A *Scheme*-based Toolkit for the Fast Prototyping of TMN-systems.

Sandro Mazziotta & **Dominique Sidou**
2229 route des crêtes, B.P. 193,
06904 SOPHIA ANTIPOLIS CEDEX, France.
email: {mazziott | sidou}@eurecom.fr

August 30, 1996

Abstract

This paper presents a TMN toolkit, its constituents and how they are integrated thanks to a *Scheme* based programming environment. This forms an effective basis intended mostly towards the prototyping of TMN-systems.

1 Introduction

Context : The toolkit presented here was originally built in the context of the TIMS[1] project. The objective of TIMS is to provide a behavior formalization and simulation environment for TMN-systems. This issues are described in [13]. The purpose of the present paper is to deal with the supporting environment of the simulator : the TMN toolkit. It is implemented in *Scheme*, which procures a simple and sound programming language support and an interactive environment ideal for prototyping. Moreover, it is a generic tool that can be used to prototype both TMN agents, managers or dual entities, in order to build entire TMN architectures. Such architectures can be composed of TMN boxes implemented with the TMN toolkit and of any other TMN box (e.g. a real agent or manager), the interactions between the TMN boxes being carried by CMIP.

Functionalities of the TMN toolkit : In terms of CMIS-API, both manager and agent sides are covered. Multiple associations can be established (both incoming and outgoing). Invocation of CMIS services can be handled either synchronously or asynchronously. The integration of any TMN Information Model (TMN-IM) GDMO and ASN.1 specifications is also supported. To implement manager interfaces (e.g. Alarm browsers), the Tk [10] toolkit is available which provides for a powerful GUI environment. Support for UNIX system and network programming is also a feature which makes the TMN box open towards other communications with its environment through usual UNIX inter-process mechanisms (pipes, sockets...). Finally, because of the variety of event sources potentially incoming to a TMN box (just think about multiple associations with other agents and managers, a GUI and communications with other processes via pipes) a coordination facility is needed in order to properly dispatch incoming events to the right handlers.

Approach : The approach taken to build such a complex system can only be based on reuse and integration of existing and available software components. Therefore, part of the OSIMIS [11] libraries have been integrated for the CMIS/P issues. And because OSIMIS is built on top ISODE [12], the ISODE OSI protocol stack is also linked to the system. In addition, for an effective ASN.1 value notation support, the Free Value Tool (FVT) [1] library has been integrated.

A prerequisite for such an approach to be feasible is to rely on a flexible and extensible kernel. A *Scheme*-based programming environment has been chosen : the Guile system [9]. *Scheme* is a simple and clean language, but still powerful. It also enables interpretation and thus interactiveness. Moreover, the

[1]This work was done in the context of the TIMS project. TIMS stands for TMN-based Information Model Simulator. This project is a collaboration between Eurécom Institute and Swiss Telecom PTT. It is supported by Swiss Telecom PTT, project F&E-288.

Guile *Scheme* system is extensible and enables the integration of any C library. The Guile distribution and contributions already integrate the Tk library (for GUI support) and a significant part of UNIX system and network programming functions.

Organization of the paper

1. Section 2 presents the features of the Guile *Scheme* system.
2. Section 3 deals with the provision for the CMIS-API.
3. Section 4 is about the ASN.1 support.
4. Section 5 focuses on the integration of GDMO specifications.
5. Section 6 presents the support for the prototyping of agent systems.
6. Section 7 describes how the coordination support is realized.
7. Section 8 gives an overview on how the TMN toolkit has been used to support the TIMS behavior simulation environment.
8. Section 9 illustrates some possible uses of the CMIS management API provided through a simple management script.
9. Section 10 is the conclusion.

2 Scheme Environment

2.1 Original Motivation

In the TIMS project, some of the formalization aspects rely on usual programming language features such as loops, conditionals, variable notation and operators.... To make such features available with minimal effort, a natural design decision was to simply reuse them from some existing small and simple programming language.

