Real Time Developer Studio

Emmanuel Gaudin
emmanuel.gaudin@pragmadev.com
PragmaDev

Dedicated to the development of a modeling tool for the development of Event driven software.

Network of partners:

- Trainings,
- Services,
- Testing,
- Products.

Distributors:

- Europe,
- Asia,
- US.
Partners

real time developer studio
References

- Universities: ENST, Polytechnica Bucarest, Telecom Beijing…

real time developer studio
Philosophy

Develop a modelling tool based on the users’ needs.
Target segment

Event driven systems
Embedded & Real time

- Decomposed in tasks running concurrently
- Communicating
- Synchronizing

Module 1
Module 2
Module 3

Message queue
Semaphore

OS-RTOS-Scheduler
hardware
“The software content is doubling about every two years. The sheer volume is making it increasingly difficult for QA and test teams to keep up with traditional tools and processes.”

*Wind River Market Survey of Device Software Testing Trends and Quality Concerns in the Embedded Industry*

*June 2010*
What modelling can solve

• Focus first on the **What** (Req) instead of focusing on the **How** (code),
• Modelling is about **Communication** and **Documentation**,  
• **Legibility** of large systems gets critical. Would you build a house without drawing detailed plans?
• Get control over **productivity**,  
• Improve **quality**.

**real time developer studio**
Requirements for a good modeling language

• The abstract model must be platform independent, as its name states.
• The abstract model must be translatable to an implementation platform.
• For that purpose, the abstract model is based on a virtual machine offering:
  • Some basic services
  • A strong enough semantic.
# Existing modelling languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDL</td>
<td>Specification and Description Language is an ITU-T standard.</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>- Event oriented,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Used by ETSI to standardize telecommunication protocols,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Formal (complete and non-ambiguous).</td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language is an OMG standard.</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>- Can be used to represent any type of systems,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Informal.</td>
<td></td>
</tr>
<tr>
<td>SysML</td>
<td>System Modelling Language</td>
<td>✗</td>
</tr>
<tr>
<td>AADL</td>
<td>Architecture Analysis Description Language</td>
<td>✓</td>
</tr>
<tr>
<td>MARTE profile</td>
<td>Modeling and Analysis of Real-Time and Embedded systems</td>
<td>✗</td>
</tr>
<tr>
<td>Z.109</td>
<td>UML profile based on SDL</td>
<td>✓</td>
</tr>
<tr>
<td>Lustre / Esterel</td>
<td>Synchronous programming languages for the development of complex reactive systems</td>
<td>✓</td>
</tr>
<tr>
<td>MATLAB</td>
<td>MAtrix LABoratory</td>
<td>✓</td>
</tr>
<tr>
<td>Autosar</td>
<td>AUTomotive Open System Architecture</td>
<td>✗</td>
</tr>
<tr>
<td>SART</td>
<td>Structured Analysis for Real Time (obsolete)</td>
<td>✗</td>
</tr>
</tbody>
</table>
Modelling languages positioning

Dynamic verification is best at identifying defects because requirements are usually dynamics

real time developer studio
No real time specificity in UML

- UML 1.x was too generic to describe a good PIM,
- UML 2.x introduced domain specific profiles,
  - Lots of tools defined proprietary profiles,
  - Z.109 is a standardized profile based on SDL.
SDL: the perfect picture

- SDL graphical abstraction (architecture, communication, behavior) brings a lot to development teams.

- SDL being formal, it is possible to simulate the system behavior on host with graphical debugging facilities.

- SDL being formal, full code generation is possible.

- SDL being object oriented, software components are reusable (ETSI telecommunication protocol standards fully use object orientation).

- SDL has the characteristics to describe a good modeling language.
SDL: the figures

Years of experience allows to quantify gains of SDL usage.

- C code: 35 to 50 mistakes per 1000 lines
- SDL code: 8 mistakes per 1000 lines

- Development time is globally reduced by 35%
  - Reduced up to 50% in the left branch of the V cycle
  - Less gain on the right side of the V because of the gap with technical reality
SDL: design problems

- All existing software modules (RTOS, drivers, legacy code) provide C APIs, not SDL,
- Some classical real time concepts are not present in SDL such as pointers and semaphores,
- SDL syntax is not suited for design.

- Integration with legacy code is difficult,
- Integration with COTS components is tricky (driver or RTOS),
- Developers are frustrated,
- Generated code is not legible,
An intermediate solution: SDL-RT

- Keep UML diagrams at high level during analysis and requirements.
- Keep the SDL graphical abstraction (architecture, communication, behavior).
- Introduce C data types and syntax instead of SDL's.
- Remove SDL concepts having no practical implementation.
- Extend SDL to deal with uncovered real-time concepts (interrupts, semaphores).
SDL-RT is:

- Available from http://www.sdl-rt.org for free,
- Legible,
- Based on a standardized textual format (XML),
- Submitted to ITU to be part of standard SDL,
- A UML real time profile.
Views: Class diagram

Relations between static classes (C++) and dynamic classes (SDL)
Views: Architecture and communication
Views: Behavior and Data

SDL abstract data types or SDL-RT C/C++ data types.

