HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory of Telecommunications Technology
S-38.128 Telecommunications Technology, Special Assignment

MAP Protocol Software Testing and Development with Emulator

Author: Mikko Aittola, 42836M
Tutor: Vesa Kosonen
Date: 25.10.1999

ABSTRACT

Extensive testing is needed to ensure the quality of mobile network software. F5EMU is a tool that can be used for DX 200 program development, building, testing and debugging in Windows NT workstation without any extra hardware.

The objective of this special assignment was to research the possibility to use F5EMU for MAP protocol software testing and development.

The results of this assignment can be summarized as follows:

- The process family that implements the MAP protocol for DX 200 can be run in the emulator environment, but a special script is needed to make the conversion process possible
- Existing test bench based test cases can be used in the emulator environment
- Easy debugging provided by F5EMU has the potential to improve the efficiency of fault analysis and correction
- Usefulness of module test cases written in Tcl is questionable
- Further development of the tool is needed (support for ASN.1 tools, F5EMU should add the correct libraries to project tree automatically)

TABLE OF CONTENTS

TERMS, ABBREVIATIONS AND ACRONYMS

API	Application Program Interface
ASN.1	Abstract Syntax Notation One
BSC	Base Station Controller
BTS	Base Transceiver Station
CAMEL	Customized Applications for Mobile Network Enhanced Logic
CAP	CAMEL Application Part
DCL	Digital Command Language
DMX	DX 200 Operating System
DX	Digital Exchange
DX 200	Product family of network elements produced by Nokia
EIR	Equipment Identity Register
ETSI	European Telecommunications Standards Institute
FISLIB	File System Library
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
gsmSCF	GSM Service Control Function
GUI	Graphical User Interface
HLR	Home Location Register
IDE	Integrated Development Environment
INAP	Intelligent Network Application Part
IP	Internet Protocol
IS	Implementation Specification
LAN	Local Area Network
MAP	Mobile Application Part
McBED	Software Engineering Environment for DX 200

MI	Module Implementation
MML	Man Machine Language
MSC	Mobile Switching Centre
MT	Module Testing
MTP	Message Transfer Part
OWL	Object Windows Library, developed by Borland
PAC	Program Block Building Procedure
PLMN	Public Land Mobile Network
Process/DMX	A unit of execution sequence which is scheduled independently. A process has unique process and family id inside a DMX computer. [8]
Process/Win32	A running instance of an application program. [8]
Process family	A set of DMX processes, which have one master process and many hand processes. [8]
QoS	Quality of Service
R&D	Research and Development
SCCP	Signalling Connection Control Part
SD	Software Design
SDL	Specification and Description Language
SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module
SIWFS	Shared InterWorking Function Server
SMS	Short Message Service
SMS-GMSC	SMS Gateway MSC
SMS-IWMSC	SMS Interworking MSC
SS7	Signalling System No. 7
Tcl	Tool command language
TCAP	Transaction Capabilities Application Part
Thread	Win32 term for a unit of execution sequence, which is scheduled independently. Conceptually similar to main or hand process in DX 200 process family. [8]

TNSDL	TeleNokia Specification and Description Language
VLR	Visitor Location Register
Windows NT	Operating system developed by Microsoft

1. INTRODUCTION

The objective of this special assignment was to become acquainted with F5EMU tool and its features and research the possibility to run, develop and test existing DX 200 MAP protocol software with it. The work was done at Nokia.

The document is divided to six parts. The first part describes briefly the motivation and some theory about software testing in the communications industry. The second part concentrates on F5EMU tool and its features. The third part is about GSM MAP protocol and the current tools and methods used for module testing. The fourth part describes the conversion process needed to be able to run DX 200 MAP protocol software in the emulator environment. The fifth part features brief descriptions on how the tool can be used for testing. The final part is based on using the tool in practice. It features proposed target use areas for the tool and lists some of the benefits that can be gained by using it. Some disadvantages and possible risks are also discussed and propositions concerning the future development of the tool are made.

2. TELECOM SOFTWARE TESTING

How to ensure the quality of millions lines of program code? How to make sure that all the customer features in various different combinations function properly? That is a challenge the communications network equipment manufacturers have to face.

Customers demand high quality product. The end users are accustomed to the high QoS that telecom networks provide. As the competition increases and the product release cycle becomes shorter, the efficiency and quality of the development and testing processes are more and more important.

