ABSTRACT

In the absence of standards for the TMN F interface, interoperability of workstation applications can be based on the Q3 interface; CMIS-capable interpreted scripting languages and associated lightweight string-based protocols can support the rapid construction of lightweight, portable TMN workstations.

A CMIS-Capable Scripting Language and Associated Lightweight Protocol for TMN Applications

George Pavlou and Thurain Tin, University College London

Over the last couple of years, the telecommunication management network (TMN) finally seems to be coming of age. The underlying open systems interconnection (OSI) management framework (X.700) has been almost completed, while a number of TMN platform products on the market have reached maturity, blending naturally the object-oriented aspects of the TMN information architecture with object-oriented methodologies and environments. As part of early research in Europe to verify the TMN concepts, study their applicability, and provide feedback to the relevant International Telecommunications Union—Telecommunications Standards Sector (ITU-T) working groups, a number of collaborative research projects specified, designed, implemented, and demonstrated TMN systems. These efforts spanned from asynchronous transfer mode (ATM) resource and routing management to the provision of international leased line services with guaranteed quality characteristics, supporting end-to-end multimedia conferencing teleservices. An aspect of this research resulted in the OSI Management Information Service (OSIMIS) object-oriented platform [1] which provided early mappings of the Guidelines for the Definition of Managed Objects (GDMO) information specifications to C++ through high-level application programming interfaces (APIs) and paved the way for the sophisticated products that have recently started to appear on the market.

While C++ has become the *lingua franca* of TMN object-oriented development environments, interpreted languages, such as Tool Command Language (Tcl), Scheme, and recently Java, provide particularly interesting alternatives for rapid prototyping. Their usefulness becomes apparent when combined with widget sets which allow the quick and easy construction of managing applications with graphical user interfaces (GUIs); that is, TMN workstations (WSs). Such applications may need to run on inexpensive desktop and even laptop computers, and also operate in non-TMN environments such as the Internet, communicating with the TMN over Transmission Control Protocol/Internet Protocol (TCP/IP).

Given these requirements, we designed and implemented extensions to the Tcl language that provide an interpreted high-level API to the common management information service (CMIS). This can be used to quickly prototype simple managing applications, such as testing scripts, or to implement combined workstation–operation system (OS) applications which communicate with the TMN through Q3 interfaces. The fact that the proposed CMIS API is string-based, as dictated by the nature of Tcl and similar scripting languages, led us also to the specification and implementation of an associated lightweight string-based Common Management Information Protocol (CMIP). This maps directly onto the OSI transport service and can be converted to full CMIP through interworking units. Lightweight CMIP (LCMIP) incurs a minimal protocol stack cost to relevant applications.

This infrastructure has been successfully used to design and implement WS applications and to provide general scripting/testing facilities in the aforementioned TMN systems. This article discusses the relevant issues and describes the work behind the CMIS-capable Tcl and the associated lightweight CMIP. The rest of the article has the following structure. Since the CMIS-capable Tcl mainly addresses the needs of TMN WSs, we first discuss our WS model, which combines WS functionality with an OS in the managing role (WS–OS). We then discuss, in some detail, issues behind the CMIS-capable Tcl and the associated lightweight CMIP. We finally give examples of the applicability of this technology and present our conclusions. The article assumes a familiarity with the TMN architectural concepts and with OSI management, which constitutes the base technology for the TMN. It also assumes that the architectural paradigm for intra-TMN communication is based on Q3 interfaces, as implied by the TMN specifications.

TMN WORKSTATION–OPERATION SYSTEM APPLICATIONS

The TMN WS function (WSF) provides the means to interpret management information for the human user [2]. It communicates such information to OS functions (OSFs) across the f reference point. The intention behind the latter is to standardize management information exchanges supporting WS functionality, enabling open communication between WSs and OSs through standard F interfaces. Remember here that a reference point expresses the logical aspect of communication between two TMN functional blocks and becomes an interface when the functional blocks become physical blocks in separate pieces of equipment, interconnected across the data communication network. The convention is that lower-case letters denote reference points, while upper-case letters denote interfaces (e.g., q3 and Q3, respectively).

The first requirements for the F interface were expressed in ITU-T Recommendation M.3300, "TMN Capabilities Presented at the F Interface" (1992). The latest version of ITU-T Recommendation M.3010, "Principles for a Telecommunications Management Network" (1996) [2], reflects the most recent views of ITU-T Study Group IV regarding the F interface.

