Webbin': A New Way To Manage Networks

Luca Deri
IBM Zurich Research Laboratory[1], University of Berne[2]

Webbin' is a research project at the IBM Zurich Research Laboratory aimed to simplify network management. I've designed Webbin in late 1994 and started to code it in early 1995 while I was developing the communication infrastructure of an IBM product for OSI network management. At that time I was very busy testing the infrastructure and I needed a simple tool able to locate and browse OSI agents running on machines attached on the network. At that time I realised that management tools were quite difficult to use and required some expertise in order to install and configure them. The thing that amazed me most was that people didn't complain too much about these tools since they were used to such complexity. In some ways my past programming experience with Macintosh made me more sensitive to usability and configuration issues. In my understanding a tool which needs a stack of manuals in order to be used is basically useless. Some people explained that network management is per se complicated hence tools must be complex. I couldn't believe accept this thesis, and then I started to think about a new way to manage networks. In late '94 the Web was a very promising technology but was not very spreaded yet. Nevertheless its incredible growth and the extreme simplicity of this technology pushed me towards it. Therefore I installed the NCSA HTTP server and I started to write some simple CGI applications just to learn this technology better and to understand its limitations. Seen that the performance was acceptable and that it was extremely simple to turn plain text in HTML I coded a couple of CGI applications which were able to visualise some OSI resources I was interested in for my tests. The result was very encouraging and then I decided to extend to make them more general and detached from the test environment I used in Zurich. The basic requirements were the following:

- the system has to drive the user and it has to prevent him from performing wrong operations;
- dynamic resource discovery: the resources have to find me and not the other way round;
- the system must use concepts users are familiar with such as folders and files in order to make them feel comfortable and to prevent them from learning new, unneeded, concepts.

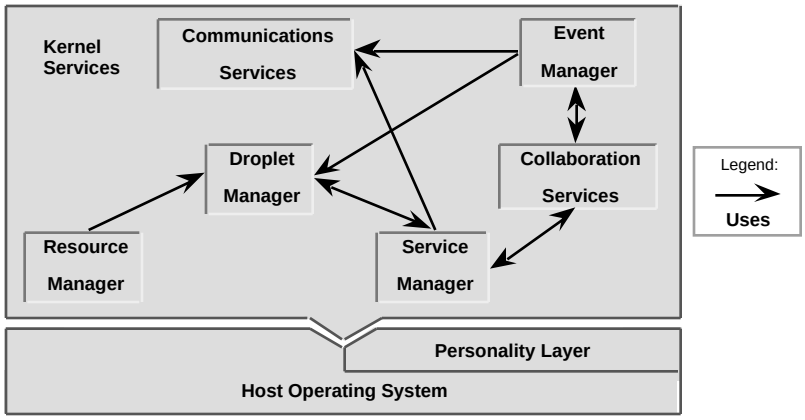
I have coded the first version on AIX, IBM's UNIX, and seen that these CGI applications were working well enough I have decided to show it around. I have selected the WWW conference in Darmstadt since my application was making extensive use of Web technologies and then such conference would have been a good place to be. Nevertheless, in order to show a demo there I would have needed an AIX box. Since I didn't have at that time a portable AIX box but just an IBM ThinkPad equipped with OS/2, I quickly ported the application to OS/2 and went to the conference where I showed it in the poster session of the conference. The name I chosen for the application was Webbin' CMIP which has now been turned into Webbin'. Encouraged by the positive feedback of people who looked at Webbin' I convinced my manager to leave me some spare time to keep working on it and due to this I have been able to turn that set of CGI applications in a platform for network management freely available for download which supports CMIP and SNMP and which runs on more that five platform, including Linux the biggest and more active Webbin' community by far.

