# 29
# Combined Application of SDL-92, OMT, MSC and TTCN

**Eurico Inocêncio,**
INESC
Praça da Republica 93 R/C
4000 Porto, Portugal
TEL: +351-2-2094224,
FAX: +351-2-2084172,
email: emi@jaguar.inescn.pt

**Manuel Ricardo,**
ISEP, INESC
Praça da Republica 93 R/C
4000 Porto, Portugal
TEL: +351-2-2094225,
FAX: +351-2-2084172
email: mricardo@inescn.pt

**Hitoshi Sato,**
NEC Corporation, Overseas Transmission Division
1753 Shimonumabe, NAKAHARA-KU,
KAWASAKI, KANAGAWA 211, Japan
TEL: +81-44-435-5572, FAX: +81-44-435-5676
email: satohh@otd.trd.tmg.nec.co.jp

**Toshimitsu Kashima,**
NEC Telecom Systems, Ltd. 2nd System Division
1753 Shimonumabe, NAKAHARA-KU,
KAWASAKI, KANAGAWA 211, Japan
TEL: +81-44-435-1172, FAX: +81-44-435-1169
email: kashima@ntes.nec.co.jp

**Abstract**

The paper describes the application of SDL-92[1] and OMT[2] to the design of a V5.x Access Network interface. While OMT is used to model the management aspects of the system, typically described as TMN objects, SDL-92 is used to describe the V5 signalling stack as well as all the distributed components. The resultant combined model is used to automatically produce an efficient C++ implementation. TTCN is the language used to evaluate the conformance of the V5 interface standards. However, complementar and service oriented testing is also required. Service provision correctness, service interaction avoidance and, particularly, quality of the services, need characterisation. For that, MSC and SDL combined languages complemented with time and probabilistic descriptions, are being used to specify test purposes, describe and verify test cases.

**Keywords**

V5 interface, testing, automata, QOS, SDL, OMT, MSC, TTCN.

## 1. INTRODUCTION

Normalisation is a key issue for the Open Network Provisioning concept (ONP). ONP devises a scenario of Telecom liberalisation where many companies compete and collaborate with each other in order to deliver cost effective telecommunication services. The Access Network is one of the segments where this specialisation is more likely to take place - a more distributed, clustered, flexible Access Network infrastructure deploying over its geographical domain a mixture of basic and advanced services.

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

The European Telecommunication Standards Institute (ETSI) has promoted a set of standards aimed at the normalisation of the interface between the Access Network (AN) provider and the Switching infrastructure: the V5 interface [5,6]. V5.1 and V5.2 interfaces cover PSTN and narrow band ISDN deployment; broadband access will be addressed by the upcoming VB5 interface. The functional split between Access Network and Switching Equipment, introduces not only the need for a normalised interface, but also arouses the need for effective open Network Management - in ONP environments various companies jointly operate a heterogeneous assortment of multi-vendor equipment.

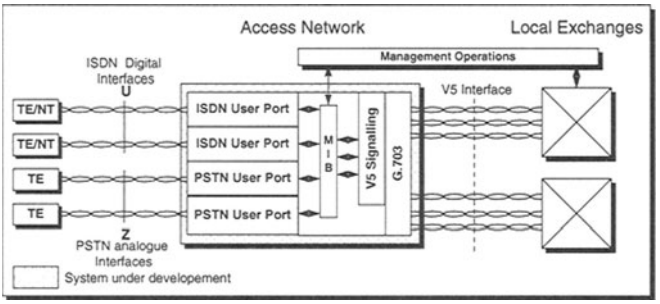


**Figure 1** Network Architecture Overview

Overall the Access Network equipment must comply with a set of standards, which can be classified into three main groups:

- Functional interface specifications [5,6]- define the signalling stacks, message sets and formats as seen across the physical interface, conceptual state machines along with the physical interface specification.
- Q3 Management information model [7,8] - these standards define the ASN.1 MIB that models the AN from the management system perspective. It covers the OAMP aspects of the system involving configuration, fault and performance management.
- Conformance Test Specifications TTCN [11,12]- System validation and testing.

Although the Interface and Management standards use informal text to describe protocols, messages and MO behaviour, they provide useful informal SDL descriptions for the signalling, and formal ASN.1 object models for the management. Hence the standards provide the initial model that is captured into the OMT and SDL tools. In a second step the models are refined and system specific implementation options are introduced, eventually an implementation specification is reached from which C++ code is automatically produced. The development of the AN and Test Emulator signalling firmware involves two teams, each with 4 to 5 engineers using the described methodology.

## 2. COMBINED OMT AND SDL MODEL.

Rumbaugh Object Modelling Technique (OMT) is increasingly being used in combination with SDL-92. Leading commercial tools are offering, or promising, support for combined methodologies, while some researchers report experimental tools and results [3,4]. Most

authors and companies advocate a waterfall model: analysis in OMT, followed by design in SDL-92 and automatic translation to the implementation language. While agreeing that it is possible to use only SDL-92 in the implementation of OMT models, this project explores the concurrent code generation from both models.

