Chapter 3: Two Taxonomies of Distributed Network and Systems Management Paradigms

Jean-Philippe Martin-Flatin, Swiss Fed. Inst. of Technology, Lausanne (EPFL)
Simon Znaty, ENST Bretagne

1. INTRODUCTION

The 1990s have witnessed the advent and wide deployment of open Network and Systems Management (NSM). Departing from proprietary management hardware and software, the IP world\(^1\) and the telecommunications world\(^2\) both migrated to integrated management platforms. Unlike their proprietary predecessors, these platforms relied on open management architectures and open protocols. Throughout the 1990s, the Internet Engineering Task Force (IETF) defined three management architectures for the Internet, known as SNMPv1, SNMPv2, and SNMPv3, while the International Standards Organization (ISO) defined several versions of the OSI model [22, 23, 104]. The OSI management architecture encountered little success in data networks; but it became ubiquitous in the telecommunications world, after having been adopted by the ITU-T (International Telecommunication Union - Telecommunication Standardization Sector) as the basis for its Telecommunications Management Network (TMN) architecture [47, 84].

The purpose of integrated management was to be able to manage all equipment—network devices, workstations, Personal Computers (PCs), telecommunication systems, etc.—from the same management station, as opposed to having many proprietary management consoles side by side. In the IP world, this usually translated into using a single station to manage a whole network; in this case, open integrated management relied on a centralized paradigm (see Section 3.1). When a network was too large to be managed from a single management station, it was split into domains, and a hierarchy of management stations were in charge of managing these domains. NSM then relied on a weakly distributed paradigm (see Section 3.2). In telecommunications, the size of telephone networks was generally large, so centralized management was rarely an option; most network operators relied on a weakly distributed paradigm to manage their networks.

Then there were changes. With the deregulation of the telecommunications market, service providers and new network operators appeared alongside traditional network operators. Because of the increased competition, these companies looked for ways to differentiate their offer from the competition. Service providers began to introduce (and manage) new services at a much faster pace than what the SNMP and OSI management architectures had experienced in their early days. They also looked for technologies that enabled them to reduce, as much as possible, the time-to-market of new services. Independently of this, but at the same time, data networks continued to grow in the IP world. Gradually, they revealed serious scalability problems in the SNMP management architectures. Larger networks required more automated management. Moreover, with the advent of multimedia applications, data networks became multimedia networks. In addition to best-effort traffic, real-time traffic now had to be managed, with its stringent demands in terms of QoS (Quality of Service—e.g., bandwidth, losses, end-to-end delay, jitter, or burstiness). In the telecommunications and IP worlds alike, traditional management paradigms (centralized or weakly distributed) were not suited to these new requirements. These paradigms had been stretched to their limits: new ones were needed.

1. That is, organizations who use Internet protocols to operate their networks (e.g., intranets). The so-called TCP/IP stack not only includes the Internet Protocol (IP) and the Transmission Control Protocol (TCP), but also the User Datagram Protocol (UDP), the Internet Control Message Protocol (ICMP), the Simple Network Management Protocol (SNMP), etc.
2. That is, organizations who use Open Systems Interconnection (OSI) protocols to operate their networks (e.g., traditional, fixed telephone networks). The OSI stack of protocols includes the Common Management Information Protocol (CMIP), the Common Management Information Service (CMIS), etc.
In response to these new requirements, the mid-1990s saw the emergence of many ad hoc solutions, based on a new family of NSM paradigms: the strongly distributed paradigms. Traditional paradigms had focused almost exclusively on solving the problem of inter-vendor interoperability; this was the core contribution of the SNMP and OSI management architectures. But beyond interoperability, strongly distributed paradigms also offered scalability, flexibility, and robustness—three features put forward by Goldszmidt [38] when he proposed his own model, Management by Delegation (MbD). Large networks require scalable NSM. The accelerated provision of new services requires more flexibility. More complex applications and services introduce new points of failures, and require robust management techniques (e.g., dispatching management scripts inside agents).

The second reason strongly distributed technologies entered the NSM arena is that much progress had been made in software engineering since the OSI and SNMP management architectures were devised. With the Common Object Request Broker Architecture (CORBA [71]), the Object Management Group (OMG) had demonstrated that distributed objects could be a viable alternative to the traditional message-passing paradigm. Parallel to this, intelligent agents [103] began to spread out from Distributed Artificial Intelligence (DAI). New languages also appeared: Java [39] was widely adopted by the Web community, and the Knowledge Query and Manipulation Language (KQML [32]) was often used in MASs (Multi-Agent Systems, a sub-domain of DAI) to support cooperation between agents. All these new technologies suggested new ways of managing networks and systems. There was an application domain, NSM, which needed distributed technology; and software engineering was coming up with the right tools at the right time.

For these two reasons, the NSM community was recently overwhelmed by a plethora of strongly distributed technologies. Today, research is going in all directions. So is the market. The buzzword in NSM was intelligent agents in 1996, WBEM (Web-Based Enterprise Management) in 1997, Java in 1998, the eXtended Markup Language (XML) in 1999—it changes all the time. The stability that was brought by the SNMP and OSI management architectures seems to have vanished. In this new context, network and systems administrators find it increasingly difficult to make investments. The amortization of software and hardware investments necessarily runs for several years, so administrators need to have some visibility; they need to know in which direction NSM is heading to make sound investments. They cannot base their choice on the loudest marketing message of the moment.

Our goal in this chapter is to provide administrators with sound technical grounds to make such investments, to choose management paradigms and technologies, and to take an evolutionary rather than a revolutionary approach to NSM. To do so, we endeavored to classify all open technologies into a limited set of paradigms, and to propose criteria to assess and weigh the relative merits of different paradigms and technologies. One contribution of our work is to show that there is no win-win solution: different technologies are good at managing different networks and distributed systems. Evolving networks and evolving user requirements call for evolving NSM solutions. The aim of vendors is to sell revolutionary solutions, because they bring in more revenue in the short term. The purpose of administrators is to find an evolutionary path in the midst of these revolutionary approaches, to save money in the short and long run.

In this chapter, we introduce two ways of categorizing NSM paradigms; we call them the simple taxonomy and the enhanced taxonomy. In Section 2, we first define a terminology, as there is a great deal of inconsistency in the terminology used by different authors in NSM, software engineering, or DAI. In Section 3, we present our simple taxonomy, based on a single criterion: the organizational model. In this taxonomy, all paradigms are grouped into four broad types: centralized, weakly distributed hierarchical, strongly distributed hierarchical, and strongly distributed cooperative paradigms. We then expose the strengths and weaknesses of this simple taxonomy, and explain why we need to enhance it. In Section 4, we draw a parallel between the ways

---

1. This chapter does not cover proprietary solutions such as Microsoft’s Distributed Common Object Model (DCOM).
2. In organization theory, people generally refer to typologies rather than taxonomies when they mean classifications by type [42]. So did we, in early versions of this work [56, 57]. Since then, it was pointed out to us that usage has opted for taxonomies in computer science at large and network management in particular, so we now use this word. In the meaning of interest to us, both words are synonymous.
in which enterprises and networks are organized; we delineate a common trend, and identify the delegation granularity as a criterion for our enhanced taxonomy. We then introduce the concepts of microtask and macrotask. In Section 5, we study the three other criteria retained for our enhanced taxonomy: the semantic richness of the information model, the degree of automation of management, and the degree of specification of a task. This leads us to our enhanced taxonomy, depicted in Section 6. In Section 7, we give practical examples of how to use our taxonomies to select a management paradigm for a given network or a given system. Finally, we present related work in Section 8, and conclude with perspectives for the future in Section 9.

2. TERMINOLOGY

Before we can classify NSM technologies and paradigms into different types, we need to define a terminology. Unfortunately, we must acknowledge that the NSM community has not yet found a consensus on this issue. What is a manager: a person or a program? What is an NMS: a machine or a program? Things become even worse when we build interdisciplinary teams bringing together people with different backgrounds. What is an agent for people with an Internet, telecommunications, software-engineering, or DAI background? To address this confusion, we therefore propose the following terminology.

In network management, a network-management application is composed of managers running in network-management stations, and agents running in network devices (also known as managed devices). In practice, several independent applications may be needed to manage an entire network, e.g. one per manager; but when we refer to a network-management application, we consider them collectively as a single application managing a network. Likewise, in systems management, a systems-management application is composed of managers running in systems-management stations, and agents running in managed systems. These systems may be related (e.g., components of a distributed system such as a distributed database) or independent (e.g., hosts in an intranet). The network-management application and the systems-management application are often, but not always, integrated. To keep the text fluid and remain generic, we will refer to the management application when we actually mean the network-management application, the systems-management application, or both. Similarly, a management station can be a network-management station, a systems-management station, or both.

A management platform is the name given to a manager in the commercial world. It is characterized by a certain version of the manager (e.g., HP Openview x.y, Cabletron/Aprisma Spectrum x.y, or IBM/Tivoli Netview x.y.), a certain version of the operating system of the management station (e.g., Windows x.y or Linux x.y), and the hardware of the management station (e.g., a Compaq Pentium III PC or a Sun Microsystems SparcStation 20).

The meaning of the term network-management system is ambiguous. To some people, it is synonymous with network management station. To others, especially in business, it means network-management platform. To yet others, a single network-management system consists of all the network-management stations and the network-management application—that is, it masks the distribution of the management application. To avoid any ambiguity, we will refrain from using this term.

By extension, managers often refer to the management stations, and agents refer to the managed devices or systems. These are clearly misnomers, for they confuse the management application running on a machine with the machine itself; but these terms are seldom ambiguous once placed in context. When we use the term managers, there is also a potential confusion between programs and people. To avoid this, the people in charge of managing networks and systems will be called administrators (as usual in the IETF community).

The meaning that we retained for the word agent is standard for the NSM community. It is inherited from the manager-agent paradigm, one of the building blocks of the OSI and SNMP management architectures. But we experienced that it is misleading to people coming from the software engineering or DAI communities, who routinely use it in a different sense. To avoid any confusion, an agent in the software-engineering sense will be called a mobile agent, which is essentially a simple technique to dispatch and execute code on remote entities.
(see Section 3.3.1). Likewise, we will speak of an intelligent agent when we mean an agent in the DAI sense, that is, a paradigm enabling elaborate (and sometimes complex) forms of cooperation between remote entities that “think” independently of each other (see Section 3.4).

To cope with legacy systems, whose internal agent does not support the capabilities described in strongly distributed paradigms, we assume hereafter that such systems make use of management gateways if necessary (see Fig. 1). A management gateway is generally dedicated to a certain legacy system and external to it. It is located between the manager and the agent, and is transparent to the management application. It can, for instance, translate a CORBA request into SNMP protocol primitives, and vice versa. When a management gateway is used, the agent embedded in the legacy system is called a legacy agent. Throughout this chapter, when we refer to an agent, we may refer either to the pair {legacy agent, management gateway}, or to a single agent.