Webbin' At A Glance

The core component of Webbin' is *Liaison* a proxy application which allows CMIP and SNMP resources to be accessed using the HTTP protocol, the one used by the Web to retrieve multimedia documents. Liaison has substituted the HTTP server and the CGI application used in the first prototype because the time and the resources necessary to start a CGI application prevent to have a high-performance application capable of handle many requests per second. Liaison has been designed to be portable and resource-savvy in order to overcome one problem very common on network management: the need to purchase specialised software/hardware necessary to run the management applications. In the case of Webbin' the perspective has been reverted: users should be able to run Webbin' independently from the operating system and from the computing resources they have. In order to make an efficient use of the computer resources and to enable scalability and tailoring, Liaison is built upon a compound architecture named *Yasmin*.

[1] IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland.
Email: lde@zurich.ibm.com, WWW: http://www.zurich.ibm.com/~lde/.
[2] Universität Bern, Institut für Informatik und Angewandte Mathematik, Software Composition Group, Neubrückstrasse 10, CH

Yasmin's core component are depicted below.

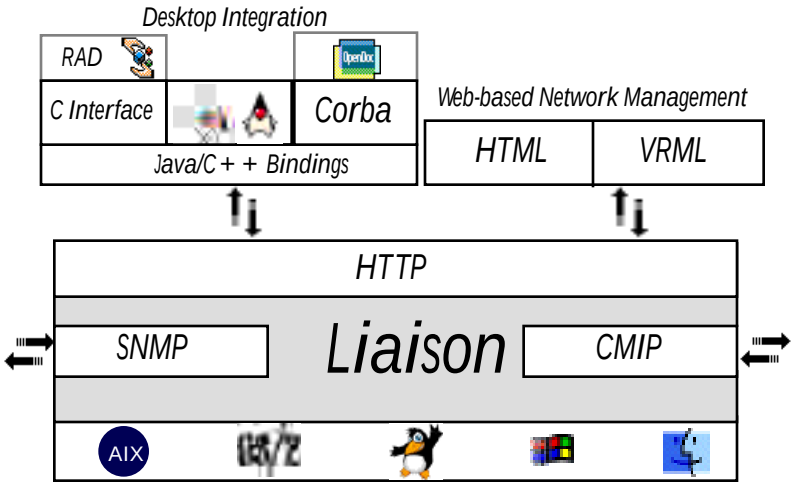


The personality layer allowed to keep the operating system-dependent code separated from the rest of the application hence to facilitate the porting to different platforms. Currently this layer contains facilities for loading/unloading shared libraries, thread management, synchronisation (semaphores) and other minor facilities. Thanks to the personality layers, the other components do not have to be modified when the application is ported to a different platform hence the code is simpler to maintain and extend. Another core component of Liaison is the droplet manager which is responsible for managing *droplets*, a new kind of software components implemented using shared libraries having the following properties:

- are not statically linked to the application but are loaded at runtime;
- have the ability to be replaced (i.e. a new version of the droplet can replace a previous one) at runtime while the application is running;
- have a well-defined interface that makes it possible to communicate with other droplets independently from the type of the services provided.

Since droplets can be replaced and added at runtime, an application which makes use of droplets (for instance Liaison) can be extended or patched dynamically without having to restart it. This facility is extremely important when the application has to provide some functionality which need to be available most of the time and also it allows to cleanly separate through the droplet interface services provided by different droplets hence to remove code cross-dependencies. The service manager handles the services provided by the droplets (for instance mapping of object identifiers from numeric to symbolic format) and it collaborates with the droplet manager every time services have to be added/replaced since the corresponding droplet has been added/modified. The other Yasmin components allow droplets to communicate, to exchange events, to share computing resources and to collaborate in order to provide a certain functionality.

