EXTENDED SDL-BASED TOOLS FOR RAPID PROTOTYPING OF APPLICATION SPECIFIC SIGNAL PROCESSORS

Philippe LEBLANC
Jean-Luc ROUX

1. Introduction

This paper presents the use of SDL supported by tools to design, simulate, validate and implement the event-driven command/control part of a typical distributed signal processing software application. SDL, for Specification and Description Language (ITU-Z100, 1996), is a design notation that uses blocks to describe the system structure, those blocks exchange messages through communication channels, and the dynamic behavior in the leaf blocks is described by means of extended finite state machines.

The main objective of the work was to enhance the design of embedded electronic systems, by putting the focus on system level design tools. Thus new capabilities for early prototyping and advanced validation have been developed. Also tool support has been extended along the design cycle in order to automate the design process down to code generation.

The paper is organized as follows: section 2 introduces the SDL notation and associated tool support; this section concludes on the limitations identified in SDL when dealing with HW-SW co-design of embedded systems (Embedsys, 1996) and presents the extensions that have been introduced for this purpose; section 3 describes the context in which the use of SDL has been explored for application-specific signal processors; then the extensions required for a proper application of SDL in this domain are detailed in sections 4 to 6, i.e. introduction of Statecharts in SDL via

F. J. Rammig (ed.), *Distributed and Parallel Embedded Systems*

hierarchical states, distributed simulation and co-execution, non-regression testing of SDL design.

This work has been initiated two years ago within the DARPA-RASSP program (RASSP, 1995), and is being completed within the current ESPRIT-COMITY project (COMITY, 1997).

2. The SDL technology

2.1. SDL, a graphical formalism for real-time system modeling

SDL is a specification and design language (ITU-Z100, 1996) which has been standardized by the ITU (International Telecommunications Union). SDL is based on communicating finite state machines. The latter, called processes in SDL, are extended by data and communicate together by exchanging asynchronous messages, called signals. These signals are stored in FIFO queues, there is one common queue for each process. A complete set of graphical and formal symbols allows the designer to describe all the dynamics of a process. Main constructs available are: sending and receiving of signals, value assignment, timer manipulation, creation and deletion of process instances, function calls.

All these features make SDL well-suited for the design of event-driven, multi-tasking and distributed systems. SDL is largely used for the development of software-intensive applications, for example in Telecom systems. SDL-based tools allow the validation of a system design, and the automatic transformation of SDL descriptions into executable code.

SDL is often used jointly with MSC, Message Sequence Chart (ITU-Z120, 1996), which is also an ITU recommendation. Basically MSC diagrams correspond to chronologies of horizontal arrows, representing exchanged events, drawn on vertical bars, representing system's components. Figure 1 below depicts SDL and the associated MSC formalism. These graphical notations allows for intuitive modeling of systems. Moreover their underlying semantics guarantee the "one diagram, one meaning" postulate, and allow to check the consistency between MSC and SDL diagrams.

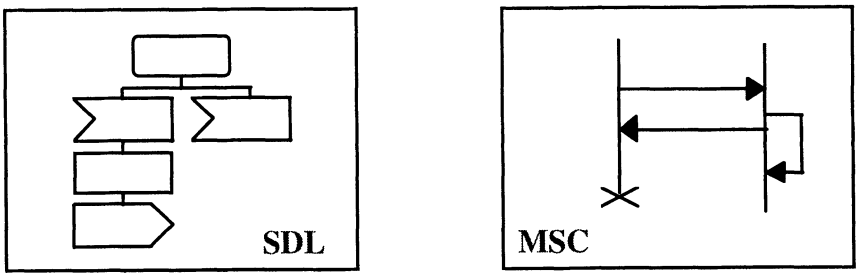

Figure 1. SDL and MSC modeling techniques

2.2. SDL-based tools

SDL has been experimented in RASSP through the application of the *Object*GEODE toolset to a signal processing application. *Object*GEODE (VERILOG, 1997) is an environment that supports the SDL language (Figure 2): it provides a graphical interface to specify in SDL, as well as with associated languages MSC and UML (OMG-UML, 1997), efficient simulation tools, and full C autocoding capability that targets usual real-time operating systems.

