

# The Formal Approach for Real-Time Design

*The increased complexity of software systems together with the increased demand for productivity and shortened product life cycles leads to a continuous search for new technologies in the effort of meeting continuously increasing demands. Currently within the software engineering community there is a large interest towards graphical modeling and design techniques as an approach to manage the task. Other groups within the software engineering community strongly believe that these demands may only be satisfied by increased usage of formal methods and formal languages.*

The Specification and Description Language (SDL) is an object-oriented programming language suitable for event driven real time and distributed systems, often applied in safety critical applications. SDL is a formal, graphical language, which combines both a high abstraction level graphical notation with formalism facilitating automatic generation of executable code. A model described in SDL is not tied to any particular implementation, software and hardware.

The characteristics of SDL give greater productivity and shortened lead-time when applied on a software development project within the domain of event-driven systems. Tools on the market offer the possibility to validate the system against requirements scenarios. It is also possible to check the robustness of the system against unexpected situations, all together ensuring designs of high quality. In addition, software maintained on the SDL level will have its maintenance cost sufficiently reduced.

The Specification and Description Language is standardized by the ITU (International Telecommunication Union) under the reference Z.100 and is frequently used within the telecommunication industry.

## DETAILS ABOUT SDL

An SDL application is described by a number of diagrams, which define the structure, and behavior of the

system. The behavior of the application is modeled by independent active objects, in SDL named processes, executing in parallel. Processes are grouped into sub-systems - blocks that may be grouped into larger sub-systems or into a system, which is the SDL diagram that defines the top scope. Each of these: systems, blocks and processes has well-defined interfaces which defines what signal primitives that may be used when interacting with a component. Signals carry information that need to be shared between processes and signaling is the only way to share data since SDL systems do not have global variables.

The behavior of a process is graphically described in a process diagram by a state machine, using symbols representing states and transitions. Each transition starts with an input symbol, which denote the consumption of the signal triggering the transition in which a set of different actions may be taking place. SDL provide all primitives needed to achieve desired behavior. However, SDL also contains language elements making it possible to use components, within a process description, implemented in other programming languages. In SDL it is possible to create libraries of reusable components such as systems, blocks and processes using packages. These components may be reused as-is or they may be specialized into new components using inheritance.

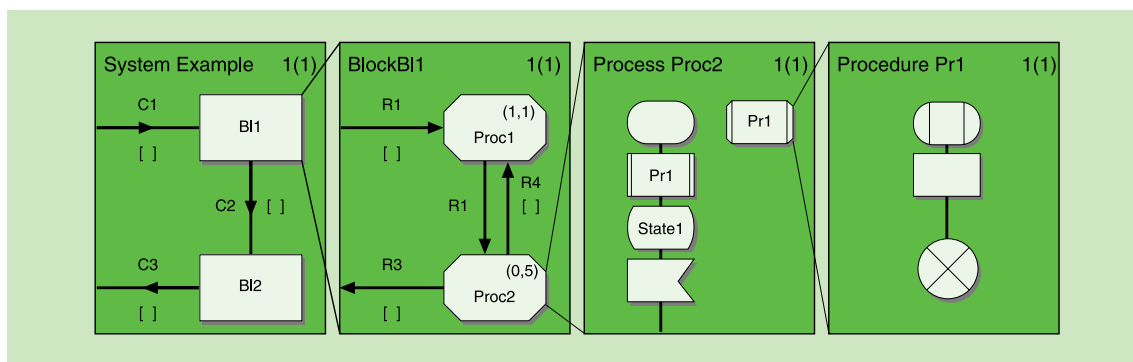


Figure 1. SDL notation

## HOW ARE THE BENEFITS ACHIEVED?

Designing and implementation in SDL is done on a higher level of abstraction compared to traditional programming languages allowing the designers to concentrate on the creative work of creating a good design rather than putting most effort into the coding of the design. The inherent complexity of a real-time problem is to a large extent decreased when working in an SDL model, since the behavior of an SDL system is formed by the total behavior of all processes within the system, which are executing in parallel allowing a process to be assigned for each task. Each process may be considered to be independent since they do not share memory, thus removing the risk of memory collisions.

Designing in SDL is often done in a top down approach. Starting with finding the interface between the system and its environment. The system is then divided into subsystems and the interfaces between the subsystems are defined. It is particularly important to have correct well-defined interfaces between subsystems in large software projects in which there will be teams of developers working in parallel assigned to each subsystem.

By using SDL strong emphasis will automatically be put on the interfaces of the system and its parts. This will help the designer to avoid the common mistake of start dealing with the detailed design of a component before its responsibilities and interface are properly defined. One of the greatest benefits is that already at the point of interface design it is possible to simulate and debug the dynamic properties of the system's external and internal interfaces. The architecture definition is hereby ensured to be of good quality and it

is not very likely that any serious errors in the architecture will remain to later phases of the project. Minimizing the risk for architecture errors has proven to be one of the most effective measures to take to keep the project within schedule and budget, since architecture errors tend to be the most costly errors to remove.

When it comes to the design of the subsystems these can be dealt with separately as if they were top level systems, thus, subsystems can be verified and validated. It is possible to simulate the subsystem's behavior independently of the rest of the system. This gives excellent opportunities to perform pre-studies of critical parts of the system without the time consuming creation of stubs. Implementing the behavior of a system using SDL is done according to the divide and conquers approach. The complex behavior of the system is divided and assigned to processes. Implementing each process one by one carries out the implementation. The behavior of a process tends to be a simple task to implement in comparison. When a subsystem is implemented existing tool support allows automatic verification to ensure compliance between the architecture definition of the system and the design of the subsystems.