Because *Scheme* is a small, simple and exceptionally clean language, it is a very sound choice in regards to this original requirement. Moreover, *Scheme* is a standardized language, the language standard [2] is only about 50 pages[2]. *Scheme* is based on a formal model (the λ-calculus), so there are plenty of nice properties for the theoreticians. In addition, features, not present in any usual programming language are available, such as garbage collection, syntactic language extensibility thanks to high level macros and the very powerful concept of continuations[3]. Finally, *Scheme* has (i) lexical scoping to enforce clean and clear programming, (ii) uniform evaluation rules to avoid confusion, (iii) uniform treatment of data types to avoid the burden of strong typing[4], and (iv) closures to provide clean encapsulation for functions along with their definition environment.

2.2 The Guile *Scheme* System

The objective of Guile is to be the GNU project's extension language library. In the context of the TMN toolkit, Guile is used only for *Scheming*[5] , but the objective within GNU is in fact more ambitious. The goal is to build a library (Libguile) providing support for multiple, integrated extension languages sharing a common object system, calling conventions, and libraries of extension code. The library is organized around a flexible implementation of *Scheme*. It is designed to mix unobtrusively with ordinary C programs. The underlying assumption justifying the effort made in the Guile project is that it is better to optimize and port one multi-lingual interpreter than several monolingual interpreters.

2.3 Interfacing C library functions

Programmers using Libguile can define new built-in procedures for their application. Built-in procedures are defined as ordinary C functions, taking arguments and returning a result of type SCM (the C type used to store any *Scheme* value). These functions are declared to the interpreter at any time (usually during program initialization). As shown in the following sections, extensive use of this facility is made to reuse and integrate the required TMN functionalities to provide a *Scheme*-based TMN toolkit.

[2] This includes a formal, denotational definition of the semantics. So, only about 20 pages are actually needed.

[3] Continuations enable to program very complex forms of backtracking that may be required to do formal reasoning such as reachability analysis. Continuations are introduced in *Scheme* thanks to a unique and very simple construct : `call-with-current-continuation`, often abbreviated as `call / cc`.

[4] Though, this can also be considered as an inconvenience or limitation.

[5] *Scheme code organized as closure*

2.4 Tcl/Tk Support

Guile has been closely integrated with Tcl/Tk [Ousterhout] so that Tcl and Tk modules can be used by Guile programs. On the other side a Tk-like GUI can be developed in pure *Scheme*. A *Scheme* closure can, for instance, define a Tk event handler associated to the click of a mouse button.

2.5 Guile and the Other Extension Languages

Many other interpreted scripting languages have been proposed as extension languages. Without being exhaustive, Tcl/Tk [10], Perl [16] and Python [15] are relevant members of this family. Some of them were originally designed for a specific purpose: Tcl/Tk for GUI, Perl for system / network programming. Python and Guile seem to have a general purpose character from the beginning. Now, each one can be considered as a general purpose scripting language fully extensible through the integration of C and even C++ libraries. They all feature module systems, packaging facilities, object systems providing the required software engineering concepts to organize the development of really complex software. Typical libraries for system and network programming are also included. Finally, they offer development environments including debugging support. Therefore, what can be achieved with such languages goes far beyond simple and unmaintainable scripts quickly hacked to solve a one day problem. Though they can still be used for such tasks, they are no more restricted to mere scripting tools.

One important distinction between Guile and the other languages is that Guile is based on a standard [2]. The other ones have been defined and are supported by a single individual or by a very restricted and closed group. The leaders are John Ousterhout (Tcl/Tk), Larry Wall (Perl), Guido van Rossum (Python).

In our opinion, the *Scheme* approach has resulted in a better design of the language itself, from which many implementations are available. As soon as application code relies only on the R4RS standard and eventually on the portable *Scheme* library [7][6], applications can easily be ported from one implementation /platform to another with reduced effort. Integrated applications involving system / network programming and GUI support, things are much more complex. The reason is that such facilities are implemented / integrated in different and non-portable ways within each implementation of *Scheme*. In contrast, this problem is avoided with Tcl/Tk, Perl or Python, where there is only one implementation portable across all supported platforms. There is up to now no general solution to this problem in the *Scheme* community. One has simply to make a right choice according to the needs.