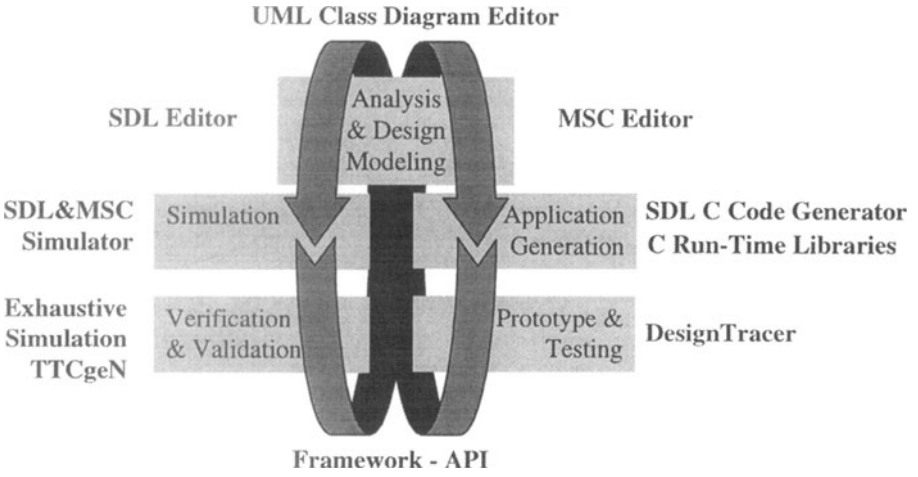

Figure 2. *Object*GEODE toolset

Therefore, the designer with *Object*GEODE can model its system, perform a formal verification by simulation (three modes are available: interactive, random and exhaustive), and generate automatically the code for producing the executable software application.

2.3. Limitations and extensions to SDL for HW-SW co-design

However the SDL notation and tools have two major limitations when dealing with HW-SW co-design and co-simulation:

- Describing the behavior of HW-SW system by means of nested states which represent the system's working modes is natural. Unfortunately SDL does not support a hierarchy of states, and actions cannot be associated to the entry or the exit of a state.
- The command program (SW part) and the signal processor (HW part) must be tested independently. Unfortunately *Object*GEODE simulation is monolithic: the complete model is simulated as a whole and parts cannot be tested separately.

In order to solve these issues, extensions have been specified and developed. The project has started in mid-1997, in the context of a sub-contracting to Lockheed

Martin Advanced Technology Labs (LM-ATL, 1997), to realize a project in the framework of RASSP.

In addition, another extension has been developed to automate non-regression testing: SDL operators are used to model the call to functions driving the signal processor; thus the SDL simulation has been extended to include these functions calls in the generated MSCs, and a semantic comparator has been developed to detect discrepancies in the chronologies of functions calls (including input parameters).

3. RASSP and the engineering of signal processing applications

RASSP, for Rapid Prototyping of Application Specific Signal Processors, is "a major DARPA/Tri-Service initiative to reinvent the development process of embedded digital signal processors (DSPs)", as announced in the statement of purpose (RASSP, 1995). They aim to achieve no less than a "4x reduction in the time from concept to fielded prototype on both new designs and design upgrades, with similar improvements in life cycle cost, quality and supportability".

3.1. RASSP design methodology

A general overview of RASSP methodology for the design of embedded DSPs is represented in Figure 3 below. The system is formally specified and the resulting model can be executed via simulation. New methods for partitioning that enable reuse of existing designs and evaluation of each HW/SW solution are applied. Finally, physical prototypes are automatically generated for both HW and SW parts.