Important part of the quality control of telecom equipment is extensive software testing. It has been estimated that testing takes about 50% of the R&D effort. [1] One of the basic rules of testing is that the earlier the faults are found, the cheaper it is to correct them and the higher is the probability of fixing the errors correctly. [2] Hence it could be said that, along with inspections and reviews, *module testing*, sometimes also called unit testing, is the key test process. The later testing phases, which are also important, are out of the scope of this document, because the tool that is presented in the next chapter is not designed for phases such as integration or system testing.

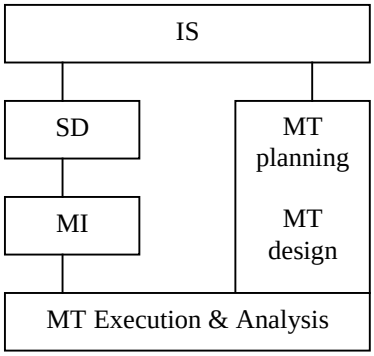

Figure 2.1 *Partial model of the program block development process [3]*

At the MSC&HLR product line of Nokia, module testing is usually done in parallel with software development. [3] In practise it means that black-box style tests can be designed before the software module to be tested is even implemented. The great advantage of the approach is that big part of the module tests can be run immediately

after the module is implemented, without the delay that designing the test cases after module implementation would cause. Figure 2.1 on previous page demonstrates the parallelism between module testing and software development.

2.1 Black and White Box

Test case design is usually divided to black-box and white-box methods. Alternative terms, *functional* and *structural*, respectively, are also sometimes used in literature.

Black-box method means that the tests are designed to verify that program produces the desired output from given input, according to specification. In white-box design method, the internal structure of the program is used to find the probable problem areas that need to be tested.

These methods should not be seen as strict and limiting rules. The situation where test engineer doesn't know anything at all about the internal structure of program is quite rare. *"In practice, it hasn't proven useful to use a single test design method. One has to use a mixture of different methods so that they aren't hindered by the limitations of a particular one. Some call this "gray-box" or "translucent-box" test design, but others wish we'd stop talking about boxes altogether."* [4]

2.2 Test Case Automation

Automated software testing provides some immediate benefits. Test cases can be saved to database from where the cases can be examined and re-used later. Regression testing, i.e. re-testing after fixes or modifications, is important part of software testing and test case automation makes that much easier.

Automated analysis of the test case results improves quality and saves time. Analysing hundred byte long messages by hand takes a long time. Test engineer could become frustrated doing that kind of work and as the result quality would suffer.

Testing automation doesn't automatically reduce the needed test effort and it is good to remember that it takes time for test team to learn to use a test tool effectively [5]. General experience is that test automation increases quality, but not always efficiency [1].

3. F5EMU

F5EMU is a set of tools for DX 200 program developing and testing. It makes possible to develop, compile, run, test and debug DX 200 process family (or families) in ordinary Windows NT workstation without any extra hardware. [6]

If needed, the process family can communicate with other process families, which are running in the same or different workstation. In addition, the emulated process families can also communicate with real DX 200 systems or test benches. [7]

F5EMU is an optional part of McBED product family. Without F5EMU, McBED offers complete and widely used set of tools for DX 200 program development and testing, but no emulator features.

3.1 DMX vs. Windows NT

DMX emulation is implemented with emulation libraries that provide the core DMX services and library interfaces. Existing source code has been used as much as possible, but some parts have been completely rewritten because of certain fundamental differences between DMX and Windows NT. Thorough explanation of the differences can be found in F5EMU documentation [8].

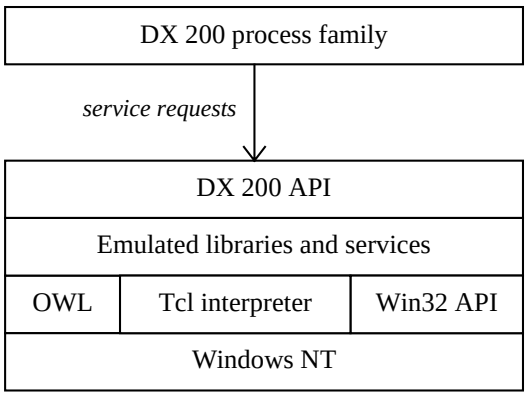

Figure 3.1 *Simplified model of F5EMU's DMX emulation architecture [21]*

3.2 Features

F5EMU has a number of useful features. The emulator is fully integrated to Borland IDE, which offers the means to reach a fast edit-build-test-debug-edit cycle. With traditional tools building a large program can take a relatively long time, but Borland's make-engine compiles only the modified parts instead of building the whole program from ground up every time. Thus, at least in some cases, F5EMU has a potential to save time and costs considerably.