The management gateway is called a proxy agent by some authors [43, 53]. The problem with this designation is that the concept of proxy is confusing and ill-defined. This was acknowledged recently by the IETF: “The term ‘proxy’ has historically been used very loosely with multiple different meanings” [55, pp. 3-4]. The definitions of a proxy proposed in RFC (Request For Comment) 2273 [55] and RFC 2616 [31] are very specific, and different from our definition of a management gateway. Our choice of the term gateway was based on the definition given in RFC 2616 [31, p. 10]: “[A gateway is] a server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway”. Therefore, we will not use the term proxy agent. For the sake of completeness, we should also mention that a management gateway is sometimes called a delegated agent [108]. This term is ambiguous, as some authors use it for programs remotely transferred to an agent [38]; so we will also avoid it.

Decentralized management is to the enterprise world what distributed management is to computer science: a management paradigm based on the delegation of tasks to other entities. These entities are people in the enterprise world, and machines or programs in computer science. Delegation is used in both contexts as a generic word to describe the process of transferring power, authority, accountability, and responsibility [29, 70] for a specific task to another entity. In distributed NSM, delegation always goes down the management hierarchy. A manager at level (N) delegates a task (i.e., a management-application processing unit) to a subordinate at level (N+1); this is known as downward delegation. In the enterprise world, we can also find upward delegation; for example, an employee delegates his tasks to his manager when he is out due to illness [70]. Downward delegation and upward delegation are two kinds of vertical delegation, typical of hierarchical paradigms. In the enterprise world, organization charts generally follow a hierarchical paradigm. They are characterized by a multi-layer pyramid, comprising a top-level manager (at level 1), several mid-level
managers (at levels 2, 3...), and operatives at the lowest level [29]. In NSM, managers globally refer to the top-level and mid-level managers, whereas operatives are called agents. Contrary to vertical delegation, we have horizontal delegation, between two peers at the same level, which is typical of cooperative paradigms used in DAI. Distributed NSM may rely on a hierarchical paradigm, a cooperative paradigm, or a combination thereof. In fact, any paradigm outside the realm of centralized paradigms belongs to distributed NSM.

Delegation is normally a one-to-one relationship, between a manager and an agent in a hierarchical-management paradigm, or between two peers (be they managers or agents) in a cooperative-management paradigm. Arguably, delegation may also be considered, in some cases, as a one-to-many relationship, where a task is delegated to a group of entities collectively responsible for the completion of the task. One-to-many delegation is forbidden by most authors in enterprise management [4, 29, 70, 101]. It can be considered in DAI though. In distributed NSM, we propose to classify it as a form of cooperation, by coupling hierarchical and cooperative paradigms: a manager delegates a task to an agent, and this agent in turn cooperates with a group of agents to achieve this task. In the case of a many-to-many relationship, we are clearly in the realm of cooperation rather than delegation.

Some people confuse paradigms with technologies. A typical example is CORBA. In the literature, we find it referred to indistinctly as a paradigm, a technology, or even sometimes as a framework. In the tradition of software engineering, and notably object-oriented analysis and design, we consider that technologies implement paradigms [35]. At the analysis phase, network and systems administrators select a management paradigm (e.g., distributed objects). At the design phase, they select a management technology (e.g., Java or CORBA). At the implementation phase, they use that technology to program the management application.

Many people also confuse architectures with frameworks. In the IETF community, these two terms are often used interchangeably. For all SNMP protocols, the IETF even uses the same name for the underlying protocol and the management architecture—unlike the ISO, for instance, which distinguishes the OSI management architecture from the communication protocol (CMIP) and the protocol primitives (CMIS). This has confused many people, especially when solutions were proposed to replace the SNMP protocol while still preserving the SNMP management architecture. This problem has been acknowledged by the IETF, and a recent RFC [41] now recommends that an SNMP protocol or SNMP management framework (SNMP framework for short) be referred to explicitly. Unfortunately, frameworks and architectures have different meanings in the object-oriented community [28]. An architecture refers to the collection of models devised at the analysis and high-level design phases of an application; it is abstract in nature. A framework, conversely, refers to both an architecture and a set of template classes that implement this architecture. A framework provides hooks for site-specific extensions. It gives a common basis for different sites to build similar applications. It comes with code that implements a specific model. A framework is related to a specific implementation and is language specific; an architecture is not. In this chapter, we adopted the definitions of the object-oriented community. Consequently, we will refer to SNMP management architectures instead of the IETF-recommended SNMP management frameworks.

The meaning of enterprise management is also ambiguous. To most people, this is something you learn when studying business; this is the sense that we retained in this chapter. But a new fashion recently appeared in computer science, promoted by the Distributed Management Task Force (DMTF). Particularly in large, geographically dispersed corporations, the problem is not so much managing network devices and systems per se: it is managing networks, systems, applications, services, and policies that are deeply intertwined. Before the days of WBEM, this was commonly referred to as integrated management. Since then, the main marketing target has shifted from networks and systems to services, so the marketing terminology has changed. But in our view, there is nothing more to this new so-called enterprise management than is already in integrated management. Since the latter is unambiguous, and since we see no reason beyond sheer marketing to change a well-accepted terminology, we will not use enterprise management in its new, misleading sense.

Finally, network management in the IP world deals mostly with the management of network devices, because there are few end-to-end management concerns (the management of QoS is still in its infancy in the IP world). This is called element management in the telecommunications world, where end-to-end management is routine
(e.g., when creating telephone connections). Under the term network management, we will therefore group both network and element management in the OSI sense in this chapter.

3. SIMPLE TAXONOMY OF NETWORK AND SYSTEMS MANAGEMENT PARADIGMS

With these definitions in mind, let us now introduce our simple taxonomy of NSM paradigms. When we built it, we tried to meet seven objectives:

- provide an intuitive categorization of NSM paradigms;
- identify a small number of types;
- separate clearly centralized from distributed paradigms;
- highlight the differences between traditional and new paradigms;
- distinguish paradigms relying on vertical and horizontal delegation;
- enable administrators and NSM application designers to find out easily what paradigm is implemented by a given technology; and
- help classify quickly a new NSM technology.

To keep this taxonomy simple, and thereby meet the first two objectives, we decided to base it on a single criterion: the organizational model. This is the approach taken by most authors (see Section 8). To meet the third objective, we began with two types: centralized paradigms and distributed paradigms. Centralized paradigms concentrate all the management-application processing in a single node, the manager, and reduce all the agents to the role of “dumb” data collectors (see The Myth of The Dumb Agent [102]). Distributed paradigms, conversely, spread the management application across several machines.

To meet the fourth objective, we further divided the distributed paradigms type. By studying the different technologies that implement distributed management, we discovered that regardless of their idiosyncrasies, they could all be classified into two categories, according to the role played by the agents in the management application. We called them weakly and strongly distributed technologies; they implement weakly and strongly distributed paradigms.

Weakly distributed paradigms are characterized by the fact that the management-application processing is concentrated in only a few nodes. Typically, the network is split into different management domains, with one manager per domain. In this scenario, all the agents remain limited to the role of “dumb” data collectors. Another example is to keep a single manager but to make a few agents smarter than the others. In both cases, we have one or two orders of magnitude between the number of smart machines and the number of “dumb” machines. Only a small proportion of the machines are involved in the management-application processing.

Strongly distributed paradigms decentralize management processing down to every agent. Management tasks are no longer confined to managers: all agents and managers are involved. Many strongly distributed technologies have been suggested in the recent past. As we will explain in Section 3.3, we found it natural to group them into three sets of paradigms: mobile code, distributed objects, and intelligent agents. The first two implement vertical delegation; we call them the strongly distributed hierarchical paradigms. The third implements horizontal delegation; we call this family the strongly distributed cooperative paradigms. This distinction allows us to meet our fifth objective.

Thus our simple taxonomy consists of four types:

- centralized paradigms

---

1. Actually, agents are not always “dumb”. They can send unsolicited notifications to managers, and some can even process an atomic SNMP set of several MIB (Management Information Base) variables, which is not a trivial task. But the bulk of their activity is to collect usage data and be polled by managers, whereas the manager does all the real management work.
Emerging Trends and Challenges in Network Management

- weakly distributed hierarchical paradigms
- strongly distributed hierarchical paradigms
- strongly distributed cooperative paradigms

We refer to the first two as the traditional paradigms, and to the last two as the new paradigms. The strong distribution of the management application is a characteristic of new paradigms. The fourth type is also called cooperative paradigms for short, because the cooperative paradigms that we consider in NSM are always strongly distributed.

Our first five objectives have now been met. Let us delve into the details of these management paradigms and review the main technologies that implement them. We assume that the reader is familiar with the traditional management paradigms and protocols, that is, the different variants of SNMP, OSI/TMN, and Remote MONitoring (RMON). New paradigms and new protocols will be presented in more detail. At the end of this section, we will summarize our simple taxonomy in a synthetic diagram that will allow us to meet our sixth and seventh objectives.

3.1. Centralized Paradigms

Centralized paradigms are characterized by a single manager concentrating all the management-application processing, and a collection of agents limited to the role of “dumb” data collectors. The two typical examples are SNMP-based management and HTTP-based management (based on the HyperText Transfer Protocol).

3.1.1. SNMP-based management

To date, in the IP world, most real-life networks and systems are managed with centralized platforms based on SNMPv1. The success of this management architecture has been phenomenal. Within a few years, the networking industry, which was entirely dominated by proprietary equipment and management during the 1980s, turned to open systems and open management. Undeniably, the management of IP networks has been one of the greatest successes of open systems.

Nonetheless, three independent evolutions soon exposed a major weakness in the SNMPv1 management architecture, and more generally in the centralized paradigm: scalability. First, the IP world has been expanding for years at a very fast pace. Once limited to Unix machines, the TCP/IP stack became available on most network devices in the early 1990s, and on most PCs in the mid-1990s. Today, it is virtually ubiquitous. Second, the size of networks has grown dramatically. The number of PCs installed worldwide increased by several orders of magnitude during the past decade. The proportion of networked machines is now close to 100%, whereas many PCs were standalone machines when the SNMPv1 specification was released in 1990. Third, the size of SNMP MIBs increased, too. In Local-Area Networks (LANs) or Wide-Area Networks (WANs), IP routers and hubs had just a few FDDI (Fiber Distributed Data Interface), Ethernet, and Token Ring ports to manage a few years ago. Now, IP switches, ATM (Asynchronous Transfer Mode) switches, and intelligent hubs have many more entities to manage (ports, cross connections, virtual LANs, etc.), and require much more data to be brought back to the manager.

