CHAPTER 2

NETWORK & SYSTEMS MANAGEMENT: STANDARDS, EMERGING TECHNOLOGIES AND THE SHIFT FROM CENTRALISED TO DISTRIBUTED PARADIGM

2.1. INTRODUCTION

In recent years, the use of information resources has dramatically increased, with organisations becoming more and more dependent upon reliable access to this information in order to remain competitive. The explosion in the size and complexity of today's local and wide area networks, combined with the increasing demands placed upon them for their resources resulted in establishing Network & Systems Management (NSM) as a factor of vital importance. To further complicate matters, the concept of single-vendor networks has vanished long time ago. Modern enterprise networks exhibit remarkable diversity of network and systems equipment necessitating the transition to *integrated management*. The primary objective of NSM is to maintain network and systems availability and health, aid in configuring the network and systems, guarantee Quality of Service (QoS), enhance performance, provide security, minimise operational overhead (execution of repetitive tasks) and decrease the cost of running the information technology infrastructure [HEG94].

As a consequence of sharing and interconnecting resources, NSM needs to meet the challenges of distribution, heterogeneity and transparency. A number of approaches and architectures that aim at standardising the management process and addressing the heterogeneity problem have, therefore, emerged. These approaches specify *management architectures*, which supply frameworks for standards of relevance to NSM. Two of the most widely used architectures are the Internet management architecture [HAR99] and the Open Systems Interconnection Systems Management (OSI-SM) architecture [ISO91a]. Other architectures, mainly applied on the telecommunication networks area include the Telecommunications Management Network (TMN) [CCITT92] and the Telecommunications

Intelligent Network Architecture (TINA) [TINA93]. More recent efforts focus on the definition of emerging management architectures, such as Web-Based Enterprise Management (WBEM) [WBEM], Java-based management [JBM] and Directory Enabled Networks (DEN) [STR99]. Distributed objects technologies, exemplified by the Common Object Request Broker Architecture (CORBA) [CORBA], represent another interesting approach which gains increasing attention in the management world.

The Internet management architecture has been criticised for exhibiting low degree of scalability[1], flexibility and reconfigurability, mainly attributed to its *centralised* architecture. The latter two weaknesses also characterise the OSI-SM framework. The emerging management technologies have only partially addressed these problems. The need for *distribution* of NSM functionality has been early recognised by researchers and developers active in this area and several initiatives have been undertaken in this direction; in fact some of them have already led to the specification of standards. However, there is still a long way to go before NSM distribution-related problems are satisfactorily addressed.

The remainder of this chapter starts with a description of the management functional areas identified by the ISO, in Section 2.2. An overview of OSI-SM and Internet management follows in Sections 2.3 and 2.4 respectively, highlighting the similarities and the differences of the two approaches. Special focus is then given to Internet management, describing the architecture, versioning history and limitations of its de-facto standard, the Simple Network Management Protocol (SNMP). Following that, several emerging technologies in the NSM field are described, including distributed object technologies (Section 2.5), Directory Enabled Networks (Section 2.6), Java-based management (Section 2.7) and WBEM (Section 2.8). The problems of centralised management are detailed in Section 2.9, while Sections 2.10 and 2.11 overview the most well-known approaches in distributed management, which however, address only certain aspects of the entire problem area; standardisation and research approaches are separately investigated. Section 2.12 comprises a synthesis and qualitative analysis of distribution approaches, classifies them in *static* and *dynamic* (depending on the degree of flexibility they offer) and identifies their strong and weak points, pointing out aspects of distributed management that could benefit by Mobile Agent (MA)-based approaches. Finally, Section 2.13 summarises the chapter.

It is emphasised that that the reader is assumed to be reasonably familiar with NSM concepts, SNMP, the way SNMP-based management platforms work in the IP world, how they

[1] *Scalability* is defined as the ability to increase the size of the problem domain with a small or negligible increase in the solution's time and space complexity [RES97]. In the NSM context, scalability is specifically defined as the ability to increase the number of monitored entities or the polling frequency, with a small or negligible decrease in performance.

are typically structured, and the management tasks they perform (for an introduction, see Rose [ROS96] and Stallings [STA99]). We therefore do not redefine here well-established concepts, but only those whose definition is not consensual and those that are relevant to this work.

2.2. MANAGEMENT FUNCTIONAL AREAS

Types of management activity have been categorised by ISO into five generic functional areas, collectively known as FCAPS from their initials [ISO91a]:

- *Fault Management*: the process of collecting information referring to network elements (NE) health. Integrated fault management systems receive reports about malfunctions (alarms), perform alarm correlation and diagnostic tests, identify faults and display various network alarms. This process can be optimised to perform root-cause analysis and suggest/take corrective measures.
- *Configuration Management*: controls the configuration state of a system/network and the relationships between components. It also initialises, configures and shuts down network equipment.
- *Accounting Management*: defines how network usage, charges and costs are to be identified in the networking environment. It is associated with tariffing schemes that generate charging/billing information.
- *Performance Management*: supports the gathering of statistical data, upon which it applies various analysis routines to measure the system performance. That way, it provides an accurate picture of network components and services. This process is capable of proactively pinpointing and forecasting potential problems before they actually occur, based on gathered information. It can predict congestion/bottlenecks and, hence, be used for network future expansions and capacity planning. Performance management represents a central application area for this thesis.
- *Security Management*: controls access to network, system, service and management components. It can offer authentication, confidentiality, integrity, access control and also handle cryptographic key distribution.

In the telecoms world, management platforms generally support most (if not all) of the OSI functional areas. This is not the case in the IP world, where most platforms support only a fraction of FCAPS. Indeed, management platforms are often simpler in the IP world than their counterparts in the telecom world.

2.3. THE OSI SYSTEMS MANAGEMENT

The OSI-SM [ISO91a] defines a management architecture with well-defined *organisational*, *informational*, *communication* and *functional* models. The organisational model assigns special roles to the management entities: the *manager* and the *agent*. A manager is an entity that controls the management process and makes decisions based on collected information, whereas the agents make available the management information to managers. Abstractions of system/network resources which need to be managed are represented by *managed objects* (MO). MOs encapsulate the underlying real resources and enable their manipulation through well-defined operations. An agent administers the MOs on its local device and provides mechanisms for performing management operations upon them, offering an interface to system resources. In essence, the agent acts as *name server* for the objects (resolves their names to internal handles), *object factory* since it creates and maintains objects and *event server* since it disseminates events (notifications are evaluated and forwarded as events to managers according to criteria preset by them) [PAV01]. The communication between the manager and the agents takes place via standardised management protocols. In general, the manager-agent paradigm can be thought of as client/server (CS) relationship, where the manager plays the role of the client and the agent the role of the server.

OSI-SM is based on a complex, object-oriented information model. MO classes are specified by templates and consist of attributes, operations that can be applied to the corresponding objects, behaviours exhibited by the objects in response to operations, and notifications that can be emitted by the objects. The functionality of a MO is defined at design time, i.e., it cannot change at runtime. MOs are logically grouped in *Management Information Bases* (MIB). MIBs are virtual, hierarchical, object-oriented databases including interrelated MOs in a managed environment. The basic architecture of OS-SM is depicted in Figure 2.1.

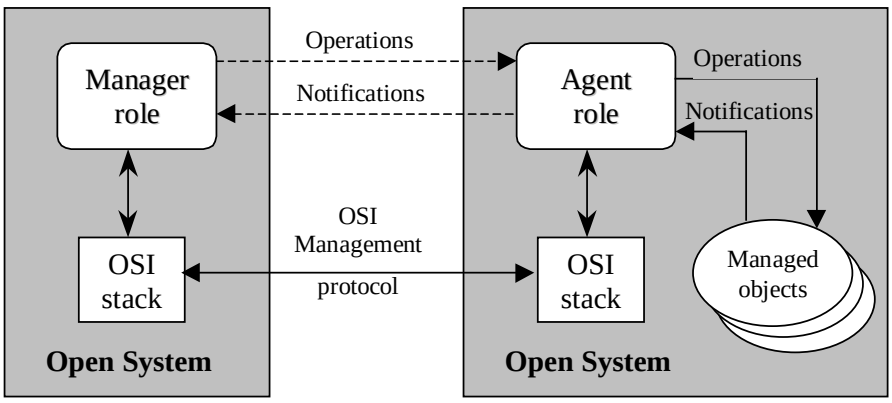

Figure 2.1. Basic OSI-SM architecture [SLO94]

The exchange of management information between managers and agents is defined by a service, the Common Management Information Service (CMIS), and its protocol, the Common

Management Information Protocol (CMIP) [ISO91b]. The CMIS provides management operation primitives that include M-GET to retrieve data, M-SET to modify data, M-ACTION to request the execution of an action, M-CREATE (M-DELETE) to request the creation (deletion) of an instance of a managed object, and M-CANCEL-GET to cancel an outstanding M-GET request. Agents may report events about managed objects using M-EVENT-REPORT. In addition, CMIS provides multiple-object access through *scoping* and *filtering* operations. Scoping allows management operations to be carried out on a selection of one or more managed objects. Filtering consists of boolean expressions with assertions on values of attributes in an object [ISO91b].