F5EMU tools run in a normal Windows NT environment, which means that all the other desktop applications, like Email, web-browser, word processor etc., can be open and used simultaneously.

3.2.1 TNSDL Source Modules

TNSDL is based on SDL-88 [9]. It provides direct support for the DX 200 conceptual model. [10] With TNSDL, matters related to finite state machines are particularly easy to implement.

Many DX 200 software program blocks are implemented using both, C-programming language and TNSDL. Usually, the finite state machine related modules are written in TNSDL and matters pertaining to algorithms and hardware are written in C. F5EMU offers full support, including compiling, linking and debugging, for TNSDL source modules.

3.2.2 McBED Tools Support

F5EMU is integrated with other McBED tools. For example, F5EMU uses a McBED tool for TNSDL translations. It also uses McBED services for obtaining the public datatype definitions.

3.2.3 Debugger

F5EMU features a versatile debugger. Debugging is possible by setting multiple breakpoints, watches, conditions and attached actions. Program can be also run step by step. Different data symbols and structures can be examined and modified "on the whim" with the debugger.

TNSDL source code can be debugged either at source level or in translated form.

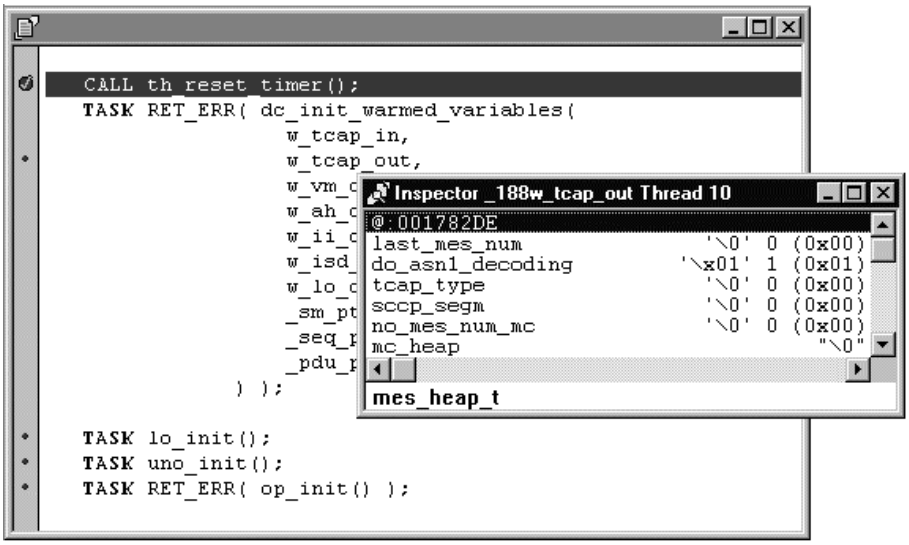

Figure 3.2 *TNSDL code can be debugged at source level*

3.2.4 Service Terminal Emulation

F5EMU features a service terminal emulation application, called *servterm* that runs as a separate process family. It looks and functions like the service terminal running on a real DX 200 computer. The main difference from the user's point of view is that not all service terminal commands are supported. The detailed list of supported commands can be found in F5EMU documentation. [11]

3.2.5 Process Family GUI

When running a DX 200 process family in the emulator environment, the behavior of the family can be controlled and monitored via graphical user interface.

The user can monitor the number and state of hands. Messages and logs can be monitored and saved for later inspection.

The user can run Tcl scripts from the GUI to automate the test cases. The GUI also features a simple interface for message sending, but it's mainly useful only for testing that the process family is running and accepting messages. Even in those cases it's easier to use some previously written Tcl script, if available.

3.2.6 Tcl

Tcl is a scripting language created and developed by John Ousterhout. It has been estimated that over half a million developers use it worldwide. Tcl is open source and extendable. The ongoing development of Tcl is collaboration between engineers at Scriptics Corporation and others in the Tcl user community. [12] The syntax of Tcl is quite clear and reader friendly compared to certain other scripting languages.

Tcl interpreter has been embedded to F5EMU. The language has been extended with some additional commands. For example, with command *dmxmsg* messages can be sent to DX 200 process family. Other extensions are for process, time and semaphore handling. Tcl scripts can be run from the GUI of the process family.