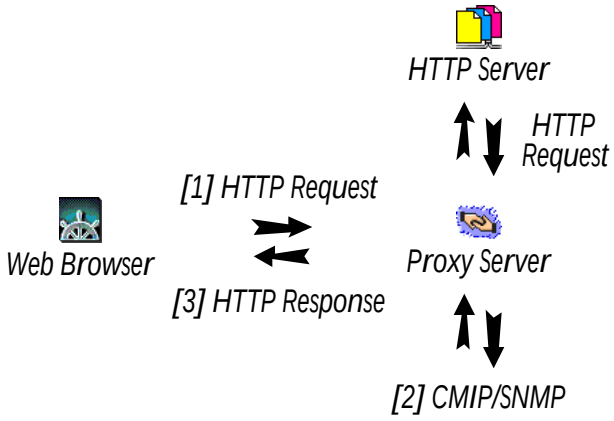
Liaison's kernel, named *Proxy*, is very simple and small since it implements Yasmin without including any management facilities which are implemented inside droplets. This design solution has been chosen since it allows to keep the Proxy very generic hence to reuse it on different contexts not necessarily related to network management and also to prevent users from installing and running code which they do not need (for instance droplets for OSI management). Additionally the smaller is the droplet granularity the better it is since it facilitates service reuse and it allows droplets complexity/size to be kept small. In fact the use of droplets is important not only at runtime but also at compile time. Since droplets are very small and use services provided by other droplets through the service manager and not directly, they do not include symbols/datatypes from other droplets hence it is extremely fast to compile and link droplets with major benefits during the development phase. Liaison comes with a rich set of droplets which range from basic network resource browsing until complex management using Corba shown in the following picture.



Liaison is basically an HTTP server which provides some services implemented inside droplets and accessible through HTTP. The droplets are divided in two sets according to the protocol they implement SNMP or CMIP[3] and they are further divided in two sets according to the type of services they provide, Web-based network management or support for application development.

Web-based Network Management

Management of network resources through the use of a Web browser has been the first functionality implemented and present already in first prototype. Basic management is performed using HTML whereas whenever it is necessary to combine a lot of information in one screen or when it is necessary to depict topological information VRML can be used instead. VRML (Virtual Reality Modeling Language) is a simple yet powerful language used to represent 3D information which is then rendered by a VRML viewer in a way similar to what happens to HTML with Web browsers. Liaison comes with droplets which allow to browse OSI and SNMP agents and to manipulate the management information from within the Web-browsers. Additionally it comes with two more droplets which allow to discover the OSI (only when the IBM OSI stack is used) and SNMP resources present on the network. Basically users connect their Web browser to Liaison and then browse network resources like if they would browse a set of HTML documents.



If Liaison can process the requests it does it, otherwise the request is forwarded to the local HTTP server, if any. Based on the requested URL, Liaison checks whether the request can be processed by a local droplet, if it relates to a file or if it cannot be processed at all. URLs are composed of five elements:
`http://<host>/<protocol>/<operation>/<context>?<parameters>`, where:

- `<host>` identifies the host where Liaison runs (Liaison's default port is 1998);
- `<protocol>` specifies the protocol used (either CMIP or SNMP);
- `<operation>` specifies the management operation (CREATE, GET, ...);
- `<context>` specifies the context used, if any;
- `<parameters>` contains the operation parameters.

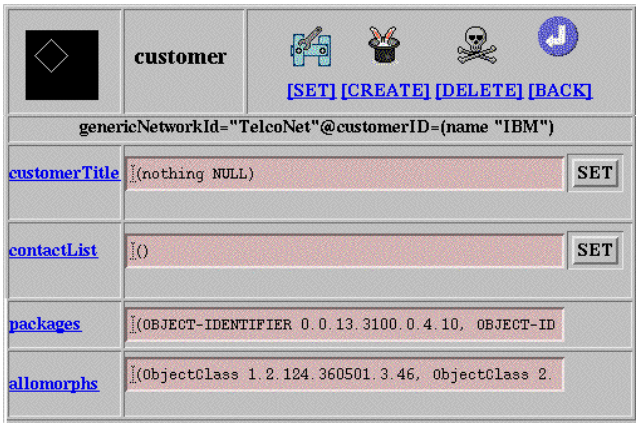
For instance, if `<protocol>` is set to CMIP, `<context>` contains the agent title and the managed object instance, whereas for SNMP `<context>` specifies the object identifier of the attribute. `<parameters>` contains operation-specific parameters