OSI-SM also addresses some aspects of management decentralisation through the standardised Systems Management Functions (SMF), which define a rich set of functionality specified in terms of generic object classes. Examples include *Metric Objects*, which measure resource performance, monitor thresholds and generate notifications [ISO93] and the *Summarization* function [ISO92], which provides a framework for the definition, generation, and scheduling of system information summary reports. These functions move intelligence in proximity to the managed resources, reducing the amount of management traffic and providing support for a sophisticated event-driven operation paradigm. Pavlou et al. proposed additional functionality that combines the capabilities of metric monitoring and summarization objects in a powerful fashion [PAV96].

The widespread interest in formal standards has generated considerable interest in CMIP, even though it has not been widely used. One factor contributing to the lack of CMIP's popularity is the slow evolutionary process of these standards.

2.4. INTERNET MANAGEMENT

Simplicity and small implementation overhead has always been the main objective of Internet management since the early days of its conception [CAS90, MCG91]. This seems to be the main reason that justifies its popularity and wide use. In this context, Internet management is characterised by a simple information model that lacks the object-orientation (MOs are nothing more than simple variables) and sophistication of its OSI-SM counterpart, but enables easier and faster writing and instrumentation of MIBs. The architecture depicted in Figure 2.1 is applied in the Internet management model as well; the difference is that the functionality of the corresponding modules is very much simplified.

The Internet management communication model relies on the Simple Network Management Protocol (SNMP) [CAS90] as the communication protocol, which defines connectionless services and primitives for getting and setting variable values and sending

notifications. SNMP has been developed with an orientation to TCP/IP networks. As it is the case with most protocols of this kind, its development and implementation occurred with considerable speed. A quick, easy and simple implementation was the first priority of its designers. Hence, the following guidelines have been adhered to:

- make it work over very uncomplicated protocols;
- keep the number of protocol message types small;
- stick to a unit of information that is a single value, such as an integer or string.

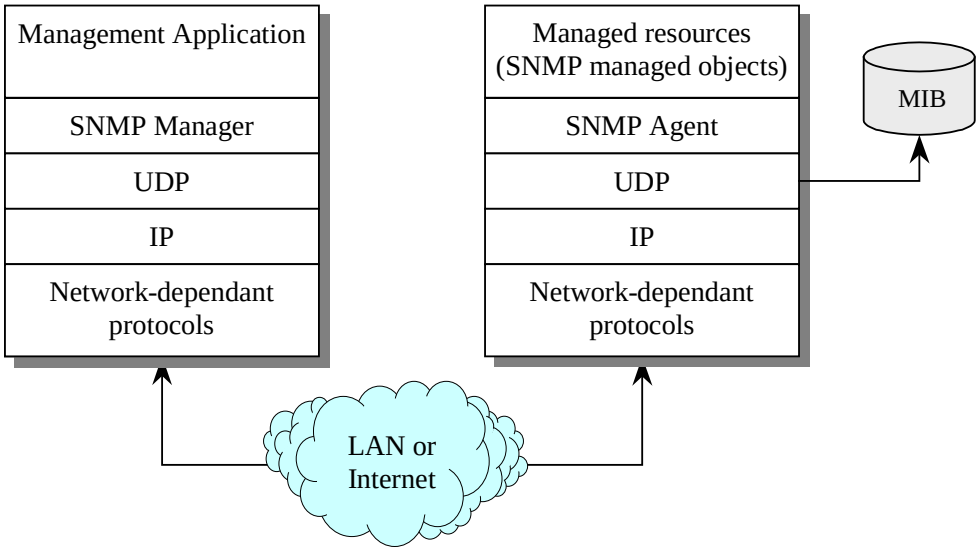

Figure 2.2. The SNMP layering [SLO94]

The User Datagram Protocol (UDP) was chosen as the SNMP transport protocol (see Figure 2.2). That decision was made mainly due to the scalability reasons. Namely, being centralised, SNMP would impose a huge demand on the manager platform system resources to be able to accommodate many open TCP connections to the managed devices. UDP also has small footprint on network resources compared to TCP, while being well suited for short request/ response type of operations, which is consistent with the connectionless nature of SNMP.

In general, the SNMP framework provides much poorer functionality and expressiveness than CMIP. However, because of the overall complexity and size of CMIP, many claim that this is a case of the cure's being worse than the disease. These problems and most importantly the domination of Internet over OSI have prevented CMIP from reaching the dominant market position that was originally anticipated. On the other hand, the inherent simplicity of SNMP has been the driving force for its wide acceptance and popularity.

2.4.1. SNMP Protocol Data Units

SNMP uses relatively simple operations and a limited number of Packet Data Units (PDU) to perform its functions. Figure 2.3 shows the types of messages exchanged between the manager and the agent. Five PDUs have been defined in the first version of the standard (SNMPv1) [CAS90]:

- *Get Request*: it is used to access the agent and obtain managed objects values. It includes identifiers to distinguish it from multiple requests.
- *Get-Next Request*: it is similar to the Get Request and permits the retrieval of the next logical identifier in a MIB tree.
- *Set Request*: it is used to change the value of a MIB object.
- *Response*: it responds to the Get, Get-Next and Set Request PDUs. It contains an identifier that associates it with the PDU it responds to. It also contains identifiers to provide information about the status of the response (error codes, error status and a list of additional information).
- *Trap*: it allows SNMP agents to report events at their local NE or to change the status of the NE.

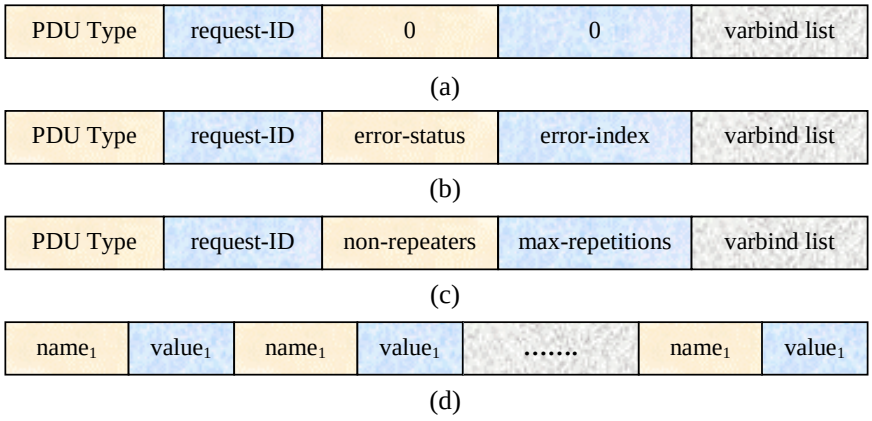

Figure 2.3: SNMP PDU formats: (a) Get request, Get-Next request, Set request, Trap, Inform, (b) Response, (c) GetBulk request, (d) varbind list

Management data are returned in a list structured as a sequence of <object ID: value> pairs, termed the *varbind* list. As shown in Figure 2.3, SNMP request and response messages have for simplicity reasons the same packet format. Later protocol versions (SNMPv2, SNMPv3), define two additional operations [STA99]:

- *GetBulk Request*: it has been devised to minimise the number of protocol exchanges required to retrieve large volumes of management data, although there is a maximum PDU size limitation. It includes a field the specifies the number of variables in the varbind list for which a single lexicographic successor is to be returned (`non-repeaters`) and

another field denoting the number of lexicographic successors to be returned for the remaining variables (`max-repetitions`).

- *Inform Request*: this PDU is used for manager-to-manager communication, i.e. it is sent by an entity acting in a manager role on behalf of an application, to another entity acting in a manager role, to provide information to an application using the latter entity.

