Architecting Specifications for Test Case Generation[1]

Richard Sinnott
National e-Science Centre
University of Glasgow
Scotland
ros@dcs.gla.ac.uk

Abstract

The Specification and Description Language (SDL) together with its associated tool sets can be used for the generation of Tree and Tabular Combined Notation (TTCN) test cases. Surprisingly, little documentation exists on the optimal way to specify systems so that they can best be used for the generation of tests. This paper, elaborates on the different tool supported approaches that can be taken for test case generation and highlights their advantages and disadvantages. A rule based SDL specification style is then presented that facilitates the automatic generation of tests.

Keywords: SDL, TTCN, Test Case Generation.

1. Introduction

The Specification and Description Language (SDL) [1] is used in major corporations for different purposes. We note that the acronym SDL is to be known as Systems Design Language in future. On the one hand SDL allows for abstract specifications to be made which can aid in the capturing and understanding of requirements for given products. Such abstract descriptions are typically part of the *early phases* of product development. On the other hand SDL allows for more detailed implementation oriented specifications to be described which capture very low level design aspects. We note that these two areas of application of SDL are not necessarily orthogonal, but can be applied in an iterative development strategy.

SDL and its associated tool support also allow for the automatic generation of tests, specifically tests given in Tree and Tabular Combined Notation (TTCN) [2]. We note that the acronym for TTCN has been changed by the relevant standards bodies to Testing and Test Control Notation due to the differences between TTCN-2 and the new standard TTCN-3 [3]. In this paper we consider only TTCN-2.

Relatively little literature exists on the optimal way to architect specifications in order to best use them for test case generation [7,8]. This paper summarises the different possible approaches currently available for test case generation (TCG) using the Telelogic TAU SDL and TTCN tool environment [6], and then shows how a specification style can be used to improve the test generation process. A rule based specification style is presented that helps to overcome some key issues in the automatic generation of tests from SDL models.

The rest of the paper is structured as follows. Section 2 provides an account of SDL usage in the early phases of product development. Section 3 presents the application of SDL and its associated tools with emphasis on test case generation. Section 4 focuses on a specification styles that can be exploited by tools for test case generation. Finally, conclusions on the work are given in section 5.

2. Early Phase Development with SDL

Before we discuss how SDL and its associated tools can optimally be applied in the early phases of product development, it is worth clarifying exactly what is meant by "early phases" as they pertain here. It could be argued that even the most complex implementation oriented SDL model falls into the general category of early phases of product development, since we are dealing primarily in the specification domain.

We regard *early phase* SDL models here as models containing that level of specification which allows to capture the "what of a product", i.e. to capture the basic functionality without specifying exactly how this will be realized. Typically such models are used to precisely capture and analyze requirements on the expected functionality of the system. One indirect way that such a classification can be ascertained is through the intended usage of the SDL model. If the SDL model is to be used for understanding the system to be developed, then the model can generally be classified as in the *early phase* category of product development. If however, the model is to form the basis for the actual product implementation, i.e. code will be generated directly from the model which, once compiled etc will represent the final product, then we should no longer consider the model as being in the *early phase* of product development. We refer to such implementation oriented models as (surprisingly!) *non-early phase*.

The distinction between *early phase* and *non-early phase* models has numerous repercussions on the strategies for applying SDL and its associated toolsets. It could be argued that TCG is only useful (or

[1] This work was undertaken whilst the author was working as a consultant at Ericsson.

meaningful!) if we consider *early phase* SDL models, since tests generated from *non-early phase* models only allow for some form of sanity check on the associated test case generation or code generation tools to be performed. Thus, if an SDL model is used to generate tests and the same model is used to generate the actual implementation then the subsequent execution of the generated tests, serve no purpose other than to check that the test case generation tool and code generation tool function correctly – or at least that they are consistent with one another. As a result, we consider the application of SDL and its associated tools here for *early phase* development only.

In addition, it is also useful to classify the role of testing in these phases. It might be considered that certain aspects related to the general testing process itself are non-early phase, i.e. testing is typically done once an existing implementation exists, as opposed to when more abstract requirements capturing models are being developed for example. Consideration of testing in the early phases of product development is a crucial part of the product development process however. Knowledge of how to test a product should be a fundamental part of the overall development of that product. This knowledge can be in the form of general understanding of how to test a given product or more concretely through the explicit tests that a given product is supposed to be able to handle.

3. Overview of TCG Approaches

To understand the impact of TCG on the development of SDL specifications in early product development it is necessary to have an understanding of the different approaches that are taken by existing tools. We present three approaches to TCG: simulation based; MSC based and rule based TCG.

3.1 Simulation Based TCG

This approach is based upon the interactive exploration of the behaviour of a given SDL model. Tools such as TTCNlink [6] allow for the static external information, i.e. the information used in testing the external behaviour of the system, of the SDL model to be automatically generated. For example, points of control and observation (PCO), abstract service primitives (ASP) or protocol data units (PDU) as well as the data types associated with these ASP/PDU’s can be generated automatically through the external channels to the environment (PCOs); the associated signals on those channels (ASPs/PDUs); and the types of the parameters associated with those signals respectively (ASP/PDU data types). Default failure test case tables are also generated automatically.

