Industrial Report on the Use of Abstractions in SDL / MSC's

Martin Kooij, Luc Provoost
Alcatel Telecom, Antwerp.
{kooijm,provoosl}@se.bel.alcatel.be

Abstract

When using SDL on large scale projects we hit the limits of the SDL language and the tools that support that language. As stated in the language definition Z.100 of ITU-T the SDL language is a description or a design language, but it does not necessarily describes a method that accompanies the language. The language is thus not constrained to a single method, and various design methods can be used. This industrial report describes the SDL design methodology we use in the Alcatel Switching Division. This methodology uses SDL in two different phases, the Top Level Design phase and the Detailed Design and Coding phase. The adding of detail when moving from one stage to the other is, however, not supported in the standard tools/methodology as supported by the tools. This report sketches the Alcatel approach and proposes a route for solution of this problem. This solution is explicitly communicated to a focussed SDL forum to propose our solution to be integrated in the tools and methods.

Keywords

SDL, abstraction, design steps, experience

1 INTRODUCTION

The Alcatel Telecom Switching Division is the largest division within Alcatel Telecom, responsible for the marketing, sales, development and maintenance of switches. The real time complexity of these switches and the architectural complexity of the software controlling these switches asks for a thorough development process. Within this process standard SDL is used for detailed design and coding, and extended to other phases in the life-cycle, notably top level design.

This paper describes the industrial experience in using SDL on such a large scale, in two distinct phases: top level design and detailed design of the software. We show how SDL is used, and how we approach the problem of keeping these phases together, as much as possible within the standard framework of SDL. First we explain how we see the use of SDL within the project. Next we describe our approach to the problem of switching between two levels of abstraction, and finally we propose routes to a more stable, industrial solution that could be of use to more projects and companies than just ours.

2 THE USE OF SDL IN THE PROJECT

The Alcatel switching division uses SDL in describing ti
both on a systems architecture level (from here on ca
implementation level (from here on called: the detailed des
various components are defined. This phase defines the sc
the blocks, and what the main communication patterns are.
build process starts. Based on the top level design the c