3 CMIS Support

The MSAP (Management Service Access Point) library and API provide in the OSIMIS platform for Dynamic Invocation / Server Interfaces (DII/DSI[7]) for all the CMIS service primitives. MSAP is a C library (`libmsap.a`) implemented on top of the ISODE OSI stack, which is itself another C library (`libisode.a`), including both ROSE and ACSE Application Service Elements (ASEs) required for CMISE. In addition, ISODE provides for the ASN.1 support needed to marshall / unmarshall CMIP-PDUs. MSAP is thus a basis for the realisation of any TMN system : managers, agents or dual role manager / agent boxes.

The purpose of this section is to describe the GMSAP module, that is how the MSAP library function has been interfaced into Guile. When interfacing an API across languages, the difficulty is to define the proper data type representations so that arguments and results for function calls can be passed in a straightforward fashion in both directions. Without any loss of generality, a choice was made from the beginning to represent CMIS primitives as uniform message records (Managed Object Messages, MOMSGs for short). Therefore, the MSAP interfacing functions need only one argument to convey CMIS calls. In *Scheme* many choices are possible to define MOMSGs. From the more elaborated ways, e.g. thanks to a full fledged object system, to the more simple ways, e.g. as flat lists. Elaborated representations allow for neat *Scheming* APIs, but are difficult to support in the interfacing C code[8]. For

[6] The SLIB is a very meaningful and general purpose set of *Scheme* functions easily portable on any R4RS compliant *Scheme* implementation.

[7] The concept of DII and DSI is borrowed from the OMG CORBA framework. CMIS [4] defines in fact only dynamic interfaces, in the sense that the same set of primitives supports all MO interactions, whatever the targeted object, operation and parameters are. In contrast, static interfaces enable invocations and receipt via dedicated and previously generated stubs. Though static interfaces are easier to use, they are obviously not general.

[8] Unless native support for objects is available, but this is not a general assumption one can rely upon.

a low level representation, the situation is exactly the reverse : the interfacing code is straightforward but the resulting *Scheme* API is less user friendly and error prone (e.g. the number and position of arguments in a flat list has to be respected).

SCM <==> C Interfacing with `recvecs` Some compromise was found by using a record module (part of SLIB, the *Scheme* Library), that is implemented with usual *Scheme* vectors (recvec). Naturally vectors (because they are standard *Scheme* data structures) are natively supported. This allow for easy and efficient manipulation in the interfacing C code. In *Scheme*, `recvecs` give a satisfactory representation for CMIS messages (`momsgs`).

To ease the manipulation of `momsgs` in the glue C code, a macro was implemented to define CMIS `momsgs` that generates as a side effect the necessary constants (as C `#define` constructs) so that an safe mapping from raw vectors indices to message record fields is available[9].

Related Work : Because this task results also in a CMIS API in *Scheme*, it may be positioned (without any pretension) with respect to other work in this area. XOM/XMP [17] was a first attempt in this direction. This joint specification from the NMF and X/Open was part of OmniPoint1. XOM/XMP has been recognized as too tedious and too low level API to be actually usable. That is the reason why the NMF has launched a task force to provide a more suitable C++ CMIS API for OmniPoint2. A proposal [3] has been submitted for public review. The interface described above may provide something similar in *Scheme* or *Lisp*. Our short term goal, however, is limited to the provision for a usable and working CMIS-API in *Scheme* that covers both manager and agent interfaces.

The manager oriented primitives of the GMSAP module can be compared to Tcl-MCMIS [14], aiming at interfacing manager side MSAP primitives in Tcl. Tcl-MCMIS was realized in the context of the OSIMIS platform itself to ease the development of managers (agent side is not covered) thanks to the Tcl/Tk toolkit. Though the approaches are comparable, the *Scheme* language support is much more powerful than Tcl in particular in regard to data types. Another problem with Tcl-MCMIS is the support for ASN.1 which gives a limited value notation support. To improve this support a dedicated ASN.1 library including the standard ASN.1 value notation has been integrated. Note that the same approach could be applied in the context of Tcl-MCMIS. More details on ASN.1 support are given in section 4.

