SDL Based Auto Code Generation: A Multi Beneficial Approach

Bhaskar Rao.G
Software Engineering Group
Motorola India Electronics Ltd.
Banglore, India - 560 042
email: gbr@miel.mot.com

ABSTRACT

Traditionally SDL was used for describing the High/Low Level Design (HLD/LLD) for identification of states and events in a protocol. This SDL design would be implemented/ coded and tested manually. Reaserach has enabled the Automatic generation of C Code, if such a design is made formal with an additional effort of 30%. It nullifies the need for an additional coding phase and promises to show impact on Cycle time, Productivity and Quality. This approach has been successfully tried in Motorola India Electronics Ltd. It has resulted in good Cycle Time, Productivity and Quality improvements without any noticeable performance problems. However a larger object code size was evident; the issue has been solved to a reasonable extent with guidelines and scripts (Cadvanced code generator). The scenario changed when Cmicro code generator was used and it obtained good results (10% more than manual code) without fore-going the primary benefits of using SDL (Portability, Readability, Maintainability etc). This paper presents the experiences and challenges faced with an aim of propagating the technology and sharing for overall benefit. This also brings out various issues in front of the community for further exploration.

Key Words

SDL Technology, SDL Experiences, Methodology, Product Development with SDL.

1. Introduction

SDL[1,2] is suitable for development of any protocol because of its enriched language constructs, Simplicity, Graphical Interface and support for Object Oriented description. Because of its Strong Finite State Machine approach, the Verification and Validation of the Protocols at Design level on host machine was made possible through powerful tools with good algorithms, which normally result in finding most of the deviations from requirements/errors in the design, hence contributing to improving the overall quality of the application at a lesser cost.

1.1 Needs of the Telecom World

- Reduced Time to Market: The visual tools, ease of usage, early verification and validation to find the problems at higher level are the some of the requirements to be first in the market.
- Best in Market: Best can be characterized by Quality, Cost, Reliability, Convenience etc. These attributes are largely influenced by Performance, Features and Code Size of the application running in the system. As the world is viewing the handset as a personal assistant, so the need for providing better performance and more functionality with optimal cost has soared in the recent years.

1.2 Industry Perspectives of SDL

- Design Language: The language and the supported tools are viewed as an utility to represent the informal design.
- Implementation Language: SDL is viewed as a coding language like C and C++ with other languages/techniques, because it facilitates the environment to fulfill the above mentioned needs of telecom world. However there are some challenges to use this perspective like optimal code size, initial learning etc.
- Test Case Development Language: The powerful constructs of SDL can be used to develop the test cases for conformance testing. The tool features can be utilized fully to automate the entire conformance testing.

1.3 Software Development with SDL Methodology

SDL, Communicating Extended Finite State Machine language consists of states, events, output events and processing constructs. These constructs are more suitable to the nature of the protocol applications. In general for software development with SDL, industry uses other languages and techniques such as OMT/UML, MSC, ASN.1 and TTCN, called as SDL Methodology. OMT/UML is used for system analysis, MSC is used for scenario capturing and test case validation and verification, ASN.1 is used for representing some aspects of the protocol which are difficult/not supported in SDL and finally TTCN is used for conformance testing. Processing intensive parts could be developed in C and can be integrated with the SDL solution. With SDL-2000, these also can be implemented directly in SDL. The legacy code can also be integrated with SDL.

1.4 Features of the tool (SDT Telelogic Tau)

Telelogic Tau[3] provides an environment to use the SDL methodology for development of software. All the projects have used this tool for SDL based development with auto code generation. The tool set comprises of various visual tools for analysis, design, testing and auto code generation. The salient features of the tool are listed below.

- Integrated environment- tool supports all the development activities from a single desk top.
- Integrated methods - OMT/UML, MSC, SDL, TTCN and C code are integrated to provide better methodology for development.
- Generation of code from formal SDL for multiple platforms with multiple code generators

The tool has three code generators namely Cbasic, Cadvanced, and Cmicro [3] to generate code from SDL to C. These code generators have been used in MIEL in many projects and prototypes.

1.5 Description of the pilot project

This project is an implementation of EIA 709.1[4] control network protocol, which is a fault tolerant, intelligent and complete platform independent network system. These networks consist of intelligent devices or nodes that interact with the environment, and communicate with one another over a variety of communication media using a common, message based control protocol. The product can also be used to control various sensor or control devices such as Motors, Valves, Encoders and Switches, and provides high level of

reliability and security[5].The protocol stack consists of 4 layers - Network management Layer, Session Layer, Transport layer and Receive side of the Network layer. The team had used the following tailored approach for maximizing the benefits.