The dynamic part of the test cases is then generated through synchronizing test case tables (corresponding to the test case being developed) with the SDL model and interactively exploring the behaviour of the specification. In order to create the dynamic part of the tests, it is necessary to create constraints, i.e. the values, associated with these data types on the input signals from the SDL environment (TTCN send events). We note that these values have to be manually input and are not generated automatically. Once the necessary constraints have been created, it is then possible to perform the sending and receiving of TTCN events/SDL signals respectively, i.e. generating the dynamic part of the TTCN test suite.

The TTCN send events are automatically placed into the associated table under development. Following their reception and consumption by the SDL model, the possible receive events generated through exploring the state space of the model, i.e. the outputs of the SDL system, are dynamically inserted into the TTCN table being created. The constraints associated with these receiving events are established dynamically based upon the outputs from the SDL system.

3.1.1 Pros/Cons of Simulation Based TCG

This approach has numerous advantages for TCG. Firstly, the tester has considerable flexibility in the development of the test cases. That is, they can decide which combinations of signals and data should be sent to the system. It is also the case that the tests developed correspond to valid traces of the SDL model.

From an SDL modellers perspective, this approach does not require any significant specification styles or conventions to be followed which TCG tools can exploit. There are some limitations in the current tools that the specifier should be aware of however, when using this approach for TCG. For example, there are some limitations on the data types, which can be automatically generated from the SDL system, e.g. it is problematic to generate tests from SDL models where SDL process identifiers (PIds) are passed as parameters in interactions with the environment.

The approach is not without its drawbacks however. For example, the development of tests with this approach is a laborious and time-consuming process. This is especially so when the constraints associated with the data are non-trivial or the SDL model has numerous traces which representing valid and interesting behaviours. This approach also requires that the test creator has considerable knowledge of the model, e.g. the test case creator may well have to know the low level behaviour of the SDL model in order for successful and meaningful tests to be generated. Such a white box approach may not always be possible, e.g. if the test case generator did not create the SDL model.

Another limitation with this approach is that it does not allow for erroneous behaviours or erroneous data to be handled; yet often these are very much of interest to the tester when testing the robustness of the software. Thus for example, “bad data” cannot be input since the tools will not allow for this. Similarly, the behaviour of the SDL model normally expresses desired behaviour, e.g. an ordering of signals is implicitly given in the model. Stimulating the model with signals that violate this ordering will generally result in those signals simply being dropped and no subsequent responses being returned to the environment (the test case table).

Another issue with this approach is that it does not allow for information related to how much of the model has been explored. The tests generated might just cover a subset of the overall behaviour of the SDL model. Furthermore, this approach does not allow to discover other interesting testing behaviours; either the tester sends the appropriate messages together with the appropriate data values and receives the appropriate responses or not. The approach does not lend itself to the automatic discovery of such new behaviours.

3.2 MSC based TCG

The primary idea behind this approach is to use Message Sequence Charts (MSC) [4] to express the dynamic behaviour of the test case and translate this directly to TTCN. This approach is often used together with other approaches. For example, it is often the case that the static information associated with a test case, e.g. the PCOs, ASP/PDU and data types are generated from an SDL specification and the MSC then used to produce the dynamic part of the test case.

This approach can be applied at various phases of product development. One example of MSC based TCG is the situation where requirements expressed through MSC interaction scenarios are given without a detailed SDL model and these MSC should be used to produce test cases. In this case, a minimal SDL model can be produced that is used to provide the static testing information, e.g. via tools such as TTCNlink discussed in section 3.1. The MSC can then be translated to TTCN via appropriate tool support. This minimal model will likely define the same external interfaces as the real system, i.e. the same channels, signals and data types will be supported; however, the detailed specification of the SDL behaviour can be omitted. Such a system would likely contain processes with null behaviour for example.

Alternatively, a more rigorous application of MSC based TCG would be to have a detailed SDL model through which manually provided interaction scenarios (given by MSCs) can be verified and subsequently used to generate test cases. The rigour in this sense stemming from the understanding of the system being developed, i.e. it can be verified that the MSCs represent valid system traces.

A third and arguably more powerful application of MSC based TCG would be to have MSC generated directly from an SDL model and subsequently converted to TTCN. Through this approach, the MSC are not expected to be provided manually as requirements based interaction scenarios, but they can be generated automatically from the model. We note that this approach and the manually provided MSC approach can and will likely be used in conjunction with one another.

3.2.1 Pros/Cons of MSC Based TCG

Perhaps the greatest advantage with MSC based TCG is the speed at which the tests can be generated. Interaction scenarios as might be produced during requirements analysis can be converted to tests cases with minimal specification effort. Of course, one of the requirements to achieve this are that the MSC specifications are syntactically correct and the associated minimal SDL specification correctly reflects the static aspects of the system under development.

