Managing in a Distributed World

Adrian Pell, Kave Eshghi
Jean-Jacques Moreau, Simon Towers
Networks and Communications Laboratory

HPL-94-121
December, 1994

Managing in a Distributed World

Adrian Pell, Kave Eshghi
Jean-Jacques Moreau, Simon Towers
Networks and Communications Laboratory
HPL-94-121
December, 1994

networked systems management, application management, distributed management, model description, print spooling, management protocols

The task of networked systems management has become increasingly complex in recent years. Reducing this complexity and permitting easy management are major challenges to the acceptance of networked systems and applications. This paper introduces a language for describing these systems and applications and gives an example of its use.

Managing in a distributed world

1 Introduction

The task of systems management has changed radically in the past 10 years. This has been caused in part by the explosive growth in computing power available in most organisations and also by the widespread distribution of these systems throughout organisations. As a consequence, it is no longer a simple matter for the MIS department to manage the available computing resources – even keeping track of where those resources are is quite taxing!

Allied to this growth in system power has been a major paradigm shift in the construction of large software systems, typified by the move towards client-server applications. Although the number of such applications in common use is, as yet, relatively small, the problems that they bring to the system manager are immense.

This paper describes a research project, Dolphin, which seeks to provide some responses to these challenges. In the next section, we describe in detail some of the problems facing system managers. We then introduce a language for describing the systems and applications that must be managed, and we give a practical example of its use. Finally, we describe our experience with this system, and outline some interesting research problems that still remain to be solved.

2 The management dilemma

One does not need to look far to observe recent trends in computing that are having far-reaching effects on the way many businesses function. The move away from a small number of large mainframe computers operated by the MIS department towards many physically smaller, yet often equally powerful, workstations is happening throughout most industries. With this decentralisation of computing have come a number of other moves – to distributed client-server applications on the one hand, and to mobile computing with its consequent change of user expectations on the other. These moves have commonly been portrayed as downsizing, partly because of the physical size reduction, but also, perhaps, because of hoped-for consequent reductions in the size and influence of the MIS department.

Looked at from a management standpoint, however, the picture is rather different. No longer is it possible to look in a single place to determine the health of a particular application – its vital signs may be spread across many machines, and these may be located over a wide area. No longer is it possible to reliably ensure the correct operation of all systems at all times – personal workstations and especially mobile workstations may come and go at a whim. Perhaps, above all, it is no longer possible to easily identify exactly *who* is managing particular systems and applications. To some extent, every person sitting at every workstation may be acting as a manager for some part of the networked systems and applications. It doesn't take much arithmetic in many organisations to recognise that, far from downsizing networked systems management, we have in fact seen an upsizing in this area.

Redressing this balance and ensuring that networked systems, and especially the business applications for which they are used, continue to serve the business most effectively are the major challenges facing businesses intent on rightsizing in the nineties.

3 Describing distributed systems

In order to effectively manage any system, it is necessary to construct a description of it. This might be an informal sketch on paper or in a manual from which the experienced system manager can work, or might be embodied in the code of an application tailored to managing a particular system. Furthermore, it is necessary to have some idea of the task for which the description will be used – will it be for system installation, configuration, fault diagnosis and so on? Often this results in management applications that only perform one of these tasks, and which may not interwork with those performing other tasks, even on the same system or for the same application. There are, of course, some description languages, such as the Guidelines for the Definition of Managed Objects (GDMO) (CCITT, 1992), which describe the information that may be examined or changed in a system or applications, but often stop short of putting real semantics on those operations, except in comments.

The Dolphin description language takes a different slant. It is declarative in nature. and makes no assumptions about the purpose for which the description will be used. Rather, it concentrates on giving a precise description of the *correct* functioning of the system. Since the number of ways in which a system can fail may be very large, and may be dependent on parts of the system outside its direct scope (e.g. network stacks), describing the correct configuration is a significantly easier task, especially for the system designer.