1.6 Overview of the tailored approach followed for development

- Requirements analysis
- System analysis, High Level Design (HLD), and Low Level Design (LLD)
- Coding in SDL and C
- Test case development
- Simulation and Validation for unit and module integration testing
- Auto code generation
- System integration and testing
- Release

This approach has proved to be effective in our study.

2. Goals and Results

2.1 Goals

- Achieving good cycle time (1.5X)
- Enhanced Productivity(1.5X to 2.0X)
- Good Quality (1.5 to 2X)

2.2 Results

The results obtained are outlined below.

Table 1: Project Results (Cadvanced code generator)

Attribute	Estimated (manual code)	Actual (auto code)
Cycle Time (weeks)	1	1.44X
Object Code Size (KB)	1	2.62X
Number of Errors	1	3X
Total Effort (staff-weeks)	1	1.3X
Performance		No Penalty

The cycle time gain was 1.44X, inspite of tool problems and learnings. Even better CTR (1.5X to 2X) is possible for new developments with same technology because of familiarity with the tools, language and techniques. The improvement in quality was very significant due to visual tools, simulation and validation. Overall productivity gain was also noticed. The object code was obtained with light integration [3] for Windows NT. It showed the penalty of ~2X with an estimated size. This penalty would have become a major bottleneck

for handset applications, which have to run in small targets with small amount of memory. So, the team analyzed the project from different perspectives to reduce the code size of the application with the help of the vendors.

2.3 Reasons for code bloat

This section briefly presents the major reasons for increase in code size.

- Software Architecture

The software architecture of the stack consists of many packages, processes and process types. This division is very modular and flexible to adapt to the new requirements and changes.

- Extensive Reuse of C Data Structures in SDL

The data structures are reused from the similar protocol. These are defined in multiple files with many structures and unions. The tool converts these header files to C files, when it is using in SDL, so it increase in the code size.

- Tool Related

The tool has provided a utility to convert all the header files to a package. The conversion generates many overhead functions on structures to perform Assignment, Default values, Equality test, Inequality, Field Modification and Extraction on structure variables and fields. Some of these functions are normally not used in the implementation, but these remain in the object code.

2.4 Results of optimization

The optimization study was done on a particular layer by considering all the problems that were mentioned above and tips were given by the vendors to reduce the code size. The results were compared with manual code.

Table 2: Optimization Results (Cadvanced code generator)

Development	Object code size	Remarks
Layer before optimization	3X	Many OO features are used. A constant overhead is more due to dependency on all headers.
Layer after optimization	1.34X	More C and pointers are used.Verification and Validation becomes difficult.
Manual code	1X	A separate buffer management module is also part of the layer.

2.5 Problems with optimization in General with Cadvanced code generator

Optimization of code size and performance is possible with SDL auto code generation, but at what cost is really what matters. If the application is more optimized then, it lacks the following important characteristics.

- More functionality need to be implemented in C, which would lead to a lower productivity, as the simulator can't go through the C code. It impacts the basic advantage of Verification and Validation on Design.
- It also impacts the readability and maintainability of SDL Code, which are also the prime reasons for using SDL.
- End up using more tool specific features, which might not be a desirable feature of the application, as it reduces the portability factor.

3. Summary of subsequent usage (Cmicro)

The organization views the above project as partially successful. The code generator technology had been inducted conditionally in infrastructure domain, where code size was not an issue. However subscriber domain require optimized code with better performance. So the work was continued in this direction. The scene was changed when the Cmicro code generator was used on a pilot work. Though it has restrictions in terms of SDL language features and automatic support of integration with RTOS, the code generator has given optimized code (10% more than manual) than with the Cadvanced code generator, without fore-going on readability, maintainability, testability, etc. The legacy code has been integrated successfully with guidelines to minimize the tool generated overhead functions, which contribute to code size increase.