[3] At the moment CMIP is supported only by the AIX and OS/2 versions only since the IBM OSI stack is not currently supported

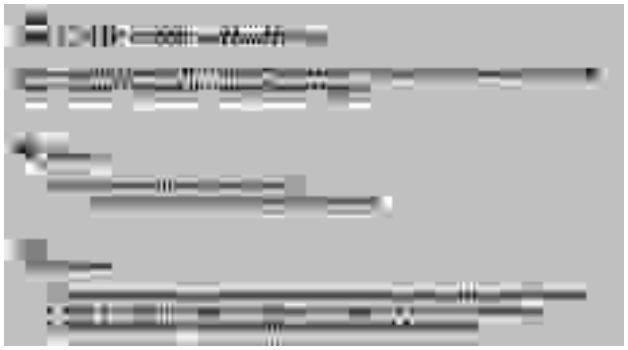
(e.g. for SNMP SET, `<parameters>` contains the MIB variable(s) to set and their new values) and other values such as timeout or name of the host on which the agent is running. Although this mapping is almost straightforward, Liaison hides it from the user. In fact, Liaison shows the user a starting point and then the user does not have to worry about the syntax because URLs are dynamically generated by the system. Supposing to run Liaison on the host kae, the starting point for SNMP is `http://kae:1998/SNMP/DISCOVERY/` or `http://kae:1998/SNMP/VRML_DISC/` depending if the output has to be respectively in HTML or VRML format, whereas for CMIP is `http://kae:1998/CMIP/DISCOVERY/`. In the case of SNMP Liaison discovers the SNMP agents running on the subnetworks specified on the configuration files whereas in the case of CMIP all the known OSI stacks are shown. The picture below shows how the VRML discovery looks like.

<PASTE HERE THE PICTURE OF SNMP VRML (USE THE ONE ON MISA)>

Notice that in the discovery configuration file it's specified the subnet type (ring, star or bus) which is then represented in VRML. Using these starting points is then possible to start the navigation and to manage the resources simply following the HTML links like if we were using static HTML files.



The picture above shows a the content of a CMIP instance of class customer. Notice that users can only perform the actions allowed by Liaison, this is in order to avoid errors (in the example, the attribute package cannot be set hence the SET button has not been displayed. Unfortunately Liaison cannot prevent all the errors but also in this case the Liaison helps the user by showing a (relatively) simple error message suggesting possible solutions to the problem as shown below.

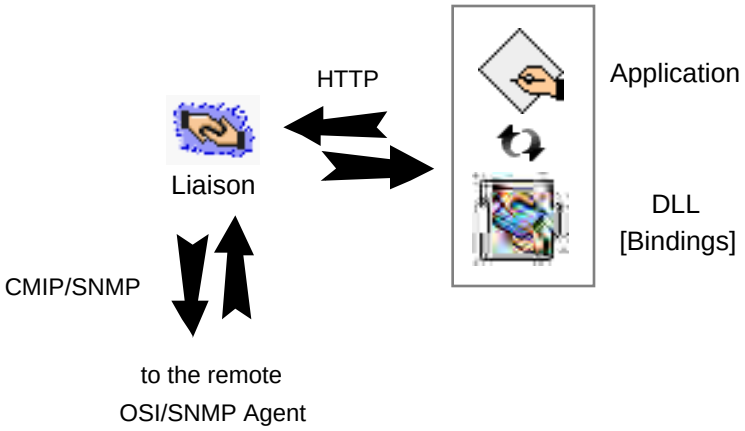


Through the Web, users have full control over the managed resources and can perform basically every operation. Nevertheless in some this is not enough. For instance in dynamic situations where the managed resources change state quite frequently or when users need more advanced or specialised tools it is necessary to create real applications the limits of HTML and VRML have been reached. In order to support application development, some external bindings have been created this is in order to allow users to create their own management application or their custom HTML/VRML pages tuned for the user's environment.

