Telecommunication protocols development, simulation and code generation tool

Roman M. Dmitrienko, *RSFLabs*

The subject of our work is an original interactive telecommunication protocols design, simulation, and prototyping tool set. The following subjects were investigated during the research phase of the project:

- Efficient modeling of dynamic, unpredictable data structures,

- Reliable yet tunable code generation routines.

The idea of this presentation is to show modern efficient methods of design of efficient complex software solutions.

API – Application programming interface,
ASN.1 – Abstract Syntax Notation 1,
CASE – Computer-Aided Software Engineering,
DTD – Document Type Definition,
GCC – GNU Compilers Collection,
GPL – General Public License,
GUI – Graphical User Interface,
IPC – InterProcess Communication
LGPL – Lesser General Public License,
MSC – Message Sequence Chart,
OS – Operating System,
RTOS – Real Time Operating System,
SDL – Specification and Description Language,
SDL-RT – Specification and Description Language for real time applications,
XML – eXtensible Markup Language

I. Introduction

Our research and development of the software is conducted as a part of an internal corporate project aiming the rapid prototyping of custom telecommunication solutions. This project has two extremely important distinguishing factors:

- The systems under development are subject to continuous changes,
- The reliability of the systems being developed is much more important than their performance.

In order to cut the systems prototyping and development costs, it was decided to employ CASE rapid prototyping tools. The two aforementioned factors imposed certain restrictions and requirements on the software development process, tools and algorithms involved, thus stipulating the need of in-house CASE software development. As an additional cost-cutting measure, only open Linux-based tools and libraries were employed during the development phase.

Manuscript received March 20, 2009.

R. M. Dmitrienko is an engineer of RSFLabs, Russia, N.Novgorod, 603104, Nartova, 6. phone: +7831-278-90-47 url: www.rsflabs.biz

II. Project Internals

The idea of the software, while quite complicated, is fairly straightforward. From the user's point of view, the standard workflow should include defining the system using SDL-RT [1] language, eliminating possible errors, running source code templates generation routine, as well as optionally simulating the work of the protocol and tracing its input/output, estimating the characteristics of the protocol, generating and running test scenarios. When all of the required phases are finished, the user should inspect and complete the generated source code templates as needed, deploy the system, run smoke tests etc. The former is achieved using the software we are developing; the latter is done by the user.

The major problems we met were:

- Efficient, reliable modeling of dynamic, constantly changing data structures during the system design phase,
- Entirely customizable yet user-friendly and non-obscure code generation.

A. Dynamic systems modeling

As already mentioned, the user should be able to define the system using the SDL-RT language. This language is powerful enough and one is able to describe procedures, processes, blocks and systems using it. However, some of the parts of the target system have to be implemented manually; this is why we are talking about the “source code templates” generation, not the source code per se. At the moment, our software provides the user with the means of describing processes and procedures. It is planned to expand it with the means of process blocks, classes and systems definition.

In order to cut the development time and costs and to improve the user experience, we have chosen the wxShapeFramework [2] library as the diagram drawing engine. It is an open source library intended to be used with wxWidgets. While being simple enough to integrate with our project, it is a powerful, flexible and easy to customize tool that suits our needs perfectly. We have expanded it with our custom set of primitives according to the SDL-RT standard.

While the user draws the diagram, a corresponding graph is being built. This graph represents the structure of the diagram; it is an internal wxShapeFramework data structure. It is used to store the information about the diagram itself, i.e. the types of elements and relations between them; it also allows for such basic operations as file input/output, clipboard operations, undo/redo framework. Hence, it is of little interest for us.

At the same time, a complex system logics data structure is being built. It stores information about the system itself, as defined by the SDL-RT diagram. This information includes the definitions of the processes, procedures, variables, signals, gates etc. Every process, procedure, system etc. is described by the corresponding C++ class instance. Such classes store information about local data definitions, as well as the graphs that represent the algorithms being described. Every node of the graph stores its parameters, i.e. those that are specified by the SDL-RT standard.