So, the very success of the SNMPv1 architecture was the cause of its decline. It proved to be good at managing relatively small networks, but could not scale to large networks (e.g. geographically dispersed enterprises), and could not cope with ever more management data. A new paradigm was needed to address scalability. The telecommunications world had already shown how to solve this problem: by distributing the load across a hierarchy of managers (see next section). But strangely enough, the distribution of management was not a priority at the IETF until the late 1990s. Since SNMPv1, four management architectures have been released: SNMPv2p, SNMPv2u, SNMPv2c, and SNMPv3. The first three only support centralized management. SNMPv2p has been rendered obsolete by the IETF in 1996. SNMPv2u had little success and “saw no

---

1. This is not necessarily true in other fields; e.g., in DAI, some forms of cooperation can rely on a central entity.
2. These are presented by many authors [43, 54, 80, 86, 89, 91]. Perkins [76] wrote a good summary of the variants of SNMPv2.
significant commercial offering” [76, p. 14]; 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 was on security [90], not scalability. Major vendors only recently began supporting it\(^1\), and its use will remain marginal in production environments in the foreseeable future. The MIBs adding distribution support to SNMPv3 were issued only in 1999, 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.

### 3.1.2. HTTP-based management

Since the mid-1990s, with the Web becoming so ubiquitous, inexpensive, and easy to use, many people have argued that Web technologies were the way to go for NSM in the IP world [45]. New vendors such as AdventNet, Metrix, and Rapid Logic saw an opportunity to enter the lucrative market of management platforms. This led to a large family of approaches called Web-based management, which supports different management paradigms. In this section, we describe the approaches implementing a centralized paradigm. In Section 3.3, we will present those implementing strongly distributed hierarchical paradigms.

HTTP-based management consists in using HTTP [31] instead of one of the three SNMP protocols to transfer management data between agents and managers. For this to work, all agents must have an HTTP server embedded, and a Web browser must run on the management station.

The simplest form of HTTP-based management relies on simple Web pages written in HyperText Markup Language (HTML); it is thus called HTML-based management [60, 69]. The manager retrieves HTML pages from the agent and displays them without processing them any further. Agents may send two types of documents: static and dynamic HTML pages. Static pages do not change over time and are stored in the agent, e.g., in flash EPROMs; a typical example is a Web page for configuration management. Dynamic pages are generated dynamically by the agent in reply to a request received from the manager. They reflect the state of the agent at a certain time; a typical example is a Web page for performance management.

The second form consists in running an applet in the Web browser or a Java application on the manager side, and using HTTP to communicate between the manager and the agent [7, 59, 102]. Management data can be pushed by the agent or pulled by the manager. Within HTTP, the data can be encoded in XML, strings, etc.

Note that an alternative to this consists in running an applet in the Web browser or a Java application, and using SNMP to communicate between the manager and the agent. Bruins calls it Java-based SNMP [16]. Clearly, this case cannot be classified as HTTP-based management. This approach changes very little compared to standard SNMP-based management platforms: the technology used to build the Graphical User Interface (GUI) is different, but the same communication protocol is used underneath. The agent cannot tell whether it is communicating with a traditional SNMP-based or a Java-based management platform.

Compared with SNMP-based management, there is no change whatsoever in all these scenarios with respect to the management paradigm. We only change the communication protocol between the manager and the agent.

### 3.2. Weakly Distributed Hierarchical Paradigms

Weakly distributed hierarchical paradigms spread the management application across several machines. The telecommunications world has followed this management paradigm for years with TMN. In the IP world, we saw one failed attempt with SNMPv2p and one successful, but partial, attempt with RMON.

---

\(^1\) For instance, Cisco supports it as of IOS 12.1, officially released in March 2000 [25].
3.2.1. In the telecommunications world

Unlike the SNMP management architecture, which proved to be successful in many sectors of activity, the OSI management architecture [22, 23, 104] encountered very little success in data networks. But in 1992, the ITU-T adopted it as the basis for its TMN model [47, 84] and for the specification of some of the TMN interfaces that mandate the use of CMIP and CMIS. Since then, OSI management has flourished in the niche market of telecommunications networks, where it is used to manage both networks and systems.

TMN/OSI is based on a weakly distributed management paradigm that distributes management along the hierarchical tree of the managers, each in charge of a management domain. If the contact is lost between a mid-level manager and the top-level manager, independent corrective actions can be undertaken by the mid-level manager. If the contact is lost between a mid-level manager and an agent, the agent is left on its own.

One of the management services offered by CMIS is \texttt{M\_ACTION}. It allows for the delegation of very simple tasks from a manager to an agent. In practice, this service is rarely used. But it is conceptually rich: any agent can execute a static, pre-defined task when requested by the manager. This gives us a flavor of strongly distributed management.

3.2.2. In the IP world

To date, the IP world is still waiting for a viable solution for distributing management across a hierarchy of managers. The first attempt was made in April 1993, when a new management architecture now called SNMPv2p was issued [36]. It relied on a new protocol and three new MIBs. Distributed management was supposedly made possible by a new protocol primitive, \texttt{inform}, and the Manager-to-Manager (M2M) MIB [21]. We call it the \textit{SNMPv2p+M2M management architecture}. This architecture was primarily targeted at geographically dispersed enterprises. But the SNMPv2p security model (based on the concept of \textit{party}) and the M2M MIB were flawed and proved to be “unworkable in deployment” [64]. In 1996, SNMPv2p was superseded by SNMPv2c and SNMPv2u, both of which only support a centralized management paradigm.

Nonetheless, the IETF managed to successfully define a weakly distributed management paradigm by making some agents smarter than others. RMON probes were the first and simplest form of delegation added to the SNMPv1 architecture. They supported the RMON1 MIB issued in 1991 [99]. RMON2 was released in 1995 [100] and became widely supported by intelligent hubs and switching hubs. By gathering usage statistics in RMON-capable equipment, administrators can delegate simple network-management tasks to these specialized network devices, thereby relieving the manager of the burden of the corresponding processing and decreasing the amount of management data to move about. Tasks achieved by RMON are static, in the sense that only the gauges and traps hard-wired in the RMON MIB are available, together with all kinds of combinations thereof (via the \texttt{filter} mechanism). If contact is lost between the RMON-capable agent and the management station, statistics are still gathered, but no independent corrective action can be undertaken by the agent. RMON is well suited to manage fairly active LANs, and is widely used today in the IP world.

3.3. Strongly Distributed Hierarchical Paradigms

Weakly distributed hierarchical paradigms address the main shortcoming of centralized models, scalability, but they also exhibit a number of limitations. First, they lack robustness. If the contact is lost between the agent and the manager (e.g., due to a network link going down), the agent has no means to take corrective action in case of emergency. Second, they lack flexibility. Once a task has been defined in an agent (via RMON, or CMIP/CMIS with \texttt{M\_ACTION}), there is no way to modify it dynamically: it remains static. Third, they can be expensive in large networks. By concentrating most of the management-application processing in managers, they require powerful (hence expensive) management stations.

To address this, a new breed of technologies emerged, based on strongly distributed hierarchical paradigms. The full potential of large-scale distribution over all managers and agents was first demonstrated in NSM by Goldszmidt with his Management by Delegation (MbD) architecture [38], which set a milestone in this
research field. The novelty of his work stems on the simple, yet insightful, idea that with the constant increase in the processing power of every computer system and network device, NSM no longer ought to be limited to a small set of powerful management stations: all agents could get involved and become active in the management application. For the first time with MbD, network devices were suddenly promoted from “dumb” data collectors to the rank of full-fledged managing entities.

MbD triggered a lot of research in strongly distributed NSM. The impact of the novel concepts it brought to this community was taken advantage of by many promising technologies that emerged, at about the same time, in other research communities. Most of these technologies came from software engineering, especially from the object-oriented and the distributed-application communities. Let us now present the paradigms underlying these technologies. They are grouped in two broad types: mobile code and distributed objects.

3.3.1. Mobile code

Mobile-code paradigms encompass a vast collection of very different technologies, all sharing a single idea: to provide flexibility, one can dynamically transfer programs into agents and have these programs executed by the agent. The program transfer and the program execution can be triggered by the agent itself, or by an entity external to the agent such as a manager or another agent.

3.3.1.1. Remote evaluation, code on demand, and mobile agents

Fuggetta et al. [35] made a detailed review of mobile code, where they clearly define the boundaries between technologies, paradigms (what they call design paradigms), and applications. As far as mobile-code technologies are concerned, they define strong mobility as the ability of a Mobile-Code System (MCS) to allow an execution unit (e.g., a Unix process or a thread) to move both its code and its execution state to a different host. The execution is suspended, transferred to the destination host, and resumed there. Weak mobility, on the other hand, is the ability of an MCS to allow an execution unit on a host to dynamically bind code coming from another host. The code is mobile, but the execution state is not preserved automatically by the MCS. (It is still possible to program this preservation explicitly, of course.)

By analyzing all existing MCSs, Fuggetta et al. identified three different types of mobile code paradigms:

- **Remote Evaluation (REV):** When a client invokes a service on a server, it not only sends the name of the service and the input parameters, but also sends along the code. So the client owns the code needed to perform the service, while the server owns the resources. This is a form of push.
- **Code On Demand (COD):** A client, when it has to perform a given task, contacts a code server, downloads the code needed from that server, links it in dynamically (dynamic code binding), and executes it. Thus the client owns the resources and the server owns the code. This is a form of pull.
- **Mobile Agent:** It is an execution unit able to autonomously migrate to another host and resume execution seamlessly. So the client owns the code, while the servers own the resources and provide an environment to execute the code sent either by the client or another server.

A number of technologies can be used to implement these three paradigms. Some of them are just languages; others are complete systems that possibly include a virtual machine, a secure execution environment, etc. Agent Tcl, Ara, Emerald, Sumatra, Telescript, and Tycoon are examples of strong MCSs, whereas Aglets, Facile, Java (or rather, to be precise, a Web browser integrating a JVM—Java Virtual Machine—and supporting Java applets), M0, Mole, Obliq, and Tacoma are examples of weak MCSs (see references in [35]). Some technologies can be used to implement several paradigms [35].

---

1. These should be compared with the client-server paradigm, where the client invokes a service on a server, while the server owns both the code and the resources (that is, it provides the environment to execute the code).
3.3.1.2. Management by delegation and variants

Mobile-code paradigms were first used in NSM by Goldszmidt and Yemini, in 1991, when they devised Manager-Agent Delegation [105]. This management architecture was later enhanced and renamed Management by Delegation (MbD); it was fully specified in 1995 [38]. MbD is a mixture of the REV paradigm (to send delegated agents to elastic servers) and the client-server paradigm (to remotely control the scheduling and execution of delegated agents).