The benefits of a good graphical notation are several. First, the designer will produce well-structured and readable code with fewer errors compared with traditional programming languages. Second, code reviews will be more effective since the reviewers will understand the code with little effort. Third the graphical notation gives excellent possibilities in communicating the result to management, customers and new project members. These are the general arguments for using graphical modeling languages in the early activities of software development. But, SDL takes the use of

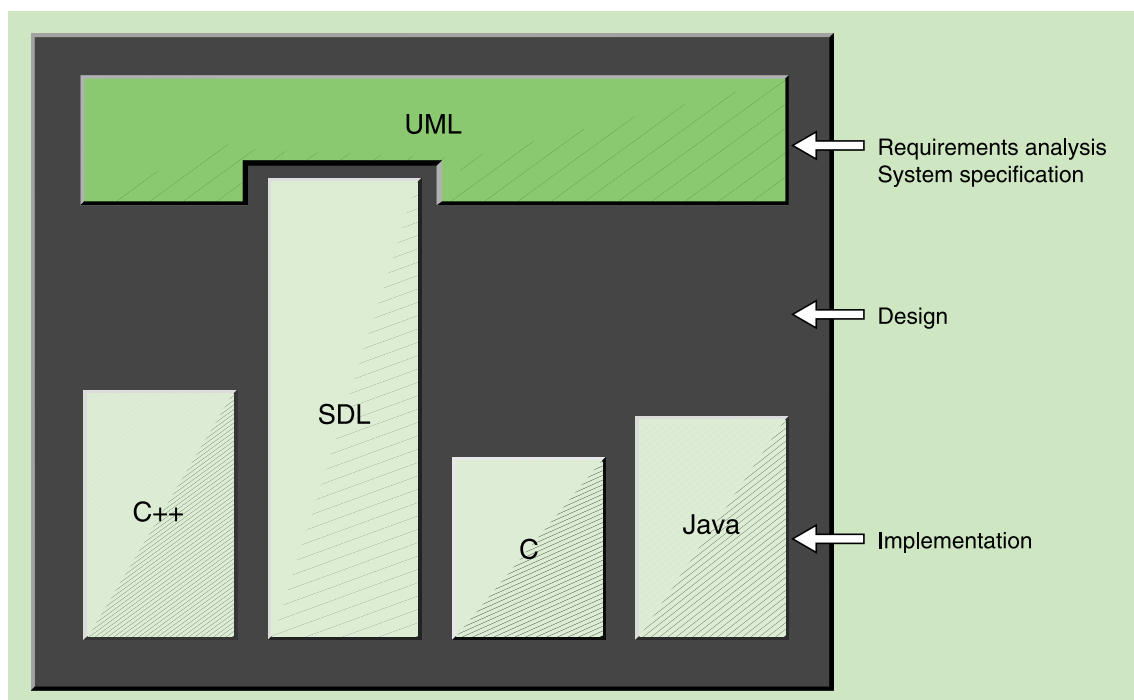


Figure 2. SDL and UML

graphical notations one step further. By making SDL the implementation language these benefits will be true not only for analysis and design but also for the implementation activity. Studies in the subject has proven that software systems produced using SDL are less error prone and more robust than systems produced using conventional programming languages.

The properties of SDL together with existing tool support allow requirements formally expressed using sequence charts to be automatically verified against the system ensuring that the tests that will be carried out are valid. SDL Tools may also reduce the routine work when creating test suites by facilitating automatic creation of test suite skeletons, thus accelerating the test phase significantly.

Considering the software life cycle, maintenance costs represent a large part of the total cost of a software system. The cliché "A picture say more than a thousand words" is certainly true when it comes to SDL and maintenance. SDL has for years been used to specify and document code written in programming languages other than SDL. Today, when it is possible to automatically generate code out of an SDL specification the often used but seldom true comment: "The code is self documenting" becomes more truthful. In fact, when programming in SDL the documentation may be considered to be the code. Although, clarifying comments provided by the designer always enhance the readability of the code also when using SDL. Thus, maintaining code written in SDL will reduce the maintenance costs significantly.

The excellent readability of code written in SDL reduces the threshold of reusing code. In-house or off-the-shelf components and frameworks written in SDL will easily be reused since these are easy to understand. Today it is possible to

procure not only small components and protocol stacks but also complete protocol specifications written in SDL ready for implementation, since most large standardization organizations within the area of telecommunication are using SDL in their standardization work.

SDL was introduced to improve the specification of computer systems. The language has proven its strengths in the early activities of software development. Recently within the software engineering community there have been much interest in the Unified Modeling Language, UML a general notation for modeling requirements and describing designs. However, UML lacks precise semantics and automatic generation of code is not possible. One interesting approach is to combine UML and SDL to achieve maximum leverage in software design. The two languages together cover the complete development process from analysis to implementation. This approach is easily utilized since the concepts of the two languages are well mapped to each other. Other benefits achieved using this approach are excellent requirements traceability between requirements and implementation. ■

---

*Niklas Landin is a methods consultant at Telelogic AB and has been employed by Telelogic since beginning of 1996. Landin holds a Master's degree in Computer science and technology, and the topic of his Master's thesis was 'Development of Object Oriented Frameworks'.*

*In addition to working with methods development, Landin has recently focused much of his work towards the integration of UML and SDL. He has been working with training in the area of UML and SDL and is co-author of the Telelogic course: 'UML and SDL in Practice'.*

# AD CADUL