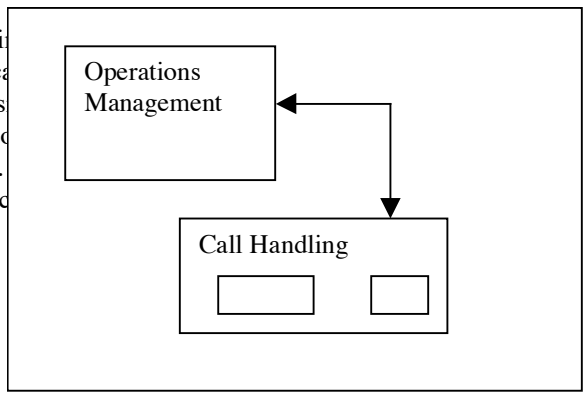


Fig 1: Top Level Design

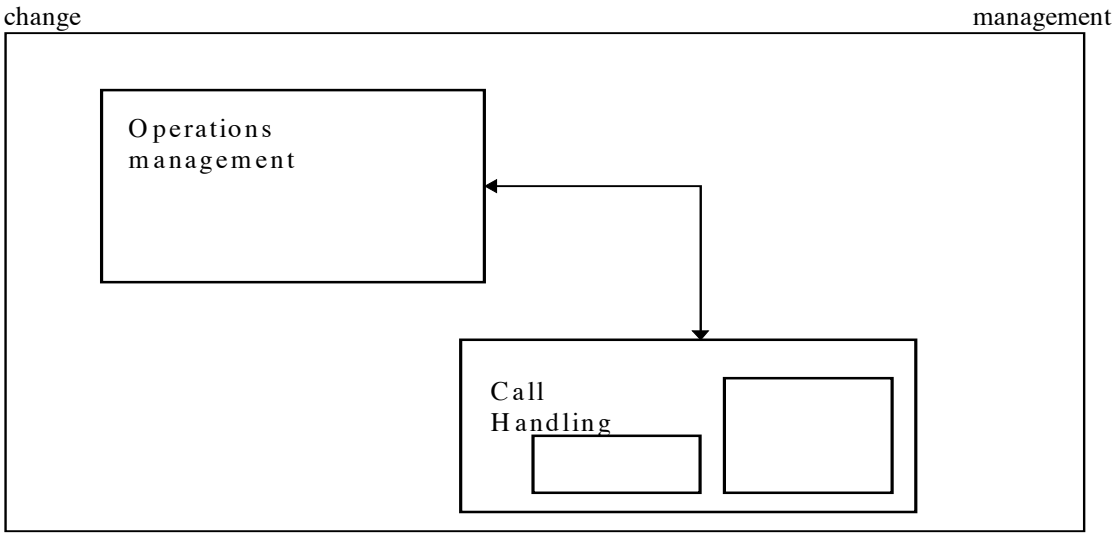


Figure 1: Top Level Design

control, even before coding starts. This top level architecture thus describes components and communication patterns, but is void detail that has to be added in later stages.

We use SDL to describe these components and communication routes and MSC's to describe communication patterns. The tools allow us checking between the component architecture and the communication patterns. Normally detailed design starts from this phase onwards and coding afterwards. With the current tools support it is indeed possible to use SDL to do the detailed design and generate code at the same time. In this way the detailed design and the coding phases are compressed to one phase. In this paper we do not advocate any waterfall or other software engineering models, but we stress the fact that the design and architecture is of special importance in a real time system with many customisations and variations from country to country and from customer to customer. This thus warrants special attention to this area, and that is why we really use a strict phase of top level design that defines the architecture. Of course, this top level design, and here we come to the crux of the matter does, as a matter of principle *not* contain as much detail as the final system. During the detailed design phase a lot of (SDL- and data-) detail is added. In this paper we state that, contrary to what is often thought, native SDL, nor the methods implemented in the tool actually supports the adding of such detail, such a transition from one phase into the next design phase. Actually, zooming in from a block to a sub-block gives you more detail on your screen. This detail, however, is already silently present in your model (the underlying PR files) and it represents detail that is just not shown on your screen but present in the model. Such zooming in from one block to its sub-block(s) does not actually describes a transition from one design level to the next. It does give the possibility to get a structured overview of the design in one level, but does not allow to consider the same system on another level of abstraction.

Switching levels of abstraction means that you can pinpoint relations that provide a mapping between elements in the abstract level and elements in the more detailed level. In general these mappings are

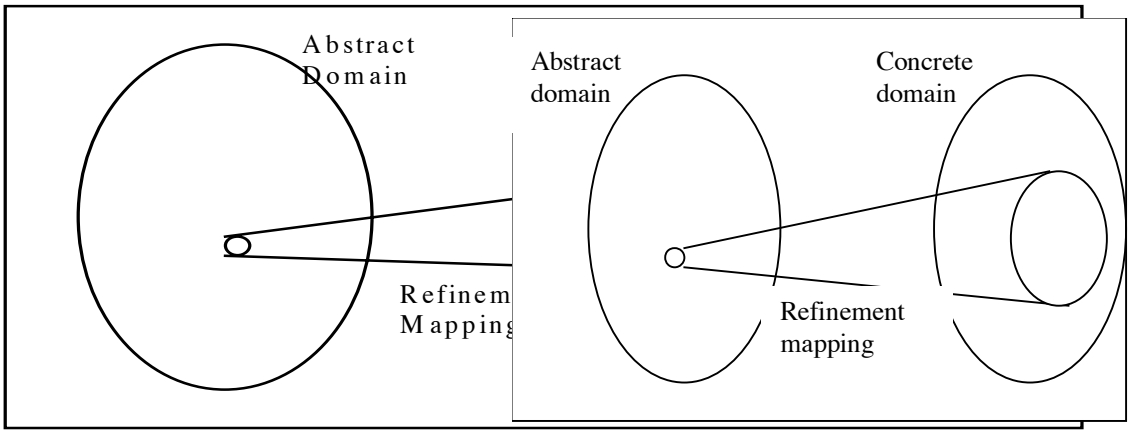


Figure 2: refinement mapping maps al possible implementations.

Fig2: refinement mapping maps abstract problem description into possible implementations…

known as refinements. To make it more concrete we state several possible, well-known, refinements that can be made when going from top level design to detailed design:

1. *Process refinement:* One process in top level design could be refined to several processes in detailed design.
2. *Data refinement:* Abstract data in the top level design is substituted by more concrete data types possibly conveying more detailed information
3. *Message refinement:* one signal in the top level design could actually be implemented by a set / sequence of messages.