- **Reasons for optimized code with Cmicro**
  - Scalable kernel, conditional inclusion of SDL features based on their usage in SDL
  - Restrictions on language features, that contribute to Code size
  - Following guidelines and further restrictions without compromising on primary benefits of using SDL such as maintainability, testability, readability etc.
  - Scripts usage for optimization

4. Advantages and Disadvanges of using SDL Auto Code Generation

Advantages

- Telelogic Tau tool provides integrated environment for developing products.
- SDL based development gives better Cycle Time, Productivity and Quality.
- Developing test cases with script language, provided by the tool, is very simple and enhances productivity for simulation.
- Testing is easier, because of tracing in SDL and MSC during simulation and Validation.
- Excellent quality could be obtained because of visual tools for error detection and correction, and also ease of the language.
- Enhancement/Maintenance is faster and easier because of graphical language and maintenance of SDL design.
- Coverage details can be obtained for test execution.
- Automation of regression testing.

Disadvantages

- Scripts or tool specific features required for code optimization lead to a deveiation from standard SDL, which consequently impacts portability.

- System and interoperability test has to be done with the generated code. At first instance the tester will find difficulty in understanding the generated code, because the code is very cryptic and consists of macros.
- Requires formal learning and practice.

5. Challenges

The following are the open challenges to maximize benefits from this technology.

- Post-Release issues such as on-site testing, feature addition etc.
- An integrated approach and tool facilities.
- Cost of the tools.
- Integration with data bases.
- Using the auto code for complex and critical systems.
- Maintaining the Design Vs Code.
- Changing the mental model of engineers to use this technology.
- Tool specific features usage Vs Performance, Code Size, Portability, Readability, Maintainability etc.
- Automatic validation for large systems.
- Universal metrics and estimation for SDL based developments
- Quality of the tool Vs Benefits

5.1 Solution proposed for few challenges

- If ITU-T and SDL-Forum consider this language as an implementation methodology for software development, focus would come on efficient algorithms to map the SDL code to target code.
- Wider publicity to be given in the telecom community in Asia-Pac, where its usage is limited to a few companies.
- Research and Implementation gap need to optimized, as the current state-of-art technology would impact significantly

6. Impact and Conclusion

The following tables summarizes the impact and conclusions.

Table 3: Impact of using the Technology in Protocol Domain

Attribute	Improvement factor
Cycle Time	1.2 to 2X
Productivity	1.2 to 2X
Quality	2 to 5X

Table 3: Impact of using the Technology in Protocol Domain

Attribute	Improvement factor
Effort	1.2 to 2X

Table 4: Conclusions

Feature	Pros	Cons
SDL	Provides better communication and understanding	Strongly typed language
Simulation and Validation	Better CTR for testing Useful for Unit and Module integration testing	Learning time needed
OMT/UML analysis	Rigorous	Good, if tailored for the project
Auto code generation	Multi target and multiple integrations CTR will be good	Learning time needed Code size will be more Portability may become an issue when tool specific constructs are used
System testing	Connectivity of the debugger with simulator	Learning time needed Interpretation of generated code may be needed.

Summary

The project and optimization studies showed that SDL based auto code generation offers many benefits - like Cycle Time, Productivity and Quality. As time progresses the code generator will be optimized resulting in better code, which can be used for tight memory and performance applications such as embedded and micro controller systems. The tool's Verification and Validation facilities are very good, which contributes to substantial improvements in Quality.

Acknowledgments

The author would like to thank Gautam Goel, Swaminathan, Albee Vimal, Jacob, Nirmal Abraham and all the members of 'SDL Users Group' for supporting and sharing their ideas.

References

[1] ITU-T Recommendation Z.100, CCITT Specification and Description Language(SDL)

[2] Jan Ellesberger, Dieter Hogrefe, Amardeo Sharma. "SDL Formal Object-Oriented Language for Communicating Systems", Prentice Hall, 1997.

[3] Telelogic Tau 3.X, SOMT Methodology Guidelines and other reference manual sets.

[4] Electronic Industries Association "EIA 709.1 Control Network Protocol Specification", Arlington, VA.

[5] Verilog, SA, ObjectGEODEV1.2 reference manual sets-1996.

Glossary of Terms

ASN.1 - Abstract Syntax Notation

HLD - High Level Design

ITU-T - International Telecommunications Union

MSC - Message Sequence Charts

MIEL - Motorola India Electronics Ltd

OO - Object Orientation

OMT - Object Modelling Technique

RA - Requirement Analysis

RTL - Run Time Library

SDT - SDL Design Tool

SDL - Specification and Description Language

SDLM - SDL Methodology

TTCN - Tree and Tabular Combined Notation

UML - Unified Modelling Techniques