The approach can be applied at different phases of the product development lifecycle. For example from the early requirements capturing phase in which testing aspects are also being considered, through to the later phases where detailed models have been produced and being used for TCG.

This approach is not without its drawbacks however. The approach in combination with a minimal SDL specification provides no guarantee that the interaction scenario as given in the MSC is a valid interaction scenario of the real system, nor does it allow for test coverage to be ascertained. Manually producing MSC based interaction scenarios whilst lending itself to a broad understanding of system behaviour, rapidly becomes unwieldy once detailed data considerations are considered both from a specification point of view as well as an understanding point of view, i.e. detailed data aspects represented in an MSC make the MSC more difficult to read and understand.

It could even be argued that MSC based TCG is not actually TCG in its purest form. That is, it is not the case that the MSCs are really being used to generate test cases as such, but rather they are simply an alternative representation of the test case behaviour. Hence this approach is more a notation conversion based approach rather than a TCG based approach. Nevertheless, this approach can be used in conjunction with other approaches and exploited by tools that allow for MSCs to be generated automatically. One such approach is rule-based TCG.

3.3 Rule Based TCG

Ideally an SDL model should be used directly for test case generation which should allow for the detailed behaviour of the model to be explored and used as a basis for generation of test cases. There should ideally be no need for manual intervention as is required for simulation based TCG and to a lesser extent through MSC based TCG. Rather, it should be possible for an SDL specification to be provided and tool support subsequently applied in generating tests with little or no tester knowledge about the inner-workings of the specification, i.e. black-box testing should be assumed.

The automatic generation of exhaustive tests from (SDL) models is still very much an area of active research. Given the complexity of realistic SDL models and data, i.e. those models where test case generation would be useful as opposed to minimal proof of concept case studies, it is unlikely that a solution for automatic and exhaustive TCG will be achieved. State space explosion is an ever present problem especially in the presence of complex environment interactions with non-trivial data as is typically the case in the telecommunication domain.

An alternative to exhaustive TCG based on rule based TCG can also be applied. Before considering this in detail however, it is worth considering the key issues to be overcome with exhaustive TCG, since this offers the most powerful and useful approach. There are at least three key issues which have to be addressed to perform automated TCG from an SDL model:

- avoiding the problem of state space explosion;
- identifying new and interesting system traces;
- dealing with erroneous behaviours and data.

To address these issues it is necessary that care and foresight be applied in the development of the SDL model. Specifically, the specification should be developed in such a way that state space explosion problems can either be minimized, or optimally, avoided altogether; new and interesting system traces can be discovered and recorded; erroneous behaviours can be catered for. One way in which this can be realized to a certain extent is through a rule based specification approach.

Rule based specification development requires that the specification be developed in a manner so that the associated TCG tools can exploit it. More precisely, the specification is written in such a manner that it is possible to check for the satisfaction of assertions on the state of the specification. Rules can be given which allow for checking of these assertions during the exploration of the behaviour of the SDL model. Rules can for example, be provided which relate directly to TCG and the key problems associated with automating this, namely: avoiding state space explosion, identifying new test purposes, dealing with erroneous behaviours. We note here the importance of the "early phase" role of SDL in the development process. That is, dealing with the early phase development allows for various modelling styles, e.g. rule based, to be applied which may not always directly reflect the architecture of the system under development.

Central to a rule based approach is the usage of SDL observer processes. These can be used to check the behaviour of the SDL model for certain conditions that might be satisfied in a given state. When this is the case, reports can be generated describing the way in which this condition arose, e.g. the sequence of interactions that resulted in the satisfaction of the condition, i.e. the test purposes.

3.3.1 Pros and Cons of Rule Based TCG

Rule based TCG offers the most powerful means of generating tests from an SDL model. It allows a multitude of tests to be automatically generated with minimal knowledge about the inner workings of the SDL model itself. In principle, all that is required is knowledge of the external inputs to the SDL model, e.g. the signals and the values of the parameters that should be sent to the system to allow for interesting tests to be generated, and the rules that apply to the model. We discuss this in more detail in section 4.3.

The approach is not without its drawbacks however. It places more constraints on the SDL modeller to ensure that the model is developed in such a way that rules can be given and meaningfully handled. Also, it is often the case that the specification is developed in such a way so that specific tool functionality can be exploited. Combining knowledge of tool capabilities with the general design of the system itself adds to the overall complexity in specification development. Nevertheless this approach is arguably the most powerful of the three documented here – where power corresponds to the number of tests that can be automatically generated directly from the model. To demonstrate exactly how such rule based models might be developed and exploited by tools we consider the specification of a protocol (PS) and its combination with a service (SeS). We note here, that the following description is based upon a case study undertaken within Ericsson related to commercial products under development and as such a more precise description of the protocol PS and service SeS used cannot be given. The principles in how to produce a rule-based specification style remain the same however, and are largely independent of the protocol or service details.

