meetings and method invocations.

3.5.2. Mobile Agent Languages

MA-based applications can, in principle, be developed in any programming language. However, there are practical issues that render certain kinds of languages potentially more

[2] The term itinerary refers to the list of devices to be visited by the MA. Itineraries may be pre-specified or determined on-the-fly (see Section 3.5.5.3).

[3] The term intelligent agent derives from Distributed Artificial Intelligence (extension of Artificial Intelligence) and refers to a software component involved in a cooperative effort to resolve a problem. Intelligent agents are typically composed of a communication mechanism, a rule base, a solution base and an inference engine [MUL98].

suitable for programming MAs. Given the heterogeneous nature of modern network devices, *portability* is a first requirement (guaranteed by interpreted languages), while additional features that enable *easy development of mobility characteristics* are also of major importance. Other factors include *object orientation, performance*, etc.

Telescript [WHI96] was an early interpreted programming language for MAs developed by General Magic Inc. The Telescript interpreter included a built-in mechanism for transparent migration. Unfortunately, General Magic does not support Telescript anymore. *Agent Tcl* [GRA95] is another MA language developed in the early period of MA technology. Agent Tcl also supports transparent migration and can be useful for running existing Tcl scripts.

The popularity of *Java* has greatly influenced MA-based application developers. As a result, *all* the MA platforms presented in Sections 3.5.6 and 3.6.1 are implemented in Java. In addition to the generic benefits of Java highlighted in Section 2.6.1.1, its suitability for MA programming is enhanced due to its inherent support for dynamic class loading, serialisation, remote cloning and distributed objects communication. These features are discussed below:

Dynamic class loading: Java architecture enables the developer to write programs that dynamically extend themselves by choosing at runtime classes and interfaces to load and use. In fact, some of those classes and interfaces may not even exist when the program is compiled [VEN98]. To enable a Java program to dynamically load classes not included within the local name space, a customised *ClassLoader* (CL) object must be provided to obtain the classes implementations (bytecode) and load them at runtime. A CL is defined by extending the abstract `java.lang.ClassLoader` class and implementing its `loadClass()` method. For instance, customised CLs may dynamically load classes received through the network. This feature makes Java particularly attractive for mobile code-based applications. More sophisticated CLs may even allow to *reload* classes that have been already loaded, in case their implementations have been modified at runtime (see Section 4.4.3).

Serialisation: Java also provides the serialisation feature [OSS97], which allows object instances to be exchanged between different JVMs. Serialisation provides a means for translating a graph of objects into a stream of bytes which can be sent as a message over the network or written in a file. Each instance of a class implementing the `java.io.Serializable` interface is eligible for serialisation. The `writeObject()` method of the `java.io.ObjectOutputStream` class defines the default behaviour for serialising an object, i.e. converting the object's state to a stream of bytes. By default, all the objects referenced by a serialised object are serialised (they must implement the `Serializable` interface); only the fields declared as *transient* or *static* [ARN96] are excluded from the serialisation process. The symmetric process of recreating the object from its serialised representation is termed *de-serialisation*. De-serialisation is achieved by the `readObject()` method of the `java.io.ObjectInputStream` class. The serialisation/ de-serialisation feature of Java is extremely useful when developing MA-based applications. As discussed in Section 3.2.3, a fundamental aspect of agent mobility is the ability of MAs to maintain their *state* information while migrating from one host to another. State can easily be obtained upon migration through serialisation and subsequently recovered on the destination host through de-serialisation. In Appendix B, we propose several ideas for reducing an MA's state size and therefore minimising the migration overhead. Appendix C presents the results of experiments that provide a better understanding of the serialisation process.

Remote Cloning: Java provides inherent support for objects cloning (i.e. creating identical copies of Java objects) through the `clone()` method of the `java.lang.Object` class [ARN96]. Cloning is inspired by the *fork* process mechanism adopted in UNIX. Although support for cloning has not been implemented with mobility in mind, it can greatly benefit MA programming, as it enables the autonomous creation of MA clones at remote sites given that specific conditions are satisfied. Remote cloning can reduce the latency and traffic involved in MAs deployment, thereby enhancing the scalability and flexibility of MAPs [LIO99].

Remote Method Invocation (RMI): Java RMI [RMI] facility provides mechanisms for objects communication in a location-transparent way (see Section 2.5.2). That can be very useful for implementing a communication scheme between agents or agents and other Java applications.

On the other hand, two problems associated with MA programming in Java have been identified:

- It is not possible to implement strong mobility through Java, i.e. to carry MAs execution stack along with their code and state. However, the majority of distributed applications employing MAs can be implemented utilising weak mobility [CAB00].
- Another limitation of Java is that it does not allow the serialisation of classes not implementing the `java.io.Serialisable` interface, e.g. threads (instances of the `java.lang.Thread` class). In Section 6.3.4, we propose a way to get around this problem.

As a result of the aforementioned advantages, we have chosen Java for the development of our MA-based management framework, described in Chapter 4.

3.5.3. Security in Mobile Agent Systems

Security issues related to MA systems are of major concern and have prevented to a large extend the adoption of agent technology by commercial management platforms. Generally, network operators are worried about the capabilities of having self-replicated MAs roaming into communication networks. This behaviour closely mirrors that of a computer virus. A slight change in the agent's executable code is enough to turn an MA to a vicious virus.

The security problem is, in fact, twofold: hosts should be secured from malicious agents [SAN98] and agents from malicious hosts [KAR98b]. The first problem has been more extensively addressed, yet both need to be solved before we can use MAs in real distributed system environments. These security problems have proved harder to solve than people initially expected. As a result, MA security is one of the hottest topics within the agent research community. For a good introduction to the whole range of security issues, the interested reader is referred to [VIG98]. In our MA framework, we have addressed the problem of malicious agents, through a security component that provides authentication, and access control services, whilst offering data encryption to protect sensitive management information from malicious hosts (see Section 4.4.1.3.2).

3.5.4. Standardisation Approaches

The well-known advantages of standards also apply in the agent mobility field, as they allow MAPs inter-operation. In order to establish a common basis for future developments and enable the interoperability of agent platforms developed by different manufacturers, two bodies promote MA standardisation. OMG has defined basic interoperation capabilities between heterogeneous MAPs in its Mobile Agent System Interoperability Facility (MASIF) [MASIF]. MASIF proposes the standardisation of agents, agent system names, agent system types and location syntax; it defines two interfaces (`MAFAgentSystem` and `MAFFinder`) which should be implemented to provide agent management and agent tracking, respectively, functionality. In parallel, FIPA [FIPA] is focusing on the standardisation of basic capabilities of *intelligent agents*. Although the scopes of the two standards are quite different, an interworking or even integration of MASIF and FIPA may be possible in the medium-term time frame [ZHA98a]. Examples of MASIF and FIPA-compliant MAPs will be given in Sections 3.5.6 and 3.6.1.

