SDL to design real time software

Emmanuel Gaudin

Emmanuel Gaudin has a technical background and developed protocol stacks in SDL. He joined Telelogic in 96 as a Field Application Engineer and as an SDL trainer to finally become technical director of the French branch. He started PragmaDev in 2001 to develop an SDL-RT tool.
emmanuel.gaudin@pragmadev.com

SDL has been developed in the first place to specify telecommunication protocols. Experience showed its basic principles could be used in a wide variety of real time and embedded systems because of its functional approach and its graphical representation of the system architecture and behavior. But when it comes to design and integration on target, the SDL high level data types and concepts become a hurdle for real time designers. Furthermore some real time concepts are not covered by classical SDL such as semaphores and pointers. In the end most of SDL developers actually wrote C code in SDL and therefore had to develop custom tools to generate code out of their mix of SDL and C description. These major drawbacks obviously prevented SDL to be widely used in the industry.

Applicable domains

It appeared the structural SDL concept gathering tasks in functional blocks was a very good way to define the architecture of any real time system. This is because a real time system is running on top of a Real Time Operating System in which the basic structural element is the task. Then a set of tasks are running concurrently to realize a basic function, and basic functions are then gathered to realize more complex functions up to the whole application. SDL was offering a standard graphical representation perfectly suited to that approach.

In a real time project the definition of interfaces between functions is next after the architecture definition. An interface is basically an exchange format made of structured data and a scenario. SDL signals were coming with typed parameters based on ADTs. That was perfect to define the static interface. The Message Sequence Chart (MSC as defined in ITU-T Z.120) provided a graphical representation of dynamic data exchange between any entity in the SDL system and with its surrounding environment such as other software modules, drivers and so on...

Since real time systems are based on independent tasks running concurrently it was very important not to waist CPU time when the task had nothing to do. That led most of real time software to be based on finite state machines where the basic principle is to hang on an RTOS object such as a message queue when that task has finished its job. Once again the SDL finite state machine provided a perfect graphical representation to design such tasks.

Technical reality

SDL looks like the perfect language to specify and design real time systems, but technical reality is quite different.

Abstract Data Types are not suited for design

When it comes to design, SDL data types (ADT) appear to be a real hurdle for several reasons among which:

- ADT manipulation syntax has been defined to specify protocols, not to design them, so developers tend to be frustrated not being able to have the same precision they used to have with traditionnal programming language such as C,
- Integration of legacy code requires mapping modules between SDL data types and C or C++ data types,
- There is no SDL compiler or cross compiler on the market so translation to C language is required to implement SDL systems on a target,

- Some ADT concepts can not be directly mapped to C or C++, so specific operators need to be generated making the generated code not legible.

SDL concepts do not always fit RTOS services

When integrating SDL generated code on an RTOS, some SDL principles are not supported any more. Let us consider two basic examples:

- the priority concept in SDL applies to messages where on classical RTOS it applies to tasks;
- an SDL transition can not be interrupted in regards of Z.100 but on an RTOS, execution can be interrupted at any time especially if tasks have different priorities or if the scheduling is based on time-slicing.

To guarantee the generated code behaves as specified, an SDL virtual machine needs to be generated making integration on target difficult and access to some RTOS principles impossible.

Semaphores are unsupported

Last but not least, semaphores do not exist in SDL where it is a very common way to synchronize task in a real time operating system. This is because SDL has focused on telecommunication protocols connecting remote machines where semaphores were useless.

End users to twist the standard

Because of the above reasons, when it comes to design real time software, SDL tools -in order to keep all benefits from SDL- are very complex to handle, very expensive, and require long trainings as well as consultants when it come to integration on target.

Hence, on the field, SDL users started to actually write C code inside their SDL graphical description. Since the mix of SDL and C is not conform to SDL standard, the tools could not handle the pseudo-SDL description any more. So SDL users started to develop their own tool chain; for example each major telecom manufacturer has its own SDL to C code generator (Alcatel, Nokia, Nortel, EADS, Sagem…). That means only big companies can afford to use such a language and even then most of the benefits of using a formal language would then be lost.

Standardization

In a few years UML has expanded to a lot of application domains because of C++ adoption. But UML has nothing to offer to a real time or embedded application and C++ is only used in 40% of real time developments. In practice only 3 or 4 diagrams are used in real time applications and the benefits are very poor. Object orientation has also showed its limits when used in telecommunication application; some major telecom manufacturers now have a “back to basics” attitude.

Suggested evolutions

UML overwhelming and intensive marketing pushes SDL to re-position itself as a graphical modeling language. UML approach is a higher level approach than SDL. In order to keep SDL interest and benefits, Z.100 should be positioned as a lower level approach complementing UML: a real time profile. The following suggestions are aiming at modifying SDL positionning:

- Open up SDL to external data types,

SDL data types are the least used data types in the industry. Prefered data types are ASN.1, C, C++, and ADA. The Z.100 should open up the data description to any other language the user might want to use.

- Break the unrealistic SDL semantic,
  “A transition takes no time”, “a transition can not be interrupted”, or “a channel has no delay” are academic semantic ideas. They do not fit the real world .

- Offer extensions to other concepts.
  Depending on the application domain some additional concept might be useful such as: semaphore, priority on processes, time freeze, time spent in a transition, and so on... Z.100 should offer extensions to any extra concept needed in a description like UML does.

SDL had all the basic elements to be widely spread in the real time domain. Unfortunately it did not, mainly because it stayed exclusively focused on the telecommunication specification niche. There is a real need to ease design and to open up to the whole real time domain, an area where 90% of the developments do not use any graphical tool. That will probably be the next challenge for the standardization body in 2004.