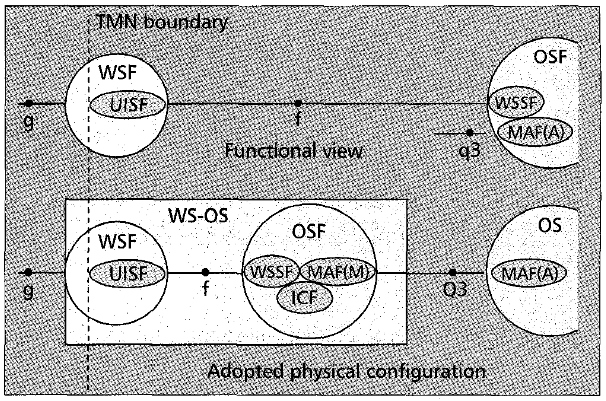
According to [2], the functionality across the f reference point is a superset of the functionality available at q3 reference points. An example of the data at f in addition to the managed objects at q3 is display support (e.g., maps), management information kept in databases, information on function or command initiation, help text, and so on. While this is the intention, it is quite a challenging problem to propose an approach for the f reference point which is generic enough not to constraint implementations and also able to cope with the needs of different types of WSFs.

Given the fact that the F interface specification has not yet been sufficiently addressed, TMN WSs in research projects targeting pilot TMN systems have followed two models: they either bundled WS functionality together with operations systems in a tightly coupled fashion, or defined proprietary F interfaces with their own protocols and information models. As part of our TMN developments in RACE projects, we have experimented with both approaches. A particularly important conclusion arrived at through this experimentation was that, in our cases at least, the bulk of information exchanges across the f reference point could, in fact, be mapped onto exchanges across q3 reference points. This mapping is not one-to-one, but the underlying assumption is that most of the information necessary to support WS functionality could be derived from management information available in q3 form. In the general case, of course, and according to the ITU-T view, q3 models will not be sufficient to support complete TMN WS functionality.

Given the fundamental assumption that f can be derived from q3, we propose the physical configuration for WS applications depicted in the lower part of Fig. 1. The upper part depicts the functional view of interconnection between the WSF and OSF functional blocks as in M.3010. The main functional component of the WSF is the user interface support function (UISF), which converts between f messages and displayable objects. At the other end of f, the WS support function (WSSF) is an OSF component that understands the f messages and responds with the requested information. As already stated, the information available across f is generally a superset of the information available across q3. Assuming that f can be fully derived from q3, we can physically separate the WS from the OS using the Q3 interface. In order to achieve this, we move the WSSF from the OS that offers the Q3 agent interface to another OSF in the managing role, tightly coupled with the WS function. The f reference point remains internal within the resulting physical block, which we will call a workstation-operation system (WS-OS), emphasizing the fact that it contains both a WSF and a managing OSF.

It should be mentioned that according to the TMN architecture, a physical block containing both a WSF and an OSF is called an OS; but we call it WS-OS for purposes of illustration. The ITU-T calls it an OS because its main role *is* as an OS, the WS being secondary. In our case, its main role is that of a WS: the managing OS is there to support the Q3 interactions and typically does not have a significant management role. The latter is provided by the TMN OSs with which it communicates.

Let us examine the WS-OS decomposition in more detail. The WSF contains the UISF component, which is responsible for managing the graphical user interface. It may also contain a security function (SF), which provides authentication and access control services for human users with respect to the WS functionality (the SF is not shown in Fig. 1). The OSF contains a WSSF, an information conversion function (ICF) and a management application function in the manager role (MAF-M). Interactions across the f reference point are effected as UISF-WSSF message exchanges. Within the OSF, the ICF maps messages and relevant information between the WSSF and the MAF-M. The latter is responsible for interactions across the Q3 interface as manager; that is, it performs operations on man-



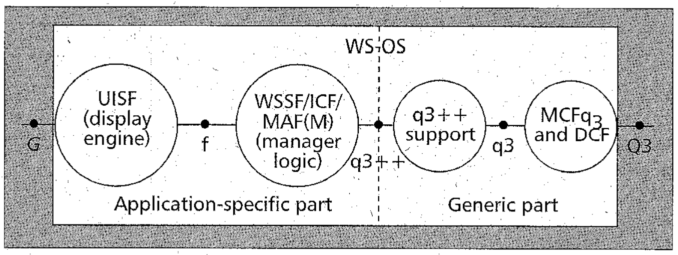
■ **Figure 1.** *Adopted physical configuration.*

aged objects and receives notifications emitted by them. Automated management intelligence, if any, is embodied in the MAF-M component, while the WSSF and ICF are responsible for f message support and information conversion, respectively. Note that the OSF may also contain a directory access function (DAF) in order to support location transparency and shared management knowledge services (the latter is not shown in Fig. 1).