The management tasks to be performed, such as configuration, diagnosis and monitoring, may now be described in a generic manner making use of the descriptions. Since these tasks are not dependent on the particular system to be managed, they may form part of the core of the management system. Descriptions of other systems and applications may now easily be added and the same management tasks performed. Of course, the interpretation and use of the description depends on the task being performed. For configuration, it can be regarded as a specification of things to install and change; for diagnosis, it is a list of things to check.

This separation of description and function in management reduces considerably the complexity of developing management tools, and ensures consistent behaviour for administrators through use of the same system description.

3.1 Models and objects

The fundamental component of the Dolphin language is the model. At one level, a model is simply a structuring component that refers to a collection of objects. It is, however, the unit by which the management system knows about systems and applications and it is conventional that a model should describe a whole system or a well-defined part of one. We can rely on a model being present in its entirety or totally absent. Models may themselves import other models, permitting the decomposition or specialisation of descriptions.

The Dolphin language is object-oriented in nature. Object definitions describe the fundamental components of the system to be managed with inheritance being used in the usual way. Instances of each of these object definitions will represent objects that are discovered in the real world. Figure 1 shows examples of some of the objects that might occur in the specification of a machine running the UNIX® operating system.

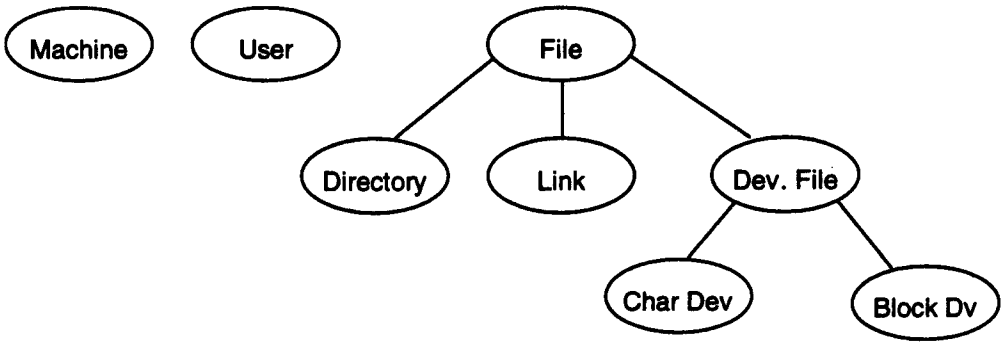

Figure 1 Object hierarchy

3.2 Location

It is insufficient, however, to only consider applications that run on a single machine. Whilst the management of such applications is not trivial, it would be short-sighted to consider that these applications are typical of those being deployed by many organisations, either now or in the future. It is necessary, therefore, to introduce a concept of the location of objects.

Some objects are fundamentally self-locating. For example, an HP-UX machine or a network printer can be reached simply by looking up its network address in a name server. Other objects, such as users and files, are not directly network addressable and information about them must be obtained via some other object, such as a computer system. Furthermore, unique identification of such objects is typically only assured within the confines of a single system – two users with the same name on different machines are not required to be the same user. Thus, the concept of the location of an object provides us with the ability to both locate and uniquely identify all objects being managed. This will be especially important when an application is distributed across multiple machines.

Here are Dolphin definitions for some of the objects in Figure 1. The ISA keyword indicates inheritance.

```
OBJECT UnixMachine
OBJECT File LOCATION UnixMachine
OBJECT Link ISA File
```

3.3 Attributes

The characteristics of a particular object are described by its attributes. There are two principal types of attribute in the Dolphin language:

® UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

- basic attributes represent information that can be obtained from the real world, for example, the name of a user or the owner id of a file.
- derived attributes, also called rules, represent higher level information and depend on the status and value of other attributes. For example, a particular user may read a given file if the user's id and the file owner id match, and the appropriate permissions are set on the file.