Burns and Quinn [19] were among the first to describe a prototype of a mobile agent used in NSM. Since then, MCSs have encountered a growing success in NSM [5, 10, 11, 15, 81]. Baldi and Picco [6] studied the network traffic generated by MCSs implementing REV, COD, and mobile-agent paradigms, and made a quantitative theoretical evaluation of the effectiveness and suitability of mobile-code paradigms in network management. The ISO integrated mobile-code concepts in its OSI management architecture by specifying a new management function: the Command Sequencer [49]. In 1999, the IETF DiStributed MANagement (DISMAN) Working Group defined a series of MIBs allowing managers to distribute tasks to agents with SNMPv3. The most relevant to us is the Script MIB [82], which allows a manager to delegate management tasks to an agent in the form of scripts. We call it the SNMPv3 + Script management architecture.

3.3.1.3. Active networks

One area where mobile code paradigms have recently encountered a large success is known as active networks. There are two approaches to active networks. The evolutionary approach, called the programmable switch or the active node, provides a mechanism for injecting programs into network nodes to dynamically program them [3, 106]. These programs may perform customized computations on the packets flowing through them, and possibly alter the payload of these packets (e.g., compress and decompress data at the edges of the network). Clearly, this breaks the principle that transport networks should opaquely carry user data. The revolutionary approach, also known as the capsule or smart packet, considers packets as miniature programs that are encapsulated in transmission frames and executed at each node along their path [97].

The concept of active networks was first proposed in 1995 by Tennenhouse and Wetherall [97]. It was first applied to network management by Yemini and da Silva in 1996 [106]. Research is now extremely active in this field [17, 51, 72, 77, 79, 83, 87]. In NSM, active networks are of great interest because they bring in flexibility and robustness. Network monitoring and event filtering [30, 96] are especially good candidates, as monitoring programs can easily be dispatched through the network. These programs are high-level filters that watch and instrument packet streams in real time. They maintain counters and report results back to the manager. Another example is active congestion control [30]. The main problems associated with active networks are security, performance, and interoperability [72]. A lot of work is going on to solve them, including coupling active software with active hardware in the area of field-programmable gate arrays [44].

3.3.2. Distributed objects

Parallel to mobile code, a second type of strongly distributed hierarchical paradigms has emerged, based on distributed object technologies. We describe the four main approaches in this section: the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), Web-Based Enterprise Management (WBEM), and the Open Distributed Management Architecture (ODMA).

3.3.2.1. CORBA

Faced with the issue of interoperability in the object-oriented world, the OMG standardized the Object Management Architecture, now commonly referred to by its main component: CORBA [85]. CORBA 2.0 [71] was released in 1995. Unlike its predecessors, this release proved very successful in the software-engineering community, particularly for large corporations with huge investments in legacy systems. Because OSI is object-oriented and SNMP managed objects can be mapped onto objects, it took little time for NSM researchers...
to begin working on the integration of CORBA with existing management architectures. Pavlou was among
the first when he proposed to use CORBA as the base technology for TMN [73].

The Joint Inter-Domain Management (JIDM) group, jointly sponsored by the Open Group and the Network
Management Forum (NMF)\textsuperscript{1}, was created to provide tools that enable management systems based on CMIP,
SNMP, and CORBA to work together. The SNMP/CMIP interoperability was previously addressed by the
ISO-Internet Management Coexistence (IMC) group of the NMF, which specified the translation between the
SNMP and CMIP/CMIS services, protocols, and information. Both CMIP/CORBA and SNMP/CORBA [61]
interworking were solved by JIDM, who addressed specification translation and interaction translation.

Algorithms were defined for the mapping between GDMO/ASN.1 (Guidelines for the Definition of Managed
Objects / Abstract Syntax Notation 1) and CORBA IDL (Interface Definition Language) [62], and between
SNMP SMI (Structure of Management Information) and CORBA IDL [63]. The JIDM mappings allow
CORBA programmers to write OSI or SNMP managers and agents, without any knowledge of GDMO, ASN.1,
or CMIP. Inversely, these mappings also allow GDMO, CMIS or SNMP programmers to access IDL-based
resources, services or applications, without knowing IDL.

CORBA has also been integrated with Web technologies. One example from the industry is IONA’s
OrbixWeb, a Java Object Request Broker (ORB) coded as an applet. Once loaded into a Web browser, it runs
as a CORBA server communicating via the Internet Inter-ORB Protocol (IIOP). Another example is
CorbaWeb [66], from academia.

CORBA has been well accepted in the telecommunications world, where it is becoming a \textit{de facto} standard, a
rarity in this industry traditionally based on \textit{de jure} standards. One of the achievements of the Telecommuni-
cations Information Networking Architecture (TINA) [8, 9] has been to add a Distributed Processing
Environment (DPE) to TMN, and CORBA proved to be a natural choice for the DPE [74, 75]. Several major
equipment vendors are now turning to CORBA to manage their telephone switches and networks.

3.3.2.2. Java RMI

Java RMI makes it possible to program a management application as a distributed object-oriented application.
Everything is an object, and all objects can interact, even if they are distant (that is, if they are running on
different machines). When Java RMI is combined with Object Serialization, which allows objects’ state to be
transferred from host to host, management-application designers have a powerful technology at their disposal
whereby they can mix mobile-code and distributed-objects paradigms.

Distributed Java objects can be mapped directly onto SNMP or CMIS/CMIP managed objects. In this case,
they are low-level. They can also be high-level; e.g., the set up of a Virtual Path (VP) across multiple ATM
switches can be defined by a single method call on a remote object living in the source switch. This high-level
object will take care of all subsequent method invocations on all switches along the path to the destination. We
will discuss this important concept in Section 5.1.

In 1996, shortly after the release of the Java Development Kit (JDK) 1.1 that added support for Java RMI, Sun
Microsystems made the Java Management Application Programming Interface (JMAPI [95]) publicly
available. This API is a set of tools and guidelines to build management applets supporting Java RMI. It
supports the most common SNMP MIB, MIB-II [65], by mapping its managed objects onto Java objects. It also
provides a rich graphics library including all the \textit{widgets} typically used in the GUIs of management platforms.

In 1999, JMAPI was superseded by the Java Management eXtensions (JMX [93]). JMX is a management
framework (in the object-oriented sense) destined for object-oriented Web-based management. It implements
a component-oriented management architecture that is typical of the current trend in distributed software
engineering. JMX is far more comprehensive than JMAPI and builds on the experience acquired by Sun
Microsystems with its own commercial offering, the Java Dynamic Management Kit (JDMK). Most of the

\textsuperscript{1} Since then, the NMF has become the TeleManagement Forum (TMF).
work so far has concentrated on the agent side [92]. SNMP and WBEM APIs have already been specified, and a TMN API is currently under definition. The agent and the manager can communicate via Java RMI, HTTP, or SNMP. As far as SNMP, JMX relies on a general-purpose SNMP-to-Java MIB compiler that translates the managed objects defined in any SNMP MIB into components called MBeans (short for Management Beans). It is not yet clear what technology should underpin the manager side, possibly Enterprise Java Beans (EJBs). The latest JMX specification leaves its definition for a future phase [92, p. 21].

Independent of JMX, the Federated Management Architecture (FMA [94]) was devised in 1999 by the industrial consortium in charge of developing Jiro, a Java technology for storage management. FMA is a generalization to integrated management of a management architecture initially created for the sole purpose of storage management. The initial concept of FederatedBeans was dropped in favor of standard JavaBeans and EJBs, thereby addressing the main problem with Jiro. With FMA, all agents must be Jini-enabled. FMA defines static management services (e.g., the event service, the log service, and the controller service), and dynamic services that extend Java RMI semantics to the management-application level. FMA also specifies management aspects related to security, transactions, persistence, etc. Manager-agent communication may rely on any protocol: SNMP, HTTP for WBEM, etc. Some work is underway to unify JMX and FMA into a single management architecture for Java-based management.

In short, Java-based management allows for a very powerful way of building a strongly distributed management application. But it mandates that all managers and agents support Java, or that all agents be accessed via a Java-capable management gateway, which is a strong requirement. FMA also relies on Jini, which is even more demanding.

3.3.3. WBEM

In the IP and telecommunications worlds, open standards have virtually wiped out proprietary NSM solutions, due to the large success encountered by the SNMP and OSI/TMN management architectures and protocols in these markets. But in the rest of the industry, proprietary management platforms are still the rule and open platforms are the exception. A few years ago, the DMTF issued the Desktop Management Interface (DMI [26]) specification and tried to promote open management for desktops. But this effort encountered little success [18, p. 2]. To date, most networked desktops are either unmanaged or managed with proprietary solutions.

The situation could soon change, however. Enterprises are now bearing the costs of two parallel management platforms: one to manage their IP-based network equipment, and another to manage their desktop PCs. These companies, who did not attach much importance to open management a few years ago, are now paying the high price for this lack of interoperability; and they are pushing the industry to integrate the management of all kinds of network and systems equipment: PCs, routers, printers, switches, etc.

To address this need, an industrial consortium led by Microsoft launched a new initiative in 1996: Web-Based Enterprise Management. This management architecture took the radical approach of (i) defining a new information model, a new communication model, and a new communication protocol; (ii) making IP-based NSM fully object-oriented; and (iii) relying on gateways to integrate WBEM with existing protocols and object models (SNMP, CMIP, and DMI). In the initial proposal of the consortium [12], WBEM included the HMMS information model (HyperMedia Management Schema), the HMMP protocol (HyperMedia Management Protocol), and HMOM (HyperMedia Object Manager), a new environment to manage elements as objects.

The integrated-management market is traditionally less keen on marketing hype than the PC market, and this reinvent-the-wheel approach received a very cold welcome from both vendors and customers. As a result, WBEM underwent a drastic upheaval [98]. The DMTF took over the responsibility for this initiative, which no longer appeared to be mainly promoted by Microsoft. The core of the work went into the specification of a new, object-oriented information model: the Common Information Model (CIM [18, 27]). Its main strength is that

1. Jini is Sun Microsystems’ universal plug-and-play solution based on Java.
it is object-oriented, unlike SNMP MIBs. Its main drawback is that it was designed as a set of schemata strongly inspired by the database world. By not conforming to the OMG’s modeling terminology and by mixing database and object-oriented terminologies, it has caused some misunderstanding in the NSM community.

HMOM was replaced with the Common Information Model Object Manager (CIMOM). As for HMMP, the new protocol, it did not survive a reality check and was swiftly dropped. The DMTF instead opted to use two standard Web technologies: HTTP and XML [13]. CIM schemas are currently defined by a dozen Working Groups at the DMTF in all areas covered by NSM plus several others. Some work is underway to integrate CIM with CORBA and Java-based management. SNMP/CIM, DMI/CIM, and CMIP/CIM gateways are also 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.

3.3.3.1. ODMA

The purpose of ODMA [46] is to extend the OSI management architecture (thus also the TMN architecture) with the Reference Model of the ISO Open Distributed Processing (RM-ODP) architecture, which provides for the specification of large-scale, heterogeneous distributed systems. This joint effort of the ISO and the ITU-T has led to a specialized reference model for the management of distributed resources, systems, and applications. It is based on an object-oriented distributed management architecture composed of computational objects. These objects offer several interfaces, some of which for the purpose of management.