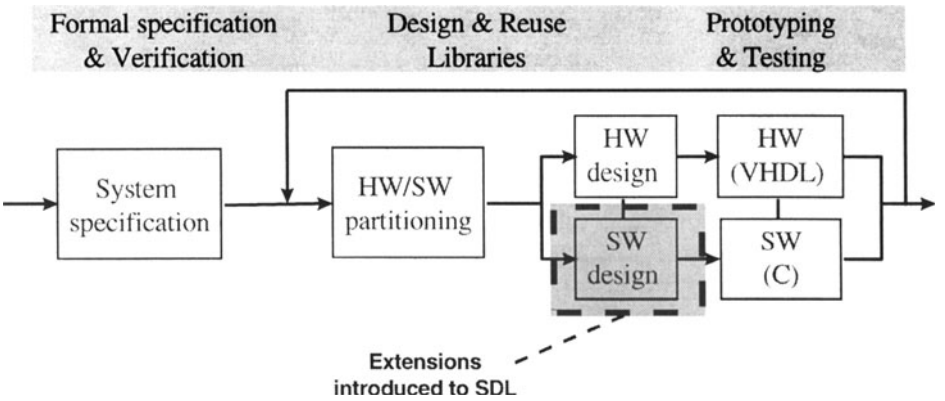

Figure 3. RASSP design methodology

3.2. Autocoding project with Lockheed-Martin ATL

In the RASSP context, Lockheed-Martin Advanced Technology Laboratory has sub-

contracted Verilog to develop a solution for autocoding from SDL using the *Object*GEODE toolset (LM-ATL, 1997).

A typical large scale architecture consists of a signal processing system which communicates with a command/control system. The signal processing system performs the high-bandwidth encoding/decoding and executes on DSPs. The signal processing system is modeled using a dataflow graph (DFG), and the command program (CP) is represented by a finite state machine. The command program is strongly related to the signal processing system, but it is often designed as part of the command/control system.

The command program (Figure 4) can be seen as an interface between the signal processing system and the rest of the command/control system. It translates system-derived or user inputs into commands accepted by the signal processing system and forwards the results produced.

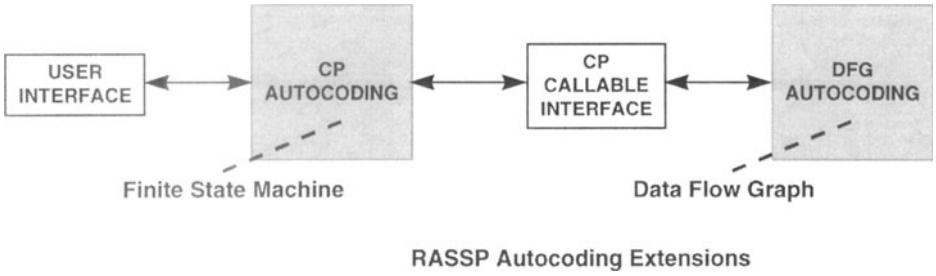

Figure 4. Command Program (CP) autocoding

4. Hierarchical states

SDL users have often expressed the need to describe hierarchical state machines. This is for example the case in the autocoding project (Figure 4), where hierarchical states are used to model the system operational modes monitored by the command program. In addition when handling hierarchical states, we need entering/leaving mechanisms; this can be achieved by entry/exit actions that are used in the autocoding project, to reprogram the signal processor when switching the modes.

Thus based on Harel's Statechart notation (Harel, 1996), we have introduced nested states and entry/exit actions in the SDL language.

Compared to the standardized SDL, the solution has consisted in introducing the concept of macro-state: a macro-state corresponds to an SDL Transition diagram, i.e. a diagram made of transitions starting from sub-states, the latter can themselves be macro-states.

Entry and exit actions have also been introduced in a similar way: two partial transitions, one starting from a label named "Entry", another one starting from a label named "Exit", can be added to a Transition diagram associated to a macro-state. The resulting behavior is that all the transitions starting from this macro-state are enriched

at the beginning by the "Exit" transition, and all the transitions ending to this macro-state are enriched at the end by the "Entry" transition.