According to the approach described above, interoperability between WS-OSs in managing roles and other OSs in agent roles is based on Q3 interfaces. The f reference point remains internal within a WS-OS and becomes a matter of design discipline rather than a TMN architectural prescription. In other words, it is up to the designer of a WS-OS application to separate GUI support functionality (UISF) from the application's management intelligence (WSSF/ICF/MAF-M). Such a separation can be effected through a well-defined internal software interface (i.e., a f reference point) rather than an "on-the-wire" interoperable F interface. This separation provides for a cleaner design and makes it possible to change the technology used for the GUI without affecting the rest of the WS-OS application. According to our experience, this separation is possible and desirable, but it is not easy to make the f reference point independent of the application-specific functionality. This is one of the reasons we believe interoperability should be based on the Q3 interface; as a consequence, we suggest that the f reference point should not be standardized.

In Fig. 2 we present another view of the internal structure of the WS-OS based on implementation considerations and internal software interfaces. The Q3 interface is supported by the message communication function (MCFq3) over the data communication function (DCF) (i.e., a Q3 upper- and lower-layer protocol stack, as in the ITU-T Q.812 / Q.811 standards). A q3 reference point is provided internally, realized typically through a CMIS API such as the X/Open XOM/XMP (directory access and file transfer APIs also need to be supported because they are part of q3 functionality). Most OSI/TMN software platforms provide higher-level access APIs that hide the complexity of "raw" CMIS access through higher-level abstractions. Such infrastructure is built on q3 APIs such as the XOM/XMP and is depicted in Fig. 2 as "q3++ support." Support for q3 and q3++ constitute the generic part of the WS-OS, while the rest is application-specific, implementing the various functional components (i.e., WSSF/ICF/MAF-M and UISF).

Given the fact that the f reference point is application-specific while the q3 one is generic and interoperable, we decided to produce generic infrastructure for the rapid prototyping of WS applications based on the q3 reference point. In fact, our infrastructure supports the "q3++" reference point, concealing much of the underlying CMIS/q3 complexity and harnessing its power. The key aspect of this infrastructure is that it is interpreted, based on the Tcl language and its object-oriented



■ Figure 2.

extensions, while it provides built-in support for the easy construction of GUIs, facilitating the rapid prototyping of WS applications decomposed according to the presented model. The application-specific part in Fig. 2 can be written fully in an interpreted language such as Tcl, while the UISF component can be prototyped using an associated widget set (Toolkit, or Tk, for Tcl) in a fraction of the time taken by the traditional compiled approaches.

A Scripting Language for the TMN WS-OS

Motivation

In today's distributed system environments, there are emerging trends in the usage of interpreted scripting languages and associated graphical toolkits. Such scripting languages are available in many varieties: procedural or object-oriented; some even provide built-in memory management facilities. Above all, their main attraction lies in the ability to prototype applications quickly, without having to go through the compile/link development cycle. In addition, their built-in GUI support makes the construction of graphical applications easy. The portability of application scripts across the confluence of different hardware and OS technologies is an extremely useful additional feature; it suffices to port the associated interpreter, done once for every hardware and OS platform.

TMN object-oriented APIs operating at a higher level than CMIS are supported by most TMN platforms today. Such interfaces help the TMN application developer a lot by providing the opportunity to spend more time on the application functionality rather than handling the complexity of the "raw" CMIS service. Work on high-level, object-oriented Q (CMIS and directory access) APIs in C++ has been conducted in the RACE Integrated Communications Management (ICM) project [3], resulting in the OSIMIS object-oriented TMN platform [1, 3]. Work is also ongoing in this area for the standardization of such APIs, driven by the NM Forum [4].