Here are Dolphin definitions for some attributes of the objects in Figure 1.

```
[User u] name [String s]
[File f] ownerId [Integer id]
[User m:u] canRead [File m:f]
IF
    [u] id [id] &
    [f] ownerId [id] &
    [f] ownerMode ['read']
```

The notation "m:" in the last attribute is used to refer to the location of the users and files. In this case, since the same name is used, the rule can only be true if the user and file are located on the same machine. As can be seen, it is only necessary to specify this location once for each variable in a rule.

3.4 Connecting to the real world

Without further adornment, the language presented here could describe many things. In order to be able to obtain information from the real world, it is necessary to ground the management system to some access mechanisms. This is done through request and action definitions.

A *request definition* describes how some particular information source in the real world may be interpreted. For example, a request about users on an HP-UX system would relate the /etc/passwd file to a collection of users with their names, user ids, home directory name and so forth. Alternatively, the online status of a network printer might be obtained by a request using some known SNMP variable.

By contrast, an *action definition* describes how a particular change may be made to the configuration of a system in the real world. For example, an action to disable a user on an HP-UX system might make a change to the /etc/passwd file to put a * in the password field. The action definition must also express any side-effects that may occur whilst performing the action.

The exact mechanism for doing this is not described here, but the interested reader is referred to Pell (1993).

4 Practical experience - a print spooler

In this section, we give a practical example of the use of the Dolphin language to model a distributed application. The application that we have chosen is the print spooler of the UNIX® System V operating system, and its derivatives. We first give a very brief tutorial on the relevant internal workings of the spooler. For more details, the reader is referred to the manuals (Hewlett-Packard Company, 1993).

4.1 Background

For clarity, we distinguish between a *physical printer* that produces paper and a *logical printer* that is the representation of a physical printer internal to the print spooler. One physical printer is likely to have a logical printer representing it on many computer systems. Indeed, it is possible for one physical printer to have multiple logical printers representing it on a *single* computer system, each providing the appearance of a separate personality for the printer. We shall not, however, deal further with this latter case.

Every computer system that has the print spooler installed will have a number of configured *logical printers*. These are the printers to which a user of the system may send print jobs. They may be of three types:

- A *local printer* is one for which the corresponding physical printer is directly connected to this computer system. Information about the physical local printer will also be obtained from this computer system.
- A *remote printer* is one for which another computer system acts as a server. Jobs destined for this printer will be sent to the server for further handling. Note that it is possible for a job to pass through many servers before reaching one that can actually print the job.
- A *networked printer* is one for which this computer system has final responsibility for printing jobs, but where the corresponding physical printer is connected directly to the network. Information about the physical printer can, therefore, be obtained directly from the printer.

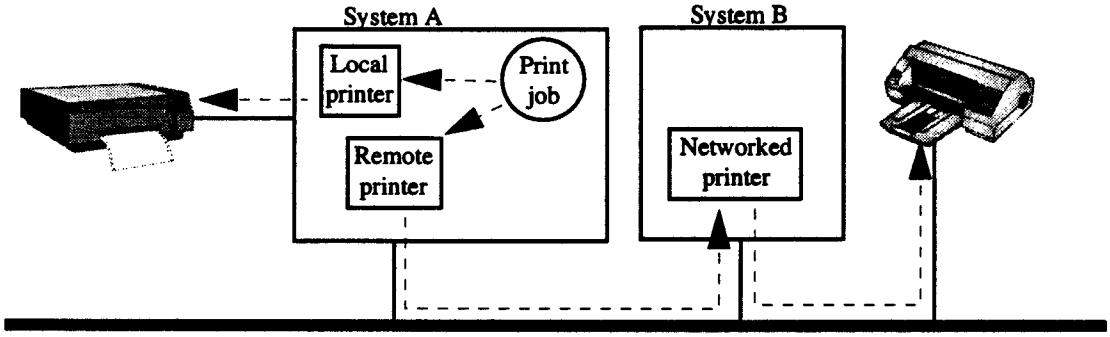

Figure 2 print spooling system

Figure 2 shows how these logical printers interact in a typical spooling system. At each node within the printing system, there are additionally a number of requirements of the local configuration, such as provision of adequate disc space, permission checking and so forth.