4. Engineering Rule Based Specifications

A typical architecture of an SDL system used for TCG is given in Figure 1, where the TCG System is decomposed into several key parts/development phases including:

- PS data (ASN.1) module
- Protocol Specification (PS) + data model
- Service Specification (SeS) + PS'
- Test Case Generation (TCG) + PS''+SeS'

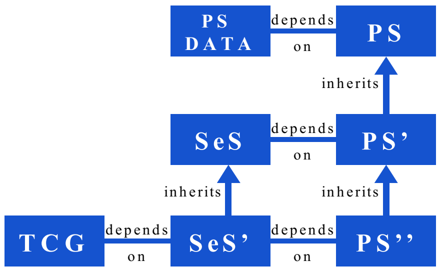

Figure 1: TCG Oriented Specification Architecture

We describe the contents of each of these development phases in the following sections. One thing worth noticing is the association relationship between the protocol and the service specification models. Protocols and services are commonly related directly through inheritance based relationships. Association relationships are useful when some detailed decomposition of the service specification might be required, which need not necessarily be represented in the protocol specification or vice versa. Similarly such association relationships are useful when the necessary knowledge of the protocol or service is not yet available, i.e. it is not known exactly which behaviour will be inherited etc. Dealing with inheritance based relationships typically implies dealing with the inheritance of the associated structures and decomposition of the inherited entities.

From an external, i.e. testing perspective, provided appropriate guidelines are followed which we elaborate upon shortly, there should be no distinction between association or inheritance as the relationship between protocol, service and in this case, TCG systems.

4.1 Early Phase Protocol Modelling Aspects

The PS protocol used in the case study is used to support a wide variety of services. A key guideline followed during the development of the SDL model of PS, was that it should avoid having detailed data dependent behaviour. Rather, the protocol was required to carry various complex data structures, whose detailed evaluation and processing would be undertaken by the associated service, e.g. SeS. As such, the PS model was developed so that it followed the basic state machines given by the design documents, but did not deal with the detailed processing of the parameters passed into the model from the environment. The ASN.1 data itself was provided as one of the inputs to the specification development and saved in a package (*OIPMessages*) and *used* in the necessary specifications of PS and SeS.

The architecture of the PS itself was based upon the separation from the originating and terminating sides of a call is depicted in Figure 2.

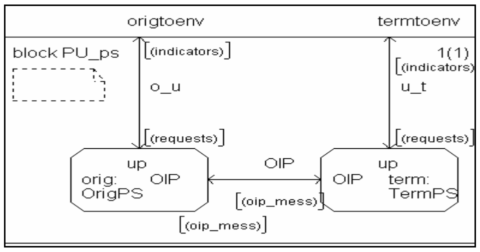

Figure 2: Decomposition of the Protocol Specification

We note here that the signallist *oip_mess* contains those messages defined in the ASN.1 package *OIPMessages*. It is these communications via the *oip_mess* signals that reflect the protocol behaviour as described in the design documents for the PS.

Having a complete system for the protocol allows investigation of the detailed behaviour of the protocol. Thus for example, the PS could itself be validated to ensure that it supported the necessary interaction scenarios and had the appropriate functionality.

To validate the PS specification via model checking or simulation requires inputs to be defined. Existing model checking tools such as the Telelogic TAU Validator [6] are able to generate certain values which can be used for state space exploration, e.g. if a signal carries a single integer parameter, the Validator will generate 3 test values (-55,0,55) which can be used to explore the state space of the specification. With more complex data structures (as in PS) however, users have to explicitly provide meaningful values with which to explore the state space. In our case, given that the PS simply forwards data means that trivial data inputs could be provided, e.g. empty sequences. However, as will be seen in section 4.3, consideration of the input values for signals used to explore the state space of the specification cannot always be treated so lightly and will have marked effects on both the service behaviour as well as the TCG possibilities.

We note that apart from supporting a certain structuring, the PS was developed without any need to follow other specification styles or features that could be applied by tools to exploit TCG. That is, the specification was a straightforward SDL model of two state machines with complex data which was input and forwarded to the associated state machine or environment without being explicitly processed.

4.2 Early Phase Service Modelling Aspects

Once a protocol specification has been developed, i.e. the upper level of Figure 1 has been realized, it can be used in the development of a service specification. To support this, the type information of the protocol model should be saved as a *package* and *used* in the description of the service model. Given that the PS was developed with structuring reflecting the originating and terminating sides of a call, the architecture of the

SeS service with the PS was developed to take this into account as shown in Figure 3.

