Use of SDL for the implementation of Abstract Test Suites

C. Kavadias*, G. Thanos*, N. Ploskas*, G. Gerolimos*, V. Kollias*, G. Stassinopoulos** and M. Loupis***

*TELETEL S.A; 124, Kifisias Avenue, Athens, GREECE
Tel: +30 210 6983393; Fax: +30 210 6983391;

**National Technical University of Athens (NTUA)
Dept. of Electrical and Computer Eng.
9, Heroon Polytechniou Str., 157 73, Zographou, Athens, GREECE
Tel.: +30 210 7721493, Fax: +30 210 7722534

***SOLINET GmbH Solutions for Innovative Networks
Mittlerer Pfad 26, 70499 Stuttgart, Germany
Tel +49 711 1398 13-0, Fax +49 711 866 1240
E-mail: C.Kavadias@teletel.gr

Abstract: This paper describes the use of the Specification and Description Language (SDL) for the implementation of Abstract Test Suites (ATS). The Tree and Tabular Combined Notation (TTCN) is the standard language for specification and implementation of validation tests for signalling systems. On the other hand, SDL is a Formal system Description Technique (FDT), which is widely used in the telecommunications field for the specification and implementation of signaling systems. This contribution first provides a description of the main characteristics of Finite State Machines (FSMs) (implemented in SDL) as well as the main characteristics of the test suites (implemented in TTCN) used for their validation. In the sequel, the common characteristics of the FSMs and their validation test suites are identified and finally a top-level comparison of common terms in SDL and TTCN is provided. A roadmap for the use of SDL in signaling systems testing is also provided.

Keywords: FDTs, SDL, TTCN, testing, TTCN and SDL overlap

1. Introduction

The descriptive and formal specification of the behaviour of reactive information processing systems is of high relevance in many technical applications. Typical application areas are telecommunication and embedded reactive systems as they are used for instance in automotive or avionics applications. For these systems often a complex behaviour is required which includes a close co-operation between the system and its environment. In general, hard or at least soft real time aspects are involved.

As it is generally agreed, specifying such systems cannot be done by a verbal description runs into major difficulties. Verbal descriptions tend to be lengthy,
incomplete, to contain phrases that may be misinterpreted, and to be not well structured. Moreover, they do not follow any description standards. Therefore formal description techniques (FDTs) have been suggested as a better way to describe complex reactive systems with respect to their information processing behaviour.

Typical examples for such FDTs are, among many others, SDL, Esterel, statecharts, Lotos, Lustre, and Estelle. They provide a graphical and/or textual syntax for the description of reactive systems.

FDT is a formal method for developing telecommunications services and protocols. FDTs range from abstract to implementation-oriented descriptions. All FDTs offer the means for producing unambiguous descriptions of OSI services and protocols in a more precise and comprehensive way than natural language descriptions. They provide a foundation for analysis and verification of a description. The target of analysis and verification may vary from abstract properties to concrete properties. Natural language descriptions remain an essential adjunct to formal description, enabling an unfamiliar reader to gain rapid insight into the structure and function of services and protocols.

FDTs are used for several purposes. Their main use is seen in providing a description of the behaviour of complex reactive systems. Such descriptions are needed in several phases of the development of a system. These phases comprise roughly requirements capture, design of the system architecture and implementation of the system components.

2. Background

2.1. Specification and Description Language (SDL)

*Specification and Description Language* (SDL) is a general-purpose description language for event-based systems that is widely used from major manufacturers in the telecommunication industry, standardised as ITU (International Telecommunication Union) Recommendation Z.100.

The basis for description of system behavior is extended *Finite State Machines* (FSM), which consists of a number of *states* and a number of *transitions* between the different states. Extended FSMs are presented by SDL *processes*.

A process is an autonomous processing entity that interoperates with other processes through standard interfaces called *signals*. A process has an infinite input queue, where incoming signals are queued and it is either in a state waiting, or performs a transition between two states. A transition is initiated by the first signal in the input queue. When a signal has initiated a transition, it is removed from the input queue (and is said to be consumed). In a transition, *variables* can be manipulated, *decisions* can be made and signals can be sent to other processes.