4.2 Object structure

Our brief description of the spooling system has identified some of the principal objects that must be modelled. Besides these "visible" components of the application, however, we must also model some hidden components – the application components that do the real work. We identify two:

- The print spooler application itself. This represents the application that the end-user would recognise. A user complaint might refer to some part of this application, e.g. printing a file, obtaining a queue listing, deleting a spooled file. The way that this object is modelled must, therefore, reflect in part its use in the real world.

- The scheduler. This represents the permanent "live" part of the print spooler. It must be running at all times and receives requests from the application mentioned above and acts on them.

We can now construct definitions for all these objects as follows:

```
OBJECT LogicalPrinter LOCATION Machine.
OBJECT LocalPrinter ISA LogicalPrinter.
OBJECT RemotePrinter ISA LogicalPrinter
OBJECT NetworkedPrinter ISA LogicalPrinter
OBJECT Lp LOCATION Machine
OBJECT Scheduler LOCATION Machine
```

4.3 Rule definitions

In order to describe the attributes of the objects that we have now defined, we apply a process of stepwise refinement, always keeping in mind the structure of distribution in the application. Note that, as we refine and develop our rules, we encounter the need for information that we will later request from the real world. This will be the basis for our basic attributes.

The print spooler application

At this stage, we must decide how to present the functionality of the spooler to the administrator. Our spooler does not permit restriction of the use of printers to particular users, so we only model whether it is possible for *anybody* to print on a given printer. The specification of this rule looks as follows:

```
[Lp m:_] canPrintOn [LogicalPrinter m:p]
```

This attribute will be true for a certain logical printer (p) if it is possible for any user to print on that printer. Note that this is an attribute of the print spooler application (Lp), since this is likely to be the source of a user complaint. The notation "_" is used to refer to an arbitrary object of the given type. Often, as in this case, there will only be one object instance, e.g. the print spooling application, on a given machine.

In the body of this rule, we express the conditions that must hold in order for it to be possible to print on the given printer. These are that we must be able to determine the name of the machine on which the application resides and that of the desired printer, and that the local scheduler must be capable of printing to the named printer on behalf of the local machine (the scheduler's client). Thus, the full rule becomes:

```
[Lp m:_] canPrintOn [LogicalPrinter m:p]
IF
    [m] name [clientName] &
    [p] name [printerName] &
    [Scheduler m:_] canPrintOn [printerName] for [clientName]
```

The scheduler

In considering the management view of the scheduler – the active component of the print spooler – we must be aware that it will be accessed remotely when a print job originates on a machine other than the server for the corresponding physical printer. In this case, the

machine which originates the print job must determine both the name of the remote server, and the corresponding printer name on that server. There is no requirement that the given printer name actually exists on the remote server, however! From a management viewpoint, therefore, we model the ability of the scheduler to print on a named printer for a particular named client. The type DomainName is a special form of string whose content is restricted to the form of Internet domain names.

```
[Scheduler m:s] canPrintOn [String printerName] for [DomainName clientName]
```

In order that the scheduler can perform as indicated, the following conditions must be satisfied. The scheduler must itself be running and there must exist a logical printer of the given name, which must be accepting jobs and be able to print for the named client. This is expressed in the full form of the rule:

```
[Scheduler m:s] canPrintOn [String printerName] for [DomainName clientName]
IF
    [s] running &
    [LogicalPrinter m:p] name [printerName] &
    [p] acceptingJobs &
    [p] canPrintJobFrom [clientName]
```

Logical printers

So far, we have made no distinction between the various types of logical printer introduced earlier. Now it is time to do this. Recall earlier that we defined an object LogicalPrinter to represent any type of logical printer. There will not, however, exist any "logical printers" in a real system – only its subclasses will be instantiated. So we can define the canprintJobFrom attributes of these subclasses of logical printer.

