36

Testing Management Applications with the Q_3 Emulator

Kari Rossi, Sanna Lahdenpohja
Nokia Telecommunications Oy
P.O. BOX 33 02601 Espoo
tel: +358−0−5060 3857
fax: +358−0−5060 3876
email: kari.rossi@ntc.nokia.com

Abstract

Testing Q_3 based management applications is often a laborious and complex task. The Q_3 emulator agent (Q3E) is a tool for improving the effectiveness of testing the semantic functionality of management applications. An emulator agent is able to participate in OSI network management communication as the agent part: an emulator agent is an OSI agent in every sense, but it emulates to be running in a network element. For testing purposes, the operation of emulator agents can be controlled using the Q_3 emulator language (QEL) designed to decrease the test case design and implementation effort of management applications. In QEL, managed objects can be created or deleted, their action behaviours can be defined, and the sending of spontaneous events can be caused. Based on QEL definitions, the Q3E is able to respond automatically to requests from management applications. For the management application there is no difference: the agent, whether in network element or an emulator, responds similarly and handles the same managed objects.

Keywords

Testing Q_3 applications, testing CMIS/CMIP applications, Q_3, CMIS, CMIP, GDMO

1 INTRODUCTION

Testing Q_3 (ITU–T, 1992) based management applications is a demanding task and often requires significant development effort. One of the main reasons for this is the inherent complexity of the Q_3 interfaces and the specification formalisms Guidelines for the Definition of Managed Objects (GDMO) (ISO, 1992[2]) and Abstract Syntax Notation One (ASN.1) (ISO, 1990). Testing requires also deep knowledge and skill of both the management application and testing practices. In addition, it may be impractical or even impossible to maintain a realistic testing environment for

A. S. Sethi et al. (eds.), *Integrated Network Management IV*

the testers due to the high costs. Therefore, in order to decrease the development and testing effort and costs tools that support high level abstractions are needed. Unfortunately, the abstraction level of most currently available tools, such as XOM/XMP (X/Open, 1991) (X/Open, 1992) are low.

The Q_3 emulator agent (Q3E) (Rossi and Toivonen, 1994) is a high level tool for testing the semantic functionality of management applications. An emulator agent can be used to test management applications in an operation environment close to the real environment: the CMIS (ISO, 1991[2]) messages sent and received correspond to the real messages, and Q3E can emulate a network of managed objects. Q3E is not targeted at the OSI protocol or interoperability testing (ISO, 1991).

In this paper we first summarize the background of the Q_3 interface and the objectives of the Q3E. Section 4 describes the functionality of Q3E and section 5 explains how Q3E is used for testing management applications. Section 6 presents the conclusions.

2 BACKGROUND

The management concept of the Q_3 emulator agent is based on the Telecommunications Management Network (TMN) information architecture (ITU–T, 1992). The principles of the architecture are object oriented and are based on the OSI systems management concepts (ISO, 1992[1]), and the fundamental concepts are managed objects, manager and agent roles.

In the model the managed network and devices are structured into managed objects which have attributes, operations and notifications. Network management applications are distributed: an agent provides an object oriented view in the terms of managed objects of the resources it manages, and the manager issues management requests to the managed objects of the agent, and receives notifications from these managed objects.

The standardized interface between the manager and agent is Q_3. The managed objects are specified in GDMO and the attributes of managed objects in ASN.1. Each device type managed by Q_3 needs its own GDMO object model characterizing the special properties of the device. The communication protocol used for exchanging operation requests of managed objects is CMIS and CMIP (ISO, 1991[1]).

3 OBJECTIVES

The objectives of the Q_3 emulator agent are the following:

- Provide automation for the semantic testing of Q_3 management applications. OSI protocol and interoperability testing are out of the scope of this application, they are tested using other tools.
- Support testing of a network: Q3E has to support the emulation of a network consisting of many network elements.
- Q3E has to be programmable by an interpreted script language.
- Communication has to be based on OSI protocols, and CMIS, CMIP, ASN.1 and GDMO have to be fully supported.
- The system architecture has to be based on automatic code generation from GDMO and ASN.1 templates.