3.2.7 FISLIB Explorer

F5EMU features a utility program called *Fislib Explorer* that makes possible to use File System Library (FISLIB) files in the emulator environment. [13]

The user can load DX 200 data file images from disk to FISLIB system. The files can be modified with FISLIB Explorer and the modified versions can be saved back to disk. [13]

3.2.8 Support for Other Tools

F5EMU features support for testing tools that are originally designed to be used with real DX 200 computer test benches. Instead of connecting to the test bench, the user can run the same test cases in the emulator environment.

It is also possible to use test case coverage analysis tools and performance analysis tools like profilers.

4. MAP

4.1 Concept

MAP consists of a set of MAP services, which are provided to MAP service-users by a MAP service-provider. The MAP service-users interact with the MAP service-provider by issuing or receiving MAP service-primitives at the service interface. [14]

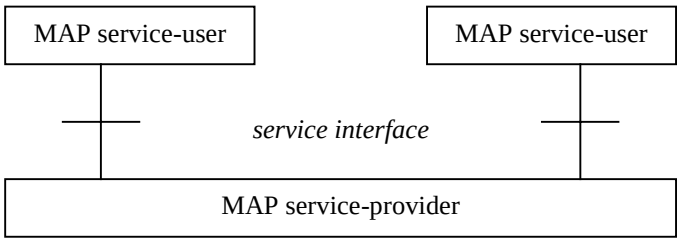

Figure 4.1 *The concept [14]*

4.2 GSM Network Architecture and MAP

GSM PLMN can be divided to subsystems. Each subsystem consists of different network elements. *Mobile Station* (MS), for example a cellular phone with SIM card, communicates with *Base Station Subsystem* (BSS) through radio interface. BTS and BSC network elements belong to BSS. BSS communicates with *Network Subsystem* (NSS) through A-interface. NSS includes many different network elements, for example MSC and HLR. BSS and NSS are controlled by *Operation Subsystem* (OSS). It contains functions related to operation and maintenance. [15]

MAP is used between the network elements inside the NSS. The NSS entities can also belong to different PLMNs. A picture describing the network elements using MAP and the interfaces between them can be found on the next page. It should be noted that some of the interfaces may be implemented internally. For example, MSC and VLR are represented by single network element and the interface between them is not standardized.

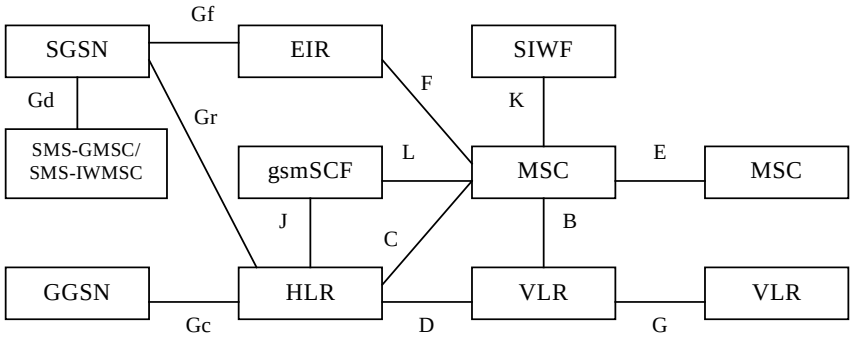

Figure 4.2 *MAP interfaces between GSM network elements*

4.3 Protocol Stack

MAP uses the services provided by lower SS7 protocol layers TCAP, SCCP and MTP.

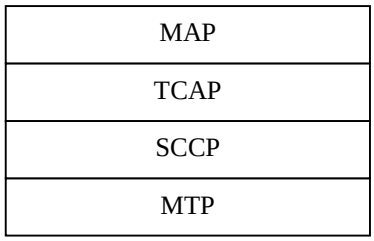

Figure 4.3 *Protocol stack*

4.4 MAP Services

MAP offers services related to mobility management, operation and maintenance, call handling, supplementary services and short message management. Detailed description of the services can be found in MAP specification [14].

Some of the MAP services are common services, which are available to all MAP service-users. Part of the services are user specific, which are services available to one or several, but not all, MAP service-users. [14]

4.5 Specification

The way MAP messages are represented in the network, i.e. the way they're encoded to TCAP messages is standardized. However, the internal data structure between MAP service-user and MAP service-provider is not standardized.

MAP specification specifies the abstract syntaxes for MAP using ASN.1. The abstract syntaxes are encoded to TCAP messages according to Basic Encoding Rules (BER) for ASN.1 [16]. [14]

MAP developer or tester cannot rely solely on the ASN.1 specifications, because the syntax allows that many of the data structures can be extended with proprietary extensions. There are also cases where a parameter is defined optional in ASN.1, but is mandatory in certain situations that are described elsewhere in the specification.