2.4.2. SNMP versioning history

Two of the reasons for the initial success of SNMPv1 have been its lightweight design compared to OSI management and also the fact that it avoided the four-year standardisation cycles of the ITU-T [MAR00]. Yet, experience has shown that SNMP evolves at an even slower pace. The phenomenal success of the SNMPv1 architecture has also been the cause of its decline. It proved to be good for managing relatively small networks, but could not scale to large networks (e.g. geographically dispersed enterprises), and could not cope with large volumes of management data [MAR99b]. The telecommunications world had already shown how to solve this problem: by distributing the load across a hierarchy of managers. But strangely enough, management distribution was not a priority at the Internet Engineering Task Force (IETF) until the late 90s. Since SNMPv1 (standardised in 1990), four management architectures have been released: SNMPv2p, SNMPv2u, SNMPv2c, and SNMPv3 [STA99].

The first three only support centralised management. SNMPv2p has been rendered obsolete by the IETF in 1996 [PER97, STA99]. SNMPv2u had little success and "saw no significant commercial offerings" [PER97]; it is thus no longer used. SNMPv2c is often used to manage busy backbone routers, because it supports 64-bit counters and offers better error handling than SNMPv1; but it brings nothing new as far as distribution is concerned. As for SNMPv3, its main focus is on security [STA98], not scalability. In effect, it took eight years before the IETF delivered a substantial new release, SNMPv3, and another two years before major vendors began supporting it[2]. Its use is thus expected to remain marginal in production environments in the foreseeable future. Note that the MIBs adding support for one kind of delegation in SNMPv3 were issued only in 1999 [LEV99], so it will take even more time before they are implemented and deployed. In short, vendors of SNMP management platforms are currently forced to resort to proprietary extensions to support hierarchies of managers [MAR99a].

2.4.3. Strengths/Limitations of SNMP

According to the main characteristics of SNMP described in the previous section, the main strengths and contributions of SNMP-based management are the following:

- interoperability;
- simplicity;
- wide support by IP-equipment vendors;
- small footprint on agents;

On the other hand, SNMP also exhibits several weaknesses:

(I) Scalability

Scalability issues can be classified into the following categories:

Network overhead: In the context of NSM, network overhead is the proportion of a link capacity used to transfer management data, and thus unavailable for user data. The purpose of a network is to transfer user data, not management data, so an important goal of NSM is to keep network overhead low. SNMP is characterised by high network overhead, which is mainly due to the *polling*-based nature of monitoring and data collection process (pull model) [GOL91]. The manager repeatedly requests and retrieves specific MIB object values at each poll cycle from remote agents. In many monitoring applications, a considerable portion of network bandwidth is typically wasted to learn nothing other than that the network is operating within acceptable parametrical boundary conditions.

Latency: For polling, latency is the time elapsed between the moment the manager requests the value of a MIB variable and the time it receives it from the agent. It is important to keep latency reasonably low, so as to quickly detect and correct operational problems. End-to-end latency depends on networking conditions (the capacity and error rates of links, the speed of the IP routers traversed between the agent and the manager, etc), the amount of retrieved data and the number of protocol exchanges. When large sets of NEs need to be managed through SNMP, latency can very be high, especially when the 'control loop', i.e. the network distance, between the managing and the managed entities is large.

Manager's processing capacity: The manager's hardware resources (CPU, memory, etc.) dedicated to management applications cannot be continuously increased, due to cost and hardware constraints, setting a limit on manager processing capability [MAR98]. The centralised structure of SNMP architecture and the lack of data filtering capability characterising SNMP agents result in transferring vast amounts of NSM data, subsequently processed at the manager platform. This forces manager's processing capacity to its limits and intensifies the need to relieve the manager from performing routine data processing tasks.

---

[2] For instance, Cisco supports it as part of IOS 12.1.5, officially released in December 2000 [Cisco00].

Capacity of the manager's local segment: The management data sent by all the agents converge toward a single point, the network segment where the manager is connected to, inevitably creating a bottleneck [BAL97].

Inefficient bulk management transfers: As the amount of data to transfer grows, it makes sense to reduce the overhead by sending the data in bulk, that is, to send unlimited number of MIB variables at a time, while keeping the number of network interactions low. However, SNMP has not been designed for transfers of bulk management data (typically stored in SNMP tables[3]). In SNMPv1, tables are retrieved through successive `get-next` operations. If the table includes many rows, the manager must perform at least one `get-next` per row [SPR99]. For tables with hundreds or thousands of rows, an equal number of requests/responses will be transferred through the network increasing the network overhead and latency. The situation improves with the `get-bulk` operator offered by SNMPv2c and SNMPv3 frameworks [STA99], which allows transferring more data per SNMP message. However, the manager should guess the length of the table to be retrieved and accordingly choose a value for the `max-repetitions` parameter. Using a low value will cause more PDU exchanges than necessary. Using a high value, however, can result in an *overshoot* effect [SPR99]: the agent can return data of no interest for the manager. The problems associated with bulk management data retrieval are elaborated in Chapter 7, which proposes MA-based applications to address all these problems.

SNMP message maximum size: The large number of protocol exchanges required to complete bulk management data transfers is mainly due to the maximum size limit of SNMP messages. All SNMP agents must accept SNMP messages that are up to 484 bytes in length, but may legally refuse longer messages [CAS96]. Yet, many open-source implementations of SNMP have used the maximum size limit of 1472 bytes (in LAN environments), proposed in [ROS96]. Clearly, when transferring data in the order of Mb, a large number of PDUs will be exchanged; the exact number will depend on the maximum size limit used in the particular SNMP implementation.

Poor efficiency of BER encoding: The SNMP protocol uses the Basic Encoding Rules (BER) [ITU94] to encode management data prior to sending it over the network. BER encoding can be implemented with very compact code (small footprint on agents) and causes a reasonable overhead on the agent and manager for encoding/decoding. Yet, the amount of administrative data (identifier and length) is large compared to the payload (content). This makes the network

[3] Simple, two-dimensional tables are the only form of structuring data in the SNMP architecture [STA99].

overhead unnecessarily large. It also increases the end-host latency, because more data take more time to transmit [MIT94].

OID naming scheme: This relates to the information model of SNMP-based management, i.e. the naming conventions for MIB variables. The Object Identifiers[4] (OID) transferred in SNMP messages exhibit a high degree of redundancy. For instance, all objects stored in MIB-II[5] are prefixed with 1.3.6.1.2.1. If this prefix could be omitted, a significant proportion of the space dedicated to the OID name would be saved. Furthermore, the prefixes of the table object OIDs are all identical up to the column number. In this case, more than 90% of the OID name is redundant [SPR99]. All these observations indicate a highly inefficient OID naming scheme.

No compression of management data: The first two versions of SNMP (v1 & v2c) did not allow the transparent compression of management data in transit. This unnecessarily increases network overhead and also network latency due to transferring larger volumes of data. As of SNMPv3, it is possible to compress management data by adding encryption envelopes to SNMP messages [SPR99]. Although this feature was initially intended for encrypting data, it also allows for data compression. When large chunks of data are compressed, the overall latency is also reduced, as the compression time is typically negligible with respect to the time saved to transmit the uncompressed data.

II) Unreliable transport protocol

Another problem of SNMP-based management is the transport protocol used for transferring SNMP messages, UDP. UDP operates in a connectionless fashion, which saves the three-way-handshake overhead of TCP [TAN96], and is ideal for exchanging short messages. However, due to the lack of acknowledgements, it is not suitable for communicating critical notifications to the manager. By using an unreliable transport protocol, the management system runs the risk of losing important notifications for trivial reasons such as buffer overflows in IP routers [MAR99b].

III) Security

SNMPv1 and v2c adopt a *weak security scheme*. Passwords for configuring routers, hubs and servers are passed across the network as clear text, in unprotected packets. Identification, which is based on *community strings*, is so simplistic that cannot be considered as secure. The main advancement brought by SNMPv3 is security. SNMPv3 supports identification,

---

[4] An OID (Object IDentifier) uniquely identifies a MIB object.

[5] MIB-II [McC91] is the standard MIB in the IP world, supported by virtually all SNMP-compliant network devices and systems.

authentication, encryption, integrity, access control, etc [STA98]. However, the support of SNMPv3 by major vendors takes place at a slow pace.