In the analysis phase OMT is used to capture the implementation requirements. The "entity relationship" [10] diagrams of the management standards were effortlessly transformed into a preliminary OMT object model. In this OMT model one can find a number of active objects: user ports, transmission resources, packet routing definitions, maintenance tests, etc. In turn the functional standards provide informative SDL descriptions that define the behaviour of these entities as seen from the V5.x interface.

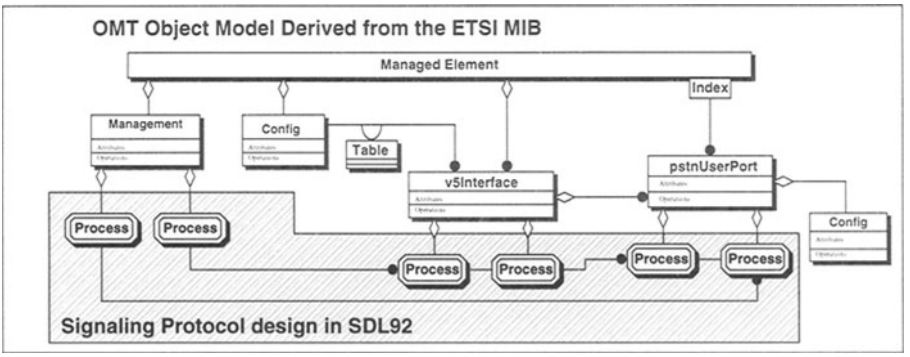


**Figure 2** Example of OMT object model with aggregation of SDL92 process types.

From the coalition of both specifications results a OMT object model that defines the object containment, inheritance and relationships. SDL-92 is used to specify the dynamic behaviour of certain objects, i.e. some OMT objects contain SDL processes. This approach is depicted on Figure 2, two overlapping descriptions, OMT and SDL92, are used to specify the V5.x signalling sub-system:

- The OMT model instantiates all objects, including the aggregated SDL processes. These SDL processes receive the required process Identifier (PId) and configuration values by means of the formal parameter list. Other objects without SDL behaviour, are generated according to predefined C++ generation templates.
- The SDL description defines the process types, the inherent protocol FSM behaviour, message (signal) set and static signal routing graph between the SDL process sets. This description uses normal SDL structuring concepts: system, packages, block types, packages, process types, routes, etc. The only deviation from a traditional SDL specification is the omission of the number of object instances, therefore from the SDL all process sets start with one initial instance and are unbounded. In the actual implementation, the creation and instance binding is performed by the constructors and destructors of some OMT objects.

An OMT object that aggregates one or more SDL process types becomes very close to the SDL-92 block type concept. But unlike the SDL-92 block type, the OMT object permits

associations, attributes and operations that are not necessarily visible by the SDL specification. Instead of "forcing" the whole information model into the SDL description, it was felt that the typical management operations are more efficiently implement by mapping some OMT objects directly to C++, without a intermediate SDL-92 step. For example, it is possible to maintain dynamic tables and lists of complex OMT objects instances, while SDL-92 dynamics is restricted to process by process creation.

## 2.1 OMT GENERATION TEMPLATES

The fact that OMT lacks formal semantics, allows an infinite number of implementation mappings for each object or association. In the Verilog LOV OMT tool, the user defines Generation Templates to customise the implementation of objects and associations. The present system uses the following class generation templates:

- **SDLClass** - The object has SDL behaviour and will be generated from a SDL specification, the formal parameter are given by the associations;
- **UnionClass** - The object is a container upon which one of the aggregated objects can be constructed;
- **PackedClass** - Used for objects that require a controlled memory layout, for example in device drivers or protocol messages;
- **AlarmMask** - Special objects used to efficiently implement bit oriented alarms, from the SDL standpoint these objects are viewed as ADT with an operation for each controlled alarm.

Besides these, the project defined G-Templates for pointer, reference, array, linked list and hash table associations. The next two sections provide an explanatory example of the **SDLclass** G-Template, which is the key OMT/SDL integration mechanism.

### 2.1.1 Embedding SDL into the OMT objects

Figure 3 presents a simplified view of the *pstnUserPort* MO implementation. For the purposes of this presentation it is assumed that, besides its own attributes and operations, the MO contains two SDL processes, A and B. These SDL processes implement user port signalling and communicate with process types C and D which are part of the common signalling stack. Since each process type is translated into a C++ class, the SDL process aggregation is translated to plain C++ membership::

```
class pstnUserport {
  private: /* OMT object attributes */
    Type 1 Atribute1;
    Type 2 Atribute2;
  public: /* Two aggregated SDL process */
    A Process_A;
    B Process_B;
  ....
};
```