The OSIMIS high-level C++ APIs encapsulate the underlying CMIS and directory access services and the Q protocol stack by using object-oriented principles and techniques. One particular abstraction for TMN OSFs functioning in the managing role is the remote management information base (RMIB) [1, 3] access infrastructure, which involves the use of proxy objects (of an `RMIBAgent` class). These act as the local image of the remote agent applications (one-to-one mapping) and provide CMIS-like message passing, simplifying many of the CMIS access details (e.g., linked replies may be assembled transparently, high-level event reporting facilities are provided, etc.). As such, requests to a remote application in the agent role are effected simply through local invocations on the representing proxy object, concealing much of the Q interface interaction. Both synchronous and asynchronous invocation semantics are supported. In the latter case, the results/errors are eventually passed back via callback facilities. An abstract `RMIBManager` class is provided to specialize the different callback behaviors.

The OSIMIS RMIB manager support infrastructure maps onto the "q3++ support" component depicted in Fig. 2.

The RMIB API provides argument-passing facilities based on string representations of attribute, action, and notification types and values, filter expressions, and distinguished names. A more detailed consideration behind the issues of a string-based CMIS approach is presented later, in the section on the lightweight CMIP. These string-based methods may be used even when programming an application in C++ because of their simplicity. However, they are obvious candidates for use in unifying the RMIB environment with string-based interpreted environments such as Tcl. This unification constitutes a high-level interpreted access facility and is discussed next.

Tcl and Tk

Tcl [5] is a general-purpose scripting language designed to be both embeddable and extensible in a wide variety of applications. The core of Tcl is the Tcl interpreter, existing in a library of C language procedures. The built-in interpreted procedures, which mirror the underlying compiled procedures and any "application-specific" extensions, can either be embedded in user applications or used interactively in a shell environment. Tcl is a weakly typed language based on strings and is relatively easy to use compared to programming languages such as C/C++ which follow the compiled approach. In addition, object-oriented extensions such as Object Tcl provide classes, inheritance, and polymorphism, which are important for structuring complex applications.

One of the main reasons for the popularity of Tcl is its extension for the MIT X Window System and MS Windows, Tk. This is a graphical toolkit that extends the core Tcl facilities with additional commands for constructing GUIs. Tk exists in the form of a collection of display classes (or widgets), providing a user-friendly way to compose graphical primitives. The combination of Tcl and Tk presents a suitable environment for the rapid construction of GUI-based applications. This approach is quicker than traditional compiler-based GUI development, where a longer learning curve is required and programming involves complex data structures.

A Tcl-Based High-Level CMIS Language

As explained above, the Tcl-based high-level CMIS scripting language mirrors the compiled RMIB functionality and owes much to string-based management syntax support. The process of creating an interpreted layer on top of an RMIB was relatively simple: the two domains, C++ and Tcl, are bridged by using a well-defined mechanism for exporting C++ classes into Tcl. The relatively easy integration of Tcl with compiled languages such as C++ and C was an important feature in its choice, in addition to the easy-to-use Tk widget set that complements it. The resulting language, called *Tcl-RMIB*, provides a number of object control and management commands for relatively high-level CMIS interaction. RMIB C++ objects are manipulated by their interpreted counterparts. Commands are provided for the user control (creation, deletion, etc.) of the agent and manager objects locally in Tcl, and for management operations through those objects. Both synchronous and asynchronous modes of operation are supported, the latter essential in single-threaded execution environments.

The control commands allow indirect handling of the proxy agent and callback manager objects in C++, the latter totally transparent to the user. Each control object (RMIBAgent or RMIBManager) is assigned a unique identifier for selection in the management commands. A manager object which receives asynchronous results and event reports is bound with specific