3.5.5. Taxonomies of Mobility Patterns

This section attempts a classification of the mobility patterns in terms of several characteristics: their support to retaining MAs execution state, their migration strategy (the number of hops realised by MA objects) and their itinerary control (predefined or dynamically configured itineraries).

3.5.5.1. Weak vs. Strong Mobility

Existing mobile code languages provide support for at least one of the following [CAR97]:

- *Strong mobility*: the ability of processes to move their code and execution state to a different site. Processes are suspended, transmitted to the destination site, and resumed there. For instance, Telescript provides mechanisms to implement strong mobility.
- *Weak mobility*: the ability to transfer code across different execution environments; code is accompanied by its persistent state, but no migration of execution state is involved (see Figure 3.2). For instance, Java supports only weak mobility.

Sun's JVM does not allow capturing of processes' execution states and, as a result, very few Java-based MA systems provide strong mobility. Those that do, fall into three categories: systems using a *modified JVM* [ACH97, PEI97], a *custom JVM* [SUR00] and systems using a *pre-processor* [FUN98] approach. Clearly, the implementation of frameworks supporting strong mobility is not a trivial task, whilst introducing performance penalties in agent transfers [FUN98]. In addition, management tasks of configuration, maintenance and control typically involve the execution of repetitive tasks on every node. That means that the requirements of MA-based NSM can be comfortably met by frameworks that only support weak mobility [CAB00, CHE00b]. This statement is also proved by the remarkable precedence of Java over other programming languages that support strong mobility. Our MA platform, described in Chapter 4, supports only weak mobility.

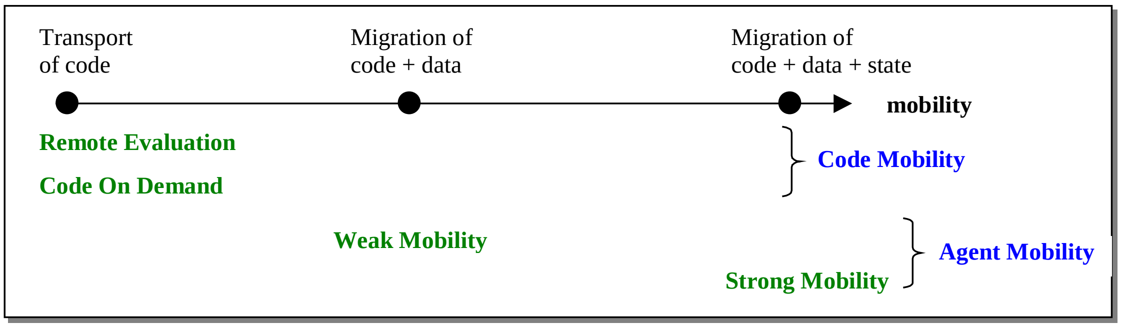

Figure 3.2. Classification of code mobility and agent mobility paradigms

3.5.5.2. Single-hop vs. Multi-Hop Mobile Agents

A second classification of MAs is based upon their migration plan, i.e. on whether MAs visit one or more hosts. *Single-hop agents* travel to a target host, start their execution and remain there until they terminate. This type of agents do not need any data when migrating to the target host (except maybe initialisation data) nor any methods for itinerary control. Therefore, single-hop MAs compare to downloadable code, i.e. they represent a direct application of REV paradigm. Bohoris et al. [BOH00b] use the term *constrained mobility* for single-hop agents.

In contrast, *multi-hop* or *itinerant agents* can travel to several sites during their lifetime. Multi-hop MAs are suitable for performing repetitive tasks over a set of devices. They can also perform different tasks, and can adapt their behaviour depending on the tasks achieved in the previously visited hosts. In [BOH00b], multi-hop MAs are further categorised in *weak* and *strong* MAs, with the former referring to the migration of an MA without preserving information gathered from previous visits and the latter involving the migration of MAs that preserve their state formed during previous visits (standard use of MAs). Weak MAs are termed *memoryless*[4] agents in [CHE00b]. To avoid confusion with the well-established definition of weak and strong mobility given in the preceding section, we adopt the term memoryless (multi-hop) MAs to refer to agents that cannot (can) preserve their persistent state when migrating. As shown in Chapter 5, the choice between single-hop and multi-hop MAs is application-dependent.

3.5.5.3. Itinerary Control

A last classification of MAs is in terms of the control mobile objects maintain on their itinerary. Thus, MAs can either have *fixed* or *dynamic* itinerary [CHE98]. When the itinerary is fixed, the agent migration path is known at the MA's creation time and it does not change during the agent's execution. For instance, fixed itinerary agents are suitable for visiting a pre-determined list of devices to collect data, where the itinerary is typically supplied by the user. In contrast, agents with dynamic itinerary may change their migration path during their execution. The support for dynamic itinerary is achieved though at the expense of increased complexity and size on the MA's code. A discovery agent that is sent to discover new components in a network is an example of agent with dynamic itinerary, since its migration path is specified depending on the detection of new components or subnets. The migration schemes incorporating MAs with fixed and dynamic itineraries are termed *passive* and *active migration*, respectively.

It is noted that migration decisions are either made by the MAs themselves or other entities co-located at the same execution environment. Our MA framework, presented in Chapter 4, supports only fixed itineraries (in performance management applications, the list of monitored devices is known in advance), while migration decisions can be made by both the MAs and the local MASs.

[4] Both single-hop and memoryless mobility involve the transfer of agents code but not their persistent state. However, this is inconsistent with the definition of mobile agents, according to which an MA is a computational entity that carries both code and state information. In other words, the term Mobile Agent is used abusively herein.

3.5.6. Commercial Mobile Agent Platforms

The phenomenal popularity of MAs is reflected on several industrial initiatives that led to the development of numerous MAPs [MAL]. State-of-the-art reports on general-purpose MAPs can be found in [KRA98, COR98b, INCO], with interesting comparative performance, robustness and scalability tests reported in [SIL00, MARINE, MIAMI98]. In this section, we briefly review four representative and popular general-purpose MAPs, all implemented in Java: Aglets, Concordia, Voyager and Grasshopper.

Aglets [Aglets]: The oldest and most well-known platform, developed at the IBM Research Laboratory in Japan. The first version was released in 1996. The migration of Aglets is based on a proprietary Aglets Transfer Protocol. The Aglets Software Development Kit (ASDK) runtime consists of the Aglets server and a visual agent manager, called *Tahiti*. The ASDK provides a modular structure and an easy-to-use API for Aglets programming and also extensive support for security and synchronous/asynchronous agent communication.

Concordia [Concordia]: It has been developed by Mitsubishi Electric. This platform provides a rich set of features, like support for security, reliable transmission of agents, access to legacy applications, inter-agent communication, support for disconnected computing, remote administration and agent debugging.