In ODMA, there are no longer managers and agents with fixed roles, like in the OSI management architecture. Instead, computational objects may offer some interfaces to manage other computational objects (manager role), and other interfaces to be managed (agent role). Moreover, by adopting the computational viewpoint of ODP, ODMA also renders the location of computational objects transparent to the management application. As far as the management application is concerned, computational objects may live anywhere, not necessarily inside a specific agent or manager. Consequently, agents may execute advanced management tasks, like managers. In short, the ISO and the ITU-T have gone from a weakly distributed management paradigm, with the OSI management architecture, to a strongly distributed management paradigm, with ODMA.

3.4. Strongly Distributed Cooperative Paradigms

Unlike centralized and hierarchical paradigms, cooperative paradigms are goal-oriented. What does this mean? For example, in REV-based mobile code technologies, agents receive programs from a manager and execute them without knowing what goal is pursued by the manager. Managers send agents the “how”, with a step-by-step modus operandi (coded in the program), and keep the “why” for themselves. Agents execute the program without knowing what it is about, they are “dumb”. Conversely, with intelligent agents, managers just send the “why”, and expect agents to know how to devise the “how”. In this sense, agents used in cooperative paradigms are “intelligent”. Obviously, there is a price to pay for this: cooperative technologies are much more complex to implement than centralized or hierarchical technologies. They also consume considerably more resources (processor, memory, and network bandwidth).

Cooperative technologies were only recently considered by the distributed NSM community. Until recently, most NSM authors simply ignored them [38, 43, 54, 67, 86, 91]. They originate from DAI, and more specifically from Multi-Agent Systems, where people are modeling complex systems with large groups of intelligent agents. This research field is fairly recent, so its terminology is still vague. Specifically, there is no consensus on the definition of an intelligent agent. Many authors have strong and different opinions about this (Franklin and Graesser [34] listed 11 definitions!), which does not help. In 1994, Wooldridge and Jennings took a new approach. Instead of imposing on others what an intelligent agent should or should not be, they defined a core of properties shared by all intelligent agents, and allowed any other property to be application specific. This approach has encountered a great deal of success, and contributed significantly to the dissemination of MASs outside the realm of DAI. For these authors, intelligent agents (or, to be precise, what they call weak agents) must exhibit four properties [103]:

14
• **autonomy**: An intelligent agent operates without direct human intervention, and has some kind of control over its actions and internal state.

• **social ability**: Intelligent agents cooperate with other intelligent agents (and possibly people) to achieve their goals, via some kind of agent-communication language.

• **reactivity**: An intelligent agent perceives its environment, and responds in a timely fashion to changes that occur in it.

• **proactiveness**: An intelligent agent is able to take the initiative to achieve its goals, as opposed to solely reacting to external events.

Proactiveness is a very discriminating property. While most intelligent-agent implementations are reactive, only a few of them qualify for proactiveness, particularly outside the AI community. We believe that this is the main difference between mobile agents from the software engineering community, and intelligent agents from the DAI community.

For Wooldridge and Jennings, optional properties of weak agents include mobility, veracity (intelligent agents do not knowingly communicate false information), and rationality (intelligent agents are not chaotic, they act so as to achieve their goals). In addition, they define **strong agents** as weak agents modeled with human-like characters, e.g. by using Rao and Georgeff’s Belief, Desire, Intention (BDI) model [78]. Strong agents are the type of intelligent agents generally used by the DAI community, whereas weak agents are the type often used by other research communities.

Two years later, Franklin and Graesser [34] compared the approaches taken by many authors and, as Wooldridge and Jennings, distinguished between mandatory and optional properties. For them, intelligent agents must be reactive, autonomous, goal-oriented (proactive, purposeful), and temporally continuous (an intelligent agent is a continuously running process). Optionally, they can also be communicative (that is, able to communicate, coordinate, and cooperate with other agents,), learn (they improve their skills as time goes by, storing information in knowledge bases), be mobile, and have a human-like character. In our view, the fact that intelligent agents should be continuously running processes is an important property. It also distinguishes intelligent agents from mobile agents.

Because we consider intelligent agents in the context of cooperative paradigms in distributed NSM, their ability to communicate, coordinate, and cooperate should be, in our view, a mandatory property. The ability to learn is often expected from intelligent agents in NSM; but like many authors, we do not consider this property to be mandatory. We therefore propose that in distributed NSM, intelligent agents should always be:

• goal-oriented (proactive)
• autonomous
• reactive
• cooperative (communicative, coordinating)
• temporally continuous

When intelligent agents are cooperative, they are exposed to heterogeneity problems, and therefore need standards for agent management, agent communication languages, etc. Two consortia are currently working on such standards: the Foundation for Intelligent Physical Agents (FIPA [33]) and the Agent Society [1]. Among all the agent communication languages that emerged in DAI [103], one, KQML [32], has encountered a certain success in the distributed NSM community.

More and more researchers are now trying to use intelligent agents to manage networks and systems [24, 68, 88, 107]. But we should remember that the limits between mobile agents (following a mobile-code paradigm) and intelligent agents (following a cooperative paradigm) are sometimes fuzzy. And when Knapik and Johnson [52] advocate the use of **OO agents** (object-oriented agents), to combine the advantages of both worlds, the classification becomes even trickier. In fact, OO agents implement two paradigms simultaneously, distributed objects and intelligent agents, and can even implement a third: mobile code.
3.5. Synthetic Diagram

Our simple taxonomy is summarized in the following synthetic diagram:

<table>
<thead>
<tr>
<th>Distributed Paradigms</th>
<th>Not Distributed</th>
<th>Hierarchical Paradigms</th>
<th>Cooperative Paradigms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SNMPv1, SNMPv2c,</strong></td>
<td><strong>SNMPv1 + RMON,</strong></td>
<td><strong>SNMPv2p + M2M,</strong></td>
<td><strong>intelligent agents</strong></td>
</tr>
<tr>
<td><strong>SNMPv2u,</strong></td>
<td><strong>SNMPv3,</strong></td>
<td><strong>OSI/TMN</strong></td>
<td></td>
</tr>
<tr>
<td><strong>WBEM,</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HTTP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SNMPv3 + Script,</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>mobile code,</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Java RMI,</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>distributed objects</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. Simple taxonomy of NSM paradigms

The chief contribution of this simple taxonomy is that it highlights some similarities between apparently very diverse approaches. Despite the fact that new technologies appear at a fast pace, network and systems administrators are no longer overwhelmed by the variety of approaches offered to them: they have a simple way to analyze them and group them, which reduces the scope of their investigation.

The main disadvantage of this simple taxonomy is that it is more of academic than industrial interest. By only considering the organizational model, it remains theoretical, and does not give many clues as to what paradigm or technology should be used in the context of a given enterprise. Administrators and designers of management applications need more pragmatic criteria. They have difficult software-engineering decisions to make at the analysis and design levels. They need to think twice before investing in expensive technologies such as CORBA, or before embarking for uncharted territories inhabited by roaming intelligent agents! Administrators would rather base their choices on sound technical grounds. Ideally, they would like case-based studies or “cookbook recipes”. The purpose of our enhanced taxonomy is precisely to fulfill this need. Although we cannot provide cookbook recipes, we will describe four selection criteria in the next sections, and highlight the software engineering trade-offs of the different approaches with respect to each criterion. In Section 7, we will also present several practical examples that can be used as simplified case-based studies.

4. A STROLL THROUGH ORGANIZATION THEORY

The topology of an enterprise computer network tends to be modeled after its organization chart. The main reason for this is that the people accountable for the smooth operation of these networks and systems belong to this chart, and it makes their life a lot easier if different managers (people, this time) have a hold on different computers and network devices. In addition, such a network topology is often justified in terms of budget: different departments pay for their own equipment. Sometimes, it also makes technical sense, for instance when different departments are located on different floors or in separate buildings. In short, NSM is not orthogonal to enterprise management.

In this section, we show that the first criterion of our enhanced taxonomy was derived by comparing the organizational models in NSM with organization structures considered in enterprise management. To do so, we studied how delegation works in enterprises, how it maps onto organization structures, and how the two fundamental paradigms that we identified in NSM (delegation and cooperation) map into the enterprise world.
4.1. Organization Structures in Enterprise Management

Mullins [70] distinguishes eight ways of dividing work in an enterprise:

- by function (one department per function: whether in production, R&D, marketing, finance, or sales, all staff share a common expertise within a department);
- by product (autonomous units, all functions are present in each unit);
- by location (geographically dispersed companies, subsidiaries abroad);
- by nature of the work to be performed (e.g., by security clearance level);
- by common time scales (e.g., shift work vs. office-hours work);
- by common processes (e.g., share a production facility in the manufacturing industry);
- by the staff employed (e.g., surgeons, doctors, and nurses in a hospital); and
- by type of customer or people to be served (e.g., home vs. export sales).

For Mullins, delegation can take place at two levels: enterprise or individual. At the enterprise level, it is depicted in the organization chart (at least it is supposed to be!) and relies on federal or functional decentralization. Federal decentralization is defined as “the establishment of autonomous units operating in their own market with self-control and with the main responsibility of contributing profit to the parent body” [70, p. 276]. As for functional decentralization, it is “based on individual processes or products” [70, p. 276]. At the individual level, delegation is “the process of entrusting authority and responsibility to others” [70, p. 276] for a specific task.

Weinshall and Raveh [101] identify three basic managerial structures: entrepreneurial, functional, and decentralized. The entrepreneurial structure is typical of an organization recently created, fairly small, and growing fast. It must be managed in an informal and centralized fashion in order to survive. Everything is centered on one person, the entrepreneur who created the enterprise. When organizations grow beyond a certain size, they must go through a major transformation. An entire set of rules by which the work is managed and carried out needs to be formalized, “in order to cope with the growing quantities of product and services, their variety, and the complexity of the organization” [101, p. 55]. This is called the functional structure. The chief executive officer directly controls the various functional heads, such as the production manager, the marketing manager, the sales manager, etc. The formalized and centralized nature of the functional structure must, at some point, give place to the decentralized structure. As a result of expansion, the number of managers (people) grows far beyond the number that can efficiently report to a single person. At this stage, the organization must slow down its growth and introduce a new formal and decentralized structure, organized by product/service line or by geographical area.
These three structures are uniform, in the sense that “all subordinates of the chief executive are structured either in an entrepreneurial, or a functional, or a product line or area structure” [101, p. 188]. Beyond a certain size, this uniformity cannot be maintained: the decentralized structure needs to change into a multistructure, i.e. a federated managerial structure where “different building blocks may be combined into different kinds of structures” [101, p. 189]. The Japanese, according to Weinshall and Raveh, were the first to operate their large organizations in multistructures, in a type of organization known as zaibatsu. The multistructure is inherently flexible, in that it enables changes in the composition of the federated basic structures. This natural evolution as enterprises grow from the entrepreneurial structure to the multistructure is depicted in Fig. 3. The actual values of the time on the x axis and the logarithm of the size of the organization on the y axis depend on the sector of activity of the enterprise, and change dramatically from one type of industry to another.