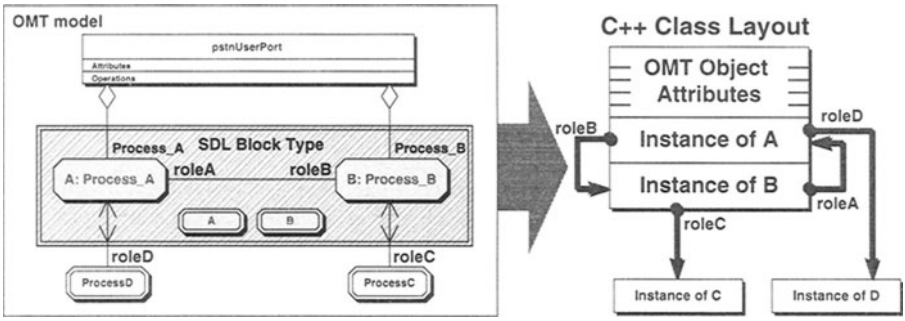


**Figure 3** OMT object with two aggregated SDL processes.

The only special property of members A and B is that these "Object/Process" are proclaimed to be SDL processes in the OMT model (**SDLclass** G-Template). Instead of generating C++ classes for B and A, the OMT code generation tool scans the roles of A and B and retrieves the SDL processes constructor signatures, in the example the SDL process formal parameters are:

```
PROCESS A FPAR roleB, roleD PID; ...
PROCESS B FPAR roleA, roleC PID; ...
```

which in turn are converted to C++ class constructors:

```
A::A(...,PId B,PId D);     B::B(...,PId A,PId C);
```

By now it is known that A and B implementations are generated from the SDL92 specification and B needs a pointer to A and vice versa. It is also known that the `pstnUserPort` object requires pointers to instances of D and C, otherwise A and B can not be correctly constructed. Since A needs a reference to B and vice versa, and the `pstnUserPort` object holds unique instances of both A and B the following code is automatically produced:

```
pstnUserPort::pstnUserPort(PId Dr, PId Cr) :
      Process_A(..., (PId) &Process_B, Dr) ,
      Process_B(..., (PId) &Process_A, Cr) { .... };
```

Using the described matching algorithm, the tool automatically generates constructors that initialises all SDL routes with both endpoints terminated within the container object. This is specially useful when creating complex objects that have a few internal processes connected by a number of SDL routes (OMT roles). Up to this point, the OMT model point of view has been assumed, the next section describes the `OMT_object` as viewed from the SDL process perspective.

### 2.1.2 Referencing OMT Objects from SDL

One SDL limitation often encountered by implementers, is the lack of shared variables between processes: *VIEW* and *REVEAL* constructs are discouraged and the *EXPORT*, *IMPORT* and remote procedure concepts rely upon SDL asynchronous signal exchange. In the described system it is necessary to periodically retrieve the status from a few hundred processes with minimum disturbance to the signalling activity. Conversely, the management

interface should not be affected by delays associated to overloaded signalling queues. In other words, while retrieving the system status the management processes should not be blocked by pending signalling operations. In this scenario shared variables are essential to decouple management and signalling processes.

The solution embraced is the use of a static configuration object tree shared by the management and signalling stacks. Each SDL process opens a "window" of shared variables into a part of the OMT object tree; therefore each SDL process has access to the attributes of the container object. In C++ this approach is readily realised by adding to some SDL processes a reference to the container object. This gives the following container object C++ constructor:

```
pstnUserPort::PstnUserPort(PId Dr, PId Cr) :
      Process_A((pstnUserPort &)(*this), (PId)&Process_B, Dr) ,
      Process_B((pstnUserPort &)(*this), (PID)&Process_A, Cr)
      { .... };
```

From the SDL perspective the shared "window" is a *NEWTYPE* structure that contains each attribute of the container object (father). Hence the SDL process formal parameters become:

```
NEWTYPE pstnUserPort
STRUCT
 Atribute1 Type 1;
 Atribute2 Type 2;
 ...
ENDNEWTYPE;
```

```
PROCESS A FPAR
  roleO pstnUserPort;
  roleA, roleC PID;
...
PROCESS B FPAR
  roleO ptsnUserPort;
  roleB, roleD PID;
...
```

Where *roleO* is the name of the role from which the SDL process views the container object (owner). From the SDL perspective the configuration is a structure passed by value, but actually the C++ implementation passes the value by reference. This reality is totally transparent to the SDL-92 to C++ translator; the translator simply maps the SDL "!" operator into a C++ "." member access. ADT operators are also employed to declare C++ member functions, the rule being that the leftmost operator specifies the object instance.

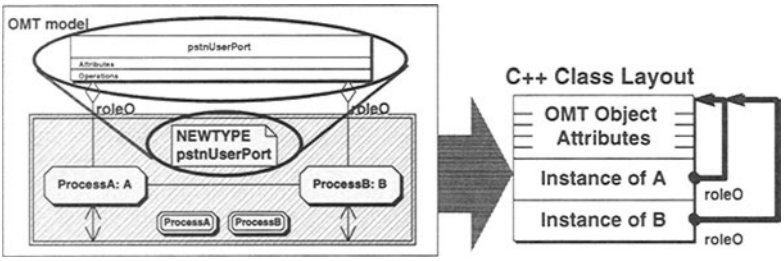


**Figure 4:** Referencing OMT configuration from SDL.

## 2.2 C++ GENERATION TOOLSET

When analysis and specification is based on two object oriented methodologies, it is natural to extend the O-O approach right through the implementation phase. In the field of embedded

systems, C++ still has a undeserved reputation for run-time inefficiency; yet, with wise usage and a good C++ compiler it is possible attain efficiency levels that match, or even surpass C based SDL implementations.

Commercial off the shelf tools are used to edit the OMT and SDL92 specifications (Verilog LOV and SDT3.02). However the SDT tool so far does not generate object oriented C++ code and the LOV support for SDL was found incipient. Therefore the project opted for a proprietary SDL-92 to C++ code generator and a "perl" script to generate the OMT C++ implementation. The complete translation tool chain is depicted in Figure 5.