Voyager [Voyager]: It is probably the most popular MAP, in terms of number of users. It has been developed by ObjectSpace. Voyager is an object request broker with support for MAs. The agent transport and communication is based on a proprietary ORB on top of TCP/IP. Voyager has a comprehensive set of features, including support for agent communication and agent security and also provides support for CORBA and RMI.

Grasshopper [Grasshopper]: Grasshopper has been developed by IKV++, with its main power lying on its compliance with FIPA and MASIF standards. MASs in Grasshopper comprise a *core agency* and a set of one or more *place*s (runtime environments where agents run). The core agency offers a set of services to support agent migration and execution. The communication service that supports agent communication and migration may use a variety of protocols: CORBA IIOP, Java RMI and plain sockets. Grasshopper also offers registration services to keep track of running places and the agents running at each place. Security in Grasshopper allows protecting both MAS-region interactions and agent-MAS interactions. Fault tolerance mechanisms are also integrated, ensuring that agents and MASs can recover in case of crashes or faults [BAU99].

The advantages of using one of the MAPs presented above, are that (a) they are relatively easy to use and typically well-documented, providing attractive frameworks for the rapid development of MA-based distributed applications; (b) most of them are robust, reliable and well-tested. Experiments, however, have demonstrated that these MAPs do not satisfy all performance requirements, as they either involve increased migration latency (Aglets, Voyager, Grasshopper) or poor scalability and robustness under stressing conditions (Concordia) [SIL00]. In fact, Grasshopper has shown to perform worse than others MAPs [MARINE], however its rich functionality, security features and compliance with well-established standards have been the main factors that contributed to its recommendation as the appropriate development platform for several MA-related EU projects [MIAMI, MARINE].

From the management viewpoint though, there are several weaknesses shared between commercial MAPs:

- *Rich, but unnecessary functionality*: Being general-purpose frameworks, most available MAPs incorporate rich functionality, yet, usually unnecessary for management applications. Some of these features are ‘hard-coded’ within an ‘Agent superclass’ that has to be extended in order to implement application-specific MAs and result in large MA sizes that affect the usage of system and network resources. A subset of the provided features would suffice for the majority of management tasks, however the exclusion of the non-desired features is not feasible;
- *Lack of essential features*: Some features considered as essential when designing a flexible management system are not supported by most general-purpose MAPs. For instance, the ability of MASs to distinguish between different versions of the same MA class, which may reflect the update/modification of an existing management task;
- *Heavyweight migration schemes*: The minimisation of MAs migration overhead is of major importance for large-scale monitoring applications that involve frequent polling of a large set of NEs. Existing MAPs incorporate complex and heavyweight migration protocols that result in increased network overhead, which typically exceeds that of static distributed objects communication mechanisms [KNI99, BOH00a];
- *No open-source MAPs*: At the time this research commenced, there was no open-source MAP that would allow the author to modify the code and perform application-specific optimisations;
- *Questionable support*: Companies shipping commercial MAPs, often suspend their support, e.g. General Magic has discontinued the Odyssey project, while that also seems to be the case with IBM’s Aglets [INCA].

All these weaknesses make difficult the selection of a general-purpose commercial MAP as implementation platform and led to the development of numerous application-oriented MAPs, many of which are tailored to management applications (see Section 3.6.1).

3.5.7. Applications of Mobile Agents

MAs can be useful in several application fields, although none of them necessitates their use; in fact, each application can be designed based on existing technologies [HAR95, MAG96]. However, the use of MAs can contribute to build these distributed applications in a more simplified and effective way. In the following, we identify some areas in which MA technology can actually give a positive contribution. NSM applications are not mentioned herein as they will be elaborated later.

- *Information retrieval*: MAs can be an effective tool for retrieving information within a distributed system; in fact, an agent encapsulating the user’s query can migrate to the place(s) where the information is actually stored; therein, the agent can obtain and filter data, and return the user only the useful information [ISM99]. This idea has been used in [PAP99] to reduce the latency involved in remote database interactions.
- *Electronic commerce*: E-commerce is an increasingly expanding area in the Internet; MAs can help users to search the products that meet their requirements, find the most cost-effective offers, etc. [DAS99].
- *Mobile computing*: Users want to access network resources from any position, notwithstanding the band limits of current wireless technologies. Thus, users can submit their requests through an agent, which runs their request within the network and returns the results later (so the user does not need to remain connected, waiting for the results) [CHE95, MIL99].
- *Distributed Computation*: MAs represent new paradigm for parallel execution of computation-demanding tasks on a distributed network of workstations [SIL99b, GHA99].

3.5.8. Performance Evaluation of Mobile Agents

It is often argued that the advantage of agent migration lies in the reduction of (expensive) communication costs by moving the code to the data rather than the data to the code [HAR95]. Although this argument is understandable from an intuitive point of view, not much research has yet been conducted to evaluate the cost of agent migration on a quantitative basis. Performance models regarding network load and execution time are needed to identify situations on which agent migration is advantageous compared to RPCs and help to decide which interaction model to be used.

A performance model for MA systems is introduced in [STR97]. The conclusion drawn from this model is that an alternating sequence of RPCs and agent migrations may perform better than a pure sequence of RPCs or a sequence of agent migrations. In particular, it is argued that agent mobility reduces both the overall latency and network load in cases that MAs visit devices where the amount of data to be processed is large compared to the size of the agent and the *selectivity* of the agent, i.e. its ability to reduce the size of the returned data by remote processing, is high. This conclusion was validated by experimental measurements in a realistic Internet-scale network scenario, using a prototype MA system implementation (Mole [STR96]). Along the same line, Chia et al. proposed *strategic mobility* [CHI97], whereby agents may choose to migrate to selective resources or instead to communicate with the needed resource over the network, depending on the problem's characteristics and the underlying computing and network infrastructure.

NSM-oriented performance evaluations of MA platforms have been reported in several research papers [BAL98, PIC98, RUB99, BOH00a], reviewed in Section 3.6.3.

3.5.9. Discussion

It should have already become evident that MA technology cannot be considered as an ideal solution for structuring *every* distributed application. A major problem that has prevented the wide spread of MA technology is the difficulty to identify 'killer applications' [MIL99, KOT99], i.e. applications that would strongly enforce the use of MA technology and become the driving force towards its further spread, adoption and exploitation.

That is a rather controversial issue which reminds of the object-oriented versus procedural programming debate[5]. Although killer applications probably do not exist, there is a number of application scenarios where MAs may improve the performance or efficiently complement REV/COD paradigms (see Section 3.4). In that sense, MAs should be considered only as another tool in the arsenal of distributed application designers [PIC01]. Concluding, we believe that the decision on whether to use MAs or alternative mobile code technologies should follow an unbiased quantitative and qualitative evaluation and depend on the specific characteristics and requirements of the examined applications.