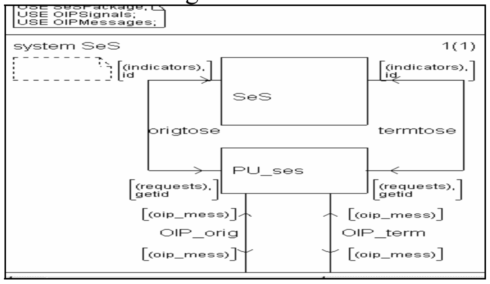

Figure 3: Architecture of SeS and PS

We note here that external interactions with the SeS/PS system are based upon the signals contained in the signallist *oip_mess*, which correspond to those given in the ASN.1 input package. The internal interactions between PS and SeS (*request/indicators*) correspond to the ASN.1 signals defined in *OIPMessages* appended with *req* or *ind* respectively.

The internal interactions between the SeS/PS system also support handshaking between the SDL processes representing the protocol and service, i.e. PIds are exchanged between the protocol and service as part of their initial internal behaviour and subsequently used to ensure future interactions between the processes are meaningful, e.g. checks on message senders/receivers made. We note that handshaking is not necessary when inheritance is used between protocol and service. This handshaking along with the introduction of specific procedures to be exploited by state space exploration and hence by TCG are shown in Figure 4.

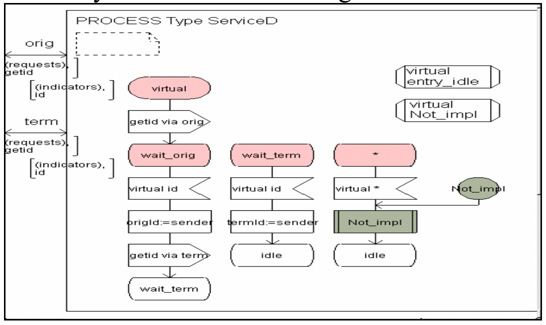

Figure 4: Service Handshaking and Procedures for TCG

Before discussing the details of the service specification itself, we note that the two procedures *Not_impl* and *entry_idle* have been introduced into the specification especially for TCG purposes. Thus there is no counterpart to these procedures in the design documents or technical reports describing the functionality of SeS/PS. We further note that these procedures have a null behaviour.

Whilst the PS was largely based upon the PS design documents, it was regarded as unrealistic, non-scalable and impractical to specify completely the SeS service. As a result the SeS developed, was based on a subset of the overall service functionality described in the design documents. This subset was chosen to demonstrate a realistic TCG investigation.

An example of the behaviour selected is given in Figure 5. Here procedures were called upon reception of the appropriate signal from the PS system. Thus for example, when the environment sent the signal *IAM* to the originating side of the PS, this process would forward the signal *IAMind* to the SeS, which in turn would call the appropriate procedure to handle *IAM*.

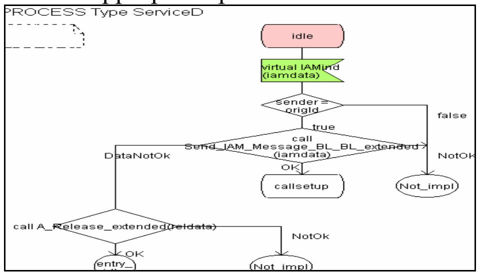

Figure 5: SeS Behaviour and Calling of Procedures

It should be pointed out that the SDL behaviour here is specified in such a way that it can be exploited for rule based TCG. For example, when the procedure handling the IAM invocation is made, three possibilities exist (assuming the sender was from the originating side and hence the first decision is true):

- the details associated with the *IAMind* message (*iamdata*) passed to this procedure are correct – in which case the procedure sends an *IAMreq* to the terminating side and returns with *OK*;
- the details associated with the *IAMind* message (*iamdata*) passed to this procedure are incorrect and result in a specified erroneous behaviour – in which case the procedure returns with *DataNotOk*;
- the details associated with the *IAMind* message (*iamdata*) passed to this procedure are incorrect and result in a non-specified erroneous behaviour - in which case the procedure returns with *NotOk*;

The two erroneous cases given here, both result in the *Not_impl* and *entry_idle* procedures being called. As stated, in the SeS, these have a null behaviour, however, this need not necessarily be the case as will be shown in section 4.3.

As stated previously, it was the case that some detailed behaviour specification based on the data handling of SeS was to be supported. As such, it was necessary that the SeS had behaviour which accessed the input data, e.g. *iamdata* in Figure 5, and could result in all three of the previous bullet points arising. Given that the PS data was represented through pointer lists referencing individual parameters, decomposing the data access of the SeS was based on the knowledge of the well-defined parameter ordering in the PS. Thus for example, it is known that the pointer in the fifth position in the list refers to the *CalledPartyNumber*. This fact could be exploited by checking that the pointer in the fifth position of the pointer list supplied in the input data references something, i.e. it is greater than zero, and if so the appropriate procedure was then called to perform the checks needed on that parameter.