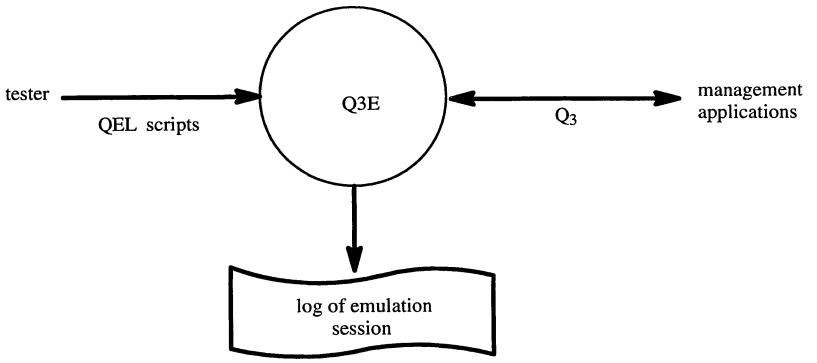

Figure 1 Controlling the execution of Q_3 emulator.

4 Q_3 EMULATOR AGENT FUNCTIONALITY

4.1 User Interface

The use of the Q3E is based on operation requests: the tester operates the emulator by writing a Q_3 emulator language (QEL) script, and when the script is ready, he submits the script to the Q3E for execution. For examining and monitoring the results of the execution of scripts and CMIS management operations the tester uses the Q3E log file. QEL scripts can be executed both interactively at run time or as batch scripts. See Figure 1.

From unix shell, the tester submits a QEL script to the Q3E using the **qrc** program. For instance, the unix command (1) executes the QEL script 'event.qel':

qrc event.qel (1)

4.2 QEL Language

Managed Objects

In QEL, managed object classes are referred to with the names given in GDMO templates. Managed object instances are referred to by distinguished names that are relative to the global root, as shown in the example (2):

[/, networkId = 1, managedElementId = 53] (2)

Distinguished names can also be constructed by specifying the path relative to another object such as a QEL variable. Managed object instances are stored in the Management Information Base (MIB) in the unix file system as ASCII files.

Attributes of object instances are referred to with the dot notation. For instance, the attribute State of the managed object (2) is referred to by:

```
[/, networkId = 1, managedElementId = 53].State                    (3)
```

Creating and Deleting Managed Objects

The tester can create managed objects using the **create** command. As a result of the command, the object is updated to the MIB. The parameters are similar to those of the CMIS create request. The tester can delete objects with the **delete** command.

Overriding Operations of Managed Objects

CMIS indications are served automatically by the Q3E. For changing this default behavior, QEL scripts can be assigned to CMIS indications to be called by Q3E when serving indications. Operations can be overridden based on object instance or class, or globally.

In the script (4) the **set** operation of the class 'trailTerminationPoint' is changed to run the script 'disabledEvent.qel' (5). The purpose of the script (5) is to send an event if the 'operationalState' attribute of the managed object is disabled:

```
change–operation trailTerminationPoint {
    set = "disabledEvent.qel"
};                                                                 (4)

– – script 'disabledEvent.qel'

if (%mo–instance.operationalState = "disabled") then
    event–req send {
        mode = non–confirmed,
        mo–class = %mo–class,
        mo–instance = %mo–instance,
        event–type = "communicationsAlarm"
    };
else
    emulate;          – – executes the default emulation operation of CMIS set
end–if;                                                            (5)
```

When changing the way to serve indications, the tester can call the automatic emulation using the **emulate** command. This is useful when the tester wants to extend the default emulation behavior as in scripts (4) and (5).

QEL Variables and Expressions

Two basic types of variables are supported: integer and string. In addition, variables of any ASN.1 type defined in the ASN.1 specification files can be created. Variables are declared by the **declare** command.

Integer expressions can be constructed with the arithmetic operators (+, –, *, /, %) from other integer expressions. String operators are + (concatenation) and – (removes the first occurrence of the second string from the first). Expressions can be grouped with parenthesis.

Assignment statements are begun with the **let** keyword. Variables may be assigned values of compatible types: type cast to integer is achieved by **integer()**, and to string with **string()**. For instance, strings 'prefix' and 'nodeId' and an integer 'i' are declared and assigned values by the script (6):

```
declare integer: i;
declare string: prefix, nodeId;
let i = 100;
let prefix = "node_";
let nodeId = prefix + string(i);                                    (6)
```

As a result of the script (6) the value of the variable 'nodeId' is "node_100".

QEL language provides two sets of predefined variables: global variables, beginning with '$', and references to CMIS indication parameters, beginning with '%'. The advantage of QEL variables is that they are more general and easier to use than absolute values since they contain emulator context specific information. References to the CMIS parameters of indications can be used when sending responses. This makes it possible to set appropriate context sensitive default values for the response parameters.

The **let** command is also used for assignment of attribute values of managed objects, e. g.

```
let $mo.systemTitle = {pString "node_5"};                           (7)
```

In the script (7) the value of the attribute 'systemTitle' is an ASN.1 string, but its type is an ASN.1 choice. **$mo** is the CMIS indication parameter referring to the managed object of the latest CMIS indication.