Process A
Views: Operating system services

- A semaphore take
- A timer is started
- SDL-RT state

When the timer goes off
A semaphore give
Views: Distributed systems

Physical deployment
Model views

- Library of components
- System architecture
- Interface definitions
- Application deployment
- Real time concepts
- Key points in the design
MSC: dynamic view

Message Sequence Chart

- Vertical lines represent a task, the environment or a semaphore,
- Arrows represent message exchanges, semaphore manipulations or timers.

Can be used:

- As specification
- Execution traces
RTDS: supported languages

Analysis
- UML

Specification
- SDL Z.100
- SDL-RT
- TTCN-3

Design
- C
- C++

Informal
- Fully formal
- Semi formal
- Testing
RTDS: supported languages

**Informal** modelling for requirements: UML
- Edition
- C++ stubs generation

**Semi-formal** modelling for design: SDL-RT
- Edition
- Syntaxic et semantics checking
- Code generation
- Graphical debugging

**Fully formal** modelling for specification: SDL Z.100
- Edition
- Syntaxic et semantics checking
- Simulation
- Verification
- Code generation
- Graphical debugging
- Test
Tools: the Project manager

It is the « hub » of the toolset:

- File organization (UML, SDL, SDL-RT, C, C++, H, traces and others) with packages,
- Calls the editors, the debugger, and the code generator,
- Handles code generation profiles.
Documentation generation

- Logical publications (state, transition, partition, diagram)
- Comments preceding or following the publication
  - Styles for paragraphs
  - Styles for characters
- Export format
  - RTF
  - OpenDocument
  - HTML
  - SGML
- Exported elements
  - Texts with publications
  - Index entries
  - Table of contents entries
Documentation generation

A document

The generated documentation

A publication

real time developer studio
Model simulator

A graphical debugger for fully formal models and TTCN-3 test cases

- Set breakpoints and step in the model,
- Dynamic traces.

real time developer studio
Code generation

- C++ skeleton for static classes
- C or C++ for dynamic classes
- Generated code is legible
- Generation profile wizard
- The code is:
  - Integrated with: VxWorks, OSE, OSE Epsilon, CMX RTX, Nucleus, uiTRON, Posix, ThreadX, and Win32,
  - Provided with an scheduler,
  - Royalty free,
  - Documented for customization.
Debugging architecture

The Model debugger relies on a traditional C debugger or cross debugger to provide graphical debugging.

- Model
- C code generator
- Generated C code
- Compiler
- External C/C++
- SDL-RT editor
- MSC editor
- TTCN-3 editor
- SDL-RT debugger
- C debugger
- Binary code

Real Time Developer Studio tools:
- Tornado
- Tasking
- gdb
- XRAY
- Multi

Third party tools:
Model debugger

Relies on the target semantic: processor and RTOS.

Debug in the model:
- Breakpoints, stepping, in the SDL/RT diagrams or in the generated C files,
- Dynamic MSC traces,
- Connecting an external tool is possible through a socket.
Debug features

- Switch between
  - Model
  - Generated C/C++ code
Graphical traces

Execution traces:
- States,
- Events,
- Semaphores,
- Timers.

Trace level configuration
Display of system time

MSC Diff allows to check:
- Conformity,
- Non-regression.
Model coverage

- Graphical model coverage analysis
- Merge feature
Prototyping interface

➢ Connects automatically to the simulator or the debugger.

➢ Knows about the model inputs and outputs.
Standard testing language

- Relies on basic services
  - Messages
  - Procedures
  - Timers
  - Parallel execution
- Based on TTCN-3 international standard:
  - Data types definitions or ASN.1,
  - Templates definitions,
  - Test cases,
  - Verdict,
  - Execution control.
Continuous integration

specification → simulation → validation testing

design → simulation execution → integration testing

code → execution → unit testing

real time developer studio
Model checking

- Partnership with Verimag on IF technology.
  - Exhaustive simulation,
  - Observers,
  - Test generation.
- RTDS feature
  - Export to IF,
  - Execute a script
  - Generate an MSC feedback.
Implementation

- SDL
  - Observer file
  - translate to file
  - execute

- MSC
  - TTCN

- IF
  - IF compiler
  - executable
    - state file
    - transition file
    - error file
    - Python script
      - Resulting scenario

- RTDS
  - Script file

real time developer studio
ESA Taste framework integration

Matlab → RTDS → Scade

Continuous Event driven Synchronous

Encoders / Decoders

ASN.1 → AADL → ASN.1

Pragmadev real time development tools
Project management

- Textual storage format, graphical diff tool, and an automatic merge tool provide a consistent integration with configuration management tools.
- Single diagrams can be exported as PNG, JPEG, PS, and HTML.
- Full documentation of the model can be generated.
- Traceability information and Reqtify integration.
- System requirements:
  - Solaris,
  - Windows,
  - Linux.
- Floating licenses.
Conclusion

- Three levels of modelling:
  - Informal,
  - Semi-formal,
  - Formal.

- Tools to:
  - Document,
  - Simulate,
  - Validate,
  - Test.

- Based on international standards.

- Integrated in Taste framework.
Stand alone MSC Tracer

- Trace on-line or off-line your target behavior with a standard graphical representation.
  - Self-document your test campaigns,
  - Check non-regression,
  - Easy connexion,
  - Free version available.

real time developer studio