3.6. MOBILE AGENT-BASED NETWORK MANAGEMENT

The potential of MAs in structuring distributed application has been early recognised by the management community, triggering intense research activity on MA-based distributed management. In the field of Network and Services Management, Magedanz [MAG96] was the

[5] At the time that object-oriented (O-O) programming was at its infancy, several developers often claimed that the O-O approach did not have a killer application either, still, it is now well accepted and widely used. While this claim is questionable, it is true that O-O applications can also be implemented in more traditional ways (procedural programming).

first to signal the potential of MAs, describing several scenarios in which their use can offer important benefits. MAs can encapsulate management scripts and be dispatched on-demand where needed. An MA can be sent to a network domain and travel among its elements collecting management data, and return with the data filtered and processed. Sending an MA for this task is a substitute to performing low-level monitoring operations and processing them centrally. Through semantic compression of collected data, the agent size can remain small and save bandwidth usage.

In an interesting study on code mobility, Baldi et al. [BAL97] presented another advantage of using MAs; when the network administrator is connected via an unreliable, costly or lossy link, he/she can create MAs off-line, connect to the network to dispatch the agents, close the connection and then reconnect later to get his agent back with the results. This principle is actually implemented in Astrolog [SAH97].

This section starts with a description of several MAPs tailored to NSM, overviews several MA-based management applications classified in terms of the mobility scheme they utilise, discusses performance aspects related to these applications, surveys the research activity on the field of active networks which is relevant to the scope of this thesis, and finally discusses efficient organisation and mobility models for MA-based NSM frameworks.

3.6.1. Mobile Agent Frameworks for Network Management

The limitations of general-purpose MAPs, as highlighted in Section 3.5.6, have led to the development of numerous management-oriented MAPs, aiming at optimising flexibility and performance aspects. Simply viewed, these MAPs consist of two components: The MAs themselves and the nodes where they can migrate and execute. There are several names synonymous to nodes where MAs execute, such as Mobile Code Daemon [SUS98], agency [SIL99a], place [ZAP97, BEL99], *lieu* [SAH97], etc. We adopt the term Mobile Agent Server (MAS) for the remainder of this section. In the following sections, we present the most representative platforms explicitly tailored to NSM applications.

3.6.1.1. IMA

The Intelligent Mobile Agents (IMA) framework [KU97], developed in Arizona State University (USA), has been the first MAP intended for building decentralised NSM applications. As such, IMA provides minimal functionality and poor integration with management standards. IMA consists of three major components: the *MA launcher* or managing entity, the *MA code* and the *agent host*. The managing entity is responsible for launching the MAs and processing the data collected by them. The MA is a software program, which migrates among managed entities to collect information based on the policies defined by the managing entity. The agent host is capable of receiving MAs and providing them access to local resources. The agent host runs as a daemon process at each managed entity able to receive and authenticate MAs.

3.6.1.2. MCT

The Mobile Code Toolkit (MCT) has been developed as part of the Perpetuum Mobile Procura project [PMP] in a pursuit of the ultimate goal, a plug-and-play network [BIE97], offering the basic functionality required to deploy NSM MAs. Every node to which MAs can migrate has to run a Mobile Code Daemon (MCD) that includes a Migration Facility and a Mobile Code Manager. The Migration Facility provides transport facilities to the agents. The Mobile Code Manager manages the lifecycle of the agents present on the MCD. The access to managed resources is handled by Virtual Managed Components, which provide a uniform interface for the MAs to monitor and control the visited NE [SUS98]. In [WHI99], an MA injection client is also introduced in order to create, deploy and manage MAs in the network. Also, authentication and data integrity features have been incorporated in the MCT framework.

Several kinds of MAs are defined [BIE98a]. Applets, servlets and extlets are single-hop downloadable components where applets are used for COD and servlets and extlets are used for REV. Deglets are multi-hop agents with limited persistence, that terminate as soon as their task is completed. Finally, netlets are persistent multi-hop MAs.

3.6.1.3. INCA

INCA (Intelligent Network Control Architecture) [NIC98] is an open architecture for the distributed management of multi-service networks. It supports three code transfer schemes: *Push code distribution* is used when the itinerary of the agent is known at creation time, where the code of the agent can be pushed to the end nodes before the agent is launched. In the *pull code distribution*, a station that receives a new MA has first to fetch its code. Finally, in the *migration code distribution*, the code travels along with the MA state. Another original feature in INCA is that the network administrator can assign priorities to MAs, depending on the urgency of their task. Other interesting features include reliable and fault-tolerant communications between stations, monitoring of the agent population deployed in the network and facilities to launch and control MAs execution. In addition, INCA includes a *location service* and a *naming servic*e, used for inter-agent communication. It is noted though, that INCA architecture has *not* been implemented.

3.6.1.4. MAGENTA

MAGENTA (Mobile AGENT environment for distributed Applications) [SAH97], developed in the Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA, France), targets mobile user applications where users are connected to the network with unreliable and expensive connections. Hence, MAGENTA assists the administrators to remotely control their managed network (possibly through portable computers), through launching MAs to carry out distributed management tasks. Places where MAs can travel (MASs) are called *lieu*s. The heavier part of the management functionality is integrated into a static *management server*. The framework is enriched with features such as fault tolerance, adaptability of MAs to changes in the environment (they are capable of detecting the disappearance of a lieu), etc [SAH98]. MAGENTA also offers a limited support of security based on access control.

3.6.1.5. AMETAS

The Asynchronous MEssage Transfer Agent System [AMETAS] developed in University of Frankfurt (Germany), uses the concept of a *place* as a MAS. Each place offers a mailbox system that allows agents to communicate asynchronously. A place can extend its capabilities by installing services that can be accessed through a control interface. MAs can therefore access managed resources after installing appropriate services. Another feature of AMETAS is that it defines human-agent interfaces through the definition of *user adapter*s. Finally, AMETAS uses a complex security mechanism including authorisation, access control and encryption [ZAP97].

3.6.1.6. SOMA

The Secure and Open Mobile Agent (SOMA) architecture [SOMA], developed in the University of Bologna (Italy), concentrates on security and interoperability as its two main design objectives. SOMA has been developed on the top of an MA environment used for NSM, MAMAS (Mobile Agents for the Management of Applications and Systems) [COR98a]. It enforces a strict security model with authorisation, authentication, integrity and secrecy features. Interoperability with other MAPs is achieved through the support of the OMG MASIF specifications [MASIF]. Moreover, a software add-on ensures interoperability with CORBA-compliant distributed applications and allows inter-operation with legacy systems through CORBA gateways [BEL99]. Several abstractions are defined within the MAMAS architecture: The *place* abstraction where agents can execute (i.e. the MAS); the *domain* abstraction which encloses a set of places; domains typically represent LANs and include a default place that embeds the *gateway* abstraction responsible for interconnecting different domains.