4 ASN.1 Support

The main purpose of this section is to explain how full ASN.1 support, in particular with respect to the value notation, was incorporated in the TMN toolkit. In effect, because GDMO specifications can potentially reference any ASN.1 abstract syntax, a general mechanism is necessary :

1. to manipulate corresponding values in *Scheme*, i.e a manager may require to create such ASN.1 values from an user interface (API).
2. then, the transfer of such ASN.1 values between *Scheme* and the protocol stack have to be done in some way.

[9] This automatically generated mapping has been very useful in the development phase, where `momsgs` were still subject to changes. A hard coded mapping would have resulted as tedious index maintenance for each modification. Though `recvecs` with automatic generation of mappings does not intend to be a general way to interface complex types between *Scheme* and C, it still seems to be usable in many circumstances.

In terms of ASN.1 support, the OSIMIS platform relies on the ISODE ASN.1 compiler (`pepsy`). `Pepsy` is not complete in regards to a value notation. Support is present for BER encoding / decoding and for printing[a] values but no read function is provided. That is the reason why the OSIMIS system has included, in an add-hoc fashion, read and print routines to and from a non-standardized (but usable) string notation. However, this support is limited to the types from CMIP and the DMI [6] used in the OSIMIS platform. Therefore, in order to provide a suitable ASN.1 value notation support, an ASN.1 C library has been integrated into the Guile *Scheme* system. The library used is the Free Value Tool library [1], which provides a standardized string based notation for ASN.1 values. This representation, is suitable to be used as a machine readable notation. In addition this notation can be converted to *Scheme* and conversely[b]. Such a *Scheme* representation is very useful, because it allows a complete manipulation of ASN.1 value in terms of usual *Scheme* operators on lists. A more ASN.1-ish API (ASN.1 value operators) can even be implemented on top of the basic *Scheme* list functions. On the other side the Free Value Tool provides also for BER encoding and decoding from / to the string based notation, to send / receive ASN.1 values to / from the CMIS API. Finally, Generic ASN.1 operators were implemented to build, access, and update any ASN.1 value in their *Scheme* representation. Figure 1 summarizes how ASN.1 values are conveyed from *Scheme* to the wire, and conversely.

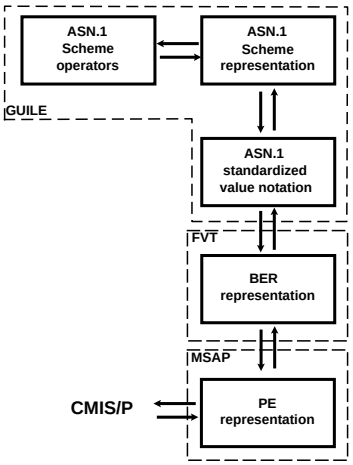

Figure 1 : ASN.1 value processing.

[a] Pepsy print routines give a user readable format, and this format would not be usable as an input format.

[b] A simple technique consists mostly to change curly braces ("{" and "}") into oval braces ("(" and ")"), and to remove comas ",", and then to apply a normal *Scheme* reader function on the resulting string. A reverse transformation is also quite easy to implement.

Note that FVT although is a C library, its required functions were interfaced in the *Scheme* environment thanks to the Guile extension facilities mentioned in section 2.

5 GDMO Support

GDMO support is based on the OSIMIS GDMO compiler. This compiler was designed in a nice way allowing one to manipulate a symbol table resulting from a flex / byacc parser by means of a very simple interpreter. This back-end interpreter which usually generates code for the automatic creation of OSIMIS agents was customized to generate a *Scheme* file. Then, this file can be loaded to fill in a Managed Object Repository (MOR). The *Scheme* environment of the TMN toolkit thus gains access to the information from the GDMO specification, through a simple and well defined API.

6 Agent Support

The following agent functions are supported on top of the DSI (Dynamic Server Interface) provided by the GMSAP module presented in section 3 :

- MIB support realized in the Managed Objet Instances Repository (MOIR) module. The MOIR is implemented as a hash-table[10] indexed by MOI names (*Scheme* representation for `DistinguishedNames`) and stores for each MOI all of its properties (attribute values...).
- CMIS scoping and filtering which provides for the multiple object selection support.