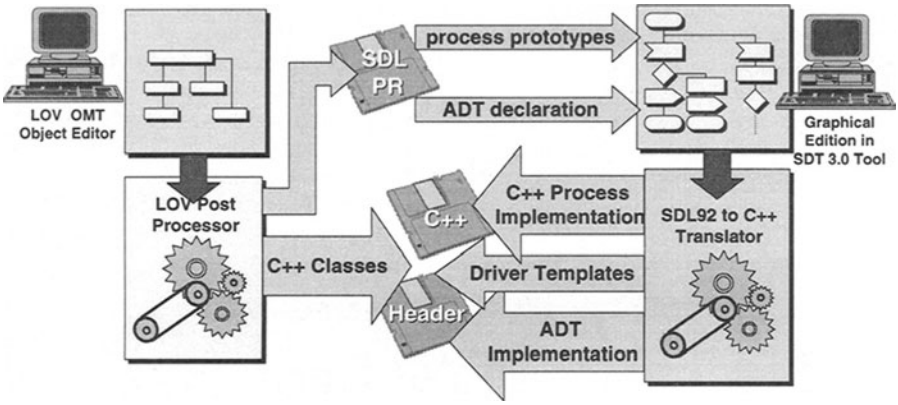


**Figure 5** - SDL/OMT to C++ tool chain.

The OMT model is translated into a set of C++ implementation classes, this set contains all classes that are not declared as SDL processes. Only the object model is used, therefore the OMT model does not directly implement behaviour. Instead the tool produces type and object declarations and instances of hand coded C++ templates (lists, hash tables, arrays, etc.). Besides the C++ declaration the LOV post processor also generates SDL ADT declarations that are incorporated into the SDL to ensure semantic correctness. Furthermore these ADT provide a interface that permits the manipulation of configuration database by the management processes. For example, an SDL process can manipulate object lists by calling operators ADT operators upon ADT instances. It also possible to generate SDL process prototypes, but after these processes are refined in the SDL tool it is not possible to revert changes back to the OMT model.

### 2.2.1 C++ SDL implementation

The tool used to generate C++ code from SDL92 is an evolution of a previous SDL88 tool [9]. This tool generates an object oriented SDL implementation: each SDL92 process type is translated into a C++ "active" implementation class and each SDL abstract type into a "passive" C++ class.

The SDL process data part is directly translated to C++ data members, along with formal parameters and timer objects. Each incoming SDL signal generates a homonymous member function with equivalent parameter signature. All signals with overloaded signatures call the same overloaded "output" function to encode the parameter and append the signal to the

process queue. The state machine implementation function consumes the signals at the queue head. All signal input and output member functions are automatically generated by the tool and a minimal library completes the support for SDL execution (signal memory management, process dispatching and SDL and target specific timer handling).

As can be seen in Figure 6, the SDL process implementation is a relatively self contained C++ class. Signal handling member functions define the external interface hiding the internal queuing and signal encoding details from the caller. From the process's body it is possible to invoke other SDL process or classes that provide a compatible interface but whose internal behaviour is not an SDL EFSM. Such is the case of the devices drivers that interface the SDL description to the surrounding hardware and software environment. These are SDL processes that have a empty state transition graph, the translator just generates a class declaration and omits the process body generation that shall be hand coded and liked to the generated code.

The use of empty driver processes appears to be more in line with O-O practice than the traditional SDL environment where all processes environmental signals are bundled together in a unstructured common space. The interface between SDL processes and drivers is potentially asymmetric: nothing implies that a driver queues the received signals (member function calls), therefore the process to driver interface is typically synchronous while the process to driver interface remains asynchronous.

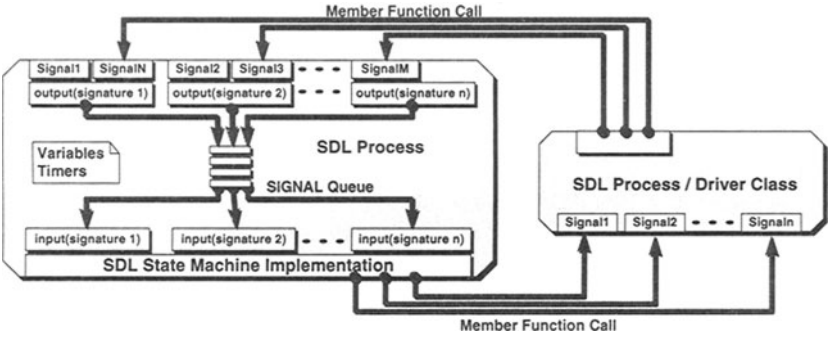


**Figure 6** SDL process implementation overview.

One of the key points of the translator tool, is the simplicity of the generated code, not only the interface is easily understood and handled by programmers not acquainted with SDL, but also the state machine implementation is effortlessly traced in the debug sessions. The efficiency is reasonable even when compared with typical hand coded implementations and while enjoying the benefits of O-O structuring most implementations are smaller that the SDT tool C implementation (about 70 % for the V5 Data Link process).

## 3. SYSTEM TESTING

The types of tests and methods used in the validation of this system are described below.