As stated, due to the complexity of the SeS, only a subset of the behaviour (and hence data) described in the design documents was selected. The example chosen for the detailed modelling of the SeS was focused upon behaviour which allowed to perform the following with the input signals and data:

- passes information on as is
- modifies/discards information in the forward or backward messages
- generates new information/messages
- releases the call and sets the appropriate End of Selection (EOS) code

Typically, these design documents were very low level and considered access and checking of the individual bits or octets passed in. The applicability of SDL for such modelling is considered in more detail in section 4.3. An example of the SDL design based on these low level requirements is shown in Figure 6

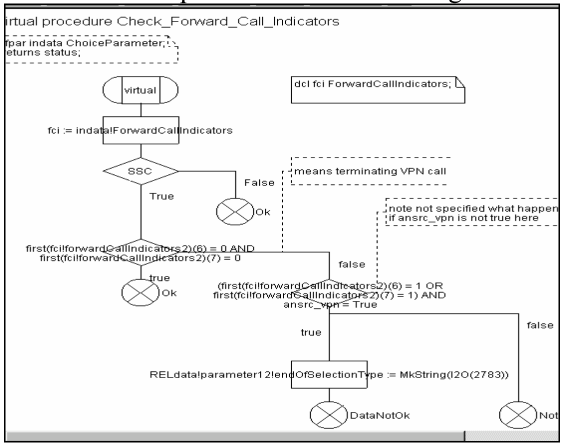
irtual procedure Check_Forward_Call_Indicators
fpar indata ChoiceParameter;
eturns status;
virtual
dcl fci ForwardCallIndicators;
fci := indata!ForwardCallIndicators
SSC
False
means terminating VPN call
True
Ok
note not specified what happen
if ansrc_vpn is not true here
first(fci!forwardCallIndicators2)(6) = 0 AND
first(fci!forwardCallIndicators2)(7) = 0
false
true
Ok
(first(fci!forwardCallIndicators2)(6) = 1 OR
first(fci!forwardCallIndicators2)(7) = 1) AND
ansrc_vpn = True
false
true
RELdata!parameter12!endOfSelectionType := MkString(I2O(2783))
DataNotOk
Not

Figure 6: Detailed SDL Design of Requirements

We note that Figures 6 provides a model of the data dependent behaviour of SeS which considers good and bad data cases, i.e. the *OK*, *NotOk* and *DataNotOk* cases can occur depending on the input data.

Having a complete system for the service and associated protocol allows amongst other things to investigate the detailed behaviour of the combined service/protocol system. Thus for example, the SeS/PS can themselves be validated to ensure that they supported the necessary interaction scenarios and have the appropriate functionality depending upon the design documents and associated use case etc.

It should be pointed out that to simulate or validate the SeS/PS system, i.e. via model checking and state space exploration, requires inputs to be defined. The values associated with these inputs need careful consideration to allow for the interesting scenarios to be discovered and investigated. Thus for example, in the case given previously, it is necessary to provide inputs which allow for all supported cases to be explored. Hence in the case given here this implies that the parameters associated with the *IAM* signal support the investigation of the combinations of the *ForwardCallIndicator* and the *UserToUserIndicators* results, i.e. the *OK* and *DataNotOk* results should be possible based upon the values of the parameters of the different *IAM* signals that can be sent into the system. We note that it is only the inputs that should be considered and not the outputs. That is, for TCG the specification should produce the necessary outputs together with the associated parameter values. We also note that the parameter values associated with the outputs may well be different from the input values, e.g. where the specification modifies the input data before sending it to the environment.

As stated previously, existing model checking tools are often able to generate certain values which can be used for state space exploration. With more complex data structures, however, users have to explicitly provide *meaningful values*. In the case of the combined SeS/PS, these meaningful values take special significance. For example, if no value is supplied for the *ForwardCallIndicator* parameter of the IAM signal, then the tools will not generate tests showing that a release is possible when a check on this parameter is done. Similarly, if the *ForwardCallIndicator* parameter has a value that does not match the precise conditions that will result in a release being issued because of this parameter, then no release case will be generated.

4.3 Early Phase Test Case Generation

In principle, the SeS/PS system described in section 4.2 could be used directly for TCG since it represents the model of the real system that we would like to generated tests for, i.e. it is the model of the SeS/PS. It is certainly the case that simulation based TCG approaches as described in section 3.1 can be applied to generate tests to this system. However, a better approach is to automatically generate tests based upon the satisfaction of rules. The SeS/PS itself does not have any rules which could be applied directly. It did however, allow for the placeholders for those rules to be given. Specifically, the empty procedures *Not_impl* and *entry_idle* were called at the appropriate places depending on the input data.

To support a rule based automatic TCG approach, observer processes are needed. The architecture of the system which can be used for TCG is shown in Figure 7. We note here that this architecture includes the specification of the SeS and the PS through packages. In addition certain new channels are declared that allow for control information to be used which can be exploited for TCG.

Several points are worth noting here. Firstly the Observer process need not be connected, e.g. via channels, to the other processes in the specification. Secondly, the two new channels *ctrl_ps* and *ctrl_ses* are used purely to support erroneous tests in TCG. That is, these channels and the signals that they carry do not arise in the SeS/PS design documents.