[10] Hash-table are part of SLIB, the *Scheme* Library.

- Static and dynamic checking of messages. Static checking ensures that CMIS messages are correct with respect to the TMN Information Model itself, i.e. the GDMO specification. Dynamic checking ensures correctness with respect to the dynamic execution context of the CMIS messages, i.e. the current state of the MIB.
- Finally, message processing and the composition of responses (including multiple responses) are handled according to the semantics of CMIS operations.

Note that no particular support is given for interfacing real resources. This could be added thanks to the coordination mechanism presented in section 7, but this has not received any particular attention until now. The reason is that the objective is not to build some kind of agent toolkit. Also, no support for Systems Management Functions (SMFs) is provided, however, the behavior formalization environment could be used to prototype SMFs.

7 Coordination Facility

The coordination facility is required to dispatch the events coming from the various input sources that can be linked to a TMN box built on top of the TMN toolkit. For the time being, this facility is supported by the already existing event handling mechanisms provided by the Tk toolkit library integrated within the Guile *Scheme* system. The Tk `fileevent` command is used to register the CMIS processing handler function to each UNIX socket descriptor opened for a management association. Naturally, X11 Window system handlers for each opened X11 display are registered by default by the Tk event handling facility. Additionally, other handlers can be registered for other event sources, e.g. files, pipes.... Thus, here again nothing was actually reimplemented, the existing facilities provided by the Guile/Tk support is reused to provide a satisfactory coordination mechanism.

Though there is no a priori defined limitation upon the number and variety of input sources that can be dispatched by the coordinator, the incoming events are handled in a simple first come first served basis. Therefore, there is no actual support of asynchronous processing of incoming events. This can reveal as a limitation if one needs, for instance, to prototype an agent system that requires asynchronous processing – Just think about `M_CancelGet` CMIS service, which should be able to stop a currently processed `M_Get` –. There is also no underlying support for multi-threading such that several requests could be served concurrently. Because multi-threading may come in a future release of Guile, it will be more relevant to think about the provision for asynchronous processing at that time.

8 Using the Toolkit in TIMS

The way in which the TMN toolkit has been used to support the TIMS behavior simulation environment is shown in figure 2.

Integration of the TMN-system's Specifications The bottom of figure 2 concerns the integration of the different specification files. GDMO and ASN.1 specifications are processed as described in sections 5 and 4. A General Relationship Model (GRM) [8] parser enables for the integration of relationship specifications. Its back-end phase generates *Scheme* code in order to fill in the RElationship Repository (RER). In TIMS, an extensive use of relationships is made for the purpose of behavior formalization. Finally, a behavior specification is simply integrated into the TMN box by loading one or more behavior specification files as usual *Scheme* files. In effect, the behavior language is only a set of *Scheme* language extensions supported by a high level macro module provided in the *Scheme* Library [7].

Management Communications : The left hand side of figure 2 concerns the OSI Management communication facilities, that is the OSIMIS and ISODE libraries integrated as described in section 3. This enables the simulator to interact with any other TMN component through CMIP. This can be a real manager within a management platform, a real agent, or any other TMN box.

Behavior Simulation Environment : The top of figure 2 describes the facilities offered by the TIMS behavior simulation environment. A global object view of the simulated TMN system provides a nice visualization of the evolutions of the system. The object view is the graph of Managed Objects along with their relationships. Because, such graphs can become big and complex, finding a proper layout is very far from trivial. That is the reason why a dedicated graph visualization tool, the *daVinci* system [5],

