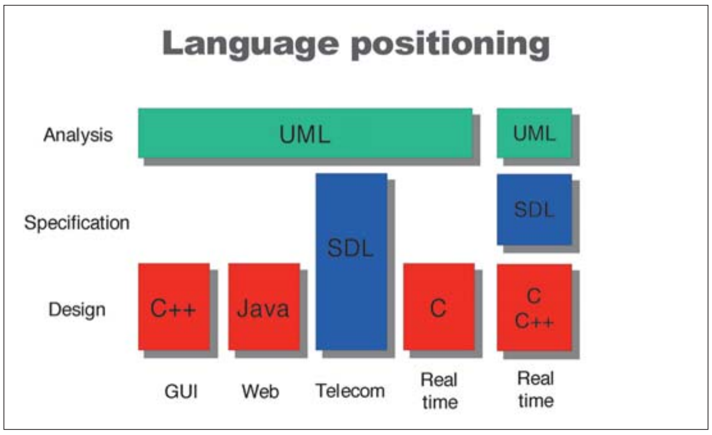
Model-driven design languages for real-time and embedded applications

By Emmanuel Gaudin, PragmaDev

Model-driven engineering is an approach to software development based on abstract models of the system to be developed.



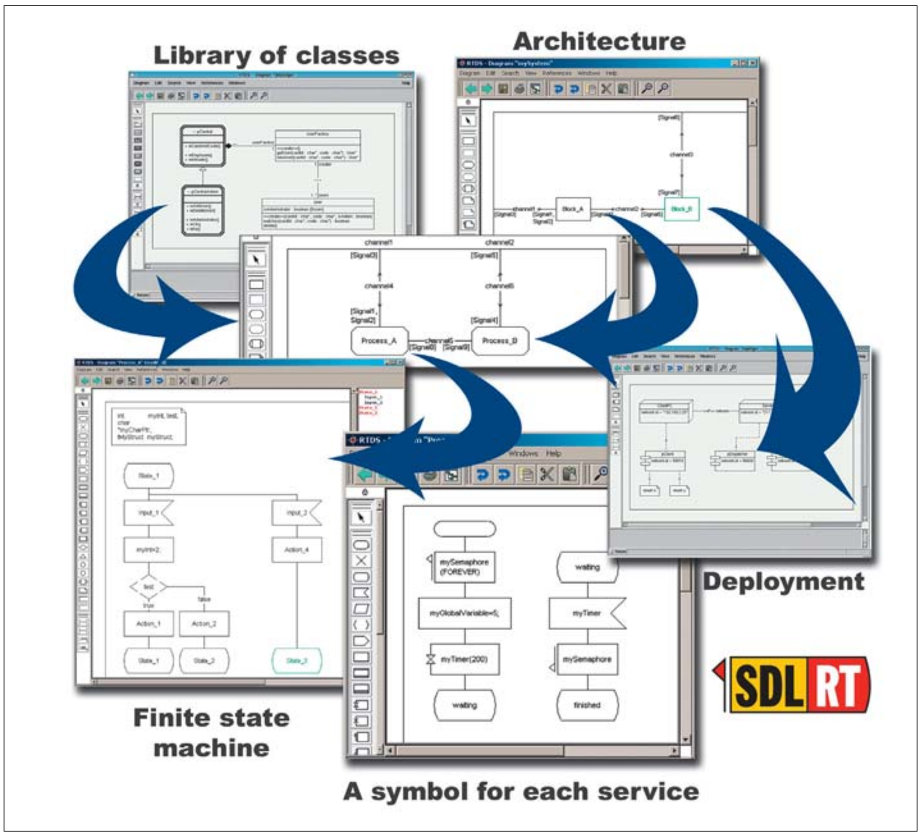
■ The best-known MDE approach is the model-driven architecture defined and registered by the object management group (OMG) in 2001. Another well-known approach is model-driven design (MDD) where the focus is narrowed in order to work on a more precise model. A model-driven approach distinguishes three types of models: an abstract model of the system under development called the platform-independent model (PIM), a model defining the platform called the platform definition model (PDM), and an implementable model of the system called the platform-specific model (PSM). The PIM is the system under development and the PDM defines the rules to transform the PIM into a PSM. So in practice the team works on the PIM, the PDM is defined by the application domain or the company, and the PSM is automatically generated out of the PIM and the PDM.

To be efficient, the PIM must be abstract enough to be independent of the platform on which the system will be implemented, but at the same time it should be precise enough to be translated into a PSM. So in order to successfully translate the model, the PIM relies on a virtual machine the characteristics of which are a number of basic services and a semantic strong enough to be expressive. In the 80s, the International Telecommunication Union standardised a language to describe telecommunications protocols: the specification and description language (SDL), under reference Z.100. The main goal was to describe the protocols in an unambiguous way so that all manufacturers' implementations of a standard protocol would be compatible with each other (e.g. a Nokia GSM phone would work with an Ericsson base station). The European Telecommunications Standardisation Institute has extensively used SDL to describe telecommunication standards and it is obvious to state that compatibility has been successfully achieved. The SDL standard is regularly updated and major new versions have been produced every four years since 1984. The 1988 (SDL '88) version was considered to be the first usable version, the 1992 version introduced object orientation, and the 2000 version aligned SDL with some UML concepts such as the class diagram.