Management command	Description
m_connect *agentId*?-a *agent_name*? ?-h *host_name*?	Establishes association with a remote agent
m_disconnect agentId	Terminates association with a remote agent
m_get *agentId*?-c *class*? ?-i *instance*? ?-s *scope* ?*sync*?? ?-f *filter_expr*? ?-a *attribute* ...? ?-m *managerId*?-*o*??	Allows the M-Get invocation; may return an invoke identifier for callback correlation
m_cancel_get *agentId invokeId*	Allows the M-Cancel-Get invocation
m_set *agentId*?-c *class*? ?-i *instance*? ?-s *scope*?*sync*?? ?-f *filter_expr*? ?-w|a|r|d *attrType*?=*attrValue*? ...? ?-m *managerId*?	Allows the M-Set invocation; may return an invoke identifier for callback correlation
m_action *agentId*?-c *class*? ?-i *instance*? ?-s *scope* ?*syn*?? ?-f *filter_expr*? ?-a *actionType*?=*actionValue*?? ?-m *managerId*?	Allows the M-Action invocation; may return an invoke identifier for callback correlation
m_create *agentId*?-c *class*? ?-i *instance* | -*s superior_inst*? ?-r *reference_inst*? ?-a *attrType*=*attrValue* ...? ?-m *managerId*?	Allows the M-Create invocation; may return an invoke identifier for callback correlation
m_delete *agentId*?-c *class*? ?-i *instance*? ?-s *scope* ?*syn*?? ?-f *filter_expr*? ?-m *managerId*?	Allows the M-Delete invocation; may return an invoke identifier if callback correlation
m_notify *agentId managerId* ??-c *class*? ?-i *instance*? ?-e *event_type*? | ?-f *filter_expr*? | ?-a??	Allows the callback registration of a manager to receive event reports and also request termination of notification; event reporting may be canceled (using -s)

■ **Table 1.** *Tcl-RMIB management commands.*

behavior via the attachment of user-defined callback scripts. A typical callback script may be to process an event report, receive an asynchronous management result/error, or even be notified of a problem such as the remote application's termination, loss of connectivity, and so forth.

The management commands operate on the control objects and essentially provide the CMIS functionality in Tcl. They fall broadly into three categories: management association commands, CMIS manager operation commands, and a command for requesting event reports based on a logical assertion. The complete set of management commands and their syntaxes are listed in Table 1.

The management association commands are `m_connect` and `m_disconnect`; they operate on a proxy agent object. The management operation commands closely model CMIS operations but are simplified as much as possible. These are `m_get`, `m_cancel_get`, `m_set`, `m_action`, `m_create`, and `m_delete`, and they incorporate the standard CMIS mechanisms for scoping, filtering, and synchronization. Filtering is based on a string-based expression language. In order to allow asynchronous results/errors and event reports, the operations are extended to register the callback entity (i.e., an `RMIBManager` object with the specific behavior). Finally, an `m_notify` command is used to allow an `RMIBManager` object to be registered with a proxy agent in order to receive event reports based on specified filtering criteria.

The above approach mirrors the compiled RMIB one in Tcl and provides a CMIS-like interpreted interface. The full CMIS expressive power is available: scoping, filtering, linked replies, and fine-grain event reporting based on filtering. Control over management associations is also provided if necessary, while it may be also left to the underlying infrastructure. Two modes of operation are supported, synchronous and asynchronous. In the latter case, linked replies may be assembled or passed back to the manager object one by one, according to its requirements. It should be noted that only the request commands are shown in Table 1. A generic list structure that applies to all the requests is used for replies and errors; its structure is described in detail in [6].

The Tcl-RMIB approach provides semantics of a dynamic invocation interface, similar to the RMIB one. This means that there is no need for precompiled knowledge of the GDMO information model accessed across the Q3 interface. Meta-data is used to map attribute, action, and notification types to user-friendly names and to manipulate the relevant syntaxes. While this approach is protocol-oriented, higher-level abstract approaches are possible. A shadow MIB (SMIB) approach is described in [3], and an interpreted Tcl layer may be put in front of it in a similar fashion to Tcl-RMIB. Alternatively, similar infrastructure may be built directly over the Tcl-RMIB using one of the object-oriented Tcl extensions. The advantage of more abstract approaches lies in their simplicity; on the other hand, some of the CMIS expressive power may be lost when precompiled knowledge of the accessed GDMO model is necessary. We may investigate such more abstract approaches in the future.

Tcl offers most of the control constructs found in other high-level programming languages, while object-oriented extensions provide classes, inheritance, and polymorphism. Combining these with the Tcl-RMIB extensions described above enables developers to build arbitrarily complex but well-structured scripts that operate in managing roles. In addition, the Tk widget set supports the quick construction of GUIs. We have found this infrastructure extremely helpful in building WS-OS applications.