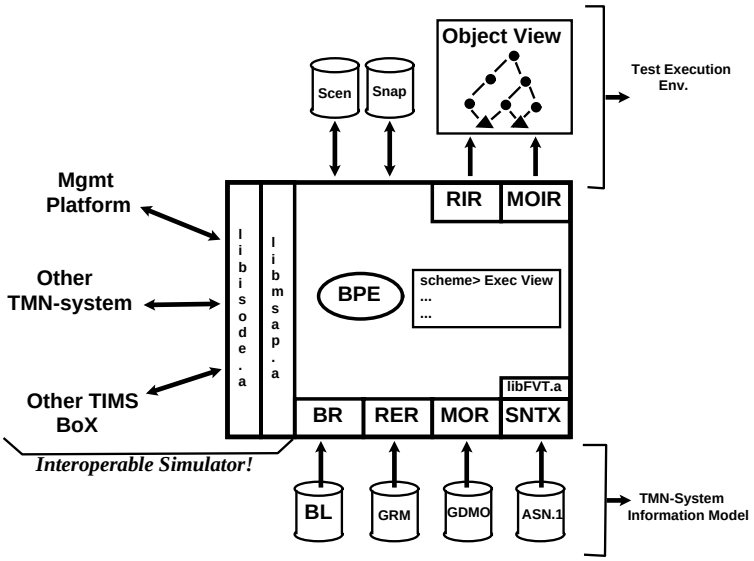

Figure 2: TIMS Environment.

is used for this purpose. In addition to the layout of a graph *daVinci* provides nice animation mechanisms such as scaling, hiding subgraphs, changing colors, icons for nodes, multi-views.... Moreover, this tool can be simply integrated as a separate UNIX process interacting via pipes according to a string graph manipulation API. In addition, a Tk-based GUI have been implemented to provide, easy to use, scenario player and a snapshot facility. The scenario player enables to submit test cases to the simulator. The snapshot facility is very useful to save a given state of the TMN-system (e.g. obtained as a result of a long behavior simulation). This snapshot then allows for fast loading the number of times required, so that any other relevant scenarios can be easily played from this "snapshoted" state of the system.

9 Example of Management Scripts

1. Between two nodes belonging to a network, a service should be created. The manager verifies that there is sufficient resources available at these nodes. That means that the manager verifies that there is enough "networkCTPs" having their assignmentState "free".

```
(for-each
 (lambda (node-elem)
   (display
    (xbif:mget
     *msd*
     node-elem
     'wholesubtree
     (filter:equal  "objectClass" "networkCTP")
     "assignmentState")))
 (list '(("systemId" "localhost") ("adminDomainId" 1) ("nodeId" 1))
       '(("systemId" "localhost") ("adminDomainId" 1) ("nodeId" 2))))
```

2. After the previous verification, the manager creates the service.

```
(xbif:mcreate!
 *msd*
 "serviceResource"
 '(("systemId" "localhost")("adminDomainId" 1))
 '("serviceId" 1)
 '("aNode" ,(moi:scm->asn
             '(("systemId" "localhost") ("adminDomainId" 1) ("nodeId" 1))))
 '("zNode" ,(moi:scm->asn
             '(("systemId" "localhost") ("adminDomainId" 1) ("nodeId" 2))))
 '("serviceSignalId" 2)
 '("supportedByObjectList" ())
 '("lifeCycleState" planned)
 '("administrativeState" unlocked)
 '("operationalState" enabled)
 '("assignmentState" free))
```

3. Then, the manager activates the service. This corresponds to put the assignmentState in an "assigned" state. This causes (by behavior propagation) to put the selected resources (networkCTPs) to be also

```
(xbif:mset!
 *msd*
 '(("systemId" "localhost")("adminDomainId" 1)("serviceId" 1))
 'confirmed
 *unspecified*
 *unspecified*
 '("lifeCycleState" inService replace))
```

4. Finally, the manager verifies that the activation of the service was done correctly. "networkCTPs" participating in the service should be now "assigned".

```
(for-each
 (lambda (node-elem)
   (display
    (xbif:mget
     *msd*
     node-elem
     'wholesubtree
     (filter:equal  "objectClass" "networkCTP")
     "assignmentState")))
 (list '(("systemId" "localhost") ("adminDomainId" 1) ("nodeId" 1))
       '(("systemId" "localhost") ("adminDomainId" 1) ("nodeId" 2))))
```