More complicated refinement mappings that allow processes to take the role of data and vice versa are possible candidates too, but will not be mentioned any further in this paper and are also not used in our projects.

We can easily see that the native SDL mechanisms allowing to get a good overview of the system by structuring the systems in blocks and sub blocks do not allow to describe two abstraction levels of design to coexists simultaneously in one SDL system. In smaller projects this can be mitigated by slowly refining your SDL code, without too much tool support and write a new, more complicated model, where the data definitions are refined, processes and messages are added. The consistency with your previous SDL specification can then checked by hand.

However, we want to stress that considering the size of the project the tooling plays a major role. Without tooling the design activity would too abstract and not give much advantage over the paper and pencil work that already worked for years. With tools the power of automatic checking, automatic code generation and automatic consistency between the various design levels become reality. This meant that were looking for an efficiently implementable methodology using present tools and the native power of SDL as much as possible.

Of course one can show that concerning MSC's the same abstraction problems exist. In our project we strongly advocate to use SDL in an integrated way by using MSC's already in an early stage to

- Convey the semantics of the interface,
- to structure incremental development,
- to serve as input for test.

We see that MSC's created in an early design stage are effectively unusable from the tools perspective in the later design stages due to the detail added in later stages of the development. We state that an MSC written in terms of top level design objects (processes, channels and data parameters) is not trivially usable during detailed design. For example, a message could carry different data , or even be redefined to several messages. This impairs the (automatic) reuse of these message sequences.

3. OUR APPROACH TO SDL ABSTRACTION LAYERS

Our strategy was to develop a method that is in the "spirit" of SDL and does not make tool support of the process impossible or unduly expensive. One can easily dream of very powerful abstractions and mappings between the top level design view and the various possible implementation. SDL, however, is known for its practicality and usability by non highly mathematically trained engineers. Moreover we wanted to keep the step from top level design to detailed design traceable by tools and understandable by humans. To this end we restricted the number of abstractions that can be made when going from top level design to detailed design. Maybe to our own surprise these restrictions appeared to be practical and workable and also made it possible to reason on the system in several levels without changing too much in the tools. The restrictions were as follows:

- *Process refinement:* This is made possible, but the processes that refine a top level design process can only communicate to (their) outside world via channels that are of the *same name* as the signalroutes in the top level process.
- *Message refinement:* not allowed. One signal in the TLD will always be implemented by a signal with the same name and same parameters over the same route during detailed design. Of course, a designer is allowed to create extra messages and signalroutes when refining a top level process into a number of smaller detailed design processes., but these should not be used to communicate with other components.
- *Data refinement:* Only structural refinement. Names of data types in the top level design are exactly the same as the data type names in the detailed design. In top level design the data types are just (singleton) enumerated sets, during detailed design the data types are expanded to contain full information.

As an example we give a possible refinement:

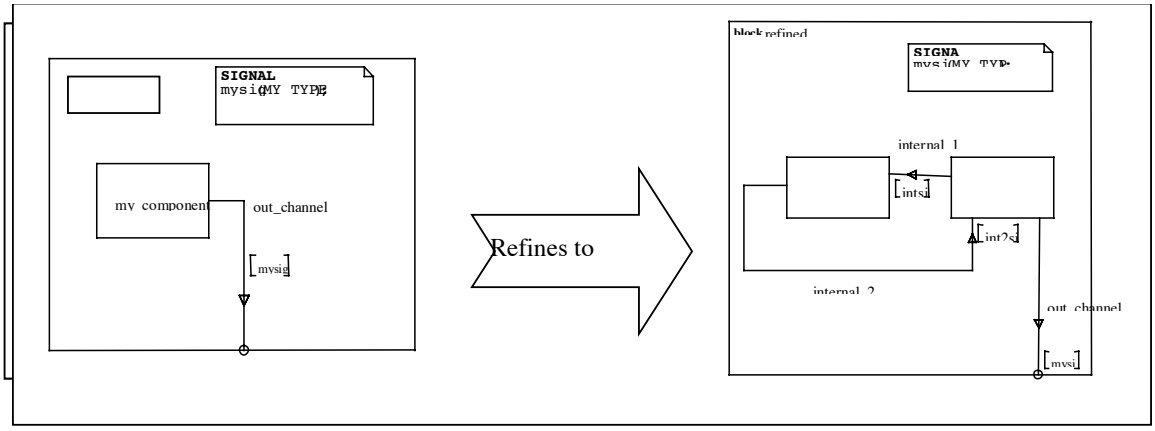


Fig 3: Valid refinement of an top level design process into a detailed design block.