A Lightweight String-Based CMIP Protocol

While Tcl operates on UNIX/X-Windows, MS Windows, OS/2, and Mac System-7, portability of WS applications written in Tcl-RMIB is only possible after the latter has been ported to those platforms as well. This necessitates porting the relevant Q protocol stack infrastructure, which is far from trivial. Given the fact that all the interactions across the Tcl-RMIB interface are string-based, it should be relatively easy to map it on a simple on-the-wire interface. In that case, a generic interworking unit could be used to translate the string-based interactions to Q3 interoperable messages. The question that arises is at what level to define the string-based protocol, and the obvious answer is at the q3 reference point because it is well-defined, standardized, and generic. By defining a lightweight string-based CMIP protocol, we could replace the MCFq3/DCF part on Fig. 2 with a lightweight MCF/DCF offering similar functionality, moving essentially the complexity of the full Q3 stack to an interworking unit.

Motivated primarily by the need to run WS-OS applications on various platforms without having to go through complex, time-consuming, and possibly prohibitively expensive porting exercises, we have designed and implemented a lightweight string-based CMIP (LCMIP) operating over a reduced protocol stack. The added value is, of course, the reduced memory and processing power requirements when operating on limited systems such as laptop PCs. In addition, the lightweight protocol is mapped over both TCP/IP and ITU-T OSI lower layers. It should be mentioned that the ITU-T has also recently endorsed a TCP/IP mapping for the lower-layer Q3 profile in the latest version of Recommendation Q.811. Another benefit of the lightweight CMIP approach is that precompiled knowledge for new GDMO attribute, action, and notification syntaxes is only needed for interworking units, because LCMIP treats these values as strings.

The reasoning behind a lightweight CMIP protocol can be taken further through the possibility of managing simple network elements unable to support full Q3 stacks, most notably devices in mobile network environments. Such a protocol might also serve local/metropolitan area network (LAN/MAN) environments, currently dominated by the Simple Network Management Protocol (SNMP); the main advantages would be the richness of the associated GDMO information model, event-based operation, and the generic functionality of the systems management functions (SMFs), combined with lightweight protocol stack operation. It should be noted that similar efforts in the past, such as CMIP over TCP/IP (CMOT) and CMIP over logical link control (CMOL), failed. CMOT failed because the OSI management framework was far from mature enough to be adopted in the Internet at the time (1989); in addition, the CMOT protocol stack does not save much compared to the full CMIP stack (it still uses full presentation and application layers). On the other hand, CMOL has been suitably lightweight but lacked generality, operating only over the LLC-capable LAN/MAN.

The requirements behind a lightweight CMIP protocol are the following:

- To be economical and efficient
- To be general and not sacrifice any functionality with respect to full CMIP
- To be easily implementable on different platforms
- To operate over both ITU-T and Internet protocols
- To interoperate seamlessly with full CMIP through generic interworking units

A fundamental decision behind LCMIP has to do with the communication paradigm: a reliable connection-oriented approach has been chosen in a similar fashion to CMIP; this aligns with CMIP, the needs of telecommunication environments, and the nature of the emerging broadband technologies. As such, LCMIP is essentially a convergence protocol, mapped directly onto the OSI Connection-Oriented Transport Service/Protocol (X.214/X.224) which operates over the lower layers as in Recommendation Q.811 (e.g., X.25/LAPB, CLNP/LLC1, RFC1006/TCP/IP, etc.).

The LCMIP protocol itself is specified in Abstract Syntax Notation One (ASN.1), while the Basic Encoding Rules (BER) are being used to encode/decode its packets. Association control and remote operations functionality is embedded within LCMIP, while presentation and session functionality is not necessary. Session functionality is not used in CMIP at all (despite the fact that the session layer should be present for interoperability reasons), so it is not an issue in LCMIP. Presentation facilities are not needed because the values of attributes, actions, and notifications are transferred as "pretty-printed" strings. This implies that well-defined string representations for these are necessary. Currently, most management platforms define their own (proprietary) representations; a common approach should be agreed on. Of course, ASN.1 is a very powerful abstract syntax language, and it is not possible to map syntaxes to user-friendly string representations and vice versa in a generic fashion. This means that manual coding of the relevant mapping logic may be necessary, certainly a limitation. On the other hand, the use of a lightweight protocol with rudimentary presentation facilities satisfies most of the requirements set above. A detailed discussion of the proposed LCMIP together with performance analysis and comparisons will be the subject of a future paper, presenting the arguments for its potential standardization. The full LCMIP protocol specification can be found in [7].