4.6 Protocol Engine

Nokia has implemented MAP protocol by using Protocol Engine. The engine takes care of common TCAP-user tasks like address handling, application context handling, dialogue control, TC-component handling and SCCP registration. Other TCAP user parts, CAP and INAP for example, can also be implemented with the engine. [17]

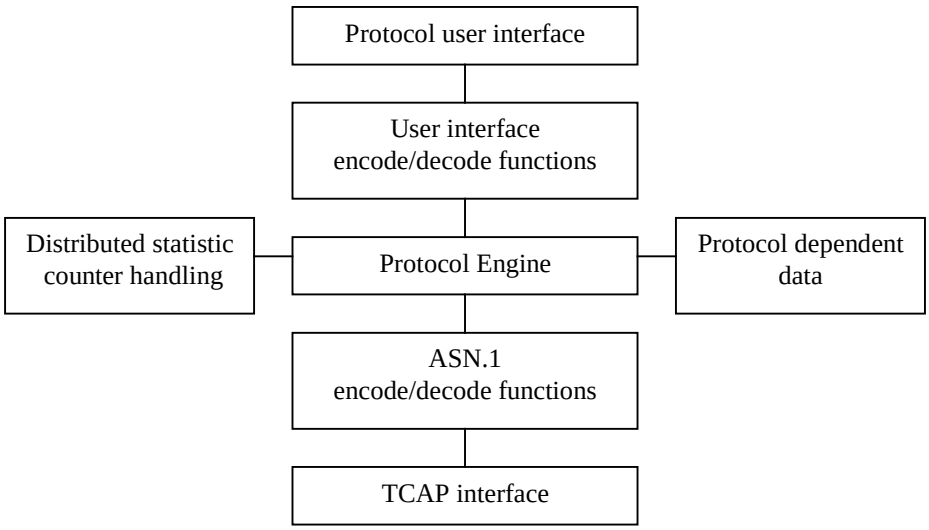

Figure 4.4 *Protocol Engine architecture*

4.7 Testing

MAP protocol is under continuous development. As new features are added to GSM networks, they often affect MAP. Examples of these new features include GPRS, location based services and CAMEL.

Systematical and extensive testing is needed to control the quality and functionality of the new features. It is also a known fact that modifying an existing program often causes errors to the parts that have worked well previously. Hence, the functionality of the old features needs also to be tested again and again.

MAP implementation by Nokia also has to work with the implementations by other manufacturers. When network user roams outside her home network, the VLR of the visited network exchanges information with the HLR of the home network according to MAP protocol. In such cases the VLR and HLR may be from different manufacturer. Many operators also use network elements by more than one manufacturer in their networks. For example, in some operator's network HLR can be from different manufacturer than the MSC. Again, systematical and extensive testing is needed to make problem-free interworking possible.

4.8 Existing Tools and Methods

Currently there are two main tools that are used for MAP module testing at Nokia. Test bench based tests cover all the blocks seen in figure 4.4, but most of the encode/decode routine tests are done with workstation based tool called Pctest.

4.8.1 Test bench

Test bench based tests are run from a Windows NT workstation that is connected to a DX 200 test bench through LAN. Test cases are described in custom-made macro language. Additional tools for message parsing and analysis of the test case results have to be used. Available debuggers are not very user friendly.

4.8.2 Pctest

Pctest works in an ordinary Windows NT workstation and doesn't require any extra hardware. It features automatic analyzing of the test case results and it prints the results along with message monitoring to test report files. [18]

Test cases are described in C. Debugging is easy with commercial C-debuggers.

Pctest is only suitable for testing the encode/decode routines. Real MAP dialogue sequences cannot be tested. TNSDL cannot be debugged at source level.

4.8.3 Need for F5EMU?

The motivation to research the possibility to use F5EMU for part of MAP module testing and development springs from the limitations of the two tools described in previous paragraphs.

With F5EMU every part of MAP could be module tested without extra hardware. Also, it would be possible to debug both TNSDL and C at source level. Real MAP dialogues could be tested as well.

F5EMU might also provide the possibility to use commercial test case coverage and performance analysis tools. The possibility to use standard scripting language like Tcl was considered worth investigating as well.

To get the process family that implements the MAP protocol running in the emulator environment was not a straightforward process. In fact, it was the most time consuming part of this special assignment. The process is explained in the next chapter.