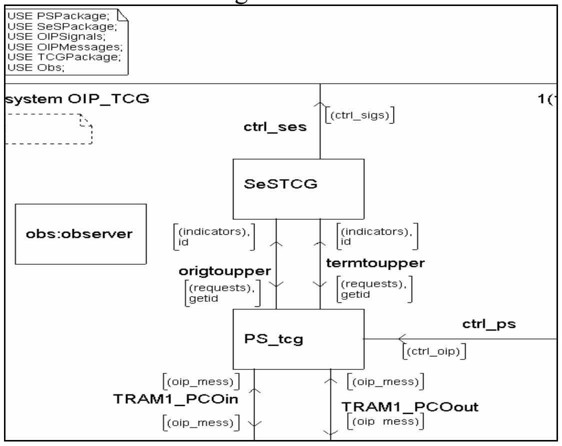

Figure 7: System Architecture Suitable for Rule Based Automatic TCG

The PS/SeS themselves are modified to allow for the exploration of these error cases as shown in Figure 8.

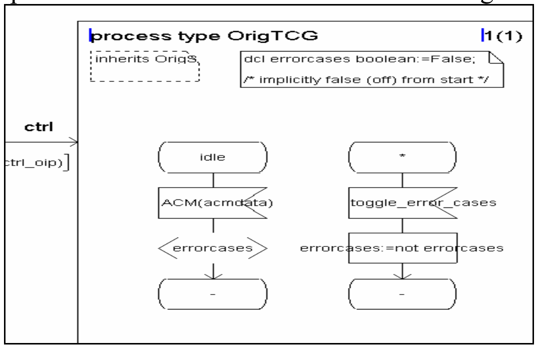

Figure 8: Supporting PS Error Scenarios in TCG

Here the PS is modified so as to allow to investigate the scenarios where certain signals should not be possible in certain states. Thus in this example here, it is the case that the *Address Complete Message* (*ACM*) in state *idle* should be possible or not possible depending upon whether the appropriate signal (*toggle_error_cases*) is sent or not. We note that if this signal is defined as being one of the possible inputs to the specification, then both system traces will be generated during TCG. Such an approach thus allows for error cases to be supported in TCG as required in section 3.3.

The rule based TCG approach can itself be realized through the appropriate specification of Observer processes. One example of an Observer process used to support a rule based approach to TCG for the SeS/PS is given in Figure 9. This Observer process allows to monitor the SDL specification as the behaviour develops and when certain conditions are matched write an appropriate report. Specifically, the observer process accesses the appropriate process identifiers for the SeS (*serv:1*) and the originating/terminating sides of the PS (*orig:1/term:1*). It then enters a state with an associated continuous signal which checks that the SeS (*serv:1*) is in state *idle*, and that certain variables have particular values. When this is the case, a report is written and the previous state is returned to.

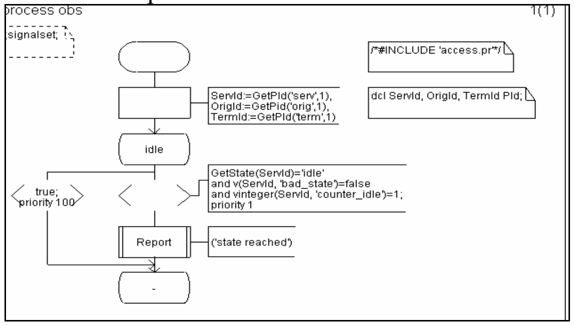
process obs
1(1)
signalset;
/*#INCLUDE 'access.pr'*/
ServId:=GetPId('serv',1),
OrigId:=GetPId('orig',1),
TermId:=GetPId('term',1)
dcl ServId, OrigId, TermId PId;
idle
GetState(ServId)='idle'
and v(ServId, 'bad_state')=false
and vinteger(ServId, 'counter_idle')=1;
priority 1
true;
priority 100
Report
('state reached')
-

Figure 9: Observer Process to Support TCG

These variables referred to here are defined in the two procedures defined as placeholders in the SeS which have been redefined for TCG purposes. Specifically, the procedures *Not_Impl* and *entry_idle* have been inherited and redefined for TCG purposes as shown in Figures 10.

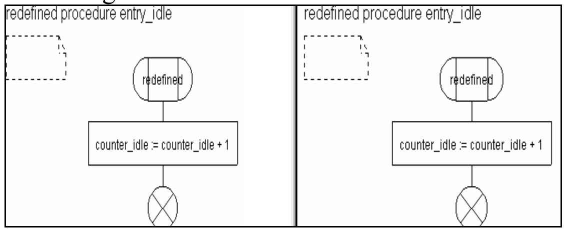
redefined procedure entry_idle
redefined
counter_idle := counter_idle + 1
redefined procedure entry_idle
redefined
counter_idle := counter_idle + 1

Figure 10: Refining SeS Procedures for TCG Purposes

These redefined procedures together with the Observer process given in Figure 9 and the specification style adopted for the SeS specification, e.g. as depicted in Figures 5 and 6 provide an example of a rule based specification style. The logic behind the rules are presented within the Observer process. The natural language description of these rules which are to be used to generate tests can be given as follows:

Write a report when the SeS has returned to state idle (implying the counter_idle=1) and the specification has not reached an erroneous behaviour, i.e. Not_Impl has not been called as this would make the bad_state variable equal to true.