CMIS Commands

QEL provides commands for direct CMIS control: **create–rsp**, **delete–rsp**, **get–rsp**, **set–rsp**, **action–rsp** for sending CMIS responses and **event–req** for sending event report requests. For instance, script

```
get–rsp send {
      mo–class = $mo–class,
      mo–instance = $mo–instance,
      current–time = $current–time,
      attr–list = {delay = 10,  bufferSize = 21}
};                                                                  (8)
```

sends a CMIS get response in which the managed object class and instance are the same as in the get indication, and attribute list contains two attributes 'delay' and 'bufferSize'. **$current–time** is a predefined QEL variable.

QEL supports also the sending of linked responses and CMIS error messages.

Delays and Timing

In order to closer emulate the response times of network elements, the tester can define a delay for specified managed object instances. Delay specifies the time in seconds to wait before executing the response. In the example below, response delay will be 10 seconds:

```
set–delay {
        [/, networkId=1, managedElementId=53] = 10
};                                                    (9)
```

In order to time the scripts to be executed by the emulator the tester can use unix scripts.

Control Structures

Conditionality can be represented with the **if** structure, and repetition in turn with the **loop** and **exit–loop** commands. The script (10) demonstrates one way to implement a 'for' loop from 1 to 10:

```
declare integer i;
let i = 1;
loop
        – – do the job here...
        if (i = 10) then
                exit–loop;
        end–if;
        let i = i + 1;
end–loop;                                             (10)
```

A script can be run from another script with the **run** command, and a script can be exited with the **return** statement. The only way to 'pass parameters' is to use global variables.

4.3 System Architecture

The Q3E system architecture is based on C++–code generation tools (see Figure 2). The Q3E emulators are generated from the same GDMO and ASN.1 definitions used to specify the management interface between the management applications and the actual network elements. Two tools are used in the generation: the GDMO compiler of the Q3++ framework (Pohja et al., 1993) generates code from the GDMO definitions, and the Q3++ ASN.1 compiler produces the code necessary for handling ASN.1 types. The generated code is compiled and linked with the static Q3E code and with the Q3++ CMIS/CMIP communication library.

At run time Q3E consists of two unix processes, Q3E Run Script (QRC) client and the Q3E Emulator Server (QES) (see Figure 3). These two processes communicate through unix datagram sockets. QRC client program is the run time user interface of the emulator, and the QES server emulates the managed objects. The QES server listens to the requests of the QRC client process and CMIS indications of the management applications: QES executes each request or indication completely before starting the execution of the next request or indication.

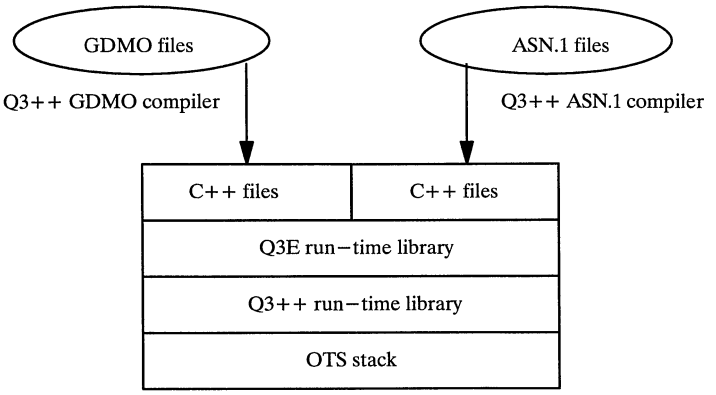

Figure 2 Q3E system architecture: generation principle.

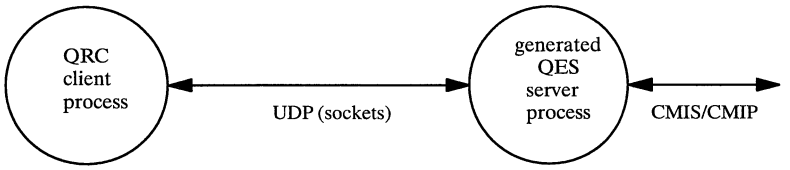

Figure 3 Q3E system architecture: run−time processes.

As the application programming interface towards the OSI stack two interfaces are supported, Nokia OSI stack, and HP OpenView XOM/XMP. In both cases, HP OTS/9000 stack provides the transport services. The current hardware platform is HP 9000 System 700*.