3.6.1.7. JAMES

JAMES [JAMES] is a MAP designed for the management of telecommunications networks, developed as part of a collaborative project between University of Coimbra (Portugal) and Siemens. MASs in JAMES are called *agencies*. The *JAMES manager* allows the network administrator to control active agents and agencies. A *code server* provides a central repository where MA codes are stored [SIL99a]. Moreover, JAMES agents have a passive migration strategy according to which the itinerary of each agent is known at launch time. Agencies make use of this property to pre-fetch the agent code from the code server [SOA99]. This leads to improving MA migration performance. JAMES also uses a checkpoint mechanism to provide fault-tolerance. Agencies states are periodically saved on a persistent medium so that they can be recovered after a crash and restarted from the last saved checkpoint. Moreover, MAs are saved and maintained until they reach the next checkpoint.

3.6.1.8. MAP

The Mobile Agent Platform [MAP] has been developed as part of a collaborative project in Universities of Catania and Messina (Italy). MAP architecture includes the following components [PUL00a]: the *Server* (MAS), able to accept and activate MAs; the *Daemon*, listening on a specific port for visiting MAs; the *Context*, that maintains a list of locally executing MAs and manages inter-agent communication; the *NetworkClassLoader* that enables MAs to run on a Server, even when their class is not present therein; the *CodeServer*, integrated within the Context, which stores the classes available at the Server. Additional features allow for synchronous/asynchronous inter-agent communication, to remotely retrieve information about MAs and change their execution state, etc. A key feature of this platform is its compliance with the MASIF standard, ensuring interoperability with other MAPs.

3.6.1.9. CodeShell

CodeShell [BOH00c], developed in University of Surrey (UK), is an optimised mobile code platform supporting the constrained mobility paradigm (see Section 3.5.5.2). As such, CodeShell cannot strictly be classified as MAP, as it does not provide migration facilities to launch MAs. The motivation that led to the development of this platform has been to address the performance limitations (mainly large migration delays) of general-purpose MAPs (see [BOH00a]). The basic components of the CodeShell architecture are a *communication service* (using Java-RMI) which provides a mechanism for delegating management logic along with initial parameters to remote machines and a *naming service* that distinguishes between objects and also binds one object to another.

3.6.1.10. Discussion

In this section, we described a number of MAPs designed and implemented for NSM applications, which address some of the limitations of general-purpose platforms, discussed in Section 3.5.6. All the presented MAPs support multi-hop agent applications, with the exception of CodeShell, which exclusively supports constrained mobility, however none supports memoryless or strong mobility. Interestingly, only SOMA and MAP comply with the OMG MASIF standard (see Section 3.5.4), while Java is the implementation platform in all cases. However, a number of limitations have been identified on these MAPs:

- *Heavyweight migration scheme*: With the exception of few (e.g. [PUL00a]), existing MAPs involve the transfer of both state *and* code at each MA migration. The transfer of code though is unnecessary, unless the MA visits a device for a first time, as the Java CL stores every loaded class on a local code table. That inefficient scheme may result in serious scalability problems both in terms of latency and migration overhead. This problem is partially addressed through the migration strategy proposed in [SOA99] and [PUL00a], where *only* the MA state is transferred and should the corresponding code is not present at a visited device, the device's CL contacts and downloads the code from a remote code server. This approach is very efficient in terms of network traffic (MA code is transferred only when necessary), however it increases the latency (the MA's execution cannot start until its code is downloaded). In addition, it involves more complex migration mechanisms, which are not necessary when MAs itinerary is known in advance. Instead, we have chosen to adopt the 'push' scheme (defined in [NIC98]), whereby bytecode is distributed at the MA's construction time with only the persistent state transferred thereafter, resulting in minimal usage of network resources (bytecode size is typically much larger than state size [BAL98]) and faster class loading. This migration scheme is described in Section 4.4.1.4.1 with a more refined design detailed in Section 6.3.5.
- *MA services customisation*: The development and customisation of MA-enabled NSM tasks is not effortless with available MAPs, as it requires programming skills and detailed knowledge of the MAPs' design. In Section 4.4.1.4, we introduce a tool that automates the generation of service-oriented MAs in a user-friendly manner, according to specified operational requirements.
- *Class loading*: Most MAPs include a CL component, able to receive and load at runtime visiting MAs bytecode. Yet, to the best of our knowledge, there is not any MAP which

allows to modify (overwrite) the bytecode, i.e. to dynamically upgrade MA-enabled management tasks. This problem is related to a limitation of Java class loading mechanism, which we address through a customised CL, described in Section 4.4.3.

- *Security*: Not all management-oriented MAPs sufficiently address security issues related to MAs. Examples of platforms without any support of security mechanisms or adopting weak security schemes, include IMA, INCA, JAMES, CodeShell. In Section 4.4.1.3.2, we describe a security component, which is integrated within our MAP and provides authentication, authorisation and encryption services.
- *Fault tolerance*: MAPs should be able to survive situations where link or node failures disrupt the normal migration process of roaming MA objects or the communication with the manager station. However, only few platforms (MAGENTA and JAMES) have addressed fault tolerance issues. In Sections 4.4.2 and 6.3.9, we discuss how network or node failures are dealt within our platform.
- *MA organisation models*: The reviewed MAPs define architectures comprising two hierarchical levels, corresponding to the manager and the NE ends, with the MAs used to delegate NSM functionality from the manager to the NEs. This organisation model is suitable for the management of small or medium-sized LANs but not adequate for large-scale, geographically dispersed enterprise networks, typically structured in logical hierarchies. This problem is discussed in Section 3.6.5.1.

3.6.2. Mobile Agent Applications in Network Management

MA-based distributed management has been a very hot research topic in the past few years, with a large number of proposed applications reported in the literature. This section provides an overview of MA-based approaches in a broad spectrum of applications, including network, systems, fault, configuration and service management. In order to provide a more structured overview, we follow the classification of Section 3.5.5.2, grouping these applications according to the mobility scheme used (single vs. multi-hop agent applications) and the control on agents' itinerary (passive vs. active migration). Most of the applications described below use the MAPs described in the preceding section as underlying platforms, whereas others have chosen general-purpose platforms such as Voyager [FER01] or Aglets [PIN99, CHI99].

3.6.2.1. Single-hop Agents in Network Management

As a general rule, single-hop migration is useful to encapsulate a function or a service into an MA and to deploy it to a remote location. A number of applications make use of this idea. White et al. [WHI99] suggest that a Virtual Managed Component (VMC) resides at each NE, providing incoming MAs an interface to managed resources. The manager side of the NMS requires a similar interface called the Virtual Managed Resource (VMR), which can be viewed as a remote wrapper of the VMC. The VMR can be supplied as a single-hop MA that travels to the remote NE, thereby, seamlessly enabling the management of newly installed NEs. In addition, this allows network component suppliers to transparently use any NSM protocol (even a proprietary protocol) for the management of NEs.