While the system logics data structure is being built, it is being analyzed in real time; the errors and warnings are being reported to the user.

The diagrams are serialized in XML [3] files by the means of wxShapeFramework. Those are enough to store all the data needed to recreate the diagram, the locations and the parameters of all of the elements and their relations.

In order to provide our software with basic means of interoperability with existing and future third-party products, we are also working on a textual SDL-RT exporting routine according to the SDL-RT standard definition. Textual SDL-RT representation is actually an XML representation; its DTD can be found in the text of the SDL-RT standard.

B.Customizable source code generation

When the system is defined and all the possible errors that prohibit code generation are eliminated, the user is able to launch the target code template generation routine. While designing this routine, we have faced several challenges; not all of our solutions are considered final yet and hence the information given here is a subject to change.

Our goal at this stage is to provide efficient, robust source code templates generation routine. Our specific requirement is that it should be flexible enough in order to both provide users with advanced means of controlling the code generation process and to allow for code generation for different target platforms and architectures.

While designing the code generation functionality, we kept in mind the fact that different real-time operating systems have different APIs and possibilities. However, in order to provide for implementation of the SDL-RT defined systems, a predefined set of functions should be supported by the target OS API. These functions include real-time process creation and destruction, fast signal-based IPC mechanism (including, at least, means of signal description, allocation, exchange and destruction), timers and semaphores. The debugging output functions are also needed. Most of the other actions described by SDL-RT are simple components of algorithms (flow control, loops, jumps etc) and thus are possible to implement using basic C89 [4] instructions.

Hence, it was decided to design an intermediate pseudo-language that should be able to describe both basic algorithms and basic IPC and synchronization functionality (signals, semaphores, timers). The textual SDL-RT representation is not quite suitable due to its structure (XML data). This pseudo-language source code is generated upon the user request when the system definition is finished and all the possible errors are eliminated.

This pseudo-language code may be stored in external files, but is not intended for user manipulation. As soon as this representation is built, the parser is run and the target C89 source code template is built.

In order to allow for source code generation for different target platforms (that obviously have different APIs), different rule sets should be applied to the generated source code. They describe the means of working with signals, timers and semaphores. Those rule sets are defined in external files and are available for user inspection and manipulation in order to provide further flexibility and extensibility of our software.

C.Optional functionality

Apart from the system definition and source code templates generation, our software should also be able to implement other functionality. In order to allow for such extensibility, we have designed the plug-in architecture which is quite simple itself. The plug-in modules receive the definition of the system as generated on the step 1, perform their actions and report results to the user.

We are considering implementation of following plug-in modules: debugging, tracing, simulation, performance analysis (including bottlenecks detection), fine-tuning.

The overview of the workflow is given at the figure 1.

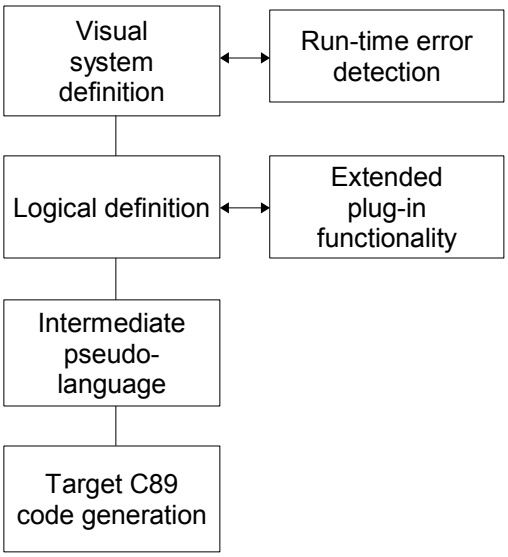

Figure 1: Workflow overview

III.Languages And Notations

There is a variety of protocol description's methods, languages, and algorithms. Most of them are governed by such well-known international organizations as ETSI, ITU, OMG, ISO etc. In order to achieve our goals, we had to choose the most appropriate, relevant, recognized ones.