To conclude with enterprise management, let us take a look at the evolutions and revolutions cycle depicted in Fig. 4, an evolution trend that Greiner identified back in 1972. In retrospect, it is amazing to see how this cycle, devised for enterprise management, suits NSM well. If the first three phases look similar to those identified by
Emerging Trends and Challenges in Network Management

Weinshall and Raveh, the last two, coordination and collaboration, are incredibly visionary and predicted three decades ago what intelligent agents are now striving to achieve in NSM! It is also interesting to notice that the last crisis is left as unknown. Would the combination of hierarchical and cooperative paradigms be the ultimate solution? Or was the idea of collaboration so new in the enterprise world that Greiner, lacking hard evidence, did not want to speculate on what could go wrong then?

4.2. Delegation Granularity

What do Mullins, Greiner, and Weinshall & Raveh tell us that could apply to NSM? First, all the management paradigms they consider are hierarchical, except for the last two phases described by Greiner, which are characterized by a mix of hierarchical and cooperative paradigms. Likewise, most distributed NSM paradigms are hierarchical today, and cooperative paradigms have only just begun to appear in hybrid structures similar to Weinshall and Raveh’s multistructures. Second, delegation schemes should evolve as enterprises grow in size, otherwise they become inefficient. Similarly, distributed NSM should rely on different distributed NSM paradigms as networks grow in size and complexity. In this respect, the recent explosion of new distributed paradigms seems justified, because networks have grown by at least an order of magnitude in terms of size and complexity since the SNMPv1 and OSI management architectures were devised.

Third, if there are many ways of dividing up enterprise organization structures, depending on the granularity of the analysis, most authors agree with Weinshall and Raveh that they all coalesce in three broad types: by function, by product or service, and by geographical area. We can see some similarities here between enterprise management and distributed NSM. The division of management domains by geographical area, for example, makes sense in both worlds. But there are clear discrepancies too, since the basic entities are people in one case, and machines or programs in the other. The division by function only makes sense in the enterprise world. For instance, it takes many years for a person to become an expert in accounting or electrical engineering, and an accountant cannot be turned into an engineer overnight; conversely, a computer can be equipped with new competencies in a matter of minutes or hours, by simply transferring a few programs. We will show next that Mullins’s federal decentralization and Weinshall and Raveh’s decentralized structure map onto our delegation by domain scheme in distributed NSM, while Mullins’s functional decentralization and Weinshall and Raveh’s functional structure map onto our delegation by task scheme.

The fourth and most important thing we can learn from enterprise management is this common evolution trend, of which Greiner and Weinshall & Raveh give two different, but compatible, versions. There is a natural evolution of companies from centralized structures to decentralized ones, and from lightly decentralized structures (organized by function) to more decentralized ones with independent units (organized by product/service or by geographical area), to even more decentralized structures based on federation or cooperation. In NSM terms, these four stages map easily onto the different types presented in our simple taxonomy. The evolution which occurred in enterprise management over the 20th century suggests that the same evolution may take place in NSM in the next decade or so. The time scale may be different, but the evolution trend toward more distributed and cooperative management is the same.

4.2.1. Delegation by domain vs. delegation by task

How do the eight types of delegation identified by Mullins translate into NSM terms? We saw that delegation by geographical domain applies equally well to both worlds, but what about the other types? In 1994, Boutaba [14] identified a number of criteria to define domains in NSM. Resources are grouped into domains when they share a common feature. This may be the organizational structure (same department, same team), the geographical location, access permissions (resources accessible to a user, a group of users, or everybody), the type of resource (same vendor, same management protocol), the functionality of the resource (printer, mail system), or the systems management functional area [48]. Some items in this list resemble Mullins’s types, although they were made in very different contexts. But both of these lists are far too detailed for our taxonomy. In NSM, we propose to group all possible delegation policies in just two types: delegation by domain and delegation by task.
Delegation by domain relies on static tasks. The manager at level (N) assumes that the manager at level (N+1) knows all of the management tasks to be completed within its domain (N=1 for the top-level manager, N=2,3,4,... for the mid-level managers). In today’s networks, delegation by domain typically translates into delegation by geographical domain, to manage geographically dispersed enterprises. For instance, let us suppose that the headquarters of a multinational organization are located in Sydney, Australia. This enterprise cannot afford to manage its large subsidiaries in the USA, Asia, or Europe over expensive and relatively slow transcontinental WAN links. Let us consider its European subsidiary, located in Geneva, Switzerland. The manager in Sydney delegates the entire management of the Swiss subsidiary to the manager located in Geneva, and expects it not to report that a local printer goes down, but to report that the number of errors per minute exceeds a given threshold on the Switzerland-Australia WAN link. The point here is that the Australian manager does not tell the Swiss manager what to report; instead, the Swiss manager is expected to make this decision by itself. In practice, this translates into a human being, the LAN administrator, hard-coding in the Swiss manager what to report back to Sydney and how to manage the rest of the LAN. There is no mechanism for the Australian manager to alter the way the Swiss manager manages its domain: it is a carte blanche type of delegation, whereby the Geneva-based manager has total control over its own LAN. Network management is not automated, and there is no way for the Australian network administrator to enforce a management policy over all of its subsidiaries. Clearly, these are serious limitations.

Delegation by task, conversely, offers a finer-grained vision at level (N) of the management processing occurring at level (N+1). As a result, the manager at level (N) can see the different tasks at level (N+1), as well as other tasks of its peers at level (N). Tasks need no longer be static and hard-coded in every manager: they can also be modified dynamically. This idea was first applied to NSM with Management by Delegation, as we saw in Section 3.3.1.2. Goldszmidt departed from the well-established notion of static tasks underlying the centralized paradigm, and introduced the notion of dynamic tasks, transferable from the manager to its subordinate agents. This paradigm was later generalized to transfer dynamic tasks from a manager at level (N) to a manager at level (N+1).

4.2.2. Microtasks vs. macrotasks

A manager at level (N) has several ways of driving a subordinate at level (N+1). With traditional approaches such as SNMPv1, the basic unit in the manager-agent dialog is the protocol primitive: the manager issues a series of get and set requests to the agent. The data manipulated are MIB variables, which are statically defined when the MIB is designed. With large MIBs or large networks, this leads to the micro-management syndrome [38], which entails significant network overhead and poor use of the resources of management stations, managed devices, and managed systems.

Recent approaches avoid this syndrome by splitting the management application into many different units, or tasks, and by distributing these tasks over a large number of managers and agents, while still letting the manager at level (N) be in control of what subordinates at level (N+1) do. The underlying mechanism of this distribution is independent of the tasks being delegated; it can rely on program transfer, message passing, Remote Procedure Calls (RPCs), etc. The focal point for the management application is the granularity of the delegation, that is, the way the work is divided. Clearly, there is a wide spectrum of task complexities, ranging from the mere addition of two MIB variables to the entire management of an ATM switch. We propose to distinguish only two levels in our enhanced taxonomy: microtasks and macrotasks.

A microtask (µ-task) simply performs preprocessing on static MIB variables, typically to compute statistics. It is the simplest way of managing site-specific, customized variables. There is no value in these data per se, which still need to be aggregated by the manager one level up. If contact with the manager is lost, statistics are still gathered, but there is no way for the subordinate to take corrective action on its own. In the case of a macrotask (M-task), the entire control over an entity is delegated. A macrotask can automatically reset a network device, or build an entire daily report, etc. If contact is lost with the manager one level up, corrective actions can be automatically undertaken.
5. OTHER CRITERIA FOR OUR ENHANCED TAXONOMY

In previous work [56], we studied the features that designers of management applications want from strongly distributed management paradigms. We identified a number of criteria and showed that in addition to interoperability and scalability, which are already addressed by weakly distributed management paradigms, the two most critical criteria for designers are (i) the semantic richness of the information model, and (ii) the degree of automation of management allowed by a paradigm.

5.1. Semantic Richness of the Information Model

The semantic richness of the information model of a management application is an indication of the expressive power of the abstractions used in this model. It measures the facility for designers of management applications to specify a task to be executed by a manager or an agent. The higher the level of abstraction used to model a management application, the higher the semantic richness of the information model, and the easier it is for someone to build and design a management application.

It is well known in cognitive sciences that computers can easily be programmed to deal with low-level abstractions, but cannot easily manipulate higher level concepts; people, conversely, find it easier to think at a high level of abstraction, but are easily overwhelmed by too many low-level concepts. This is also true for NSM administrators, particularly when they design large or complex management applications. Unfortunately, management architectures have traditionally offered fairly poor APIs, thereby constraining designers to model management applications with low-level abstractions.

In this section, we show that this limitation has been addressed recently by some of the new management paradigms. Today, designers of management applications have the choice among three types of abstractions to build an information model:

- managed objects, offering low-level abstractions;
- computational objects, offering high-level abstractions; and
- goals, offering very high-level abstractions.

Let us review these three types of abstractions. We will introduce and compare the concepts of protocol API and programmatic API, and will identify a new criterion for our enhanced taxonomy: the degree of specification of a task.

5.1.1. Managed objects

Both the SNMP and the OSI management architectures offer a protocol API. In these architectures, there is a one-to-one mapping between the communication model and the information model, to use the ISO/ITU-T terminology [23]. The semantics offered to the designer of a management application—that is, the kind of entities and actions that can be defined in the information model and constitute the building blocks of a management application—are constrained by the communication protocol primitives used underneath. The protocol is not transparent to the application; this breaks a well-established rule in software engineering. For instance, with the APIs available in the different SNMP architectures, administrators have to think in terms of SNMP get and set when they write a management application.

We call this the managed-object approach, as both the IETF and the ISO use this term to describe a unit of the information model in their respective management architectures. All technologies based on centralized or weakly distributed hierarchical paradigms share this approach. When a management application is designed with managed objects, a protocol is automatically imposed; the managed objects must live in full-blown agents (in the case of TMN, these agents need to implement a large part of the OSI stack, including especially CMIP and CMIS); and the manager-agent style of communication is imposed. These are very strong constraints imposed on management-application designers.
We stress that the identity between the communication and information models has nothing to do with the protocols themselves. It is implicit in the SNMP and OSI management architectures. The limitations entailed by this approach reflect on the apparent limitations of some technologies. In the recent past, some Java-based management platforms used a simple port of the very basic SNMPv1 API developed at Carnegie Mellon University in the early 1990s (the `snmpget` and `snmpset` C programs). But nothing inherent in Web-based management prevents Java-based technologies from using richer APIs that deal with higher-level abstractions.