IV) Information model: low level of semantics

As far as semantic richness is concerned, the main shortcomings in SNMP are the absence of high-level MIBs, the limited set of SNMP protocol primitives, and the data-oriented nature of the SNMP information model. Due to these limitations, developing high-level management applications is a difficult task, which partly explains why NSM applications are often limited to little more than monitoring in the IP world [MAR00]. In particular, due to the way the SNMP market evolved over time, SNMP MIBs offer only low-level Application Programming Interfaces (API), often called *instrumentation MIB*s. SNMP frameworks provide no support for management applications to dynamically define external data models as part of the MIBs. Although it is possible for applications to retrieve raw MIB data and compute the appropriate data model at the platform host, this is highly inefficient [GOL95]. In addition, SNMP protocols support limited protocol primitives vocabulary, which allows getting/setting atomic variable values and notifying about important events. This is extremely restrictive and is typical of a data-oriented information model, unlike object-oriented models, which are widely used in industry today. The absence of an object-oriented information model in SNMP is generally regarded as one of the main limitations of SNMP [GOL96]. When the Distributed Management Task Force (DMTF) endeavoured to define a new management architecture in the late 90s, it came as no surprise that its first delivery was a new object-oriented information model: the Common Information Model (CIM)[6] [BUM00].

The *scalability* problems of SNMP are mainly attributed to its centralised model and can be efficiently addressed through MA-based approaches. The latter enable the dynamic delegation of NSM functionality to managed elements, where MA objects may filter/correlate management data, adopt an event-driven (instead of polling-based) approach to notify managers about important events and apply data compression, thereby reducing network overhead. Chapter 7 describes MA-based monitoring applications that address these issues. In addition, as described in Section 4.4.1.2, the administrator is given the option to use *either TCP or UDP* for MA migrations, depending on the application reliability requirements. Our model also addresses the authentication, authorisation and encryption *security* aspects (see Section 4.4.1.3.2). Finally, *semantically rich* management operations can be built using low-level

[6] CIM is defined as a conceptual information model for describing management that is not bound to a particular implementation. This allows for the interchange of management information between management systems and applications. This can be either ‘agent to manager’ and ‘manager to manager’ communications which provides for Distributed System Management.

protocol primitives as a basis and dynamically deployed to managed systems. This issue is discussed in Chapter 7.

2.5. DISTRIBUTED OBJECTS-BASED MANAGEMENT

A new programming paradigm has emerged in the ’90s, Distributed Objects Technologies (DOT). DOTs define an object-oriented paradigm, where objects can interact even if they do not reside on the same system. Since OSI-SM is object-oriented and SNMP managed objects can be mapped onto objects, it took little time for NSM researchers to start working on the integration of DOTs with existing management architectures. Thus, DOTs enabled a paradigm shift from the protocol-based approaches of the early 90’s, exemplified by the SNMP and OSI-SM, to distributed object-based approaches in the mid to late 90’s. In this section, we briefly describe two representative DOTs, the Common Object Request Broker Architecture (CORBA) and Java Remote Method Invocation (RMI), discussing their relevance to NSM.

2.5.1. CORBA

CORBA [CORBA] is the product of a consortium of over eight hundred companies, known as the Object Management Group (OMG). The OMG has approached the problem of handling the interaction of distributed components by creating interface specifications, and *not* code. Distributed components of the system are able to describe their interfaces using the Interface Definition Language (IDL) and subsequently inter-operate through the underlying Object Request Broker (ORB). Namely, the ORB provides the communication backbone through which distributed components are able to interact. To perform requests or return replies, objects use a generic RPC-like request/response protocol, the General Inter-ORB Protocol (GIOP) or its TCP/IP mapping, the Internet Inter-ORB Protocol (IIOP). Distributed components communicating via an ORB do not need to be aware of the mechanisms used in that communication and are able to discover each other at run time. A number of CORBA *services* provide basic functions, e.g. the *Naming* service that allows clients to locate objects based on their names, the *Trading* service that enables objects location based on their properties and the *Event* service which allows asynchronous messaging between objects.

The applicability of distributed objects on NSM has been a subject of intense research in the past few years [MAZ96]. Along this line, Pavlou pointed out the suitability of DOTs in large scale distributed environments and proposed the use of CORBA in TMN open interoperable interfaces, replacing OSI-SM [PAV00]. However, a main direction of the research efforts has been on the seamless integration of legacy systems into emerging distributed object environments. In that context, Mazumdar proposed the use of a gateway, which achieves the

inter-operation of management applications in CORBA domain and agents in SNMP domain [MAZ96]. The main function of the CORBA/SNMP gateway is to dynamically convert method invocations on object references in CORBA domain to SNMP messages for MIB entries at remote agents. Likewise, a CORBA/CMIP gateway has been proposed in [CHA97], which hides the complexity of OSI management from developers.

2.5.2. Java RMI

Java RMI [RMI] of Sun Microsystems is an API defined for remote communication by the Java development team. It allows method invocations between objects residing in different address spaces in a seamless and location-transparent way. This facility requires that the remote objects implement a specified interface and that their hosting devices run a dedicated service (*rmiregistry*).

However, Java RMI technology provides a low-level communication infrastructure. The syntax and the content of the messages are not defined (or remain in a very poor form). Another major drawback of RMI is that unlike CORBA, it is *not* language-independent, but exclusively intended for distributed Java-to-Java applications communication [MOR97]. RMI uses the object *serialisation* (described in Section 3.5.2) facility of Java to *marshal* and *unmarshal* parameters. Methods invoked at remote objects are reachable through the RMI's transport protocol, the Java Remote Method Protocol (JRMP), which has its own lookup service. In the context of NSM, Java RMI has been used in the JMAPI and JMX initiatives of Sun Microsystems (see Section 2.7.2).

A third emerging DOT is Microsoft's proprietary solution, the Distributed Component Object Model (DCOM) [DCOM], which has not yet found wide acceptance in NSM applications development. This is due to its limited platform support (Microsoft platforms) and the fact that DCOM is still an immature technology, in comparison with CORBA.

2.5.3. Limitations

Despite the undeniable suitability of DOTs for building distributed management applications, a number of disadvantages have been identified in the literature. The first three are specific to CORBA and RMI, while the last generally applies to all existing DOTs:

- Even though a main aim of distribution is scalability, CORBA retains a reliance on centralised information stores for such things as the name service, the trading service and implementation repositories. Although that has the benefit of simplicity, the centralisation of data represents a potential performance bottleneck and a single point of failure [McK00].

- The communication protocol of CORBA has been criticised for its efficiency, as it gives rise to a high message overhead on certain operations [GOK98] and is slower than traditional Remote Procedure Calls (RPC) [LIP00].
- Distributed applications relying entirely on Java RMI have been criticised for low scalability due to the limitations of JRMP (low protocol speed, and out-of-control socket creation). However, the situation improves in the latest releases of RMI that support the IIOP transport protocol, which offers improved scalability, while allowing the inter-operation of RMI and CORBA-enabled applications [MOR97].
- In DOT-based approaches, the functionality of the distributed objects is static and cannot be altered, in a similar way to OSI-SM and SNMP support object facilities [BOH00b]. The required degree of flexibility and reconfigurability can be achieved through the mobile code paradigms described in Chapter 3, which allow the dynamic deployment of management services at runtime.

2.6. DIRECTORY ENABLED NETWORKS

The emerging Directory Enabled Networks (DEN) [STR99] framework signifies a shifting from DOT-based approaches back to protocol-based approaches (exemplified by the SNMP and OSI-SM architectures). Like OSI-SM, DEN uses OSI directory technology to store information about networks. Changes in network configuration are handled within directories, which store information hierarchically, with each entry (object) containing a set of attributes. DEN directories distribute management information across multiple machines through a process known as *directory replicatio*n. DEN directories are typically accessed via the Lightweight Directory Access Protocol (LDAP), which supports three types of protocol operations: query, update, and authentication. The purpose of DEN framework is to facilitate the management of large and complex configurations of heterogeneous systems and deal with dynamic networks with frequent changes in their configuration. Therefore, DEN tackles the dynamism and scalability challenges in network management.

2.7. JAVA-BASED MANAGEMENT

Another approach that emerged in the 90's has been based on Java [JAVA], which in the NSM world is considered as a *technology* rather than as a programming language. This section starts with an overview of Java, highlighting its generic advantages and deficiencies (especially those with relevance to NSM applications). This overview is necessary, as it justifies the selection of Java as the implementation platform for our MA-based management framework. A

state-of-the-art report on Java-enabled management technologies follows, along with an overview of research initiatives undertaken in Java-based NSM. For further information on research activities and commercial products related to Java-based management, the interested reader is referred to the links given on [JBM].

2.7.1. Overview of Java

Since its first release in 1995 by Sun Microsystems, Java has become one of the most popular development platforms within the Internet community, mainly due to its strong connection with Web-based applications, inherent portability, simplicity and support for Internet-oriented programming. Java is described as "*a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language*" [JAVA].