5 TESTING MANAGEMENT APPLICATIONS

5.1 Setting up the Testing Environment

The testing environment at the OSI agent side consists of the OSI stack, QRC client process, QES server process and Q3E MIB storing the managed object instances as ASCII files. MIB is most conveniently created with a QEL script, but it can be also created manually or by a unix shell script.

When setting up the testing environment, the first step is to generate the QES server process binary code from the GDMO templates and ASN.1 definitions specifying the management interface, and to configure the OSI stack used by the emulator.

* HP and OpenView are trademarks of Hewlett−Packard Co.

5.2 Principles of Writing Test Scripts

The test cases have to be planned carefully. Once they are written they can be repeated and reused. If no scripts are given to the Q3E, it receives indications and responds to them according to the standards and it's MIB. Other behavior of Q3E has to be defined using QEL scripts.

The Q3E is able to keep log file on various aspects of its operation. Since the log file is the only output from the Q3E, it is also the main source to be checked for the test results. Consequently, test planning should include also logging targets.

The QEL scripts implement the test cases and depend thus on the management application and the testing objectives. In the following paragraphs examples of different kinds of test cases are given.

Startup Scripts

If a Q3E were invoked without a startup script it would in most cases not be usable due to lack of information of the managed object instances. The purpose of a Q3E startup script is to define:

- the MIB containing the managed object instances of the emulated network elements;
- managed object class action behavior;
- the default usage of CMIS parameters;
- emulator specific defaults, e.g., logging parameters.

Testing CMIS Semantics

The most elementary test scripts respond to simple CMIS requests of management applications. The MIB is sufficient for many test cases testing the correct behavior of management applications, e. g., creating object instances, setting attribute values and combinations of CMIS parameters. For testing CMIS defined errors, it is convenient to override the default behavior of some object instances or classes to respond with CMIS errors.

Failure Scripts

A QEL failure script should be written for each kind of failure of a network element. A failure script executes all the emulator commands modelling a fault, such as modifying the managed object instances of the emulator to represent the new faulty state or sending an event or a set of events for the management application to inform of the failure. For example, the script 'communicationsFailure.qel' (11) changes the 'operationalState' and 'probableCause' attributes and sends an event report:

```
-- script 'communicationsFailure.qel'

[/, networkId = 10, equipmentId = 2].operationalState = "disabled";
[/, networkId = 10, equipmentId = 2].probableCause = "ProbableCause:{ localValue : 8}";
-- {localValue: 8} means loss of signal

event-req send {
   mode = confirmed,
   mo-class = "equipmentX",
   mo-instance = [/, networkId = 10, equipmentId = 2],
   event-type = "communicationsAlarm",
   event-info = asn1[AlarmInfo : {
            probableCause localValue : 8,
            perceivedSeverity major,
            notificationIdentifier 20}]
};                                                                    (11)
```

Combining Unix Scripts with QEL Scripts

QEL scripts can be combined with unix scripts to achieve even more complex tests. It is possible for example to create dynamically new QEL scripts, execute QEL scripts periodically, or automate a test session. For instance, the C shell script (12) calls the communications failure script (11) once in a minute:

```
while (1)
      qrc communicationsFailure.qel
      sleep 60
end                                                                   (12)
```

Unix shell must be used in this timing test case. If only QEL were used, the QEL script would block the execution of other QEL scripts and CMIS indications in the QES emulator server process because the QES executes one script (and CMIS indication) at a time.

Emulating a Network in Failure Scripts

The **mo-class** parameter of the event report request command defines the distinguished name of the object instance. Since the attribute values in the distinguished name can be arbitrary QEL expressions, it is possible to program a script sending events from multiple managed objects. If event report requests are sent in a loop, the loop variable can be used to construct attribute value assertions for the distinguished name. As a result, the management application receives event report indications from seemingly different managed objects.

In the script (13) a 'for' loop is implemented using the generic **loop** command and the variable i as a loop counter. An event is sent in every iteration. The distinguished name varies according to the value of the loop counter 'i': the last attribute-value-assertions in the distinguished names will be equipmentId = "node_1", equipmentId = "node_2", etc.:

```
declare integer: i;
let i = 0;

loop
        if (i = 100) then
                exit–loop;
        else
                let i = i + 1;
        end–if;
        event–req send {
                mode = non–confirmed,
                event–type = "communicationsAlarm",
                mo–class = "equipmentY",
                mo–instance = [/, networkId = 1, equipmentId = "node_" + i],
                event–time = $current–time,
                event–info = asn1[AlarmInfo : {
                        probableCause localValue :  2,
                        perceivedSeverity minor,
                        notificationIdentifier 20,
                        additionalText "Equipment Y specific fault text!"}]
        }
end–loop;                                                              (13)
```

Generating Side Effects

With QEL scripts it is possible to extend emulation of network elements from the simple receiving of indications and sending of responses and event requests. When receiving an indication, a special QEL script can be executed which changes other attributes of the managed object or modifies other managed objects in the MIB than requested by the CMIS indication.