U
would be channeled to the outside block. Also the outsig message should carry only one element of data , of the type M_TYPE. The actual content of M_TYPE, however, can be an enumerated type "LOW, HIGH" in the Top Level design model, but an integer, or maybe even a very complicated structure in the detailed design. For the detailed designer, however, the rule is clear. He should use the message outsig to communicate with the outside world for this block, and this message carries one data-element of type M_TYPE.

Under our rules it is also not possible that the signal outsig is implemented as a cascade of signals e.g. implementing a handshake version of the signal. In our approach this has already been defined in the top level design phase.

The consequences of this strategy of limitations are twofold. Because the messages are not to be refined the top level design is *a full interface model* between the software components. This means that the message flow of the MSC's written during top level design can easily be reused during testing

to check the behaviour on the component interfaces, traditionally an area that is prone to errors. Also these test MSC's can be generated during a simulation of the top level design, using the abstract data definitions.

Secondly, because the data was refined in a strict way (using the same names in the two models) we are able to check, verify and even simulate the top level design because the data names are all present. Here we should explain that all the data type definitions are kept together in a data dictionary, called the "context". Thus we could simulate the same SDL system as a top level design system by linking it to the simple, top level design data-types, and simulate the system as a detailed design system by linking it to the more detailed data types. As all data types have the same names one can see that the semantics will be different (i.e., more detailed), but the model will pass the SDL static semantic check. Tool support can be limited to generating and checking lists of names (of data types, of blocks and of signals) to keep several versions of SDL systems consistent.

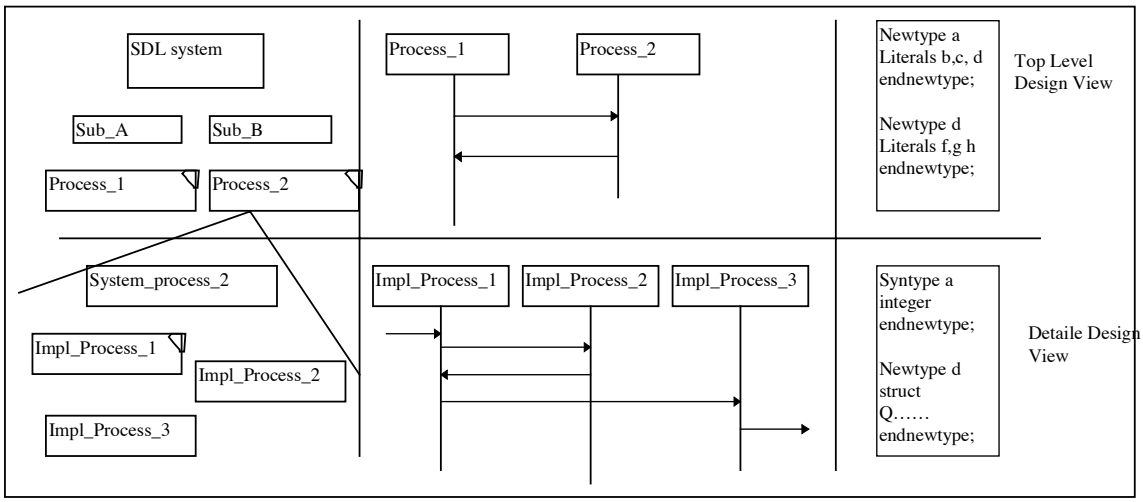
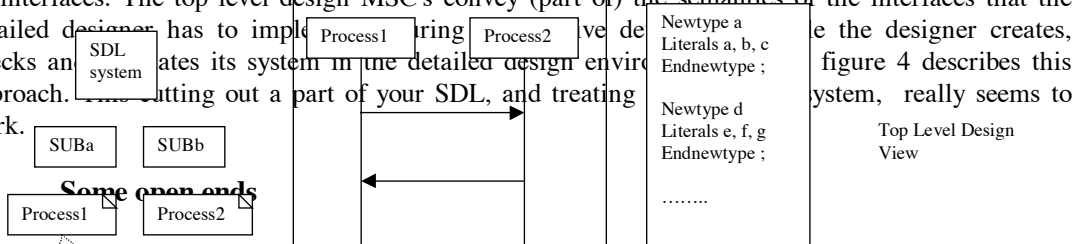


Figure 4: Relations between top level elements and detailed design elements

During detailed design a designer uses an SDL system based on a top level design SDL process with its interfaces. The top level design MSC's convey (part of) the semantics of the interfaces that the detailed designer has to implement during the detailed design. Above the designer creates, checks and validates its system in the detailed design environment. figure 4 describes this approach. This cutting out a part of your SDL, and treating it as a system, really seems to work.