2.7.1.1. Generic Benefits & Deficiencies of Java

Architecture-neutral and portable: Java offers the advantages of architecture neutrality and portability to networking application developers, which led to the Sun's marketing phrase "write once, run anywhere". That allows developing applications that address the problem of increased heterogeneity characterising modern networks, in terms of operating systems and hardware architectures. To achieve platform-independence, the Java compiler does not generate "machine code", i.e. native hardware instructions, but *bytecode*: high-level, interpreted, portable code. The format of this binary code is architecture-neutral. Given that the run-time platform, i.e. the Java Virtual Machine (JVM), is available for a platform, any Java application can execute there.

Object-orientation: Object-oriented design is a very powerful programming paradigm as it facilitates the clean definition of interfaces and maximises software re-usability. Although there are several object-oriented languages, Java is the only one that *enforces* it, i.e. the programmer has no choice but to encapsulate all data in objects.

Multithreading: In most languages, writing programs that deal with many processes (*threads*) simultaneously can be a very difficult task. Built-in multithreading support is one of the most powerful features of Java, as it allows lightweight process concurrency. The Java Development Kit (JDK) library provides a rich collection of methods to start, stop, suspend, resume or check the status of a thread.

Strong networking support: Java is particularly suitable for network programming. It offers an extensive library of classes and routines, which include TCP and UDP socket communication, support for broadcast and unicast messaging, invocation of remote methods through RMI, etc.

Native methods: When developing an application, situations might arise that necessitate the integration of code written in other languages (*native code*) within the Java application. The use of native code might be required for one of the following reasons:

(i) Code re-usability: Most of existing software is not written in Java. In some occasions, it is time consuming to translate it to Java, so the direct integration of non-Java with Java front-end programs might be preferable.

(ii) Performance: Being an interpreted language, Java exhibits worse performance compared to other languages that involve compiled code. Hence, native code could be more suitable for time-sensitive applications, in order to maximise the execution speed.

(iii) Tasks that cannot be implemented in Java: In certain cases, the implementation of certain tasks in Java might not be feasible (e.g. when low-level, platform-dependent functions have to be used), making the need for co-operation with native code imperative.

The integration of Java programs with native code is achieved through the Java Native Interface (JNI) [JNI]. An important feature of JNI is that it allows 'bi-directional' communication between Java applications and native code, i.e. Java methods may invoke native functions and vice-versa.

Security: Java is intended for use in networked/distributed environments. Along this line, particular emphasis has been placed on security. A built-in security mechanism is provided with the core JDK API: the `java.lang.SecurityManager` class that developers may extend to customise the access rights of individual applications on system resources [AUS00]. In addition, the Java Security API [JSAPI] is a framework for developers to easily include security features in their applets[7] and applications. The Security API includes support for digital signatures, encryption and authentication.

Despite all its advantages, Java has been often criticised for performance-related issues. The fact that Java is an interpreted language makes its performance substantially poorer than compiled languages, such as C++. However, with the advent of Just-In-Time (JIT) compilers [JIT], Java programs may be executed much faster. JITs basically compile the bytecode into platform-dependent native code, which is subsequently executed (it is faster to read the bytecode, compile it and run the resulting executable, than it is to interpret it).

[7] Applets are Java programs that can be included in an HTML page and subsequently uploaded and executed by Java-enabled Web browsers JVMs.

2.7.2. Java Technologies for Network & Systems Management

The self-evident suitability of Java for developing management applications has been recognised since its early days, with the first commercial implementations of the SNMP stack in Java released in 1996 [AdventNet]. Since then, AdventNet, a company that specialises in Java-based management products, has gone much further providing a complete suite of Internet management tools. These include a visual builder tool used to build SNMP management Java applets and applications, an Agent Toolkit that automates the process of creating SNMP MIBs and instrumenting agents for these MIBs, etc.

The following section summarises the technologies recently emerged in the field of Java-based management. All these technologies represent initiatives undertaken by Sun Microsystems and comprise toolkits (JMAPI and JMDK) or standards (JMX) expressly oriented to building management frameworks.

JMAPI - JDMK - JMX

In 1996, shortly after the release of the JDK 1.1 that added support for RMI, Sun Microsystems made the Java Management API (JMAPI) [JMAPI] publicly available. This API is a set of tools and guidelines to build management applets supporting RMI. It supports the most common SNMP MIB, MIB-II [McC91], by mapping its managed objects onto Java objects. It also provides a rich graphics library to ease the development of sophisticated GUIs, allowing the visual representation of management information.

Soon after, Sun released the Java Dynamic Management Kit (JDMK) [JDMK], a component-oriented management toolkit written in Java. Like JMAPI, JDMK is publicly available. It is based on Java Beans (JB)[8] and comes with a library of core management services. It also contains adapters to enable communication via RMI, HTTP and SNMP. Unlike JMAPI that deals only with MIB-II, JDMK includes an SNMP-to-Java MIB compiler, which translates the managed objects defined in any SNMP MIB into JB components (called management beans, or MBeans). The toolkit supports push and pull from agents and offers a powerful framework for developing management applications.

In 1999, JMAPI was superseded by the Java Management eXtensions (JMX) [JMX]. JMX is a management framework intended for object-oriented Web-based management. It is far more comprehensive than JMAPI and builds on the experience acquired by Sun with the JDMK. The specification does not only focus on the agent part of the management system (as it is the case with JDMK) but also specifies the manager part. In other words, JDMK can be considered as an integral part of JMX. It should be noted though that being relatively new

[8] A JavaBean [JB] is a reusable component that can be visually manipulated by means of a builder tool.

technologies, JDMK/JMX need extensive evaluations and performance studies to test their performance and scalability.

2.7.3. Research approaches to Java-based management

In addition to commercial activities, the power of Java in building management applications has been also signified by the profound interest of the research community. Early approaches focused on developing centralised frameworks entirely built in Java [LUD97, PAR98]. Both these works introduced a multi-threaded Java Agent engine, and proposed the replacement of BER encoding by a heavyweight bytecode-based mechanism. In particular, a simple `get` request involves the creation, compilation and transfer of a Java class that encloses the requested OID string, with the receiving Agent loading the class and retrieving the OID string before returning the requested value using the same mechanism. This method also imposes heavy network traffic (unnecessary transfers of classes) and exhibits the drawback of delay imposed by the time-intensive processes of compilation and class loading. These problems were addressed by a paper describing preliminary work of the author, which introduced a lightweight socket-based communication mechanism [GAV99].

More recent approaches to Java-based management concentrate on encompassing the Sun's standards, described in the preceding sections, in distributed NSM frameworks. For instance, [KEL99] described a case study for dynamic management of Internet telephony servers based on JBs and JDMK. Lee described the design and implementation of a management platform based on the TMN and discussed how Java technologies can support a variety of management interfaces as service components in a distributed computing environment [LEE00]. Anerousis introduced *Marvel*, a sophisticated Java-based management framework that enables the development of scalable NSM services [ANE99]. Scalability in Marvel is achieved by supporting computed views of low-level management information that convey high-level network operational status statistics and distribute the view computation task to a hierarchy of processors (servers). Information stored in Marvel objects is accessible through a Web interface.

2.8. WEB-BASED MANAGEMENT

Java owns its popularity mainly to its strong connection with the Web through applets. Since the Web is now ubiquitous, several proposals (see references in [Web]) have been made to use the Web technology in NSM. Wellens and Auerbach introduced the concept of *embedded management applicatio*n, where an applet is stored in the managed device and loaded by the administrator into a Web browser; communication between the applet and its

origin agent later relies on HTTP instead of SNMP [WEL96]. Since the time of this proposal, new technologies, such as Java *servlets*[9] and RMI have appeared and been used in Web-based management, for instance to open persistent sockets between applets and servlets, etc. Recent approaches realised a step towards decentralisation through the transition from *pull* to *push* management [MAR99b, ADA00]. In the push model, management data transfers are always initiated by the agent, similarly to SNMP notifications, thereby reducing the network overhead, and moving part of the CPU burden from managers to agents [MAR99b].

In the standardisation arena, an industrial consortium led by Microsoft launched a new initiative in 1996: the Web-Based Enterprise Management (WBEM) [WBEM]. WBEM took a revolutionary approach by replacing all existing protocols and object models with new ones. The main motivation was the integration of the Desktop Management Interface (DMI), used to manage cheap desktops, with SNMP, used to manage network equipment and expensive workstations. In this context, a new object model, CIM, has been devised. Today, SNMP/CIM, DMI/CMI and CMIP/CIM gateways are under development. WBEM is backed by most vendors in the NSM industry and is likely to emerge as one of the main management architectures of the decade [MAR00].