For example, the scripts (14) and (15) extend the behavior of the create indication of 'managedElement' class to create a 'log' object instance for the current managed object:

```
change–operation managedElement {
    create = "createManagedElement.qel"
};                                                                     (14)

– – script 'createManagedElement.qel'

let nextLogId = nextLogId + 1;     – – global variable, initialized to 0 in the startup script
create {
        mo–class = log,
        mo–instance = [%mo–instance, logId = nextLogId],
        attr–list = {operationalState = "enabled"}
};
emulate;                                                               (15)
```

5.3 Performance Testing

The performance and stability of management applications can be tested using scripts that cause heavy communication loads. This could for instance be achieved with event generation scripts that send events at fast rate. Another alternative would be using several Q3Es.

6 CONCLUSIONS

This paper has discussed the testing of Q_3 based management applications. The testing of management applications is a demanding task, because, among other things, the Q_3 interfaces and the specification formalisms GDMO and ASN.1 are complex. Therefore testing tools are needed to decrease the effort and costs involved. The Q_3 emulator agent tool covers the semantic testing part of Q_3 management applications.

The main advantage of using Q3E lies in the reduction of testing costs. The testing costs affected are those for test equipment, man power, training and testing time. This is achieved because Q3E provides a high abstraction level for the testing personnel, and new emulators can be generated at short notice with minor effort.

The first version of Q3E that supports event sending over XMP has been in use since February –94. Initial experiences have been encouraging. This first version has been generated for three network element types, and their generation required about one day's effort from one person. The first two emulators are used by development teams in module testing and the third by a system testing group. The complete version of the Q3E will be released during first half of 1995.

The system architecture has been proven to be sound. The generation mechanism makes Q3E suitable for testing a very wide range of management applications. A considerable engineering effort was however required to achieve this kind of generality.

7 REFERENCES

ISO (1990) Specification of Abstract Syntax Notation One (ASN.1). ISO/IEC 8824, ITU–T Recommendation X.208.

ISO (1991[1]) Common Management Information Protocol. ISO/IEC 9596–1, ITU–T Recommendation X.711.

ISO (1991[2]) Common Management Information Service Definition. ISO/IEC 9595, ITU–T Recommendation X.710.

ISO (1991[3]) Conformance Testing Methodology and Framework. ISO/IEC 9646–1.

ISO (1992[1]) Systems Management Overview. ISO/IEC 10040, ITU–T Recommendation X.701.

ISO (1992[2]) Structure of Management Information Part 4: Guidelines for the Definition of Managed Objects. ISO/IEC 10165–4, ITU–T Recommendation X.722.

ITU–T (1992) Principles for a Telecommunications Management Network. ITU–T Recommendation M.3010.

Pohja, S., Kaski, J. and Nurmi, E. (1993) Application Programming Interface for Managed–Object Communications, in *IEEE First International Workshop on Systems Management*, Los Angeles.

Rossi, K. and Toivonen, H. (1994) Q3E: Q_3 Emulator Agent, in *1994 IEEE Network Operations and Management Symposium*, Orlando.

X/Open Company Ltd (1991) OSI–Abstract–Data–Manipulation API (XOM). X/Open CAE Specification.

X/Open Company Ltd (1992) Management Protocols API (XMP). X/Open Preliminary Specification.

8 BIOGRAPHY

Kari Rossi received his M. S. and Licentiate of Technology degrees in Computer Science at Helsinki University of Technology in 1986 and 1991. Mr. Rossi was the R&D project manager of the Q3E and Q3++ GDMO++ compiler projects. He is currently the R&D project manager of Nokia OMC for Fixed Network project which is developing a management system for Nokia DX 200 switches.

Sanna Lahdenpohja received her M. S. in Computer Science at Turku University in 1992. She was a senior engineer in the Q3E project. Currently she is a senior engineer in the Nokia OMC project.

9 ACKNOWLEDGEMENTS

Hannu Toivonen, Timo Posio, Lasse Seppänen, Saku Rahkila, Marko Setälä, Markku Rehberger and Susanne Stenberg have been working in the project team and have contributed essentially to Q3E.

The Q3E project has been partially funded by the Technology Development Centre of Finland (TEKES).