To address this, some people developed richer APIs to managed objects. One of the authors proposed an API [108] based on the Structured Query Language (SQL) to leverage the natural mapping between SNMP tables and tables found in relational databases. One advantage of making SQL queries from the management application, rather than SNMP `get`’s, is that table handling in SQL is less tricky than SNMP-table handling. (SNMPv1 is fairly poor at dealing with sparse tables.) But SQL-based APIs have not encountered great success so far, probably because relational databases are too slow and too demanding in terms of resources to become widely available in agents.

5.1.2. Computational objects

Protocol APIs for distributed systems are based on ideas that began to be criticized in the late 1970s and early 1980s, in particular by the software-engineering research community which was then promoting the concept of objects. Since the mid-1980s, this community has been advocating the use of programmatic APIs instead, which have been one of the selling points of the object-oriented paradigm for distributed systems. With such APIs, any object belonging to a distributed system is defined by the interface it offers to other objects. The distributed object model is independent of the communication protocol: it only defines a programmatic interface between the invoker and the operations (methods) supported by the invoked object. This programmatic API relies on a protocol at the engineering level, but this protocol is completely transparent to the management-application designer.

We call this the computational-object approach, with reference to the terminology used by the ISO for ODP and ODMA. In this approach, designers of management applications can use class libraries that offer high-level views of network devices and systems. Few constraints are imposed on the design: objects may be distributed anywhere, they need not live in the specific agents that implement specific protocol stacks. The only mandatory stack is the one that implements the distributed processing environment. No specific organizational model is imposed or assumed. The management application relies solely on object-to-object communication. The administrator may define site-specific classes and use them in conjunction with libraries of classes that implement standard MIBs.

The computational-object approach is the main strength of many recent management technologies, most notably distributed object technologies. In NSM, it accounts to a large extent for the recent success of Java in the IP world. Strangely enough, it is not responsible for the even greater success of CORBA in the telecommunications world: CORBA has mostly relied on a managed-object approach so far. Telephone switches are very complex to manage, considerably more than IP routers. The sole fact that, with CORBA, the millions of lines of code necessary to manage a switch could be written in parallel by many independent programmers, from different companies, in different languages, was in itself a blessing. And dealing only with well-known managed objects was a guarantee of interoperability. Even though CORBA objects could have conveyed higher levels of abstraction than CMIS/CMIP managed objects, the telecommunications industry has been happy so far to simply translate managed objects into CORBA objects. Will the success of Java-based management in the IP world suggest new ways of using CORBA in telecommunications in the future?

5.1.3. Goals

The third type of abstraction that may be used in information models is the goal. In Section 3.4, we saw that cooperative paradigms are goal-oriented. The management application is split into tasks, which are modeled
with very high-level abstractions and partially specified with goals. Once these goals have been sent by the
manager to the agent, it is up to the agent to work out how to achieve these goals.

This approach is fundamentally different from the one taken by weakly or strongly distributed hierarchical
paradigms, whereby the management application is broken down into fully specified tasks. Whether the
implementation of the task relies on calls to communication-protocol primitives or method calls on objects, the
agent is given by the manager a step-by-step
modus operandi
to achieve its task. With strongly distributed
cooperative paradigms, it is not.

Goals may be specified via a programmatic API, a protocol API, or both. They do not require an object-oriented
distributed system to be used underneath. But the coupling of agents and objects looks promising in NSM.
Knapik and Johnson [52] describe different styles of communication between intelligent agents:
object-oriented agents rely on remote method calls, whereas plain agents rely on communication languages
such as KQML [32]. The primitives (performatives) of KQML are considerably richer than those of SNMP and
CMIP, so goals are less limited by protocol APIs than managed objects.

To date, goals represent the highest level of abstraction available to management-application designers. They
rely on complex technologies known as intelligent agents (see Section 3.4), which often support some kind of
inference engine and pattern learning, and are generally not available on managed systems or network
equipment. So there is still a large market for simpler technologies that support computational objects, or even
simpler technologies that support only managed objects. But goals are a type of abstraction that makes it
possible to manage very complex networks, systems or services, for which simpler abstractions are not suited.
They are particularly well suited to support negotiation, load balancing, or resource usage optimization. We
will come back to this in Section 7.

5.1.4. Degree of specification of a task

Managed objects and computational objects rely on fully specified tasks, whereas goals rely on partially
specified tasks. In other words, the semantic richness of the information model and the degree of specification
of a task are tightly coupled. We decided to retain the latter as a criterion for our enhanced taxonomy, because
it shows two very different ways of specifying tasks in a management application. But we must keep in mind
that these two criteria are not independent.

<table>
<thead>
<tr>
<th>management-application unit (= information model abstraction)</th>
<th>managed object</th>
<th>computational object</th>
<th>goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstraction level</td>
<td>low</td>
<td>high</td>
<td>very high</td>
</tr>
<tr>
<td>where does it live?</td>
<td>MIB</td>
<td>object</td>
<td>intelligent agent</td>
</tr>
<tr>
<td>how do we access it from the management-application code?</td>
<td>management protocol primitives (SNMP, CMIP, HTTP)</td>
<td>method call</td>
<td>1) agent communication language primitive (KQML) 2) method call</td>
</tr>
<tr>
<td>degree of specification</td>
<td>full</td>
<td>full</td>
<td>partial</td>
</tr>
</tbody>
</table>

**Fig. 5.** Semantic richness of the information model (adapted from [58])

5.2. Degree of Automation of Management

Until a few years ago, the main drive behind the automation of management was to relieve network and systems
support staff from the burden of constantly gazing at GUIs and of solving problems manually as they occur.
As systems and networks grow by the year in size and complexity, administrators become increasingly eager
to automate their management: ad hoc manual management is not sufficient anymore. Yemini claimed a few
years ago that “management should pursue flexible decentralization of responsibilities to devices and maximal
automation of management functions through application software” [104, p. 28].
Today, the need for more automation of management is also determined by two factors: the deregulation of the telecommunications industry worldwide and the explosion of new services offered to end-users. First, there is fierce competition in the telecommunications market. Monopolies (or near monopolies) have given way to a plethora of competing network operators, service providers, service traders, content providers, etc. So any service provision today is likely to cross several networks, managed by different companies, with equipment from several suppliers [2]. Second, more and more services are offered: mobile telephony, electronic commerce, video on demand, videoconference, teleteaching, telemedicine, etc. Videoconferences, for example, used to be booked by fax on an ad hoc basis. End-users would contact support staff several days in advance; support staff would fax the single provider on the market (the local network operator); they would receive a reservation confirmation and an invoice within a day or so, sometimes less; and finally, they would inform the end-user that the booking had been made. This process was time consuming, very inefficient, and error prone. Today, end-users want to deal directly with a service trader via a user-friendly GUI, get the best possible deal for a videoconference scheduled at most a couple of hours in advance, and make an electronic transaction with a click of a mouse. Tomorrow, videoconferences will not be scheduled but will be provisioned immediately, like telephone today. Such demands are much more stringent than they used to be, and require considerably more work than mere faxes. As the number of such transactions grows within enterprises (from once a month to once an hour to once a second), and as the demands become more stringent, manual handling becomes less often an option. Service management must be automated to offer the online GUI that the end-user expects. Network management also has to be automated, e.g., to handle resource reservations and potential rerouting. Eventually, systems management must in turn be automated, e.g., to provide for automatic failover (hot stand-by) for video-on-demand servers.

As we show in Fig. 6, microtasks poorly automate distributed NSM; but macrotasks are very good at it, because they enable remote agents to take corrective actions independently from the manager. Intelligent agents are typically used in negotiation, e.g., to get the best deal for a cross-Atlantic videoconference from competing service providers. But they are also good at dealing with the dependencies between service, network, and systems management. To summarize the need for automation, the larger and the more complex the networks or the systems, the more automated the management application should be.

6. ENHANCED TAXONOMY OF NETWORK AND SYSTEMS MANAGEMENT PARADIGMS

Our enhanced taxonomy is now completed. It consists of the four criteria presented in the previous sections:

- delegation granularity;
- semantic richness of the information model;
- degree of automation of management; and
- degree of specification of a task.

These criteria are not independent. The semantic richness is closely linked to the degree of specification of a task, as is the delegation granularity to the degree of automation.

Note that in Fig. 6, the axes take discrete values, not continuous values. In other words, the relative placement of different paradigms in the same quadrant is meaningless.
By counting the quadrants that are populated in Fig. 6, we see that our enhanced taxonomy consists of nine types:

- no delegation with low-level semantics;
- no delegation with high-level semantics;
- delegation by domain with low-level semantics;
- delegation by domain with high-level semantics;
- delegation by microtask with low-level semantics;
- delegation by microtask with high-level semantics;
- delegation by macrotask with low-level semantics; and
- delegation by macrotask with very high-level semantics.

### 7. HINTS AND TIPS TO SELECT A MANAGEMENT PARADIGM OR TECHNOLOGY

In this section, we explain how designers of management applications can use our enhanced taxonomy to select a paradigm and possibly a technology. In particular, we give several examples of situations where changes are required in the management application due to new user needs or growth in the size of the enterprise.
7.1. What Does This Enhanced Taxonomy Tell Us?

The main message this enhanced taxonomy puts across is that there is no win-win solution. Depending on the size and complexity of the network, service, or system to manage, some paradigms are better suited than others. Certain paradigms, e.g. mobile code, encompass a very large number of technologies and are in turn subclassed into multiple paradigms. This yields a very wide variety of fine-grained design styles. Therefore, designers should not be constrained when they model management applications. Among all available paradigms, they should select one that allows them to model the problem at hand in the most natural way. The days of protocol APIs, when the focus was on network-management communications, are now over. Instead, management applications can rely on programmatic APIs, where the focus is on software development.

A second important point is that among all the technologies we reviewed herein, some are supported commercially and have been well tested and widely deployed, but others lack support and are still confined to the research community, and yet others have hardly gone beyond the proof of concept. To manage production networks and systems, it is critical to rely on well-tested and well-supported technologies; designers should therefore carefully choose the technologies that offer the best guarantees. If no such technology is available for the paradigm they selected, they should tell vendors that they are ready to pay for such technologies. This is not wishful thinking: more and more telecommunication- and computer-equipment vendors are proposing service- and network-management platforms based on CORBA. This move was driven by customers’ demands, especially from network operators, to deal with heterogeneity, interoperability, and distribution of their information systems. Similarly, the IETF resumed work on SNMP with the SNMPv3 and DISMAN Working Groups because vendors were pushed by customers to improve security and support multi-tier hierarchical management, as the multiple SNMPv2 architectures did not live up to expectations.