In [PUL00b], four types of MAs (programmed on the top of MAP platform, described in Section 3.6.1.8) are identified to perform management functions. Among them, the *daemon agent* is a single-hop agent sent to a network node to locally compute a health function (a linear aggregation function of several MIB values) and automatically notify the manager station when certain thresholds are exceeded. This scenario provides the advantage of saving bandwidth when compared to a remote polling-based scenario. Single-hop agents are also supported by the CodeShell platform (see Section 3.6.1.9), mainly applied to performance management applications: the aim is to provide traffic rates, QoS alarms and periodic summarisation reports by observing raw information such as traffic counters on NEs [BOH00c].

Other possibilities of applying single-hop agents are suggested in [MAG96]. A service or a network provider can send single-hop MAs to user end-points in order to adapt his equipment to new services. These agents can also achieve other tasks such as user accounting and capturing user requirements.

3.6.2.2. Multi-hop Agents in Network Management

Fixed Itinerary (Passive Migration)

The most typical and widely-used scenario of applying multi-hop agents is to deploy an agent to a list of hosts to locally perform management tasks and return to the manager processed management information. In this context, mobility allows the agent to perform semantic compression [BAL98], take decisions based on the past visited nodes and bring back a report result to the management station. This scenario is proposed in many works for different management activities.

Several research works on MA-based network monitoring involved rather simplistic applications, whereby a single MA object sequentially visits a predefined set of hosts, collecting a number of MIB values from each one without performing any processing upon them. This approach, used in [KU97, SAH98, CHI99] fails to solve the scalability problems of SNMP management as the state of travelling MAs grows rapidly (due to the unprocessed data accumulated into the MA state) resulting in increased network overhead and response time, in addition to imposing computational burden to the manager station where data processing takes place [BAL98]. The same principle is used in [COR98a], where MAs return system resources utilisation reports from a group of devices. Simulation results have shown that using MAs as management data collectors can be more efficient than SNMP-based management only when the management of remote domains (separated from the manager station by bottleneck links) is considered [RUB99].

The ability of MAs to return high-level information is exploited in [PUL00b], which defines a *verifier agent* that returns a list of nodes verifying a certain condition, e.g. overloaded CPU. In [SIL99a], MAs are used in a TMN environment to collect and process performance data from a set of NEs in order to produce *global* reports about the performance of the network. More advanced applications are proposed in [ZAP99] that presents NetDoctor, an application built on top of the AMETAS platform (see Section 3.6.1.5). NetDoctor addresses scalability issues by delegating NSM tasks to MAs that migrate to remote domains where they act as local managers, performing SNMP operations. Several interesting applications are proposed, including evaluation of health functions, termination of mis-behaving processes to free-up system resources, etc. Similar applications have been proposed by Pinheiro et al. [PIN99] that defined *discovery agents* to discover the existence of specific MIB variables on given hosts, and *aggregator agents* to perform computations on MIB variables (several aggregation levels may be defined). An interesting aspect is the dynamic adaptation of the proposed architecture to changing network conditions, so that MAs move closer management data to minimise their intrusiveness, in terms of the NSM-related traffic. This application has been developed using Aglets as underlying mobility framework.

El-Darieby et al. proposed a fault management application, using intelligent MAs endowed with a rule-based engine that allows to infer and diagnose possible faults on visited nodes [ELD99]. Another interesting application is described in [KNI99] that uses MAs for monitoring the conformance of Service Level Agreements (SLA) in enterprise networks. The paper suggests to use MAs that periodically roam the network to collect SLA monitoring results produced by other agents standing close to locations where user applications execute. Both the applications described in [ELD99] and [KNI99] are developed on the top of the MCT platform, described in Section 3.6.1.2.

Feridun et al. [FER01] presented the Distributed Management Framework (DMF), an architecture built on the top of the Voyager platform. The execution environments for incoming MAs are provided by the Distributed Management Nodes (DMN), characterised by a highly modular and lightweight design. The DMF has been tested on an application scenario where an *enterprise manager*, running on a DMN, analyses IP traffic characteristics of remote subnets. Upon receiving an event reporting that a performance threshold has been exceeded, the enterprise manager dispatches an MA to the remote subnet where the event originated from. On arrival, the MA starts monitoring the traffic activity on the subnet using a packet sniffer. When the specified monitoring period ends, the MA returns back to the enterprise manager to deliver its collected data.

The *Mobile Disman* architecture described in [OLI99] goes one step further, integrating the IETF's Distributed Management (Disman) framework [Disman] with a MA-based NSM framework. Disman defines an architecture where a main manager can delegate control above several distributed managers (DM), thereby improving the scalability, robustness and flexibility of centralised approaches. In Mobile Disman architecture, MAs can by used to implement DMs; providing mobility support to DMs allows them to adapt to dynamic environment conditions, offers location transparency and simplifies tasks such as data correlation and tasks distribution. However, Mobile Disman architecture is heavyweight, i.e. it comprises many resource-demanding components that would certainly increase the requirements on system resources; the fact that Disman is still only an Internet draft should also be considered. In addition, Mobile Disman architecture is *not* supplemented by a prototype implementation.

Dynamic Itinerary (Active Migration)

In their pioneer work, Appleby and Steward demonstrated the feasibility of employing multi-hop actively migrating MAs to control traffic congestion in circuit-switched networks [APP94]. A first class of MAs, called *parent agent*s, randomly navigate among the network nodes and collect utilisation information. By keeping track of this information, they gather an approximate utilisation average of the network nodes. Therefore, they are able to identify congested nodes, relatively to this average. When a congested node is found, a *load-balancing* MA is created to update the routing tables of the neighbouring nodes so as to reduce the traffic routed through the congested node. This application inspired other researchers to further develop its ideas by using *biologically* inspired agents. The work of Minar et al. [MIN99] covers this ground, using MAs to configure routing tables in a highly-dynamic radio frequency network where nodes are low-power transceivers, moving from one location to another in a two-dimensional space.

Shoonderwoerd et al. [SCH97] proposed to use *ant*-like MAs to achieve load balancing in telecommunications networks. MAs randomly roam the network and put *pheromones* depending on the distance from the source and the congestion of the followed route. Deploying a sufficiently large number of such ant-like agents allows to route calls according to the distribution of pheromones. Simulations show that such MAs significantly decrease call rejections compared to other approaches. An improvement to this work is proposed in

[BON98], where MAs no longer roam the network in a completely random manner, but follow those places where the strength of pheromones is higher.

White et al. [WHI98] applied a similar approach in a network fault location application, using multiple interacting *swarms* of MAs. Four types of agents are defined. *Service monitoring agents* monitor the compliance of service instances to the required QoS. When significant changes are detected, a *service change agent* is sent to mark the resources on which the service depends (increase the pheromone intensity when the QoS has downgraded). *Condition sensor agents* continuously roam the network and evaluate specific conditions on the visited nodes, with the tendency to visit more frequently those nodes where problems have been detected. Nodes with strong pheromone attract a fourth type of agents called *problem identification agent*s, with the capability to diagnose and repair certain patterns of problems. As a general comment, it should be stated that the applicability of ant-based solutions in the field of telecommunications is questionable; so far it has only been demonstrated in simulated environments.