Successful printing on a locally connected printer requires that the logical printer be enabled, and that its device file be accessible to the user whose name is 'lp' – the owner of the print spooling system. In this case, we choose to ignore the name of the particular client requesting service. We could, however, enhance this rule at a later stage to check this against some security policy.

```
[LocalPrinter m:p] canPrintJobFrom [DomainName _]
IF
    [p] enabled &
    [p] devFile [m:devFile] &
    [User m:lp] name ['lp'] &
    [lp] hasReadWriteAccessTo [devFile]
```

Similar constraints occur in the case of the networked printer, except that explicit checking of the ability to print is deferred to the (directly accessible) printer itself. This is embodied in the use of the isOk rule, which is defined appropriately for each type of network printer. It need not be of concern, however, to the designer of the print spooler manager.

```
[NetworkedPrinter m:p] canPrintJobFrom [DomainName _]
IF
    [p] enabled &
    [p] networkPrinter [np] &
    [np] isOk
```

The final case, that of the remote logical printer, is the most interesting since it is in this description that the fundamental requirement for managing a *distributed* application is embodied.

Here, in addition to determining whether the logical printer is enabled locally, we determine the remote server for this printer, and the name of the associated logical printer on that server. Finally, we determine whether the scheduler *on the remote node* (server) can print to the named printer on behalf of the given client.

```
[RemotePrinter m:p] canPrintJobFrom [DomainName clientName]
IF
    [p] enabled &
    [p] remotePrinterName [rpName] &
    [p] remoteServer [server]
    [Scheduler server:_] canPrintOn [rpName] for [clientName]
```

4.4 Basic attributes

Having defined all of the rules that are required, we have also determined the basic information that is required to manage the spooling service. All of this information is readily available through normal UNIX® commands or configuration files.

```
[Scheduler s] running
[LogicalPrinter p] name [String s]
[LogicalPrinter p] enabled
[LogicalPrinter p] acceptingJobs
[LocalPrinter m:p] devFile [File m:df]
[NetworkedPrinter np] networkPrinterName [String npName]
[RemotePrinter rp] remoteServer [Machine m]
[RemotePrinter rp] remotePrinterName [String rpName]
```

Each of these attributes must be represented by a (part of a) request to the real world. For example, the status of the scheduler can be checked by using the '**lpstat -r**' command. Similarly, information about the various logical printers can be gleaned from a configuration file. In each case, the return from the request (typically a string or an SNMP variable) is transformed by the request processor into some information about basic attributes, such as the running state of the scheduler.

4.5 Using the model

Having defined our model of the print spooling application, let us consider how it would be used in order to diagnose a potential fault. The symptom that we wish to resolve is that a user cannot print on the printer called **laser1** whilst working on the machine called **machine1**.

We start with the attribute that we defined earlier for the print spooling application itself:

```
[Lp m:_] canPrintOn [LogicalPrinter m:p]
```

We must instantiate the variables represented here. In this case, **m** will be a machine whose name is **machine1**, and **p** will be a logical printer whose name is **laser1**. Throughout the remainder of this section, we will represent this by the notation:

```
[Lp m:_] canPrintOn [LogicalPrinter m:p]          (m.name='machine1', p.name='laser1')
```

Note that, in order to do this, we must determine the actual type of the logical printer whose name is **laser1**. This we do by performing a request to the machine **m** for information on its available logical printers. This will tell us the precise type of each. Let us suppose, for this example, that **laser1** is a remote logical printer.