## 3.1 SYSTEM UNDER TESTING

The system considered for testing, as described in Figure 1, consists of a set of user ports that are mainly of two types: 1) analogue telephone, also known as PSTN; 2) ISDN basic rate access.

All the system interfaces are standardised. Analogue Z and digital layer one U interfaces, for the user side, the three layered V5 interface, for the exchange side, and the TMN Q3 standard, for management. Beside the remote control of user ports, V5 introduces also a digital and message oriented signalling protocol stack for the old telephone system calls.

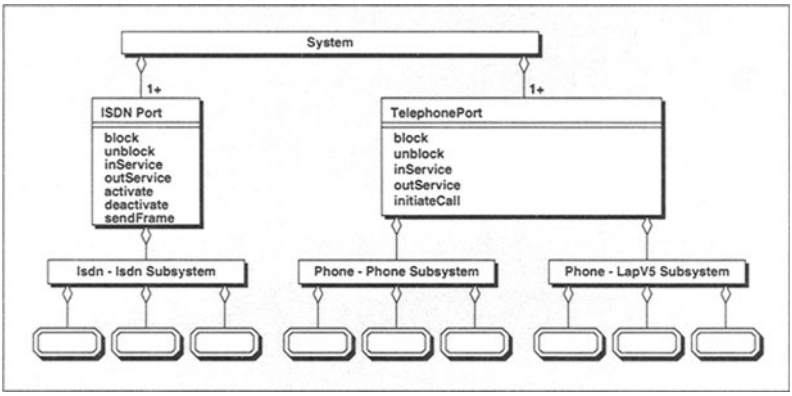


**Figure 7** - Testing view of the system

From a pure functional and testing point of view, presented in Figure 7, each user port can be requested to provide services both to the telecom operator or to a customer. The operator can, for instance, block a user port. The customer, in this model, can request a call initiation or, for ISDN, a frame transmission. Services are always requested through a system interface and, often, can also be invoked by more than one interface type. Putting a port out of service, for instance, can be requested directly through the management or V5 interfaces. During the provision of a service more than one interface can be used. A phone call, for instance, involves interactions through Z and V5 interfaces.

## 3.2 TYPES OF TESTS

According to the model presented above, and from the protocols point of view, two major types of testing were identified as required: interface testing and service testing.

### 3.2.1 Interface Testing

The interface testing main purpose is the verification of the interface conformity with the relevant standards. Although this type of tests, alone, does not guarantee full interoperability with other Local Exchanges vendors, it is required because it is used as the basis for

certification and it strongly limits the number of eventual mis-interactions of the system. Interoperability testing, assumed here as a field testing, is not discussed here.

Being Z and U, electrical and logical interfaces, only the V5 interface is related to protocol conformance testing. V5, layers 2 and 3, have test suites standardised and described in TTCN.

The conformance evaluation of each V5 state machine defines the usual types of conformance tests: 1) basic interconnection; 2) capability; 3) valid behaviour; 4) inopportune behaviour; 5) invalid behaviour; 6) timers and counters.

No formal methods [13] are known to be used in this test suite derivation, neither formal concepts, such as conformance relation, are introduced in the standards. Nevertheless, about 500 tests, for the AN side of the V5 interface, are specified on the main aspects of the protocol state machines.

For this type of testing, the following method has been selected: using the ITEX TTCN tool, tests are selected and parametrised in TTCN *mp* format. After that, the test suite or particular test cases is compiled for a SIEMENS protocol tester, where the tests will be executed. Remote single layer test method is the recommended and used.

### 3.2.2 Service testing

Since the major functions or services of the system are to be provided through the system V5 interface, conformance testing, in theory and if tests are well derived, indirectly checks the correctness of the main services provisioned.

Another class of tests, the load tests, concerned with detection of unwanted service interactions and with the characterisation of the quality of the services, need however to be also applied.

Service interaction tests [15] are intended to verify if a well-provided service is not affected by the provision of another services or instances of the same service. Quality of services tests are used to characterise, from a user point of view, the performance of the system under real usage scenarios. Parameters, such as probability of premature call release or HDLC frame transfer delay require statistical characterisation.

Unwanted interactions and quality characteristics usually deteriorate with increasing loads or service usage's. It means that a service correct under low loads, can begin to become incorrectly provided or with unacceptable quality under high or over load conditions.

The quality of the services, recommended by standardisation bodies for some network portions or elements, is defined by parameters that are rigorously associated with interfaces events, usually of layer three.

Quality of service parameters are about time intervals and behaviour quantification. Examples of the first type are call setup delay, call release delay, frame transfer delay and port blocking delay. Examples of the second type are misrouting and no tone.

For the first set, maximum mean durations and maximum durations for 95% of the usage cases are defined. For a call release delay, a mean value less that 250 ms and a 95% probability of not exceeding 300ms is recommended. Values are defined for several load conditions, being a load condition statistically characterised service usage mix. Poisson and deterministic service usage models are usually referred.

The second set of parameters is defined in terms of probabilities. As an example, the probability of a call attempt encountering no dial tone following the sending of a correct address shall be less than 0,0001.