5. FROM PAC TO IDE

DX 200 program blocks are built according to *building procedures*, which are called *PAC*s. Typical PAC features commands for fetching, translating, compiling and linking source modules from the version control database to an image that can be loaded to DX 200 system. PACs are written in DCL. [19]

```
$ fetch /version 1.2-0 foo.c
$ fetch /version 3.1-1 bar.sdl
$
$ 'tncmd' 'tnsdl_flags' bar.sdl
$
$ 'c_comp' foo.c
```

Figure 5.1 *Excerpt from example PAC*

Borland's IDE offers means to write, edit, compile, link, run, manage and debug programs. In Borland, building procedures are called *project*s. Borland's project files have a suffix '.ide'. [20]

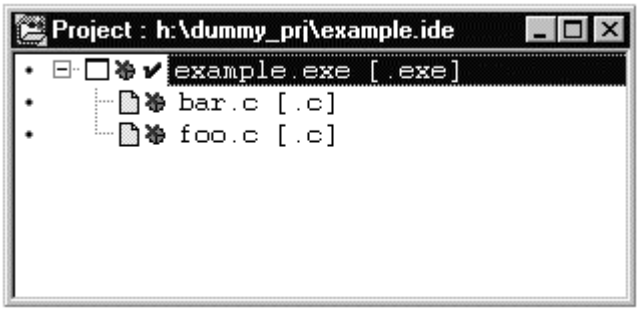

Figure 5.2 *Simple example project*

In order to compile DX 200 program block with Borland IDE, a corresponding project file ("IDE") must be created. This can be done by hand, but F5EMU also offers an automated tool, *pac2ide*, for doing the conversion.

5.1 Pac2ide

Pac2ide is a tool, which converts a PAC to an IDE. The tool analyzes the contents of the PAC and performs the necessary actions according to it. There is four possible actions, the commands are *executed directly*, *accepted for conversion*, *ignored* or *rejected*. [21]

If the DCL command has no equivalence in the project, the command cannot be converted. Such commands, for example symbol assignments, are executed immediately. [21]

The commands that have equivalence in the project are added to the commands-list for further processing. The inputs and outputs of the command are deduced for determining the dependency relations. [21]

Many PAC commands are irrelevant in F5EMU and can be ignored. Pac2ide prints a warning message in such cases. [21]

If F5EMU does not support the command it is rejected. In such case error message is printed and the conversion is aborted. [21] The PAC has to be edited before the conversion is possible. The conversion is also possible by selecting the 'ignore errors' option. In that case Pac2ide converts only the supported commands.

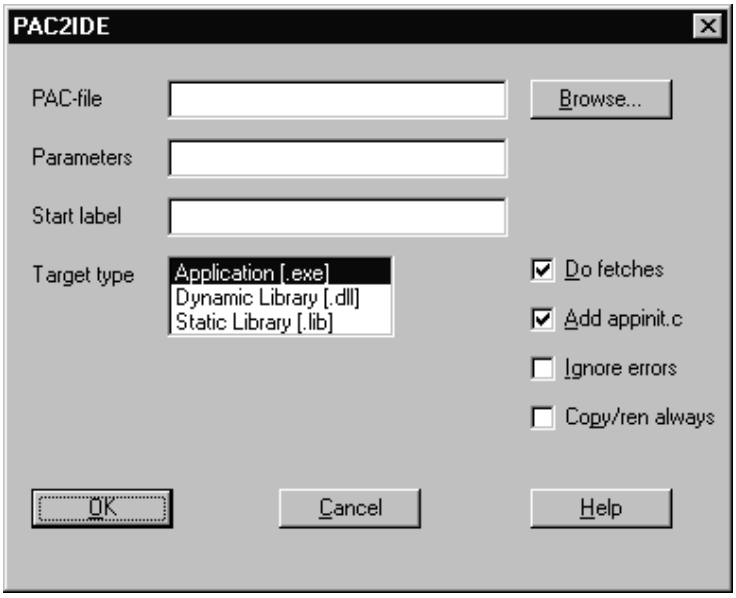

Figure 5.3 *Pac2ide GUI*

5.2 Pac2pac2ide Script

F5EMU does not yet support the ASN.1 tools. Those are used in the PAC that builds the process family that implements the MAP protocol for DX 200 systems. Because of the ASN.1 tool commands and certain other compatibility issues, the PAC couldn't be converted directly with Pac2ide.