Now, we can expand the rule above and, by using the known information about m and p, we reach the next major goal:

[Scheduler m:_] canPrintOn ['laser1'] for ['machine1'] (m.name='machine1')

In order to verify this, we must first check whether the scheduler is running. For this, we can perform a straightforward request to the machine m. We must then determine whether a logical printer called laser1 exists, and whether it is accepting jobs. We know the former condition to be satisfied because of the way that we reached this point, but we must perform another request to determine that the printer is, in fact, accepting jobs. Finally, we reach the next goal:

[RemotePrinter m:p] canPrintJobFrom ['machine1']
(m.name='machine1', p.name='laser1')

Now, we must determine whether this remote printer is enabled and what are its remote server (suppose its name is printserv) and remote printer name (suppose it is attic). Assuming all of these can be successfully found using requests, we reach the goal:

[Scheduler server:_] printOn ['attic'] for ['machine1'] (server.name='printserv')

At this point, we continue to check and expand the various rules, but this time using information from the printserv machine. Finally, either all conditions will have been found to be satisfied, or some condition will not succeed. In the latter case, this information may be used to identify a possible fault, although further work may be required to obtain a full diagnosis.

5 Experience and future work

The Dolphin management system presently operates in a centralised manner, that is, there is a single management station which makes requests to agents on a number of managed nodes for information. This has proved to be quite satisfactory so far in managing a relatively small number of systems – roughly a large workgroup or a small site. Furthermore, the approach of separating system descriptions and task specifications has proved to be a good way to build management systems since the descriptions are easily written and understood, and the quality of the management provided (diagnosis, etc.) is more than adequate. There remain, however, a number of interesting research issues to be solved.

The centralised nature of the system causes some problems of scale. We are limited principally by the ability of the management station to hold and manipulate large quantities of information from many systems. In order to progress further, to true enterprise management for example, we require the ability for multiple management stations to coexist and to cooperate in the management tasks. For example, in our print spooling example, it would be possible to configure the system to print on a printer on the other side of the world. This might mean that the remote server would be outside the scope of the local management station. So, the assistance of a remote management station must be sought to obtain and check certain information on behalf of the local manager. Determining which remote manager to use, and arranging for the diagnosis or fix to be split between the management stations is a challenging problem.

Agents are presently assumed to be passive. That is, they do not generate asynchronous events when some piece of information changes. This means that, whilst performing a management task, fresh information will be gathered from managed systems even if nothing has changed. The introduction of asynchronous events, which may be regarded as something

like an unsolicited request for information, together with persistent storage of information, would give a more responsive management system, whilst not sacrificing accuracy.

There is also no notion of time within the management system. That is, only immediately available information is used when performing management tasks. Introducing such a concept would permit more expressive modelling of applications that themselves have a notion of time, or would allow historical views of the systems being managed.

6 Related work

We divide related work into two parts – the provision of information for management, and the use of that information.

6.1 Management protocols

There are a number of emerging standards for information retrieval from managed systems. In the Internet world, and increasingly wider, the SNMP standard (Schoffstall, 1990) is used to specify data that may be retrieved from systems and devices. Historically, this has been used for the *monitoring* of network devices, although recent work on the Host Resources MIB (Grillo, 1993) has extended this monitoring to the lower levels of computer system activity. Application monitoring still remains a missing area. There has, until now, been no widespread use of SNMP for actually changing the configuration of systems and devices. This is largely because of SNMP's reliance on a single password, or community name, per device for write permission. With the emergence of SNMPv2 (Case, 1993), this objection to the use of SNMP for active management may go away. It remains to be seen, however, just how widespread is the adoption of SNMPv2.

The lack of information about applications on the desktop is being addressed by the Desktop Management Task Force (DMTF) – a industry grouping of the major players in the PC marketplace. They have recently produced a first release of their Desktop Management interface (DMI) (DMTF, 1994a), together with an initial set of component descriptions (DMTF, 1994b). The DMI provides access to an extensible set of components which may describe the desktop operating environment as well as installed applications. It is hoped that software and hardware manufacturers will move to producing components for the DMI as part of their products, thus evolving towards full management of desktop systems. It is too early to tell how well this will be realised, although one shortcoming that will need to be addressed is that the DMI is intended as a local interface only. A mapping has been defined from the component descriptions to SNMP MIBs, but this hides some of the benefits of the DMI specification, such as easy access to the component definitions.