The third point that can be derived from our enhanced taxonomy is that several paradigms span over multiple quadrants, as Fig. 6 clearly shows. So, different technologies claiming to support a given paradigm may actually offer fairly different degrees of automation, or a different semantic richness of the information model. Let us take an example. A good marketing campaign convinces a network administrator that mobile code is the right paradigm to manage his network. So, before delving into the design of a new, powerful management application, he decides to investigate the market in order to purchase a mobile-code technology. All vendors claim to sell the best product on earth, so what technology should he choose? With our taxonomy, he can see at a glance that under the same term, mobile code, he can actually buy four very different types of technologies: some offering low-level semantics, others high-level semantics; some offering a high degree of automation of management, others a low degree. This taxonomy allows him to choose the technology offering the best value-for-money ratio according to the relative weight he gives to the four selection criteria.

The fourth and last point is that we do not list all existing technologies in our enhanced taxonomy. The reason for this choice is threefold. First, we want to keep this taxonomy readable. Second, technologies evolve so quickly, and this market is currently so active, that any such effort would be doomed to fail: such information would be obsolete as soon as it is published. Java, for instance, has blurred the boundaries between mobile code and distributed objects in the recent past. Third, we believe that the criteria we selected and presented are easy to understand, and that potential buyers of such technologies should be able to decide where to locate a given release of a given technology in Fig. 6, based on a short technical description of it.

7.2. Examples

Let us now study a series of examples that illustrate how to use our taxonomy in network management (these examples could be easily adapted to systems management). In all these examples, we will see that the relative weights given by the administrators to the different selection criteria give a clear indication of the best suited paradigm(s).

In a small enterprise, to manage a small LAN or a small distributed system comprising a dozen machines, there is no need for an expensive technology offering a high level of automation with computational objects, or even
goals; a less expensive solution based on managed objects and microtasks is sufficient. Even centralized management may be suited in this case.

Let us assume that this enterprise develops and opens small subsidiaries abroad. It is now a geographically dispersed enterprise, but it still has fairly simple needs (data network, no multimedia services). A weakly distributed hierarchical technology is well-suited. The required degree of automation is medium, and managed objects are sufficient to deal with simple needs. RMON is a good candidate. If the WAN links to the remote subsidiaries are very expensive, each subsidiary may be equipped with a local management platform.

As people in this enterprise begin using multimedia services on a more regular basis, there comes a point where manual handling is no longer an option: a higher degree of automation is required. Depending on the demands of the users, and the complexity of the services they use, inexpensive Java-based distributed object technologies may suffice. Otherwise, solutions based on intelligent agents may also be put in place, to cope with these new services on an ad hoc basis.

As this enterprise develops and grows larger, there comes a time when the number of entities to manage is so large that the management application is too complex. It becomes difficult to modify, and any change may cause a new problem, due to unforeseen side-effects. The semantic richness of the information model is too poor: managed objects have become inadequate. Even for simple day-to-day management tasks, it is now time to use computational objects instead.

At this point, only complex tasks for complex services are dealt with by intelligent agents. As new services are used, new intelligent agents are added on an ad hoc basis. As this enterprise is now relatively large, intelligent agents are no longer restricted to dealing with complex services. They may be used for pattern learning, for example: they may dynamically learn what are the peak and slack hours of a Virtual Private Network (VPN) over an ATM or Frame Relay network, and automatically readjust the bandwidth rented from the service provider in order to reduce the running costs for the enterprise.

Finally, if this enterprise is later bought by a large multinational with tens of thousands of managed systems and devices, the degree of automation of management then becomes critical. Day-to-day NSM should then entirely rely on distributed objects: managed objects should be banned. If day-to-day management was already based on distributed objects in the smaller enterprise, the integration within a larger management application would be considerably easier.

In this large multinational, the number of requests for high-level services, such as multimedia services, will soon increase, and the diversity of services used will also grow. This calls for fairly elaborate systems, based on a large number of intelligent agents, which no one has yet tested on a large scale.

8. RELATED WORK

Although the literature offers many examples of taxonomies of organizational structures in other research fields such as enterprise management [4, 29, 40, 70, 101], oddly enough, fairly little has been published with respect to organizational structures in NSM.

Before the outbreak of strongly distributed management technologies, most authors [37, 86, 89] simply presented the different SNMP management architectures adopted by the IETF and the OSI management architecture promoted by the ITU-T. Since then, several people [67, 81] have showed the advantages of one technology or architecture over the centralized approach. Others have studied a single family of paradigms [35]. But few authors, in fact, have already proposed full-scope taxonomies covering the entire range of NSM paradigms.

Those we found were all based on a single criterion, the organizational model, like our simple taxonomy. Hegering and Abeck [43, p. 121] quickly sketch a taxonomy of integrated management paradigms, comprising centralized, hierarchical, and cooperative paradigms. But the meaning they give to cooperative management is
very different from ours. According to them, “both approaches [OSI management and SNMPv2 architecture] support a cooperative management of peer systems” [43, p. 198]; this seems to indicate that they consider cooperation solely as a manager-to-manager communication. Leinwand and Fang Conroy [54] propose a taxonomy of network management paradigms, more detailed than the previous one, but still rather different from our simple taxonomy. They focus on whether management databases are located in mid-level managers or in the top-level manager only, and they ignore what we call strongly distributed hierarchical paradigms and strongly distributed cooperative paradigms.

Actually, we did find one multi-criteria taxonomy by Kahani and Beadle [50], which compares technologies that implement different distributed paradigms. But unlike our enhanced taxonomy, it does not attempt to cover the full scope of management paradigms. Instead, it only considers four distributed technologies, and compares them with the centralized approach. The authors selected seven criteria to perform their evaluation: architecture (what we call the organizational model), communication method, polling method, polling interval, autonomy, extensibility, and flexibility.

In summary, we believe that our simple taxonomy is the first to integrate the entire range of NSM paradigms proposed to date (full-scope taxonomy), and that our enhanced taxonomy is the first to also propose multiple criteria to compare and weigh the relative merits of all management paradigms and technologies (full-scope and multi-criteria taxonomy).

9. CONCLUSION

Our goal was to help designers of management applications select the right paradigm and technology to manage a given network or distributed system. To achieve this, we proposed two ways of classifying all major management technologies available to date.

In our simple taxonomy, all NSM technologies were classified according to their underlying organizational model. We grouped them into four different types of NSM paradigms: (i) centralized paradigms, (ii) weakly distributed hierarchical paradigms, (iii) strongly distributed hierarchical paradigms, and (iv) strongly distributed cooperative paradigms. Faced with dozens of commercial or prototype NSM technologies on the market today, with new ones appearing every month, designers of management applications run the risk of being overwhelmed by the abundance of choice. With this simple taxonomy, they now have a simple tool to find out quickly which management paradigm lays behind a given technology.

The purpose of our enhanced taxonomy was to go beyond the sole understanding of the management paradigm by providing criteria to actually select a paradigm first, and then a technology. To this end, we identified four criteria: (i) the granularity at which the delegation process takes place (by domain, by microtask, or by macrotask); (ii) the semantics of the information model (managed object, computational object, or goal); (iii) the degree of automation of management (high, medium, or low); and (iv) the degree of specification of a task (full or partial). This enhanced taxonomy complements the previous by being more practical. It gives some arguments for designers of management applications to select one paradigm rather than another, based on the issues they face during the analysis and design phases. The examples that we gave show that different environments call for different management solutions; they also give some hints to designers of management applications, who can, by analogy, apply them to their own environment.

An interesting perspective for the future would be to demonstrate the gains of coupling hierarchical and cooperative paradigms in NSM, by integrating technologies based on different paradigms. For instance, how could mobile agents, distributed objects, and intelligent agents be used together to manage multimedia networks efficiently? As we observed, the current information models of the SNMP and OSI management architectures are based on low-level semantics. Would it be possible to devise new models that integrate managed objects, computational objects, and goals without being overly complex? The current approaches that we reported in this chapter all rely on a single management paradigm, and a single level of semantics. New approaches that combine several could prove to be very promising.
ACKNOWLEDGMENTS

This research was partially funded by the Swiss National Science Foundation (FNRS) under the grant SPP-ICS 5003-45311. The authors wish to thank all the people who gave them feedback on drafts of this chapter, especially R. Glitho, D. Hutchison, G. Pavlou, G.P. Picco, and J. Schönwälder. We also thank H. Cogliati for proofreading the final version. An early version of this work was published in [58].

REFERENCES

91. W. Stallings. “SNMP, SNMPv2, SNMPv3, and RMON 1 and 2”.


IEEE

82. J. Schönwälder, J. Quittek, and C. Kappler. “Building Distributed Management Applications with the IETF Script MIB”.

Computer Networks and

81. J. Schönwälder. “Network management by delegation—From research prototypes towards standards”. Computer Networks and


River, NJ, USA, 1996.

89. W. Stallings. Java Management Extensions White Paper

93. Sun Microsystems. Network and Distributed Systems Management

84. J.K. Shrewsbury. “An Introduction to TMN”.

Journal on Selected Areas in Communications


85. M. Sloman (Ed.). The Simple Times


Network Operations and Management Symposium (NOMS’96), Kyoto, Japan, April 1996. 1:34–43. IEEE, Piscataway, NJ, USA,

1996.


60. M.C. Maston. “Using the World Wide Web and Java for Network Service Management”. In A. Lazar, R. Saracco, and R. Stadler

(Eds.), Integrated Network Management V. Proc. 5th IFIP/IEEE International Symposium on Integrated Network Management


61. S. Mazumdar. “Inter-Domain Management between CORBA and SNMP”. In Proc. 7th IFIP/IEEE International Workshop on

Distributed Systems: Operations & Management (DSOM’96), L’Aquila, Italy, October 1996.


78. A.S. Rao and M.P. Georgeff. “Modeling rational agents within a BDI-architecture”. In R. Fikes and E. Sandewall (Eds.),

Proc. 1996 IEEE

Network Operations and Management Symposium (NOMS’96), Kyoto, Japan, April 1996. 1:34–43. IEEE, Piscataway, NJ, USA,

1996.


River, NJ, USA, 1996.

81. J. Schönwälder. “Network management by delegation—From research prototypes towards standards”. Computer Networks and


82. J. Schönwälder, J. Quittek, and C. Kappler. “Building Distributed Management Applications with the IETF Script MIB”. IEEE


89. W. Stallings. SNMP, SNMPv2, and CMIP: the Practical Guide to Network Management Standards. Addison-Wesley, Reading,

MA, USA, 1993. (Obsoleted by [91], except for CMIP).


31
The network and service management taxonomy serves as a classification system for research on the management of computer networks and the services provided by computer networks. The taxonomy has been created and is being maintained by a joint effort of the Flamingo FP7 Project and the Committee of Network Operations and Management (CNOM) of the Communications Society (COMSOC) of the Institute of Electrical and Electronics Engineers (IEEE). 

Proof: Suppose that the entire distributed monitoring system consists of a finite set $N$ of monitoring managers whose size is $n$ and there is the set of all crashed domain managers, denoted by $SCDM$. The proof proceeds by induction on the number of all the crashed domain managers in $SCDM$, denoted by $|SCDM|$ ($|SCDM| < n$).