Network Management Application Development

Besides the droplets used to manage CMIP/SNMP resources using HTML/VRML, Liaison provides the *external bindings*, a set of C++/Java classes and C functions that communicate with further droplets by enabling programmers to develop decentralised management applications/applets based on the services provided by Liaison. An application based on the bindings, transparently issues HTTP requests to Liaison which maps them in management requests and then returns the

response(s) to the management application. The application deals only with classes/functions and all these communications are hidden by the bindings.

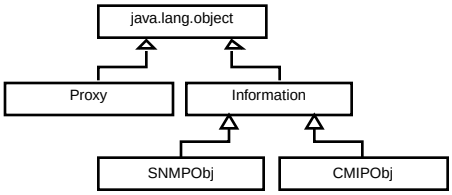


As these bindings are simple classes/functions which build URLs sent to Liaison and handle the responses, they are quite light (about 20 Kbytes in total). Hence they enable the creation of very light applications based on the services provided by a remote Liaison. Depending on the complexity of the final application and on the user requirements, three different solutions based on the external bindings are offered:

- HTML/VRML,
- Java,
- other scripting languages such as TCL, Python or Perl.

In the first case the application is composed of several HTML/VRML pages that allow people to manage network resources using a basic user interface. End-users interact with HTML elements such as buttons and menus, and an HTTP server application interprets the user commands which have been mapped transparently to URLs. In case a management operation has to be performed, developers can create simple C/C++ applications using the bindings or exploit the shell commands (snmpget, snmpset...) provided by Liaison which are then integrated inside CGI applications.
Java has the advantage that it is platform independent hence this solution has to be considered when portability is a must or when the management application has to be integrated in a HTML page and retrieved from remote. In this case the Java bindings are used.
TCL and other scripting languages allow average skilled programmers to quickly develop small yet powerful applications with a nice graphical user interface (for instance, if Tk is used). Since TCL allows external C functions to be called, in this case it is preferable to employ the C external bindings.

Once described the different techniques for application development, it is now time to see how to use the external bindings. In order to simplify the application and to hide differences between different datatype, external bindings are based on strings. Every type such as Integer and IPAddress is mapped to strings. Therefore the IP address 9.4.33.38 is mapped to “9.4.33.38” and not to a 4 byte long. The conversion from strings to real types is performed transparently by Liaison. In order to do this, Liaison uses some metadata which in the case of SNMP are data files which associate a type to each object identifier which identifies a MIB variable (for instance `sysDescr.0` has associated the `OctetString` type). The mapping to strings not only simplified the user application but also allowed almost every programming language to be used for network management since the string datatype should be supported by every language. Additionally this simplifies significantly the usage of the bindings from languages such as TCL, since there are no complicated struct to be passed to/from C/TCL but only strings.
The Java/C++ bindings are very similar. Owing to space constraints only the Java version is described; similar considerations can be done for the C++ version. The class hierarchy is quite simple.



The class `Proxy` is responsible for handing communications with the Proxy application. It transparently sends the requests and receives the responses. The class `Information` contains the information relative to the request and to the response(s),