## 3.3 LOAD TESTING ARCHITECTURE

Load tests require testing equipment that enables: i) the invocation of the most often used system services; ii) the invocation of these services by user selected sequences; iii) the statistical invocation of the services; iv) the characterisation of time intervals between events; v) the quantification of the behaviour; vi) high service usage rates.

As this system interfaces at the user side are electrical, some emulation facility was required for the testing system. For the Exchange side, some protocol emulation was also required in order to minimise the V5 stack complexity. With this emulation, the initial system has been modelled as a set of three subsystems types, also known as emulation subsystems, as represented in Figure 7: i) ISDN-ISDN; ii) Phone-Phone; iii) Phone-LAPV5. For each user port provisioned one of these subsystems types will be selected.

Each subsystem is modelled in SDL by three processes. Global real-time facilities are available within the overall testing system.

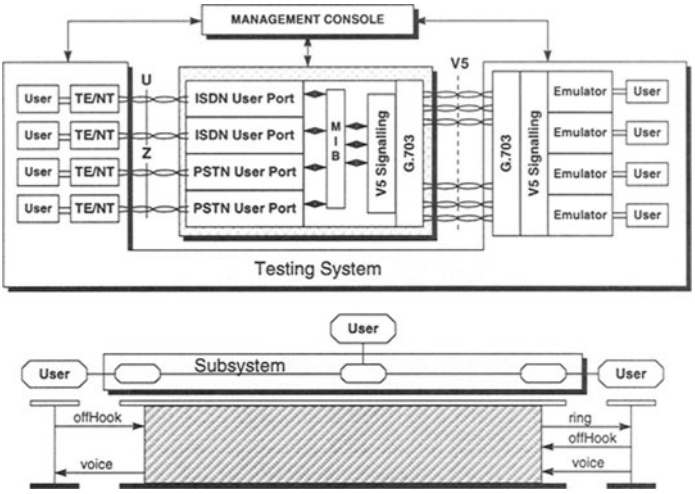


**Figure 8** Possible *initiateCall* service interactions in the Phone-Phone subsystem

## 3.4 TEST SPECIFICATION

In order to integrate test derivation methods with the combined OMT-SDL methodology described above, current MSC-SDL test derivation methods [14,15,16,17] for system and conformance testing were considered and complemented with service usage related specifications [18].

### 3.4.1 Test Purpose

Load test purposes are being specified in MSC. Two types of MSC were found required: i) usage MSC; ii) system MSC.

An *usage MSC* describes the service usage main trace and it intends to describe the users main behaviour. One of the users is assumed to trigger the service invocation regularly and, during service provision, other users can be involved.

For in order to allow the regular invocations, the *usage MSC* is complemented with the *interServiceInvocation* construct that characterises the time interval between successive service invocations by describing the statistical law to be followed, such as, Poisson, Uniform or Deterministic. MSC Timeouts can also be associated with statistical laws.

A *system MSC* describes a property of the system that needs *quantification*. Although users and system instances are also represented in a *system MSC*, the stress is put on the sequence of (constrained) messages that is to be observed as sent or received by the subsystem.

A *system MSC* can be complemented with the specification of a time interval needing to be *quantified*. Several *system MSC* can be associated with one *usage MSC*, as represented in Figure 9.

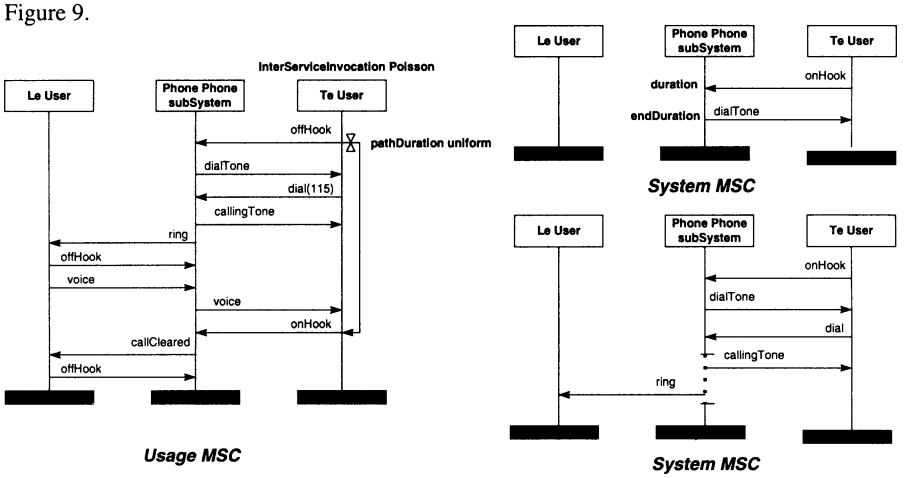


**Figure 9** Usage and System MSCs

### 3.4.2 Test case

Load test cases are specified using the combined OMT-SDL methodology described above. The automation of the process is, at the moment, being carried out.

From the *usage MSC* two SDL process types are derived, each one emulating a user. Typically, each user has always the same behaviour and one of them invokes the service according to the *interServiceInvocation* parameter.

Each *system MSC* is interpreted as one automata, as described in Figures 10 and 11, that observes and counts the events exchanged between the subsystem and its users.