The structure of the WS using the lightweight CMIP approach is shown in Fig. 3. An interworking unit is needed to convert LCMIP to full CMIP. This is pure protocol conversion and can be accomplished easily in a generic way by accepting primitives from one interface and copying them to the other after simple conversions with respect to attribute, action, and notification types/values. Such a unit must have knowledge of all such possible types/values and of the relevant ASN.1 syntaxes in order to perform the conversions. This means that every time it needs to convert for an interface whose GDMO model introduces new syntaxes, it has to be updated with additional logic and data to be able to handle them. This is a limitation, but it is outweighted by the flexibility and lightweight operation of LCMIP-based applications. Protocol conversion can be effected in both directions by such units.

In TMN terms the lightweight CMIP is a Qx protocol, while the interworking unit becomes a mediation function that performs pure protocol conversion. This classification applies to simple LCMIP-capable devices that need to be managed by the TMN. In the case of WS applications that use LCMIP as above, the same conversion takes place in the opposite direction; that is, the managing application speaks Qx while it accesses a managed OS through a Q3$\rightarrow$Qx mediation device. This type of "inverse" mediation is not catered to in the TMN architecture, but the interoperability point is the accessed OS Q3 interface, as already explained. It should also be clarified that the WS-OS does not communicate directly with the NE in Fig. 3 but with OSs at the other end of the Q3 interface. Such OSs may manage lightweight LCMIP-capable NEs through the interworking unit, as shown in Fig. 3.

Finally, our operational experience with LCMIP suggests that the size of applications is substantially reduced from the full upper-layer CMIP stack. The size of a simple application in a manager role using the full CMIP upper-layer stack is about 1.2 Mb at runtime; it becomes about 0.6 Mb over the LCMIP implementation. Performance is significantly better for applications communicating directly over LCMIP, but the need for a mediation device between CMIP/LCMIP introduces an additional delay. Summarizing, application size and performance clearly benefit from the direct LCMIP approach. In addition, the Tcl-RMIB infrastructure becomes easily portable across different hardware and OS platforms, while Tcl-RMIB-based scripts become fully portable since only the interworking unit needs to have knowledge of new ASN.1 syntaxes for attribute, action, and notification values.

Applicability Examples

We have used this technology in two RACE-II TMN projects and will also be using it in ongoing ACTS projects. In this section we look briefly at the types of applications for which we have been using it.

In the RACE II ICM project, a WS-OS application supported the GUI used by a human manager working for a value-added service provider (VASP). This GUI provides the means to effect an ATM-based virtual private network (VPN) service after receiving a request from a customer in nonelectronic form (e.g., fax message). The WS-OS can be used for naming the customer sites to be connected and describing the desired performance requirements (e.g., bandwidth, end-to-end cell delay, jitter, cost). It provides the graphical interface to add, delete, modify, and view information about customers, the networks, and their interconnection. The display part (WSF) of the WS-OS was implemented through presentation graphical objects supported by the Tk widget set. The rest of the WS-OS that accesses management information in other OSs (i.e., route design, configuration management) was implemented entirely in Tcl-RMIB.

In another RACE II project, PRE-Pilot in Advanced Resource Management (PREPARE), the Tcl-RMIB was used for the development of flexible script modules to support interdomain service management operations (i.e., resource reservation/allocation, fault handling, and various graphical user interfaces). These applications were written in Object Tcl over Tcl-RMIB, and a purely object-oriented approach was followed for their design and implementation. Also, in both ICM and PREPARE, Tcl-RMIB scripts were used for testing purposes (e.g., testing the functionality of Q3 interfaces).

Summary and Conclusion

This article has looked at the applicability of interpreted scripting languages with CMIS/Q capabilities and built-in graphical support in a TMN environment. We have found such facilities extremely helpful in building WS applications based on the presented model in which the internal f reference point is mapped onto external Q3 interface interactions. Although we recognize the need for such f reference points within graphical applications, we believe that interoperability should be based on the Q3 interface. As such, we propose that the F interface is not standardized by the ITU-T.