When building the model, the designer can use macro-states and entry and exit actions in accordance with the rules given above. In order to simulate the model or to produce the executable code from this model, it is necessary to start a utility flattening the nested transitions and introducing the entry and exit actions, in such a way that the resulting model conforms to the SDL notation (ITU-Z100, 1996).

5. Distributed simulation and co-execution

The designer may want to execute the HW and SW parts separately. For this purpose, *Object*GEODE toolset has been enhanced by introducing the concepts of MuT (Model under Test) and TM (Test Model): the MuT represents the signal processor, it is driven by the TM which represents the SW command program (Figure 5).

At the modeling stage, the complete model is built in two parts: MuT and TM. This means that the SDL system has been modeled in the form of a partition of two SDL blocks.

At the simulation stage, the designer can simulate the MuT and the TM on two separate machines, allowing real independent execution. The authorized exchanges between the TM and the MuT consist in signals and data.

At the code generation stage, the designer can generate the C code for the MuT only and then drive the execution of the generated C code on the target, from the TM side which still runs in simulation mode, i.e. the simulated TM plays the role of a User Interface for the MuT.

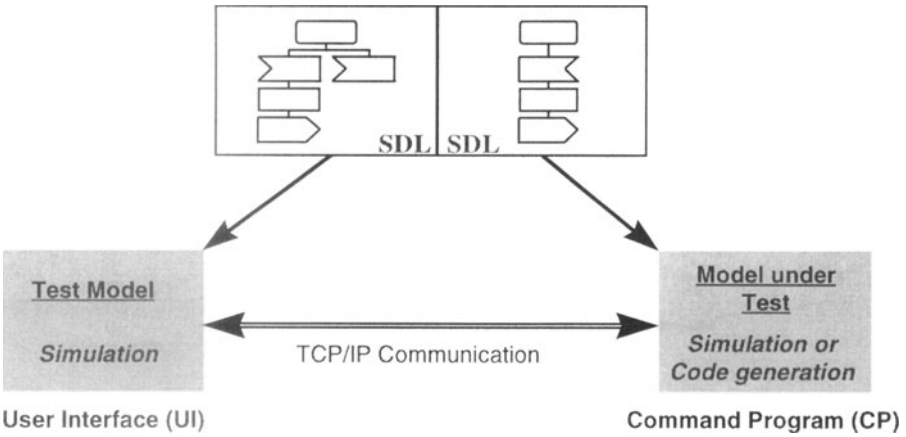

Figure 5. Co-simulation / Co-execution SDL-SDL

6. Non-regression testing

In order to automate non-regression testing, the *Object*GEODE simulator has been enhanced with a semantic comparator of MSCs. The main objective was to perform incremental testing while system design is evolving. During simulation, MSCs are produced reflecting the exchanges observed between the different components of the system (Figure 6).

First of all, the production of MSCs has been enriched by taking into account the calls of functions (in SDL, this corresponds to the call of Abstract Data Types operators). Input parameter values are also taken into account. From the HW-SW co-modeling point of view, the functions represent the lower level signal processor control commands such as *create queue*, *write variable* etc..