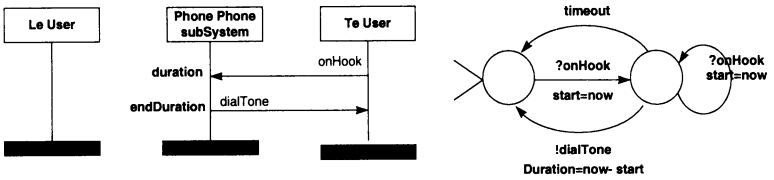


**Figure 10** Automata representing a *system MSC*, for duration quantification.

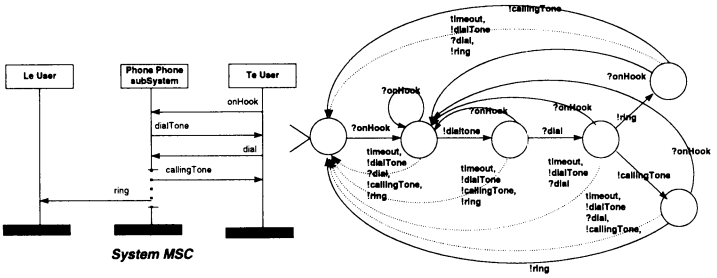


**Figure 11** Automata representing a *system MSC*, for behaviour quantification

The full set of *system MSC* are implemented in a SDL driver, see Figure 12, that routes the messages between the subsystem and its users. Drivers, as described in Figure 6, are synchronous with the processes that output signals for them.

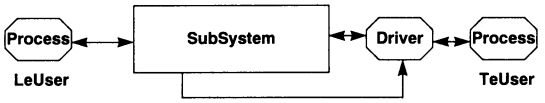


**Figure 12** Architecture of a Load Test Case.

*Test Load Veredicts*, unlike conformance testing, quantify the system by means of histograms data structures. Visual representation is presented in Figure 13.

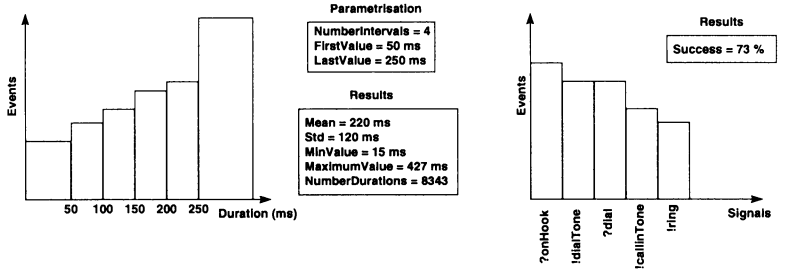


**Figure 13** Results of a load test case execution.

## 3.5 TEST EXECUTION

The test system needs to be configured before each test session execution is started. For this, each user port is instantiated with a SDL subsystem. Subsystems loaded with the same set of *usage MSC* and *System MSCs* can be grouped, thus emulating standard traffic/usage scenarios. For each group *usage MSC* and *system MSC* require parametrisation.

*Usage MSC* parametrisation consists in attributing values to the *interServiceInvocation* parameter and Timeouts.

*System MSC* parametrisation consists in selecting parameters for the duration histograms: numberIntervals, firstValue and lastValue.

## 3.6 RESULT ANALYSIS

Results of a load test session are mainly of two types: behaviour and time histograms.

**Behaviour histograms** quantify the success of an observation. The number of times that a signal has been observed, in the sequence defined by the corresponding *system MSC,* is recorded for each signal. In Figure 13, for instance, only 73% of the *offHooks* received by the subsystem from the user that invokes the service, have been completed with the desired *ring* indication. On the other side, this user has *dialled* a correct number every time he got the dialTone indication from the subsystem.

**Time histograms**, also shown in Figure 13, represent the distribution of the durations or delays requested in the corresponding *system MSC*. Mean delays and standard deviation values are obtained. Probabilities can be also evaluated from the histogram, since this graphic approximates a probability distribution function.

Confidence intervals for both histograms can also be quantified from the number of service invocations. The confidence on the results increases with the number of service requests and, for large confidence intervals, transformation of a relationship such as the 73% of ring success into a probability is permitted.

## 4. CONCLUSIONS

The combined use of OMT and SDL-92 allowed a design that can be traced back to the base standards. Improved maintenance and easy comprehension of the implementation are the approach key benefits. The O-O nature of the generated SDL implementation allows easy integration of SDL objects into the MO data structures. Customised OMT generation templates, allow efficient implementation of the configuration database and alarm management components.

Though OMT does not have defined semantics, and is often used exclusively in the analysis phase, it is also quite suitable for design as long as set of concrete mapping rules are established. This flexibility is the weakest and strongest point of OMT in design. It is weak because unlike SDL it is not possible to expect a well defined behaviour. On the contrary it is strong as it allows the implementation of functionality that falls outside the established SDL behaviour, such as: complex dynamic data structures, database access and specialised memory management.

ADT declarations allow access to OMT objects from the SDL process graph. This leaves much of the system behaviour inside the SDL description, without constraining the whole system to the SDL execution model overheads. Therefore a compromise between design formalism and efficiency was reached.

TTCN is the language used as the basis for conformance test. Automatic compilation and execution is achieved by means of commercially available tools.