A process works autonomously and concurrently with other processes. Some aspects of communication between processes are closely related to the description of *system* structure. The co-operation between the processes is performed asynchronously by
discrete messages, called *signals*. The conveyance of signals between two communicating processes is managed with *signal routes* (Figure 1).

---

**Figure 1: Structuring a block into process types**

SDL is a structured language. The aim of structuring is to cope with complexity. The process concept is suitable for structuring and must always be contained in a *block*, which is the main structuring concept. A system contains one or more blocks, interconnected with each other and with the boundary of the system by *channels*. A channel is a means of conveying signals. Figure 2 depicts the structure of an SDL system. A system can be regarded as a special kind of block, having no channels connected to it from the outside.

---

**Figure 2: Static structure of an SDL system**

The key features of the language are:

- Suitability for real-time, stimulus-response systems
- Presentation in a graphical form
- A model based on communicating processes (extended Finite State Machines)
- Object oriented description of SDL components.
- The ability to be used as a wide spectrum language from requirements to implementation.
2.2. Conformance Testing basic concepts

Conformance testing involves testing both the capabilities and behaviour of an implementation, and checking what is observed against the conformance requirements in the relevant ITU-T Recommendations or International Standards and if appropriate in the related International Standardized Profiles and against what the implementor states the implementation capabilities are.

Conformance testing does not include assessment of the performance nor the robustness or reliability of an implementation. It cannot give judgements on the physical realization of the Abstract Service Primitives, how a system is implemented, how it provides any requested service, nor the environment of the protocol implementation. It cannot, except in an indirect way, prove anything about the logical design of the protocol itself.

The purpose of conformance testing is to increase the probability that different OSI implementations are able to interwork. However, it should be borne in mind that the complexity of most protocols makes exhaustive testing impractical on both technical and economic grounds. Also, testing cannot guarantee conformance to a specification since it detects errors rather than their absence. Thus conformance to a test suite alone cannot guarantee interworking. What it does do is give confidence that an implementation has the required capabilities and that its behaviour conforms consistently in representative instances of communication.

In the context of OSI, a real system is said to exhibit conformance if it complies with the requirements of applicable OSI specifications in its communication with other real systems. Applicable OSI specifications include those that specify protocols or protocol profiles, and those that specify abstract syntaxes and encoding rules or information objects (in so far as they are implemented in conjunction with protocols). Conformance to a profile specification should always imply conformance to the relevant set of base specifications.

2.3. Tree and Tabular Combined Notation (TTCN)

IUT has defined the ISO 9646 standard, which specifies the framework for the validation and conformance testing of open systems. The Part 3 of ISO 9646 Methodology describes the Tree and Tabular Combined Notation (TTCN), which is the standard language for specification and implementation of validation tests for communication protocols used by all major telecom manufacturers and operators. All major new standards for communication protocols are accompanied by an Abstract Test Suite (ATS) specified in TTCN.

A TTCN ATS consists of four parts that are organised in a tree structure.

- The Test Suite Overview provides information about the ATS such as test case grouping of an index for test cases, test steps and defaults.
- The Declarations Part contains all definitions and declarations required by the test suite such as Protocol Data Units (PDUs), Points of Control and Observation (PCOs) or test suite variable declarations for example. Next to the TTCN specific notation the
Abstract Syntax Notation Number One (ASN.1), forms part of TTCN for the declaration of elements.

- The Constraints Part attaches individual values, sets of values or a range to declare elements like PDU fields. Like in the declarations part, ASN.1 can also be used to specify constraints of an ATS.
- The Dynamic Part describes the behaviour, which means it specifies when a message is sent and at which time a message is expected to be received. It is split into three sub-sections, test cases, test step library and defaults library.

For the transition from an ATS to an Executable Test Suite (ETS) a TTCN compiler is required. The TTCN compiler produces code that can be executed on the test platform using a simulator.

3. SDL-TTCN mapping

A system can be defined to be a combination of a state machine and its validating test scenarios. There has been extensive research in this area and several international specifications defining various aspects for implementation, validation and data types.

SDL is an international standard (Z.100) for the specification of state machines. It is a formal language and suitable for execution. Communication is using an abstract concept of signals, which can be sent and received. Signals can have parameters. There are several flavours of SDL reflecting the evolution from 1988 to present.