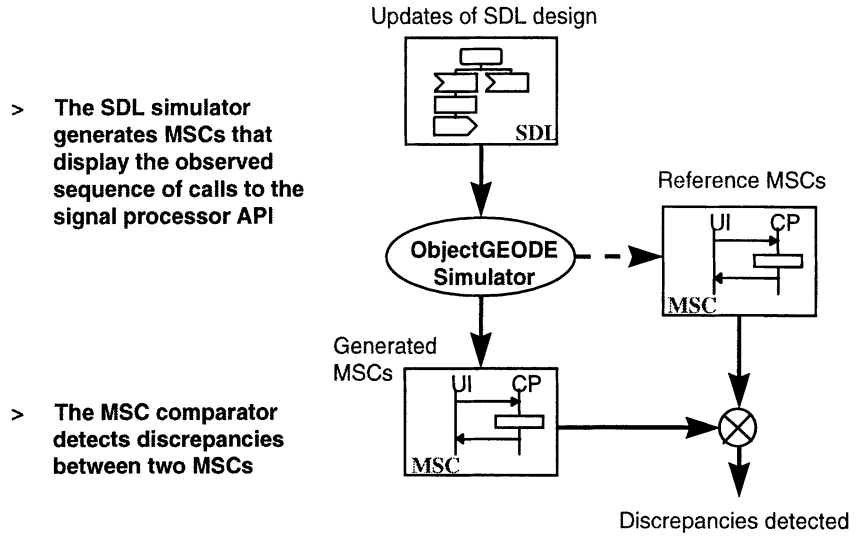

Figure 6. Non-regression testing with MSCs

Then during a simulation session, the designer can generate the MSC corresponding to the current scenario. Afterwards, the designer can compare the produced MSC with another reference MSC. Semantic discrepancies are detected and can be located graphically in the SDL design.

7. Current tool achievements & further work

The previous three extensions have been developed in the context of RASSP. They are now being completed in the COMITY project whose objective is to develop a set of techniques and tools for the design and prototyping of embedded systems, with application to telecommunication, automotive and aerospace domains.

The extension to support hierarchical states is already available in the latest commercial release of *Object*GEODE, as well as an MSC comparator but limited to a syntactical comparison only.

At last, we are currently implementing a new coupling between the *Object*GEODE toolset and a VHDL generator through a partitioning module for HW-SW co-design (EETimes, 1998).

8. Conclusion

The extensions that have been developed in this project, bring highly valuable benefits to the SDL designer in the field of embedded systems. Thus HW-SW system design can be simulated and verified earlier; the designer realizes an abstract modeling and simulation takes place at system level. Behavioral design errors are detected faster, non-regression testing is performed semi-automatically, and the design process is always repeatable.

Another important advantage with *Object*GEODE tool is that software design in SDL can be automatically transformed into full executable code. The tool delivers an optimized and portable application, that may be distributed over several processors. The last extension to the tool, under development, will offer the possibility to produce automatically VHDL code from SDL system level design.

References

(COMITY, 1997). European ESPRIT R&D program - *Co-design Method and Integrated Tools for Advanced Embedded Systems*, project N°23015, http://www.intranet.gr/COMITY - index.html.

(EETimes, 1998). EE Times OnLine, *System-level design tools brewing*, by M. Santarini, http://www.eet.com/story/OEG19981109S0014.

(Embedsys, 1996). Proceedings of the *Embedded Systems Conference*, San Jose, California, http://www.embedsyscon.com.

(Harel, 1996). Harel D, Naamad A., *The STATEMATE Semantics of Statecharts*, ACM Transactions on Software Engineering and Methodology, Vol.5, No.4.

(ITU-Z100, 1996). ITU-T, Recommendation Z.100, *Specification and Description Language (SDL)*, http://www.itu.ch - Electronic Bookshop, Geneva.

(ITU-Z120, 1996). ITU-T, Recommendation Z.120, *Message Sequence Chart (MSC)*, http://www.itu.ch - Electronic Bookshop, Geneva.

(LM-ATL, 1997). Lockheed Martin ATL, RASSP - *Command Program Autocoding* - http://www.atl.external.lmco.com/projects/rassp/RASSPQUADS/RASSP41.html.

(OMG-UML, 1997). Object Management Group (OMG), *UML 1.1 Documentation Set*, ftp.omg.org/pub/docs/ad.

(RASSP, 1995). USA DoD DARPA - *The Rapid prototyping of Application-Specific Signal Processors,* RASSP program, http://rassp.aticorp.org - index.html.

(VERILOG, 1997). VERILOG Products, *ObjectGEODE 3.2 Documentation Set*, http://www.verilogusa.com.