Active migration has been also applied in network discovery and dynamic configuration of networks and services. In [SCH98] and [WHI99], the authors proposed a scenario, where MAs (*netlets*) continuously roam the network to discover new devices and detect removed components. Netlets are suitable for dynamic networks with frequently changing topologies and supported services. An MA-based network discovery application is also described in [FER01].

In another work, Pagurek et al. [PAG98] developed a simulation testbed for the configuration of Permanent Virtual Channels (PVC) in heterogeneous ATM networks. The idea is to dispatch an MA that travels across the network switches, and progressively configures the PVC fragments on each switch. After configuring the PVC route on a switch, the MA travels to the next switch where it uses configuration information of the past switches to correctly configure the current one. Cheikhrouhou et al. proposed a similar application [CHE00a], based on static intelligent agents with dynamically extended capabilities.

It is essential to notice that in the above applications, MAs provided a very interesting tool to model either lightweight moving entities or biologically inspired entities used for management tasks. The MA metaphor allows to model a self-contained entity that evolves in its environment to accomplish simple tasks according to its *own view* of the network.

3.6.3. Performance of Mobile Agents in Network Management

While the main focus of MA-related research activity on distributed management has been on developing NSM-oriented frameworks and using them on specific applications, some researchers concentrated on evaluating the performance of MA-based management. The purpose of such performance studies is twofold. First, they suggest ways to efficiently use MAs so as to outperform centralised NSM approaches. Second, they allow to determine how MAs are best deployed for particular types of NSM operations.

Pioneer work on this direction has been presented in [BAL97] and [PIC98]. A management task carried out on a set of NEs, involving multiple CS interactions with each NE has been examined. Models of the overall traffic and the traffic around the management station are computed for CS, CoD, REV and MAs approaches. The impact of semantic compression is also studied for mobile code approaches. The purpose of these models is to provide the network administrator with an objective quantitative criterion to choose the right approach, given the management task and the managed network topology.

Liotta et al. [LIO99] presented an evaluation of MA-based monitoring systems, providing quantifiable metrics based on performance and scalability. MAs are used to perform monitoring tasks in place of traditional centralised polling. Performance is measured using the generated monitoring traffic and the monitoring delay. These parameters are evaluated for a combination of schemes based on monitoring models, MAs organisation and MA deployment patterns. Possible monitoring models consider MAs performing periodic polling. In the first model, each MA polls and analyses a set of managed objects (MO); this MA is subsequently polled by other MAs or the monitoring station. In the second model, each MA periodically polls the MOs, analyses the results, and *notifies* other MAs or the monitoring station. The third model differs from the previous on that MAs generate data (alarms) only if specific *events* are detected. The MA organisation can either be flat or hierarchical, and the deployment scheme can either be cloning-based or without cloning. This leads to four possible deployment patterns: flat broadcast with no cloning, flat broadcast with cloning, hierarchical broadcast with no cloning and hierarchical broadcast with cloning.

In a more recent work [LIO01], the authors investigated the problem of optimal placement of MAs acting as remote monitoring stations, so as to minimise data collection latency and the network traffic incurred due to (localised) polling. To address this problem, a distributed algorithm is proposed that relies on agents learning about the network topology through standard management interfaces and subsequent deployment of MAs to remote domains through a ‘clone and send’ process. Deployed MAs can possibly adapt to network changes and move again in order to maintain optimality.

The latency aspects of MA migrations, with respect to management applications, have been investigated by Lipperts [LIP00]. The results of time measurements indicate that MA migrations can be more time efficient than remote communications, especially when class loading is not necessary (the code of the MA is already at the destination host) and the operations are performed over slow media. However, the decision concerning the replacement of a remote communication scheme by MA migrations should take into account several parameters, such as the number of remote communications to be replaced, the size of the parameters involved in the remote communication, the size of the agent and the networking environment. Lipperts proposed a solution based on utility theory to aid on deciding whether MAs deployment is justified or not.

[ELD99] and [ZAP99] included brief quantitative evaluations of their proposed applications demonstrating improved scalability compared to SNMP management. Similarly, [BOH00c] compared the performance of CodeShell against that of Grasshopper, RMI and CORBA-based approaches in terms of response time, generated traffic and memory requirements. The experimental results indicated that CodeShell offers the management flexibility benefits of general-purpose MAPs, whilst achieving performance characteristics comparable to static DOT-based approaches.

3.6.4. Active Networks

Active Networks (AN), which originated at MIT’s Software Devices and Systems Group, advocate the use of *active packets* or *capsules* [TEN96]. Active packets contain user data and programs, which are executed on every node along the route of the packet through the AN. Such packets require a software layer on top of the hardware and communication protocols, which can provide dynamic configuration, improved security, interoperability, extendable protocols, etc. AN research is relevant to the context of this thesis, as programs carried by active packets, resemble the functionality of MAs: an active packet can be thought of as a multi-hop MA, which executes on every node along its path through the network; likewise, a travelling MA can be mapped to an active packet, containing both data and a program, sent from one active node to another, in an AN. Since the motivations for proposing the MA and AN concepts are on the same direction, MA and AN technologies start to converge and overlap, specifically in the management domain [KAW00]. Nevertheless, only recently have AN concepts been applied to distributed management [GRE99, RAZ00, KAW00].

Greenwood and Gavalas [GRE99] described a modular framework that exploits the ability of active mobile processes or agents to interact with one another in order to realise active network architectures. The process execution environment is integrated into the network devices. The process kernel houses both pre-loaded and visiting agent processes. The proposed framework is compared against traditional SNMP-based with the former outperforming the latter both in terms of response time and network overhead when the calculation of health functions combining large numbers of MIB objects is considered.

Raz et al. [RAZ00] described a prototype system where legacy routers are enhanced with an *active engine*, which enables the rapid deployment of new distributed management applications. In particular, the active engine comprises an environment in which code encapsulated in active packets can be executed. Physically, the routers forwarding mechanism and the active engine may either reside on different machines or co-reside in the same box. This prototype has been tested on a simple bottleneck detection application and offers the functionality of the well-known *traceroute* program, in a more efficient way.

The convergence of MAs and ANs is more clearly portrayed in [KAW00] that introduces the Active Distributed Management architecture. This architecture is characterised by a programmable middleware platform (organised in different layers of abstraction), whose active properties are drawn from the AN and MA paradigms. A number of applications are proposed including variables monitor/control, bottleneck detection, topology detection, etc.

3.6.5. Synthesis and Discussion on Mobile Agents-Based Management Applications

This section attempts to highlight the benefits and shortcomings related to the application of MA technology on NSM. The discussion is twofold: first, the effect of MA organisation models is investigated with respect to the scalability and flexibility of MA-based approaches on the management of large-scale networks; second, the effect of mobility schemes on management applications is discussed and ways to effectively exploit the mobility feature of MAs are suggested.