The main characteristics of SDL are: it is a graphical language, it is object oriented, it is event oriented, the model is independent from implementation, the language defines a strong semantic of execution and it contains abstract data types. Because it embeds abstract data types and a syntax to manipulate data, SDL models are formal (complete and non-ambiguous). An SDL model can be fully described because of this characteristic but it does not have to be. It is possible to describe non-deterministic models with, for example, the use of the ANY keyword allowing to describe "any" input or "any" execution path. It is also possible to leave out some operations with the use of undefined "OPERATORS" describing an operation interface and leaving the implementation to the designers. An SDL model can also contain informal operations written in natural language. So depending on the level of precision within the model, an SDL system can be informal or very precise. SDL has built-in concepts and services such as processes, messages, timers, and procedures. All these concepts are supported by most RTOS making implementation on a real target straightforward.

The strong semantic of SDL and its built-in services describe an SDL virtual machine on which a model is based. This is actually the main characteristic of the platform-independent model. The definition of eventual external operators, and the implementation of the SDL services provided by the SDL virtual machine constitute the actual definition of the platform, the platform definition model. From the PIM and the PDM, it is possible to fully generate the platform-specific model in an executable language such as C code. As SDL became increasingly used in telecommunications systems, it appeared its built-in concepts were pretty close to the ones used in real-time operating systems:

- the system is decomposed into tasks running concurrently,



SDL-RT diagrams

- most of the tasks are based on extended finite state machines,
- tasks communicate with messages sent to message queues,
- built-in timers help to deal with unexpected behaviour.

Most telecommunications manufacturers have used SDL to design their software and some of them have measured the added value when using this modeling language. The results are impressive: it has increased quality by a ratio of 5 and reduced the overall development time by 35% in average.

In 1997, the object management group standardised the Unified Modelling Language, a merge of different object-oriented modelling approaches, most of them coming from the database application domain. The first versions of UML, versions 1.x, were too generic to support a platform-independent model. They did not contain any of the real-time operating system concepts such as task, semaphores, and timers. They did not contain any semantic, nor any data type. In order to use UML to describe a PIM, version 2 of UML introduced the concept of profiles to make UML more precise within each application domain. A profile allows to introduce specialized concepts and some semantic within a UML model. But the OMG did not define any standardised profile. Therefore UML 2 tools have introduced their own profiles, most of the time without documenting them, making the models tied to the tools they have been designed with, and tied to the underlying profile that was used. The ITU has taken this opportunity to work on a UML profile for telecommunications systems based on SDL under the Z.109 reference. The standard should be in force by the end of 2007, at that time SDL will be a standardised UML profile for telecommunications systems.

Because UML is very abstract and informal it is mostly used in the early phases of the development process when analysing and setting the requirements on the system. When it comes to coding, traditional textual languages are at the same level as the SDL abstract data types. Because of its graphical abstractions dedicated to telecommunication systems, SDL is positioned between the very generic UML and the very specialized coding languages.

When using SDL, telecommunications manufacturers found the concepts within the language were not exactly the ones available in the real-time operating systems nor the ones in textual languages. For example, in SDL messages have priorities and processes do not; on an RTOS it is the other way around, tasks have priorities but messages do not. As another example, in SDL it is possible to define arrays indexed on reals, which is very tricky to implement in C. To fully conform to SDL, code generators had to support all the SDL concepts and produced complex and illegible code. So in order to be efficient, most of the telecommunications manufacturers broke the SDL semantic to use the one from their operating systems, and wrote C code manipulating C data types instead of the SDL data and syntax. They were actually already using what we call SDL-RT. As previously explained, SDL-RT comes from industrial practice and its first version has been written by PragmaDev so that everybody has a common way of combining UML, SDL, and C or C++. The specification is freely available on http://www.sdl-rt.otg, and is easy to read compared to an official standard. The storage format is XML so that the description is not tied to any tool. Since SDL-RT aims at all real-time and embedded applications, the semaphore concept has been introduced in the language so that each service of a real-time operating system has a dedicated graphical symbol. SDL-RT concepts are now proposed to ITU in order to be standardised. So eventually SDL-RT will most likely become a standardised UML profile for real-time applications.

SDL-RT diagrams are:

- the class diagram in which passive classes are further described in C++ and active classes are further described using the SDL finite state machine,
- the architecture diagram which decomposes the system in hierarchical functional blocks down to the process level,
- the process behaviour which is described using the SDL finite state machine because it is much more detailed than ones from UML,
- the deployment diagram for distributed systems,
- the message sequence chart (MSC) to document execution scenarios similar to the UML sequence diagram.

Each service of the RTOS has a dedicated symbol: dynamic task creation or deletion, message input and output, timer start, cancel, and time out, semaphore takes or gives.. All views are related to each other making the model of the system consistent.

Since its inception twenty years ago, SDL has been a model-driven language for the specification of telecommunications systems. Following industrial practice, SDL-RT has extended the application domain to all applications based on real-time operating systems.. In the meantime, UML 1 being too generic, UML 2 has introduced the possibility to define profiles dedicated to an application domain but has not standardised any. Each UML 2 tool has therefore implemented a proprietary profile that is rarely documented making portability to another tool impossible. The ITU is standardising a UML profile for telecommunications systems based on SDL. The profile should be finalised by the end of 2007. ■