In the telecommunications world, the CMIS/CMIP standard (CCITT, 1991 and ISO, 1991) is in widespread use. This provides a broader scope for definition of managed objects through the Guidelines for the Definition of Managed Objects (GDMO) (CCITT, 1992). However, the required implementation of these protocols is perceived by many to be much more heavyweight than SNMP or DMI, and it looks unlikely to become established in any field other than telecommunications.

The Dolphin management system adopts a liberal attitude to this diversity of management protocols, permitting the use of information from many different sources to be used in the management of systems and applications. In addition, it is possible to take these

definitions as a starting point for Dolphin object definitions, and then to build higher level semantics on top.

6.2 Management products

In most cases, the above management protocols simply convey data about the systems being managed. With the exception of some limited textual comments, there is little attempt to express the semantics of the systems which are, of course, essential for effective management. There are a number of management products which introduce these semantics.

The TME management environment from Tivoli Systems Inc. (Kramer, 1993) presents to the system manager a similar view to the Dolphin system. That is, it is an object-oriented management framework which, for the applications supported, has a similar model structure. There, however, the similarity seems to end since TME requires separate underlying programs for its supported application areas such as user management, security management and so on. This contrasts with our approach of using a single management application augmented by *descriptions* of the systems and applications to be managed, which are more easily adaptable to changing requirements.

Unicenter, a product of Computer Associates, addresses the needs of large system installations including tools for routine tasks such as storage management, problem management, help desk setup and so forth (Ricciuti, 1992). It is not clear to us how this is constructed, although we suspect that it again uses a number of separate underlying programs, with the difficulties of consistency outlined earlier.

There are, in addition, emerging products from various major computer manufacturers which provide some of the functionality available in the Dolphin system, whilst not giving the same flexibility of specification.

7 Summary

In this paper we have presented a language for describing systems and applications to be managed, and have shown how this may be done, even in a distributed environment. This language and technology are embodied in the HP OpenView AdminCenter product from Hewlett-Packard which supports configuration and change management in an enterprise.

The principal benefits from using this approach to building a management system lie in the ready capture of the necessary management understanding through a rich descriptive language, and the uniform application of this knowledge to the various facets of system management. Although the initial work in constructing a comprehensive model might appear to be somewhat more than comparable approaches, we believe that the long-term gains in productivity for system implementors and managers far outweigh this initial investment.

8 References

Case, J. *et al* (1993) Introduction to version 2 of the Internet-standard network management framework, *Internet request for comments 1441.*

CCITT (1991) Recommendation X.710 (1991). Common management information service definition for CCITT applications.

CCITT (1992), Recommendation X.722 (1992) | ISO/IEC 10165-4 (1992). Information technology – open systems interconnection – structure of management information: guidelines for the definition of managed objects.

Desktop management task force (1994a) Desktop management interface specification, version 1.0.

Desktop management task force (1994b) PC standard groups, version 1.0.

Grillo, P. and Waldbusser S. (1993) Host Resources MIB, *Internet request for comments 1514.*

Hewlett-Packard Company (1993), System administration tasks, in *HP-UX manuals release 9.0.*

ISO (1991) ISO/IEC 9595 (1991). Information technology – open systems interconnection – common management information service definition.

Kramer, M.I. (1993) Enterprise system management: the quest for industrial-strength management for distributed systems. *Patricia Seybold's Distributed Computing Monitor* **8**(6), 3–23.

Pell, A.R. *et al* (1993), Data + understanding = management," *IEEE first international workshop on systems management*, Los Angeles, California.

Ricciuti, M. (1992) Industrial strength UNIX management tools *Datamation* **38**(10) 73–74.

Schoffstall, M. *et al* (1990) A simple network management protocol (SNMP), *Internet request for comments 1157.*