Subclasses `SNMPObj` and `CMIPObj` implement some high level manipulation functions for manipulating the input/output information and invoking `Proxy` methods whenever a request has to be issued. These subclasses have been provided to further simplify the access to the `Information` and `Proxy` classes and have to be considered like pure facilities.
Requests can have single or multiple responses returned in case of a CMIP scoped requests or of a SNMP walk. When multiple responses are returned they are insert in a `java.util.Vector` that is returned as output parameter. In case of single response, the returned values replace the actual ones in the input `SNMPObj` or `CMIPObj` object. In this way the input object is transparently updated with the return values.
If a request fails for whatever reason an exception of class `ProxyException` is raised: users should not deal with protocol errors but they should interact with remote objects only using programming constructs. This is very important because programmers do not have to change their programming style using familiar concepts like exceptions. When an exception is raised, an error code is returned together with the receiver error response that does not affect the input object which remains unmodified. The `Information` class and its subclasses `SNMPObj` and `CMIPObj`, greatly simplifies and reduces the code users have to write:

- a `SNMPObj` or `CMIPObj` object represents a hook to an instance or attribute independently from the operation that will be issued: this allows to issue different operations using the same input object
- parameters such as scope, filter, sync (CMIP) or community (SNMP) are handled transparently: if not present or set to default they are not sent to the Proxy that will then use the defaults
- default values are expressed using empty ("") values instead of using special flags or data structures.

Additionally, this solution allows to save bandwidth because only the needed attributes are exchanged between the Proxy and the Java application and because unmodified attributes, for instance objectClass in a CMIP response, are not transmitted. Classes `SNMPObj` or `CMIPObj` other than issuing protocol requests, allow to retrieve metadata information and to convert object identifiers that can be expressed in both numeric or symbolic form.

Installation and Configuration

Since Webbin' has been designed to simplify network management, the installation and the configuration of it must be simple and strightforward. Basically once downladed Webbin, unzipped and untarred we're almost ready. It is only necessary to edit the file `discovery.cfg` where we have to specify the network segments (for instance 9.4.33.*) where are running the SNMP agents which we intend to discover. Done this, you can start Liaison (`./Liaison &`) and from your favourite web browser open the URL `http://<your host name:1998/SNMP/DISCOVERY/`. As you may have noticed Liaison is seen as an HTTPd running at the port 1998 (you can modify this value by setting the `BASE_PORT` environmental variable from your shell). From that URL you can start the exploration of your SNMP agents and you can also modify some MIB variables. If you like more VRML instead, you have to use as starting point `http://<your host name:1998/SNMP/VRML_DISC/` which returns a VRML 3D world instead of a simple HTML page. In the standard Liaison distribution you will also get the C/C++ external bindings (contained the the directory `ExternalBindings/`), whereas the Java bindings and the extaernal bindings documentation (in HTML) have to be downloaded separately. This separation has been done in order to simplify the packaging only. The bindings come with some examples which show how thy are supposed to be used. Moreover, Liaison comes with simple applications (snmpget for instance) usable from shell which allow them to be integrated in CGI applications or shell scripts. In one of the coming versions of Liaison (this should be available by the time you read this article) some CORBA bindings, for CMIP/SNMP management from CORBA will be released as well. Notice that since the IBM OSI stack is not available on Linux, it is not possible to use Liaison to manage OSI resources although it is possible to write applications based on the extarnal bindings which talk with a Liaison running on AIX for instance which has the support for OSI.

Final Remarks

If you have read until here, you should know how Webbin' works and what you can do with it. I don't believe in powerful management tools which do everything since if a tool is really generic then is a browser hence is not extremely flexible for every situations. The idea behind Webbin' is to give to people the chanche to manage their networks by either using the basic browsing facilities provided by Liaison or to write simple yet powerful applications for advanced management. The era in which "one management platform does everything" is about to end and will be replaced with one that enables developers to build needed management applications easily. This does not mean that large and powerful management platforms will disappear because these applications constitute the backbone of corporate management systems. It means that in the future, end-users will increasingly demand tools that allow them to write the applications they need, tuned to their environment instead of delegating this task to specialised and expensive developers. One of the reasons for the limited diffusion of management tools lies with the cost of the tools and their extreme complexity. This work is a small contribution towards the construction of simple and powerful network management tools that can be used by many people and not only by rich or large organisations but also by universities and small institutions.