We have chosen SDL-RT [1] as the core systems description language. The traditionally employed language, SDL [5], did not suit our exact needs due to the following

drawbacks:

- Out-of-date data types and data description tools. ASN.1 [6] notation has a lot of advanced means of data types definition, but it is not quite convenient for definition of systems and for automated code generation.
- Out-of-date syntax. Formalization is not enforced enough (i.e. different standards descriptions that make use of SDL often have different conventions).
- No semaphores. It is especially inconvenient during real-time systems design.
- No pointers. Most of the embedded and real-time systems are implemented in C, which depends heavily on pointers.

SDL-RT is designed to eliminate the problems of classic SDL. Some of the original SDL features were eliminated as well, as they are almost never used in modern real-time systems design.

We have also chosen MSC [7] as a message flow visualization and debugging tool. It is also extended as a part of SDL-RT project.

IV. Target Platforms And APIs

Actually, a target platform for the source code templates generated by our software is any OS (or RTOS) that both has C89-compliant compilers available and supports such common features as message exchange, semaphores and timers. A lot of different hardware platforms is currently supported by GNU/GCC [8] compilers. One of the specific requirements of our project is allowing users to tweak code generation rules, in order to make it possible to deploy the generated software for different target OS APIs. Our primary target API is ENEA LINX [9] library, which is intended to be used with Linux.

V. Operating System, Tools And Libraries

We have chosen Linux as a host operating system. It has gained popularity during last few years and is widely recognized not only as an OS for servers or software developers, but also as a desktop OS. Being a free, open source operating system, Linux is a great yet low-cost software development environment. As a consequence, we have decided to use GNU/GCC g++ compiler.

Another problem we have faced was choosing the right GUI toolkit. The most popular ones are Qt [10], wxWidgets [11] and GTK+ [12]. All of them provide developers with similar possibilities, but we have chosen wxWidgets. It is free from license limitations (Qt was not licensed under LGPL at the time when the decision was made), it uses native GUI controls under target operating systems whenever possible, its build system is not as complicated as the one of Qt. wxWidgets also provides developers with auxiliary non-GUI classes, thus simplifying crossplatform development. Hence, our software will be portable and most likely available for Windows users.

VI. Current Status And Future Prospect

At the current phase, an alpha version of the software was developed. It provides users with basic diagram drawing tools, error reporting and code generation tools.

A lot of work is being done to improve the code generation algorithms. We are planning to introduce new debugging features, real-time protocol simulation, performance estimations. We also plan to extend the list of supported target platforms to include the most popular and widely employed systems. Our long-term plans include implementing automated testing and test scenarios generation and detailed automated analysis of systems being designed. One of the possible long-term development plans is improving the project even further and releasing it as a commercial product.

VII. Résumé

The proposed solution of the efficient dynamically changing systems design problem consists of the following:

- Correct decomposition of the problem,
- Thorough planning of data structures,
- Thorough modeling of algorithms,
- Using customizable source code generation routines,
- Using open development tools, libraries and solutions

References

[1] SDL-RT. http://www.sdl-rt.org
[2] wxShapeFramework library. http://wxcode.sourceforge.net/components/shapeframework/
[3] XML. http://www.w3.org/XML/
[4] ANSI C. ANSI X3.159-1989 "Programming Language C."
[5] SDL. ITU-T Z.100
[6] ASN.1. ITU-T X.680
[7] MSC. ITU-T Z.120
[8] GNU/GCC. http://gcc.gnu.org/
[9] ENEA LINX. http://www.enea.com/Templates/Product____27016.aspx
[10] Qt. http://www.qtsoftware.com/
[11] wxWidgets. http://www.wxwidgets.org
[12] GTK+. http://www.gtk.org