2.9. THE NEED FOR DISTRIBUTED MANAGEMENT

Management world today is dominated by protocol-based approaches, exemplified by the SNMP and OSI-SM. Both the IETF and the OSI approaches are characterised by the inflexible manager-agent paradigm. Centralisation has a serious impact on management scalability since it imposes almost all the computational burden on the manager platform [BAL97, CHE98]. The operations available for accessing MIBs are very low-level. In SNMP, for instance, the manager can only get and set atomic values in a MIB. This fine grained CS interaction is often called *micro-management* [GOL91], and leads to the generation of intense traffic and processing bottlenecks.

The OSI-SM supports the delegation of monitoring activities to the NEs, reporting only QoS alarms or summarised reports to higher-level managers [OSI92, OSI93]. Nevertheless, such generic functionality needs to be first researched, standardised, implemented and eventually deployed to NEs; this process typically takes a long time. Furthermore, the same research-standardisation-implementation-deployment cycle needs to be repeated whenever any

[9] Servlets are server-side Java components; while applets provide a way of dynamically extending the functionality of client-side browsers, servlets allow the application developer to dynamically extend the functionality of network servers.

modification, e.g. for providing more sophisticated features that were not thought out in advance, is to be introduced [MOU98b, BOH00b].

The scalability problem of centralised architectures becomes more profound as the dimension of the managed network grows. The managers need to communicate with a larger number of devices, as well as store and process an ever-increasing amount of data. This leads to the need for high cost hardware dedicated to the manager platforms [MAR98] and poor performance. In addition, the network area around the manager stations is saturated due to the combination of messages sent by the management platforms with those sent by the devices. The worst shortcomings of the centralised approach show up during periods of heavy congestion, when management intervention is particularly important [PIC98, KIM98]. During these periods: (i) the manager increases its interactions with the devices and possibly downloads configuration changes, thereby increasing congestion, (ii) access to devices in the congested area becomes difficult and slow (sometimes even impossible), and (iii) congestion, as an abnormal status, is likely to trigger notifications to the manager, generating even more traffic. In order to answer the problems related to centralisation, NSM functionality must be *distributed*, i.e. complex diagnosing and information gathering activities must be moved from the managers to the managed devices.

The advantages of distributed management can be found in many research papers (e.g. [GOL91, BAL97, KAH97, CHE98, MAR99a]). In particular, management distribution:

- allows applications to efficiently exploit the increased availability of hardware resources of modern managed systems;
- improves the autonomy and survivability of NMSs. That is, when the communication with the managing process is lost, distributed management entities can continue to execute;
- reduces the need for intensive polling;
- reduces the computational load on manager platforms through pre-processing management data, filtering unimportant alarms and delivering only high-level information;
- leads to significant reduction of management traffic, as most management interactions are locally executed.

It should be emphasised though that *not all management applications should be necessarily decentralised*. In fact, centralisation is the appropriate model for applications that have little inherent need for distributed control. Such applications (a) do not require frequent polling or high frequency computation of MIB deltas, i.e. aggregation functions, (b) have high-bandwidth network connections between the manager and the managed devices, (c) exchange relatively

small amounts of data, and (d) do not need frequent, semantically rich conversations between manager stations and managed nodes [MEY95] (see Figure 2.4).

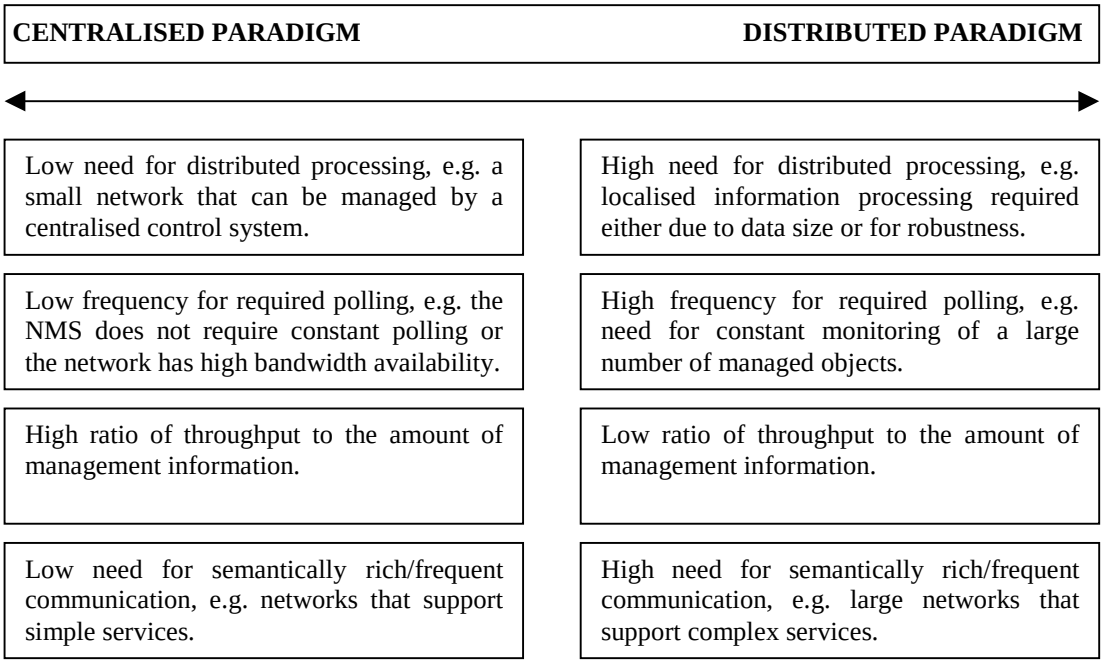

Figure 2.4. Management centralisation or distribution metrics (adapted from [MEY95])

2.10. DECENTRALISATION INITIATIVES WITHIN THE INTERNET COMMUNITY

The disadvantages of centralised management, as outlined in the preceding section, have first been admitted by the same organisations that introduced it. The following sections review the decentralisation initiatives undertaken by the IETF, with focus on IP networks management. The reader interested in extended surveys on the various approaches on management distribution may refer to [KAH97, MAR99a].

2.10.1. Management Distribution within the SNMP Frameworks

A primitive form of decentralisation (provided in SNMPv1) is the asynchronous notification mechanism. Namely, SNMP agents can send *traps* [CAS90] to the manager platform not as a result of a request, but when an important event occurs. Still, no management action can be performed locally as decisions are made centrally by the manager application.

The SNMPv2 introduces the concept of *proxy agent* [McC96], which realises a transition from centralised to hierarchical management models (see Figure 2.5). A proxy agent can be responsible for a set of devices; the manager sends requests to the proxy instead of interacting directly with these devices. Traditionally, SNMP has used proxy agents in a pass-through role, wherein a proxy passes the manager requests and agent responses through, in an essentially transparent mode. The fact that a proxy can be used as an intermediate manager for

hierarchical management is recognised by SNMPv2, however, the framework provides no means for managers to delegate tasks to intermediate managers or to communicate with them during the execution of these tasks [KAL97]. SNMPv2 has also attempted to promote management distribution through introducing the concepts of the *inform* PDU primitive (see Section 2.4.1) and the Manager-to-Manager (M2M) MIB [CAS93], which however proved unworkable in practice and have now become obsolete [MAR00].

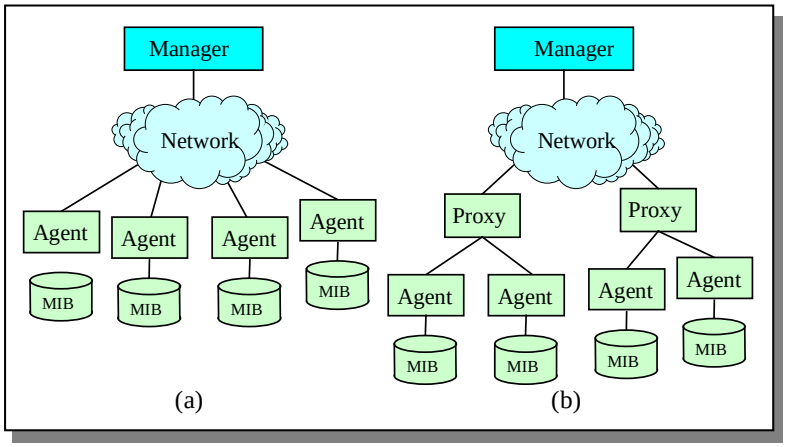

Figure 2.5. (a) Centralised management, (b) Hierarchical management

2.10.2. Remote Monitoring

The IETF has proposed another approach, known as Remote MONitoring (RMON) [WAD95], that introduces a higher degree of decentralisation. RMON assumes the existence of network monitoring systems called *monitors* or *probes*, which can either be standalone devices dedicated to link monitoring or embedded into network devices (Figure 2.6). By monitoring packet traffic and analysing the headers, probes provide information about links, connections among stations, traffic patterns, and status of network nodes. SNMP is used for communication between the manager and the agents running on probes. RMON allows the delegation of monitoring functions from the managers to the probes through the definition of suitable *filters*. A probe can detect failures, misbehaviours, and identify complex relevant events even when not in contact with the management station. In addition, the agent on the probe can perform semantic compression of data by pre-processing the information collected, before sending it to the management station.