TTCN is an international standard for the specification of test scenarios. It is a formal language and suitable for execution. Communication is using data packets, which are defined down to bit-level. Data packets can be sent and received using ASPs (Abstract Service Primitives), a concept based on the OSI model for layers to communicate with each other.

Although signalling systems are implemented as state machines and validated using test scenarios, there is no apparent common basis to the SDL and TTCN specifications, which were written by different research committees with limited contact to the industry.

Assuming that the requirements to implement and validate systems must have at least some overlap, there arises the question of the chicken or the egg. Should system implementation be based on the same mechanism as system validation, or vice versa or a complete mix of both? As the system implementation is the most natural reference point, the requirement to specify test scenarios should follow. As a matter of interest the SDL specification is generally speaking based on much cleaner and abstract concepts, making this selection easier.

3.1. Characteristics of State Machines

State machines have different requirements to other applications, such as databases, user interfaces and scientific calculations. Some researchers may try to implement databases as state machines, to implement state machines as databases or implement
state machines as low-level formatted communication and state transitions without using these basic principles.

In all cases there may be some interesting outcome, but the mainstream characteristics of state machines and the principles to implement them remain unchanged. Starting from the implementation of state machines, the basic principles can be highlighted.

a) Dedicated Purpose: A state machine is a single, specific entity able to send and receive signals; the signals sent depend on the signals received and the order they are received in. A state machine has a dedicated purpose.

b) Execution: A system is started and typically runs continuously until it is stopped by the user; for example a telephone, a router, a television.

c) Hierarchical: A state machine can be defined hierarchically in terms of other state machines, which are connected together or to the environment. A complex state machine can typically be broken down into a collection of simpler, communicating state machines.

d) Abstract Communication: State machines can communicate via connections using signals; the formatting of the communications signals is not relevant, only that they are uniquely identifiable and have parameters. Communication with objects external to the environment clearly have to have a defined format, especially for communication with telecoms sub-systems with bit-level formatting. Typically this formatting can be isolated in codecs (coders/decoders) which handle the conversion from external format to the abstract internal format.

e) State Transitions: A system has a starting state and a set of defined states. A state transition can only occur, but does not need to occur, when a signal is received.

f) Time: Time is measured using timers, which are started with a specified duration and result in a timeout signal on expiring. Timers can be defined, started and stopped.

g) Variables: A state machine can have local variables. There is a set of mathematical operations that can be carried out on them.

h) Data Types: State machine variables and signal parameters are defined with data types.

i) Decisions: Decisions can be made based on values of expressions and defined alternatives.

j) Graphical Presentation: State machines based on these principles are suitable for graphical representation in a flow chart.

3.2. State Machine Validation

An unvalidated state machine is of limited use; generally speaking there is great interest from industrial users not only to validate systems, but also to measure and document the degree of validation. Tests should be predictable, deterministic and reproducible.
Whilst this interest applies generally to all kinds of software applications, even to all machines in general, state machines can be implemented using well defined and strictly limited mechanisms that allow a high degree of automated checking.

In particular, state machines are self-contained entities that communicate entirely using signals. A straightforward approach to validating a state machine can be the 'black box' method, namely to isolate it and check the external signalling. If the external signalling is correct then the internal implementation must also be correct. Of course this makes some assumptions, namely that the external signalling can be checked for all possibilities, something that is generally not practical, but it is at least a formal foundation to build on.

### 3.3. Characteristics of Test Suites

A state machine can be validated by providing controlled inputs and observing the corresponding behaviour (outputs). For validation the state machine under test, also known as the test object, has to be isolated and connected to a simulator, which is a device able to provide inputs and observe the outputs for each of the connections.