3.6.5.1. Organisation Models

As mentioned in Section 3.6.1.10, a common attributed shared between the MAPs developed for management applications is their structuring in two levels, corresponding to the manager and the managed devices, with the MAs used to delegate functionality from the former to the later. This simple organisation model implies a ‘flat’ network topology where a single MA is launched and visits the entire set of managed systems, causing scalability problems both in terms of latency and network overhead. Flat models are not suitable for the management of large-scale, hierarchically structured networks, which can be more efficiently managed by MA platforms organised in hierarchical fashion [LIO01].

The hierarchical organisation of agents is not an entirely new idea. For instance, hierarchies of *static* co-operating agents have been proposed in [QUE97] to support the resource-control part of a signalling system. In the management domain, this problem is partially addressed by the hierarchical framework described in [LIO98] and also MAGENTA, AMETAS and Mobile Disman architectures. In particular, Liotta et al. [LIO98] proposed an MA-based management architecture adopting a multi-level approach enabled by static *Middle Managers* able to launch MAs. The same principle is used in MAGENTA platform. A second approach has been reported in [ZAP99] and [OLI99] that proposed the use of MAs as mobile mid-level managers, performing distributed management tasks. The first approach [SAH97, LIO98] does not adequately address the flexibility limitations of static hierarchical management frameworks (see Section 2.11), Still, even the latter approach [ZAP99, OLI99] is not suitable for managing networks with dynamically changing topologies and traffic patterns, as they lack mechanisms to change the location where mid-level managers execute. This approach also involves the deployment of a new MA for each introduced monitoring task, which complicates the management of mid-level managers. In addition, critical issues such as criteria for segmenting the network into management domains, explicit determination of the domain boundaries or strategies for assigning mid-level managers to these domains, are not addressed. Furthermore, the architectures described in [LIO98] and [OLI99] are not supplemented by prototype implementations.

In Chapter 6, we introduce a hierarchical MA-based management framework that deploys at runtime mobile mid-level managers (when specific criteria are satisfied), with the ability to dynamically adapt to the managed network dynamics.

3.6.5.2. Mobility Schemes

The usage of MAs covers many aspects in NSM. *Single-hop MAs* are suitable for the easy deployment of software, functionality and services [MAG96]. In particular, single-hop MAs have been used to deploy adaptable instrumentation facilities, compute health functions [PUL00b], provide on-the-fly adaptors for newly installed NEs and services [WHI99]; also to return QoS alarms and periodic summarisation reports by processing raw management data [BOH00c]. In that context, single-hop mobility can be considered as successor of the MbD paradigm.

Multi-hop MAs go one step further, allowing the execution of management tasks on a set of NEs in a sequential manner. Multi-hop mobility may be preferable in several application scenarios (see Section 3.4), for instance when short-term tasks are to be executed over multiple NEs; in such case the code deployment time is reduced, the manager station is not overloaded and the network area around it is not saturated by the simultaneous generation and transmission of agents [BOH00b]. This mobility scheme is, therefore, suitable for collecting on-line data and performing simple monitoring and configuration tasks on several NEs.

Hence, multi-hop MAs have been used to compute health functions [ZAP99, PIN99], return reports related to a set of NEs [SIL99a], discover nodes verifying specific conditions

[PUL00b], perform traffic analysis on remote subnets [FER01], etc. Additional applications include fault diagnosis [ELD99], or collection of data pre-processed by other agents [KNI99, LIO99]. It should be emphasised though that applications using multi-hop agents for collection of data, later processed by the manager station [KU97, SAH98, CHI99, COR98a] do not scale, as not only processing bottlenecks are created at the manager platform, but also the network overhead is increased. Hence, it is essential to benefit from the semantic compression that multi-hop MAs can perform while moving from one host to another, thereby preventing the rapid growth of their state size and minimising the migration overhead [BAL98].

An aspect of management scalability not sufficiently addressed by existing applications relates with the management of large sets of NEs. In such case, launching a *single* multi-hop MA with the task to collect data from all the managed devices can lead to large round-trip delays and also to increased network overhead, even when performing semantic compression of data. In Chapter 5, we address this issue by launching several MA objects, travelling in parallel, with each visiting a relatively small group of devices.

Interestingly, although the potential of using multi-hop agents to perform correlation of data collected along their itinerary has been recognised by researchers [FUG98, LOP00], none of the existing applications implements this principle. MAs can be used to realise a simple data correlation model, providing more even distribution of processing load and returning high-level information to the manager station with no need for further processing. In Chapter 7, we describe an application scenario where MAs can be efficiently used to perform correlation of management data.

Active migration is often combined with a society of MAs imitating ant-like behaviour [APP94, MIN99, SCH97, BON98, WHI98]. Although such kind of MA systems is particularly difficult to manage, they are very suitable for dynamic networks, where random mobility combined with a large number of MAs allows to easily detect changes related to performance, faults or configuration and react accordingly. However, active migration is not necessary for network performance management applications, which represent the main focus of this thesis, as the list of managed devices involved in the monitoring process is typically known in advance. Should new devices are installed, the latter can advertise themselves directly to the manager, without requiring to be discovered by continuously roaming MAs (see Section 4.4.1.3.8). Besides, the use of active migration would imply increased complexity (extra code needed for itinerary control), hence, increased MA size and migration overhead.

Concluding, management applications can benefit either by single-hop or multi-hop MAs (see Section 3.6.3). The selection of the appropriate migration scheme should depend on the type of the application, the managed network size, the duration of the management task's execution, etc. As a result, NSM-oriented MAPs *should support both single-hop and multi-hop agents*. The design of our MA framework, introduced in Chapter 4, satisfies this requirement. Last, but not least, the choice of MA-based solutions instead of alternative approaches management distribution approaches should necessarily be justified through analytical evaluations. The performance of our framework and the applications developed on the top of it, is assessed through extensive quantitative evaluations, validated by experimental results.

3.7. SUMMARY

The advent of MA technology has signalled many potential benefits in the network management arena and attracted the attention of several researchers working on the field. In particular, MAs promise to overcome the limitations of traditional centralised architectures and address the weaknesses of distributed management approaches reviewed in the previous chapter. As a result, MA-based management applications have largely increased in number. The intensity of research activity on that field is strongly related with the proliferation of MAPs expressly developed with management applications orientation. These platforms have demonstrated improved security, fault tolerance, MA control, inter-agent communication and interoperability features, while some also aimed at optimising the performance of MA migrations through sophisticated code distribution mechanisms. All these facts signify that MA community is about to reach the state of maturity.

However, there are a number of issues related to the scalability and flexibility of MA-based management that need to be carefully evaluated. For instance, appropriate organisation models and mobility schemes that meet the specific requirements of management applications need to be identified and evaluated to ensure that MAs are effectively used and offer performance benefits. The investigation of these issues comprises the main part of the research work presented in the following chapters, which describe the design and implementation details of a MAP tailored to management applications, with the main focus being on network monitoring and performance management.