We have based our scripting language on Tcl because it was the first interpreted language to integrate easily with C/C++ and to provide the easy-to-use Tk widget set for graphical applications. In addition, object-oriented Tcl extensions such as Object Tcl provide classes, inheritance, and polymorphism, which are important for structuring more complex applications. Although there is not yet industrial strength behind Tcl, we believe that object-oriented interpreted languages will become popular in the future; Java is an example. The same principles may be applied to similar languages. Such technologies could be used in parallel with "tried and tested" industrial approaches for TMN WS-OS applications, such as UNIX Motif/C++, Windows NT Visual C++, and so on. It should be added that the NM Forum has also initiated an activity on a CMIS High-level Scripting Language (CMIS-HSL), which underlines the importance of the approach.

Motivated also by the need to run such applications on inexpensive desktop and laptop computers and to increase the portability of the relevant scripts, we specified and implemented a lightweight CMIP protocol. This operates directly over a connection-oriented reliable transport service and uses well-defined string representations for attribute, action, and notification values. Such a protocol may also be used by simple network elements (e.g., in mobile environments). We experienced important reduction in size and better performance for applications using the lightweight CMIP, which also resulted in highly portable interpreted applications across different hardware and operating system platforms. We intend to present in the future a detailed justification for lightweight CMIP as a potential candidate for standardization, together with an analysis of the relevant strengths and weaknesses.

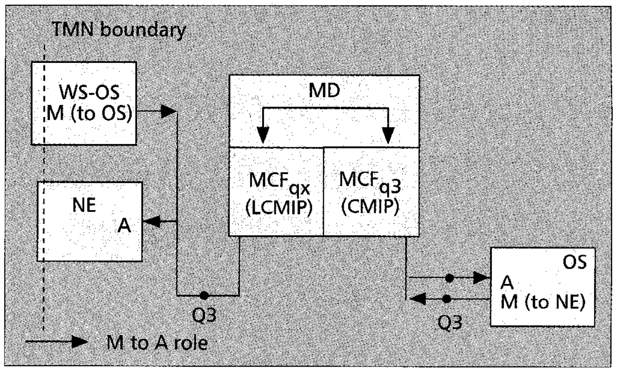


Figure 3.

Acknowledgments

This article describes work undertaken in the context of the RACE II ICM and ACTS Management of Integrated SDH and ATM (MISA) projects, partially funded by the Commission of the European Union.

References

[1] G. Pavlou *et al.*, "The OSIMIS Platform: Making OSI Management Simple," *Integrated Network Management IV*, Sethi *et al.*, eds., Chapman and Hall, 1995, pp. 480–93.

[2] ITU-T Rec. M.3010, "Principles for a Telecommunications Management Network," 1996.

[3] G. Pavlou *et al.*, "High-Level Access APIs in the OSIMIS TMN Platform: Harnessing and Hiding," in *Towards a Pan-European Telecommunication Service Infrastructure – IS&N '94*, Kugler *et al.*, eds., Springer-Verlag, 1994, pp. 181–91.

[4] NMF, "Red, Green and Blue High-Level CMIP-APIs," 1995.

[5] Ousterhout, *The Tcl Language and the Tk Toolkit*, Reading, MA: Addison-Wesley, 1994.

[6] Tin *et al.*, "Tcl-MCMIS: Interpreted Management Access Facilities," *Proc. IFIP/IEEE Dist. Sys.: Operations and Management Workshop*, Ottawa, Canada, Oct. 1995

[7] G. Pavlou, "LCMIP: A Lightweight Protocol for Data and Telecommunication Network Management and Control," UCL Comp. Sci. Res.Note RN/95/61.

Biographies

George Pavlou is a senior research fellow in the Department of Computer Science, University College London, responsible for European-funded research projects in the area of network and service management for high-speed broadband networks. Part of his research has culminated in the OSIMIS TMN platform. His main research interests include distributed management in the TMN and future integrated service engineering contexts. He obtained a Diploma in electrical and mechanical engineering from the National Technical University of Athens in 1982 and an M.Sc in computer science from University College London in 1986. He expects to finalize his Ph.D. thesis in 1996 at University College London.

Thurain Tin is a research fellow in the Department of Computer Science, University College London, where he has been involved in the area of network and distributed systems management for European-funded research projects. He has participated in the RACE II ICM project and contributed towards the OSIMIS TMN platform. He is currently working in the ACTS VITAL (Validation of Integrated Telecommunication Architectures for the Long Term) project, which involves the investigation and validation of the TINA and CORBA architectures for telecommunications management. He received a B.Sc in computer science from Queen Mary and Westfield College, University of London, in 1990.