Through this approach, reports will be generated for the valid legal traces of the specification, i.e. those that allow for a sequence of interactions from the initial (*idle*) state and returning to the same state. Hence as required in section 3.3, new and interesting traces (documented as reports which can subsequently be converted via tool support to MSC diagrams) can be automatically discovered.

In addition, to avoid the problem of state space explosion, rules can be incorporated into the appropriate tools that allow for the state space explosion problem to be minimised. An example of a rule given directly in the TAU Validator is: *define-rule serv:1->bad_state!=false*. This rule will effectively only explore the state space when the bad_state variable is false. Put another way, whenever the exploration of the model calls the *Not_impl* procedure and thereby makes the *bad_state* variable equal to *true*, the rule will immediately be enforced. The interpretation of this enforcement is that the state space exploration will stop at this point and backtrack to a previous state, i.e. a state which led to this current one. This pruning then results in a reduced state space being explored and only interesting scenarios/traces of the specification being considered. Through usage of such "cut-off" procedures and the appropriate usage of rules, a powerful mechanism to address the state explosion problem can be achieved. We note that such a constraint oriented specification style is applicable to a multitude of systems where the problem of state space explosion exists, and not only those where TCG is of interest.

5. Conclusions

Testing is an activity that should be incorporated at all stages of product development, from the early phase product ideas right through to the final product deployment. SDL is a language that can be applied throughout the lifecycle of a products development. SDL also allows generation of tests automatically.

One of the key issues to be addressed is how to use SDL in the early phases of product development, and at the same time allow generation of tests which can be applicable to the product itself, where typically the actual product is normally considered at a later time of product development. To allow for this dichotomy to be addressed an approach based upon detailed specification in accompaniment with under specification based upon rules has been shown. In addition, we have demonstrated how following a constraint oriented specification style and using appropriate rules can aid in both the identification of new and interesting traces of the system behaviour, as well as avoiding the problem of state space explosion.

We note here that this combination of very detailed specification together with under-specification through rules which effectively cut-off unwanted state space explorations, together offer powerful, complementary features. Through these combined approaches, detailed aspects of a product can be specified and unwanted details abstract away from (hidden via rules). This then enables tests to be generated focusing only upon some subset of the overall product functionality. Put another way, it is not necessary to specify the whole product behaviour in SDL in order to generate tests for that product. Rather, the most important product features which should be tested should be specified in detail, then rules applied to ensure that tests are generated only based upon these interesting cases. In terms of using SDL and its associated tools in the early phase of product development, such an approach is of course directly applicable and offers considerable advantages to product development where issues related to testing should be supported as early as possible.

Ideally an SDL model should be usable by anyone for TCG, i.e. the person performing the TCG may not necessarily be the person who created the SDL model. Whilst a rule based approach avoids the test generator having to have a detailed knowledge of all aspects of the SDL behaviour, e.g. as is the case with simulation based TCG as presented in section 3.1, the test generator will likely need some help to understand how to generate tests from some non-trivial SDL model. At a minimum this information should include the signals that should be sent into the system as well as the values of the parameters that the signals carry, together with any rules that should be applied to aid in the test case generation process. Such information can easily be documented along with the specification itself, e.g. through the TAU Organizer interface.

Finally we note that the case study did allow completely automatic generation of TTCN test cases along with their associated constraints. Given the nature of next generation telecommunication systems, with single messages often having several hundred complex parameters (the *IAM* message given above 179 extremely complex parameters), this approach looks especially promising.

6. References

[1] Specification and Description Language (SDL 2000), ITU-T Recommendation Z.100, Geneva.
[2] Tree and Tabular Combined Notation version 2 (TTCN-2), European Telecommunications Standards Institute (ETSI). Methods for Testing and Specification (MTS), ETSI TR 100 000 V1.0.0 (1998-11).
[3] Testing and Test Control Notation version 3 (TTCN-3); European Telecommunications Standards Institute (ETSI), Methods for Testing and Specification (MTS); Part 1: TTCN-3 Core, ETSI ES 201 873-1 V2.1.0 (2001-10).
[4] Message Sequence Charts (MSC 2000), ITU-T Recommendation Z.120, Geneva.
[5] Abstract Syntax Notation One (ASN.1): Specification of Basic Notation, IS-8824-1, ITU-T Recommendation X.680.
[6] TAU Users Manual, www.telelogic.com.
[7] Proceedings of Workshop on Formal Approaches to Testing of Software, A Satellite Workshop of CONCUR'01, Aalborg, Denmark, August 2001.
[8] Proceedings of Second IEEE Workshop on Industrial Strength Formal Specification Techniques, Boca Raton, Florida, October 1998.