However, RMON also exhibits several deficiencies:

- Typically, a stand-alone RMON compliant device (probe) is required to monitor the traffic activity of a single network segment, leading to considerable increase of cost when the management of multiple segments is required.

- The control operations of a RMON probe may be set/modified only at *configuration* time, i.e. *runtime* modifications are not supported [MOU98a].
- RMON is adequate for providing only *traffic-oriented* statistics since the status of the network is determined by direct inspection of the packets flowing in it, rather than inspection of the devices status, like in the mainstream (centralised) approaches that offer *device-oriented* statistics. Hence, RMON is not adequate when management operations should be applied in both system and network level.

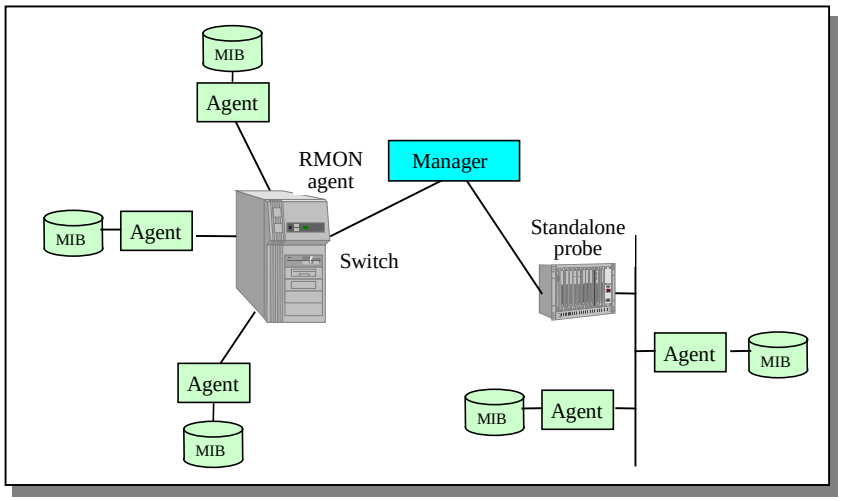

Figure 2.6. The RMON approach

2.10.3. Script MIB

Management distribution support to SNMPv3 framework has been added in 1999. In particular, the DISMAN (DIStributed MANagement) Working Group of the IETF was chartered to define an architecture where a main manager can delegate control to several distributed management stations. Among others, the DISMAN framework provides mechanisms for distributing scripts, which perform arbitrary management tasks to remote devices. This is achieved through the Script MIB [LEV99], which defines a standard MIB for the delegation and invocation of management functions (scripts), based on the Internet management framework (see Figure 2.7). According to the specification, the term script is very broad, referring to some type of executable code which can be executed by any device that implements the MIB.

In particular, Script MIB provides the following capabilities [SCH00]:

- Transfer of management scripts to a distributed manager;
- Initiating, suspending, resuming and terminating management scripts;
- Transfer of arguments for management scripts;

- Monitoring and control on executing management scripts;
- Transfer of the results produced by running management scripts.

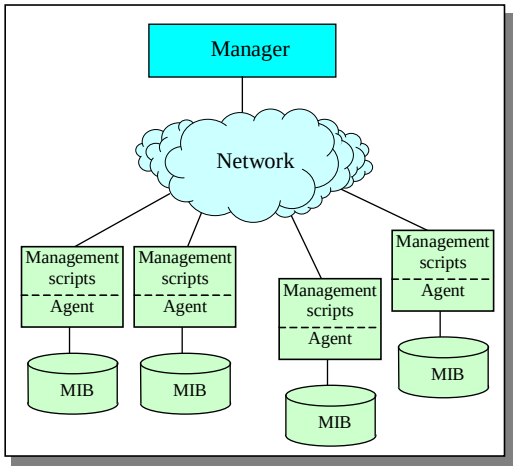

Figure 2.7. The Script MIB approach

The Script MIB is limited to the operations performed on the six tables that constitute the MIB. Before the administrator decides to delegate a script, he/she should first check the languages supported by the Script MIB implementation and select an appropriate script from a repository. Scripts are uploaded to the devices through client pull or server push models. The Script MIB enables scripts to be initiated, controlled and terminated through the SNMP management framework. For instance, the execution of a script starts with a single SNMP `set` request. The manager can also obtain intermediate or final results generated and maintained at the agent.

Although providing a powerful management distribution mechanism, the Script MIB also exhibits a number of limitations:

- The current specification makes it difficult to update existing scripts [McM99];
- Currently, the number of Script MIB implementations (such as the *Jasmin* project implementation [Jasmin]) is still limited, namely this technology has not been widely tested/evaluated yet;
- The Script MIB approach is specific to the Internet management frameworks, similarly to all standardisation approaches undertaken by the IETF or in the context of OSI-SM [BOH00c]. In contrast, MAs can provide a more generic and framework-independent mechanism for the delegation of management functionality.

It should be noted that parallel work to the script MIB has been also reported in the in the context of OSI-SM [VAS97].

2.11. RESEARCH APPROACHES ON MANAGEMENT DISTRIBUTION

The advantages of distributed/hierarchical management over centralised approaches have motivated several research works on that field, described in the following sections.

2.11.1. Management by Delegation

The full potential of large-scale distribution over managed devices was first demonstrated by Goldszmidt et al. through the *Management by Delegation* (MbD) framework [GOL91], which set a milestone in NSM research. MbD realises a distributed management architecture that enables the execution of management tasks at the end nodes by dynamically delegating management functions to stationary agents ("*elastic*" processes). The elastic processes allow new control functionality to be uploaded in the form of scripts, using a proprietary delegation protocol, the Remote Delegation Protocol (RDP). Scripts execution is usually initiated by other processes that require the corresponding functional services. This is quite similar in concept with the Script MIB approach (see Section 2.10.3); in fact, the latter follows the MbD paradigm, while the same applies to most recent standardisation and research NSM distribution initiatives.

Epitomising, the MbD paradigm represents a powerful management distribution approach, which has promoted for the first time network devices from 'dumb' data collectors to the rank of full-fledged managing entities. MbD can be considered a precursor of the ideas introduced in this thesis. The fundamental difference lies on the fact that in MA-based NSM, delegation of management functionality is achieved through the MA objects and not downloadable scripts; the execution of MAs is not necessarily restricted on a single device, as they can autonomously migrate from host to host.

2.11.2. Flexible Agents

The work presented in [MOU98a] builds upon the groundwork of the MbD paradigm, introducing an intelligent agent module, termed the "*flexible agent*". Unlike MbD agents, flexible agents do not operate at the managed device level; instead, they are responsible for a number of standard SNMP agents located within their management domain and exploit their ability to communicate and co-operate with their peers to correlate/filter collected data. The applicability of the flexible agents approach has been demonstrated in fault management scenarios [MOU99].

2.11.3. The Spreadsheet approach

A *spreadsheet* scripting environment for SNMP is proposed in [KAL97]. The spreadsheet scripting language allows a manager to prescribe computations that can be carried out by the agent and supports arithmetic, logical and relational operators. Each cell in the spreadsheet defines an expression that computes a value from other given data such as values in the MIB. Expressions can be inserted, updated or deleted according to the manager needs, through SNMP operations. Apart from computed attributes, the spreadsheet is also capable of generating event reports by evaluating predicates containing relational expressions.

2.11.4. Hierarchical Management

A hierarchical NSM system that uses the concept of the *SubManager* has been presented in [SIE96][10]. A SubManager is responsible for few agents; it collects primitive data from them, performs some calculations and produces more meaningful values that can be used by a superior manager. This method significantly reduces the volume of NSM traffic since only high-level information is sent to the manager. A similar approach is adopted in [KOO95] that proposes the use of *Area Agents*, each performing local management to a specific network section. Area Agents can be configured to collect and pre-process all data that was formerly dealt with by the manager station.