Although the direct conversion wasn't possible, it was quite clear that converting the PAC by hand would be too time consuming, especially if the task needs to be repeated often. By "pre-executing" the incompatible commands and modifying the PAC before running Pac2ide, the conversion was possible and at least partly automated. It was apparent, however, that editing the PAC manually every time would be cumbersome. Hence, a script that modifies and prepares the MAP PAC for Pac2ide was written as this special assignment's side project of sorts.

The script is a fairly straightforward batchfile that executes McBED tool commands. It fetches the desired PAC from the version control database, analyzes it using *grep* and creates and runs a temporary PAC that executes the parts that aren't compatible with F5EMU. After that the MAP PAC and some related files are modified to prevent Pac2ide error messages by calling simple *awk* [22] scripts.

Special attention was paid to keep the script as maintenance free as possible. For example, the files aren't fetched according to any specific version and wildcards are used whenever possible. It is clear, though, that maintenance cannot be completely avoided if the basic structure of the PAC changes.

6. TESTING WITH F5EMU

6.1 Tcl Scripts

Test scripts written in Tcl can be run from the GUI of the process family. The example script below sends a message containing Update GPRS Location operation from "network" to MAP.

```
dmxmsg $src $dest $msgid {
02 66 00 80 01 03 17 00 19 31 FF 39 00 67 00 18 00 05 00 06 00 00 00 00 01
04 01 00 01 03 09 04 00 01 00 00 00 00 06 06 17 00 05 00 07 00 00 00 00 01
04 01 00 01 03 08 04 00 01 02 03 04 05 07 07 04 00 00 01 00 20 03 A1 2B 00
00 FF FF FF FF FF FF 00 00 00 00 00 08 04 02 01 17 30 18 04 06 32 14 05 01
02 03 04 07 91 53 48 10 20 30 40 04 05 04 83 E4 07 11
} 0
```

6.2 Test Bench Tools

Existing test bench based module test cases ("macros") can be run in the emulator environment. The same test tool that is used for running the test cases in test benches can be used with the emulator. After the MAP process family is up and running in the emulator, the test engineer connects the test tool to the emulated family via emulated service terminal. That can be done by simply connecting to *localhost* instead of the IP-address of a test bench. Detailed instructions are available in F5EMU documentation.

The macros that test from-user-to-network-to-user cases require that the lower protocol layers are present in the environment. There’s three possible work-arounds for these cases. The process families that implement the lower layers could be run in the emulator environment. However, using a "dummy" TCAP that just forwards the messages back to MAP would probably be a better idea, because that would make testing more independent of the lower layers. It’s also possible to modify the test cases so that the dialogue is split to from-user-to-network and from-network-to-user parts.

7. PROPOSED TARGET USE AREAS

Existing test bench based tests can be executed in the emulator environment. This is useful especially in situations where test benches are either reserved or unavailable because the new software package is not yet installed.

F5EMU's easy to use debugger has the potential to make fault analysis and correction more efficient in the cases where the exact location of the fault is unknown. Messages can be cut from the service terminal logs and pasted to Tcl script template that sends a message to MAP. This has potential to decrease time spent to the fault analysis phase also when the fault is found during later testing phases or by a customer.

It is worth noting that designing and running the test bench based module tests in the emulator environment does not exclude the possibility to run the same test cases later with real test benches.

Test case design training might also be easier with the emulator, at least for users that are not yet familiar with DX 200 computers.

The emulator might also be useful during the software design and implementation phases. If the software designer is unsure how some particular part works or how it should be implemented, the designer could use F5EMU to test the different alternatives and possibly speed up the iteration process. The same applies to white-box testing in general.

7.1 Benefits

The main benefits of F5EMU can be listed as follows:

- No extra hardware needed
- Fast edit-build-test-edit cycle
- Good debugger
- Possible to debug C-code, TNSDL-code and translated TNSDL-code
- Compatible with existing tools
- Easy cut-and-paste debugging from message monitoring possible with Tcl

- The emulator environment is virtual real-time and it can be controlled with Tcl scripts
- Easier to use for new test engineers, because compared to test benches not as much knowledge about DX 200 operation is needed
- The emulator testing environment is more isolated of the surrounding DX 200 system compared to test bench environment

7.2 Shortcomings

As described in section 5.2, a special script is needed to convert the MAP PAC to IDE. F5EMU doesn't yet support the ASN.1 tools. Also, the library files included in the PAC have to be added to the IDE by hand.