The set of different tests is a test suite.

a) TSS/TP: The validation process has to identify a set of different input/output sequences suitable to confirm the intended behaviour of the state machine. Each sequence is called a scenario and has a defined test purpose (TP). The collection of scenarios is called a test suite and has a defined structure (TSS, test suite structure).

b) Execution: Normally scenarios can be run in isolation or groups of scenarios run consecutively. Each scenario will stop when its sequence has finished.

c) Verdicts: Each scenario executed results in a verdict, for example Pass or Fail.

d) Hierarchical: The test sequence of a scenario can be defined hierarchically in terms of other sequences. Also specific scenarios may have asynchronous sub-sequences running in parallel, typically for different connections. A complex scenario can typically be broken down into a structure of simpler sequences communicating with internal events.

e) Abstract Communication: Scenarios need to communicate with the test objects, which may be state machines communicating via connections using signals, but may also be external bit-level formatted data packets.

f) Message Sequences: A scenario has a starting point and a set of defined messages to send/receive. As each expected message is received, the next messages can be sent or further messages expected as defined in the sequence, i.e. progress is made through the sequence until the sequence has finished.

g) Constraints: Generally speaking, the messages received can be checked both as the correct message and with the correct parameters. The specification of the message and the expected parameters is called a constraint. However in many cases the specific values of specific parameters may not be relevant and may not even be
predictable. In this case the constraint can be enhanced with wildcards or additional information, such as range of values allowed.

h) Time: Typically timers are used to measure the response time of the test object and ensure it lies within specific boundaries. A timer can be started when a specific message is sent to the test object and stopped when the expected answer is received. A timeout indicates that test object was too slow. Alternatively, if the test object has to wait a specific time before answering, then no timeout means the test object was too fast.

i) Variables: A scenario can have local variables. There is a set of mathematical operations that can be carried out on them.

j) Data Types: Scenario variables and signal parameters are defined with data types.

k) Decisions: Decisions can be made based on values of expressions and defined alternatives.

l) Graphical Presentation: Scenarios based on these principles are suitable for graphical representation in a flow chart.

3.4. Overlap

Comparing the characteristics of state machines and test suites, the main differences represent enhancements needed for validation, namely;

1) Organisation of specific scenarios into test suites.
2) Verdicts for executed test scenarios (test report).
3) Constraints
4) Message sequences

On the other hand, there is a large overlap in the common characteristics, implying that the same concepts, methods and mechanisms can be used for both implementing state machines and validating them.

The following table depicts a top-level comparison of common terms in SDL and TTCN.

<table>
<thead>
<tr>
<th></th>
<th>SDL</th>
<th>TTCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate</td>
<td>PCO &amp; Coordination Point</td>
<td></td>
</tr>
<tr>
<td>Signal</td>
<td>ASP Type/PDU Type</td>
<td></td>
</tr>
<tr>
<td>Signal List</td>
<td>Abstract Service Primitives, PDUs &amp; Coordination Messages</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td>Timer</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td>Operations (+, -, etc)</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Test Component</td>
<td></td>
</tr>
<tr>
<td>Procedure</td>
<td>Test Step</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>&quot;?&quot;</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>&quot;!&quot;</td>
<td></td>
</tr>
<tr>
<td>Join &amp; Label</td>
<td>Goto &amp; Label</td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>If/Qualifiers</td>
<td></td>
</tr>
<tr>
<td>Save Gates</td>
<td>PCO Blocking</td>
<td></td>
</tr>
<tr>
<td>MIBs</td>
<td>MIBs</td>
<td></td>
</tr>
<tr>
<td>Procedures</td>
<td>Test Suite Operations</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>Constraint // SDL only matches on the signal, not parameters</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>Verdict</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>TSS/TP</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>Message sequences</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>PICS</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>Selection Expressions</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1: Top Level Comparison of Common Terms in SDL & TTCN**

4. **Conclusions**

This contribution provided the main concepts and characteristics of the SDL language, which is suitable the design and implementation of signaling systems as well as the main concepts and characteristics of the TTCN language, which is suitable for the specification and implementation of validation tests for signaling systems.

The top-level comparison of common terms in SDL and TTCN has shown that there is a large overlap in their common characteristics, implying that the same concepts, methods and mechanisms can be used for both implementing state machines and validating them. Another interesting point is that constraints and message sequences could be meaningful enhancements for implementing state machines.
To conclude, the analysis in this contribution indicates that a formal system description technique such as SDL can be used for both the design and implementation of signaling systems as well as the implementation of the validation test suites for them, if the appropriate enhancements will be performed.

5. References

[6] INTERVAL IST 11557 Deliverables