The *Mid-Level Manager* (MLM) [MLM] of SNMP research has been another step towards management distribution. An MLM is a dual-role entity, i.e. it acts both as an agent and a manager. When managers request information from the MLM, it plays the role of the agent, while acting as a manager for the agents located within its domain. When the MLM is placed across a WAN link, remotely from the enterprise manager, it obviates to a certain degree the necessity of performing “normal” SNMP polling. This reduces the polling traffic over the WAN link, thereby achieving significant cost savings. Furthermore, should the WAN link “go down”, the MLM continues to perform management tasks at the remote site despite its separation from the central manager. Finally, MLMs can provide a prompt first-line response to problems while leaving the manager in the position of final authority. After the first-line response, the manager can evaluate the situation more thoroughly and decide whether further action is required or not.

[10] It should be noted that the concept of hierarchical management has been first proposed in the TMN recommendation [CCITT92].

2.12. SYNTHESIS & EVALUATION OF EXISTING APPROACHES ON MANAGEMENT DISTRIBUTION

The approaches investigated in Sections 2.10 and 2.11 certainly offer useful mechanisms for realising management distribution. In general, they can be classified in:

(a) *Static* approaches (proxy agents, RMON, flexible agents and the hierarchical management models described in Section 2.11.4), which offer distribution but very limited flexibility;

(b) *Dynamic* approaches (MbD, Script MIB, spreadsheet approach), which offer distribution and increased degree of flexibility).

With the exception of RMON, static approaches achieve management distribution through building management hierarchies, where a mid-level delegation entity is responsible for managing a group of managed devices. That model is consistent with the hierarchical structure of modern large-scale enterprise networks, obviates the need for remote communication between the manager and managed elements and results in localisation of management traffic within the individual management domains.

There are two parts on building management hierarchies: (i) assigning roles ("proxy agents", "flexible agents", "area agents" "SubManager", "Mid-Level Manager") to the members of the hierarchy, and (ii) assigning members to specific physical locations where they will function under the supervision of higher-level members. This is feasible if the network is moderately small and/or not very dynamic, and if a single manager is the only user and specifier of the management policies. However, it is not in step with the dynamically evolving topological and traffic characteristics of large-scale enterprise networks [LIO01]. In addition, the distribution of aggregation and filtering computations in these approaches is manual and static. The manager of such a system is required to know the managed network well enough to build a hierarchy of distributed servers that will accommodate, to the best extend possible, all desirable computations that could be performed on NSM data. In general, the concept of management domain is not clearly defined in static approaches, nor are the criteria according to which the boundaries of such domains are determined. Also, the MLMs, SubManagers, etc, may suffer from overloads while executing their tasks [LOP00], either due to limited capacity or insufficient computing power of the hosting processors.

The problem of architecture inflexibility also applies to RMON where the filters configured by the manager are statically predefined and cannot be easily changed at runtime [KAL97, MOU98a].

The inflexible definition of static approaches can be addressed by MAs that dynamically migrate to remote management domains when certain conditions are satisfied and acting as mid-level management entities. In Chapter 6, we will introduce a hierarchical MA-based

management framework that addresses the issue of dynamic deployment and placement of agents, operating at an intermediary level between the manager and the managed devices.

Dynamic approaches offer increased flexibility in comparison with their static counterparts, as they allow on-the-fly customisation of delegated functionality. However, they exhibit a number of limitations. First, they rely on *scripts* as a means for distributing management intelligence to end nodes. Such scripts are typically coded using script languages, such as Perl, Javascript and Tcl/Tk. Script languages allow rapid coding and are ideal for implementing programs with limited capability. On the other hand, MAs are typically programmed in high-level programming languages (e.g. Java), which are suitable for developing more complex and demanding decentralised management tasks [PUL00a].

In addition, the Script MIB and the spreadsheet approach both rely on SNMP for script transfers. As a result, they suffer from disadvantages related to SNMP, that is:

a) not guaranteed delivery: SNMP relies on UDP for message transfers, which makes use of an unreliable packet transport mechanism;

b) SNMP packet length limitation: scripts are typically downloaded line-by-line [McM99] to minimise the risk of exceeding the packet length limit, resulting in additional traffic and increased unreliability and latency, especially when downloading large scripts.

On the other hand, the MbD approach uses a proprietary protocol for uploading scripts (RDP) and has been criticised because its delegation primitives have not been integrated with the SNMP framework [KAL97]. We believe that the wide spread of SNMP should certainly be taken into account, when designing distributed management architectures. The desired compliance with SNMP framework does not, however, compel the use of SNMP as a delegation protocol, neither necessarily entails suffering from the generic limitations of SNMP frameworks highlighted in Section 2.4.3. Our MA-based management infrastructure, presented in Chapter 4, is fully integrated with SNMP frameworks, however it relies on more efficient and reliable protocols than SNMP for MA migrations.

A common idea shared among all dynamic approaches is to upload code down to the managed device level, thereby performing semantic compression of management data and reducing network overhead. This idea is also behind MA-based management approaches. However, existing dynamic approaches focus is on the entities that enable delegation, i.e. the delegation agents. This represents a conceptual difference with the MA paradigm, which focuses on the mobile entities themselves and not on the entities that create, dispatch and receive the MAs [CHE00b]. That is, the MAs can be viewed as mobile *managing entities* that migrate to specific managed devices on demand, to perform arbitrarily complex management tasks, with an increased degree of *autonomy*.

Through exploiting their *mobility* feature, individual MAs can also perform, if required, decentralised NSM operations over a group of managed elements. In that sense, MAs can be regarded as a 'superset' of delegation agents, as they can provide all the functionality offered by the latter, having the additional benefit of mobility. Mobility is used only when necessary or whenever it is more efficient in terms of management cost; should a static delegation agent is sufficient for performing a management function, an MA can be sent to managed device and remain there until no longer needed. In Chapters 5 and 7, we identify application scenarios where mobility is necessary and suggest ways to minimise the impact of MA transfers on network resources.

Another limitation of existing dynamic approaches is that they imply a *device-level* view of managed resources, since delegated code executes on a single-device [LOP00]. This problem is addressed by static hierarchical models, where mid-level entities can correlate data collected from their management area, providing a *domain-level* views, which can be useful to identify problems related to large groups of devices or aid in capacity planning. We believe that the "best of the two worlds" can be achieved through MAs that visit a set of devices and correlate the data collected from them to provide domain-level views of managed elements. This approach simplifies the data correlation process, as it obviates the need for a complex co-operation mechanism between delegation agents, as proposed in [MOU98a]. In Chapter 7, we describe an application that addresses this issue.

Furthermore, dynamic approaches only consider a one-to-one relationship between two entities, where one (the *delegator*) delegates some task to the other (the "*delegee*"); this task is executed under the control of the delegator. No aspects of *cascaded* delegation are supported, limiting the degree of distribution granularity[11] as the delegator-delegee scheme comprises a two-level hierarchy. This argument is also applicable to DOT-based distributed management approaches. Quite felicitously, [MOU98b] characterises this approach as *remote* management rather than *distributed* management. Static hierarchical management approaches address this issue, as management functionality is distributed among a number of mid-level entities and is not concentrated on a single manager platform. Yet, static approaches imply a rigid configuration, which leaves a lot to be desired in terms of management flexibility.

A final point relates to the delegation *trigger mode* supported by both static and dynamic distributed management. In particular, existing approaches address this issue only for the user-driven case. That is, there is no support for delegation as a result of events. In Chapter 6, we investigate how MAs can be used to delegate NSM functionality on an event-driven basis.

---

[11] In the management context, granularity can be defined as the relative scale, detail, level of hierarchy or depth of penetration that characterises management distribution.

2.13. SUMMARY

Existing management distribution approaches offer a number of mechanisms for delegating NSM functionality to selected entities that perform management tasks as close to the managed systems as possible. As discussed in the preceding section, although the mechanisms have been defined, problems related to the delegation aspects have not been fully addressed yet. Mechanisms only represent a piece in the puzzle of distributed management. There are still open questions on what to delegate, where to delegate and under which circumstances and conditions to delegate.

The ideas behind MbD will however comprise the basis and enabling mechanism for distribution. These ideas will be driven much further by exploiting the power of the MA paradigm that promises to enhance the flexibility of the management process, whilst maintaining the required degree of distribution and scalability. As elucidated in the previous section, MAs can be used to accomplish a synthesis of the strengths identified in static hierarchical models and dynamic management approaches, serving both as a means of functionality delegation and dynamically deployed mid-level management entities. The following chapter discusses the issues related to the application of MA technology on NSM.