Load testing, that is of major importance for this type of systems, is carried out in specially developed equipment. For that, conformance testing and verification techniques based on the combination of MSC with SDL, have been reused, improved with statistical characteristics, and implemented in the OMT-SDL methodology presented.

The definition of usage scenarios and parametrisation of test cases have also been shown. The validation of the system from the performance point of view, has also been addressed.

## 5. ACKNOWLEDGEMENTS

The authors would like to thank NEC corporation and INESC management for the permission to publish this work. The work of INESC personnel was in part funded by scholarships from JNICT. At last we would like to thank our work colleagues for they review and helpful suggestions.

## 6. REFERENCES

[1] ITU: Z.100 (1993): CCITT Specification and Description Language (SDL). ITU-T Jun. 1994.

[2] Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W.Lorensen: Object Oriented Modeling and Design. Prentice Hall, International Editions (1991).

[3] Witaszekm E. Holz, M. Wasowski, Stefanie Lau, J. Fisher, A development Method for SDL-92 Specifications Based on OMT, in SDL'95: with MSC in CASE, R. Braek and A. Sarma (eds), Elseiver Science Publ, (1995) pp. 103-113.

[4] F. Guo, T. MacKenzie Translation of OMT to SDL-92 in SDL'95: with MSC in CASE, R. Braek and A. Sarma (eds), Elseiver Science Publ, (1995) pp. 115-125.

[5] ETSI: ETS 300-324-1 Signalling Protocols and Switching (SPS); V interfaces at the digital Local Exchange (LE), V5.1 interface for the support of Access Network (AN), Part 1: V5.1 interface specification, ETSI (1994).

[6] ETSI: ETS 300-347-1 Signalling Protocols and Switching (SPS); V interfaces at the digital Local Exchange (LE), V5.2 interface for the support of Access Network (AN), Part 1: V5.2 interface specification, ETSI (1994).

[7] ETSI: ETS 300-376-1 Signalling Protocols and Switching (SPS); Q3 interface at the Access Network (AN) for configuration management of V5 interfaces and associated user ports, Part 1: Q3 interface specification, ETSI (1994).

[8] ETSI: ETS 300-378-1 Signalling Protocols and Switching (SPS); Q3 interface at the Access Network (AN) for fault and performance management of V5 interfaces and associated user ports, Part 1: Q3 interface specification, ETSI (1994).

[9] E. Inocêncio, M. Fonseca, SDL to C++ translator for ISDN Basic Rate Terminal Signalling, in SDL'93: Using Objects, O. Faergemand and A. Sarma (eds), Elseiver Science Publ, (1993) pp. 353-360.
[10] A. Gillespie, Simon Rees, Access Network Management Modelling, IEEE Communication Magazine, March 1996 Vol 34 No 3 (1996).
[11] ETSI: ETS 300-324, Parts 2 to 8, V5.1 Conformance Test Specifications, ETSI (1994).
[12] ETSI: ETS 300-347, Parts 2 to 8, V5.2 Conformance Test Specifications, ETSI (1994).
[13] D. Hogrefe, Status Report on the FMCT Project, in Proceedings of the 7th International Workshop on Protocol Test Systems, Tokyo, 1994, pp. 165-180
[14] B. Algayres, Y. Lejeune and F. Hugonnet, GOAL: Observing SDL behaviours with Geode in SDL'95: with MSC in CASE, R. Braek and A. Sarma (eds), Elseiver Science Publ, (1995) pp. 223-230.
[15] P.Combes, S. Pickin, B. Renard, F. Olsen, MSCs to express Service Requirements as properties on a SDL model: Application to Service Interaction Detection in SDL'95: with MSC in CASE, R. Braek and A. Sarma (eds), Elseiver Science Publ, (1995) pp. 243-255.
[16] J. Grabowski, S. Hogrefe, I. Nussbaumer, A. Spichiger, Test Case Specification Based on MSC and ASN.1 in SDL'95: with MSC in CASE, R. Braek and A. Sarma (eds), Elseiver Science Publ, (1995) pp. 307-322.
[17] S. Leue, Specifying Real-Time Requirements for SDL Specifications - a Temporal Logic Based Approach, in Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification, Warsaw, 1995, pp. 19-34.
[18] P. Gunningberg et all, Application Protocols and Performance Benchmarks, IEEE Communications Magazine, June 1989.

## 7. ABREVIATIONS

ADT ..... Abstract Data Type.
AN..... Access Network.
ASN.1..... Abstract Syntax Notation One.
FSM..... Finite State Machine.
EFSM ..... Extended Finite State Machine.
ETSI ..... European Telecommunication Standards Institute.
HDLC..... High Level Data Link Control.
LAPV5 ... Link Access Procedure for V5.
ISDN ..... Integrated Services Digital Network.
MO ..... Managed Object.
MIB ..... Management Information Base.
MSC ..... Message Sequence Chart.
OAMP.... Operation Administration, Maintenance and Provisioning.
OMT ..... Object Modeling Technique.
ONP ..... Open Network Provisioning.
O-O ..... Object Oriented.
PId..... Process Identifier.
PSTN ..... Public Switched Telephone Network.
QOS ..... Quality of Service.
SDL..... Specification and Description Language.
TMN ..... Telecommunication Management Network.
TTCN..... Tree and Tabular combined Notation.