10 Conclusion

In this paper, it has been described how a full fledged and easy to use TMN toolkit has been built. This construction was performed with a minimized effort, by integrating all the required basic building blocks into a powerful and extensible kernel. The result is a generic TMN toolkit enabling to prototype of TMN-systems, i.e. agents, managers, or dual role entities. The kernel is based on the Guile *Scheme* system. *Scheme* provides for a simple and sound programming language. Guile is fully extensible thanks to a well defined foreign function interface for C library functions. The Guile distribution comes with Tcl/Tk (allowing GUI support) and UNIX system and network programming library functions already interfaced. This allows for the provision of complex, but still easy to use and interactive programming environments. Interactiveness is allowed because *Scheme* code can be interpreted, this avoids the burden to compile, link that would be required in C/C++ environments. Obviously, the realization of the presented TMN tool-set would have been impossible without the availability of the different software packages and libraries used and integrated.

Further Issues : It would be interesting to see how, for instance, policy interpreters could be built on top of the TMN toolkit. Because in *Scheme* as in any other *Lisp* like programming language, code and data have a uniform representation, policy interpretation code could be easily exchanged between interpreters (just like people are currently envisioning it with Java).

References

[1] ASN.1 Free Value Tool. Available at ftp://osi.ncsl.nist.gov/pub/osikit/.

[2] W. Clinger and J. Rees. $Revised^4$ Report on the Algorithmic Language Scheme. *ACM Lisp Pointers*, 4(3), 1991. Available at http://www.cs.indiana.edu/scheme-repository/doc/standards/r4rs.ps.gz.

[3] High Level CMIP API (In development), 1996. Available at http://www.osf.org/~zpope/CMISAPI/.

[4] Management Information Service Definition - Common Management Information Service Definition, ISO/IEC 9595, ITU X.710.

[5] The Interactive Graph Visualization System daVinci. Available at http://www.informatik.uni-bremen.de/~inform/forschung/daVinci/daVinci.html.

[6] Structure of Management Information - Part 2: Definitions of Management Information, ISO/IEC 10165-2, ITU X.721.

[7] T.R. Eigenschink, D. Love, and A. Jaffer. SLIB: The Portable Scheme Library, 1994.

[8] ISO/IEC JTC 1/SC 21, ITU X.725 – Information Technology – Open System Interconnection – Data Management and Open Distributed Processing – Structure of Management Information – Part 7 : General Relationship Model.

[9] Thomas Lord. The Guile Architecture for Ubiquitous Computing. to appear in the proceedings of the Usenix Tcl / Tk Workshop'95, 1995. Available at http://www.cygnus.com/library/ctr/guile.html.

[10] John Ousterhout. Tcl and the Tk Toolkit, 1994.

[11] G. Pavlou, K. McCarthy, S. Bhatti, G. Knight, and Simon Walton. The OSIMIS Platform : Making OSI Management Simple. In Chapman & Hall, editor, *Integrated Network Management IV*, pages 480–493, 1995.

[12] M. T. Rose, J.P. Onions, and C.J. Robins. The ISO development environment: User's manual, version 8.0. Technical report, PSI, July 1991. Available at ftp://ftp.psi.net.

[13] Dominique Sidou, Sandro Mazziotta, and Rolf Eberhardt. TIMS : a TMN-based Information Model Simulator, Principles and Application to a Simple Case Study. In *Sixth International Workshop on Distributed Systems : Operations & Management*, Ottawa - Canada, 1995. IFIP / IEEE. Available at http://www.eurecom.fr/~tims/papers/dsom95-paper.ps.gz.

[14] Thurain Tin, George Pavlou, and Ron Shi. Tcl-MCMIS : CMIS Manager Extension to Tcl. In H.-G. Hegering and Y. Yemini, editors, *Sixth International Workshop on Distributed Systems : Operations & Management*, pages 181–192. ©IFIP / IEEE, 1995.

[15] Guido van Rossum. The Python Language, 1996. Python home page is at http://www.python.org/.

[16] Larry Wall. The Perl Language, 1996. Perl home page is at http://www.metronet.com/1h/perlinfo/perl5.html.

[17] X/Open : OSI Abstract Data Manipulation and Management Protocols Specification, 1992.