Describing module test cases with Tcl is easy only on the surface. For example, it is not possible to fill the messages with Tcl using the DX 200 datatype definitions. Filling the messages that can be several hundred bytes long by hand is a slow process. It might be possible to implement a C-module for filling the messages, though. Also, at present time messages can only be *sent* from Tcl. *Receiving* the messages sent by MAP is needed before automated test case result analysis with Tcl would be possible. Currently the received messages can be saved to file and an existing analysis tool can be used to analyse the test case results.

Using Tcl instead of C has also other kind of problems. A module test can potentially reveal a fault in at least four different places: the specification, program, test case and test tool. It means that in unclear situations both, the test engineer and software designer may need to be able to read and debug the script language *and* the programming language to sort out the problem. By our experience with Pctest, it's is a lot more simpler when everything is just written in plain old C.

It takes some time to set up the process family up and running in the emulator environment if it has to be done from scratch. In the case where previous user has already configured the test bench in suitable way, it would be faster to use the test bench instead of F5EMU.

7.3 Risks

Using F5EMU has also some risks that need to be acknowledged. The tool is currently tied to DMX operating system and to Borland IDE. If the operating system changes, the emulator needs to be updated or changed too.

What if emulator development is too much behind DMX development? For example, are up to date library updates available all the time?

Is the tool dependable? According to F5EMU documentation, the tool is not formally verified to emulate DMX in all situations, so a user shouldn't trust it alone in tests. [6]

8. FUTURE DEVELOPMENT

8.1 F5EMU

Although F5EMU is useful even in its current form, further development is needed to make the conversion from PAC to IDE easier. F5EMU needs to support the ASN.1 tools. It should also be able to add the correct libraries to the project tree automatically.

8.2 MAP Development and Testing

Usefulness of test case coverage and performance analysis tools should be investigated. It might also be useful to examine if Pctest filler routines could be used to fill test case messages, which could then be sent from Tcl scripts to the MAP process family.

REFERENCES

[1] Auer, A., Testing at Nokia Telecommunications, Presentation slides for the Software Testing and Validation course at Helsinki University of Technology, 1998

[2] Myers, G., The Art of Software Testing, John Wiley & Sons, Inc., 1979

[3] Halme, J., Salmi, T., Aalto, H., MSC & HLR Module Testing Process Area, Nokia internal document NCSD 2 05 0172E, 1999

[4] Faught, D., comp.software.testing Frequently Asked Questions (FAQ), 1999 < http://www.faqs.org/faqs/software-eng/testing-faq/ >

[5] Dustin, E., Rashka, J., Paul, J., Automated Software Testing: Introduction, Management and Performance, Addison Wesley Longman, Inc., 1999

[6] F5EMU Frequently Asked Questions, Nokia internal document, 1999

[7] F5EMU Interfamily Communications, Nokia internal document, 1999

[8] Hippeläinen, L.., DX 200 F5 DMX Emulation Libraries on Win32 API, Nokia internal document, 1997

[9] Functional Specification and Description Language (SDL), Recommendation Z.100, CCITT Blue Book 1988

[10] Lindqvist, M., Ruohtula, E., Kettunen, E., Tuominen, H., The TNSDL Book, Nokia internal document YFR 0811/3E, 1995

[11] F5EMU Service Terminal Emulation, Nokia internal document, 1999

[12] Tcl/Tk, Scriptics Corporation, 1999 < http://www.scriptics.com/products/tcltk/index.tml >

[13] FISLIB Explorer User's Manual, Nokia internal document, 1999

[14] GSM 09.02, Digital Cellular Telecommunications System (Phase 2+); Mobile Application Part (MAP) Specification, ETSI, 1999

[15] Mouly, M., Pautet, M-B., The GSM System for Mobile Communications, published by the authors, 1992

[16] ITU-T Recommendation X.690, 1994

[17] Einamo, K., Protocol Engine, Nokia internal document, 1999

[18] Aittola, M., Pctest Suite for Module Testing of Encode/Decode Routines, Nokia internal document, 1999

[19] Mankinen, J., Commands Allowed in the Building Procedures (PACs) of a Program Block, Nokia internal document YFR 1110/2, 1994

[20] Borland electronic documentation

[21] Koivulainen, J., Generation of Build Information with Dependence Relations from a Simple Building Procedure in List Format, Information Science, Special Assignment, Helsinki University of Technology, 1998

[22] Schulz, R., comp.lang.awk FAQ, 1999
< http://www.faqs.org/faqs/computer-lang/awk/faq/ >