3.1 Some open ends

Methodology: With this approach we advocate that one could [illegible] of SDL and use it to add detail during a detailed design phase in a separate matter. The above picture here is however, still a little bit optimistic. We do know how to generate a "system" that the designer gets, containing all the interfaces [illegible] according to the [illegible] all. We use the MSC's that are automatically connected and also once the component is "cut" difficult to put it back into the system [illegible] feature such as "reverse engineer from" given in tool would come in handy. Especially we would like to merge scenario's into bigger scenario's to allow testing.

Tools: Although the refinement rules are relatively simple we create our own tools to check the consistencies. The tool for keeping the MSC's consis[illegible] rather complicated and maybe strongly dependent on the actual allowed refinement mappings.

User interface: Besides that the fact that we have to keep consistent a number of SDL files that are actually just views on the same system is sometimes counterintuitive to the designers working with the too

Figure 4: Relations between top level elements and detailed design elements

4. A ROUTE TO A SOLUTION

It is our strong opinion that to be really successful, SDL tools should be enhanced with a feature that would give the opportunity to view the system at different levels of abstraction in addition to the current possibility to divide an SDL system into blocks.

Above we explained our approach of linking top level design and detailed design. Based on a variation of this approach we could imagine that the tools allow for the selection of a block. This block can then (independently) taken by a designer or a group of designers. This block can be treated fully as an SDL system and refined as such. The tool would allow independent code-generation, independent simulation etc. based on this block. Within the Rational Rose 98 tool, one finds such features within what they call the "component view".

The great advantage to keep such a block accessible in both the top level design view and the detailed design view is that the consistency can be guarded. A "nice to have" feature would be a feature that allows the re-importing a refined block and automatically try to fit it into the top level design view. In this case the top level design is continuously consistent with the detailed design, and because of automatic code generation also consistent with the produced code. Such a chain of linked models will greatly reduce interface and implementation errors due to incomplete or out of date specifications! The proposed features where the tools would keep several SDL files (each showing a different view) consistent would give the user thus more freedom in implementing design and implementation methods using SDL.

Of course the definition of consistency between top level design and detailed design has to be stated. This boils down to the question "When is a certain SDL system a valid implementation of a architectural block?". We propose that this consistency is guarded by a number of simple (but possibly user-defined) mapping relations. Special attention should be given on the effect on the MSC's in the mapping relation. The implementation MSC's should be consistent on both levels.

We could think of support for mapping functions that define:

- *Process refinement:* Message sequences are allowed between blocks, processes can become blocks in more detailed views. Of course, as one cannot dynamically create blocks there are some restrictions on which kind of process can be refined to blocks with processes inside.
- *Message refinement:* Not allowed now at all. In practice this may give unexpected problems. We currently have under study the idea that messages with less (data) parameters during top level design as compared to detailed design .
- *Data refinement:* data could be structured in different ways, depending on the abstraction level. Simulation would natively use the abstraction level chosen. Data consistency between upper and lower layers are based on name consistency.

5. CONCLUSIONS

In this paper we showed how Alcatel Switching division uses SDL in two ways. The first is to describe top level design (the architectural view) and the scenario's that describe the behaviour on that level. Secondly SDL is used to do detailed design and for skipping the traditional phase of coding that is now largely done automatically.

Next we showed how we agreed on a number of very simple mapping functions between these phases that defined the consistency between top level design and detailed design and what that means for working with scenario's on the two levels. We also argued that the basic mechanism of SDL of defining blocks and zooming in on these blocks actually does not always help in adding detail necessary for the step between top level design and detailed design.

Finally we proposed an integration of the mapping functions into the tools, that will make it possible to "independently" cut out a certain part of the SDL system and refine it, while keeping it consistent with the overall picture, and thus finally, with the other parts that are not yet refined, or that are

refined within other groups. By using these features SDL could be used during a much longer part of the life-cycle boosting the efficiency and usability.