

# Real Time Developer Studio



Emmanuel Gaudin  
emmanuel.gaudin@pragmadev.com

---

**real time developer studio**

# PragmaDev

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

Network of partners:

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

Distributors:

- Europe,
- Asia,
- US.



---

**real time developer studio**

## Partners



# References



- Universities : ENST, Polytechnica Bucarest, Telecom Beijing...

---

**real time developer studio**

# Philosophy

Develop a modelling  
tool based on the  
users' **needs**.

---

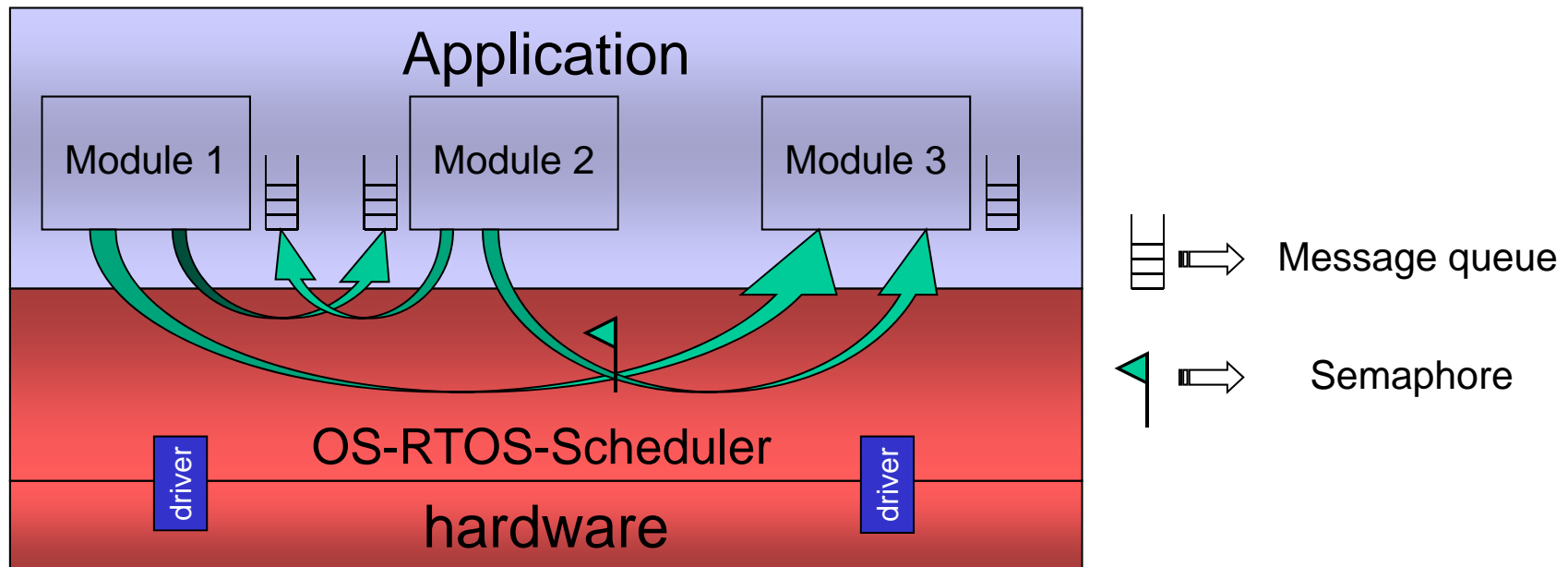
**real time developer studio**

# Target segment

## Event driven systems

Embedded & Real time

- Decomposed in tasks running concurrently
- Communicating
- Synchronizing



**real time developer studio**

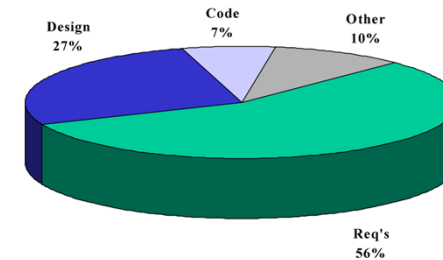
# Issues

“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*

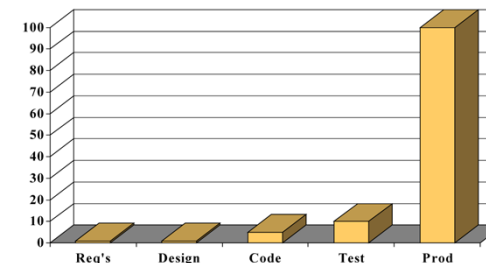
*June 2010*

Where Defects Originate



Never forget the 1:10:100 rule

The Relative Cost of Fixing Defects



Source: James Martin study

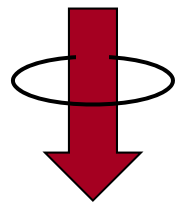
## What modelling can solve

- 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**.



# Model driven development

- Vocabulary
  - PIM: Platform Independant Model
  - PDM: Platform Definition Model
  - PSM: Platform Specific Model



The goal is to transform the abstract model (PIM) to a concrete model (PSM) using the platform definition model (PDM).

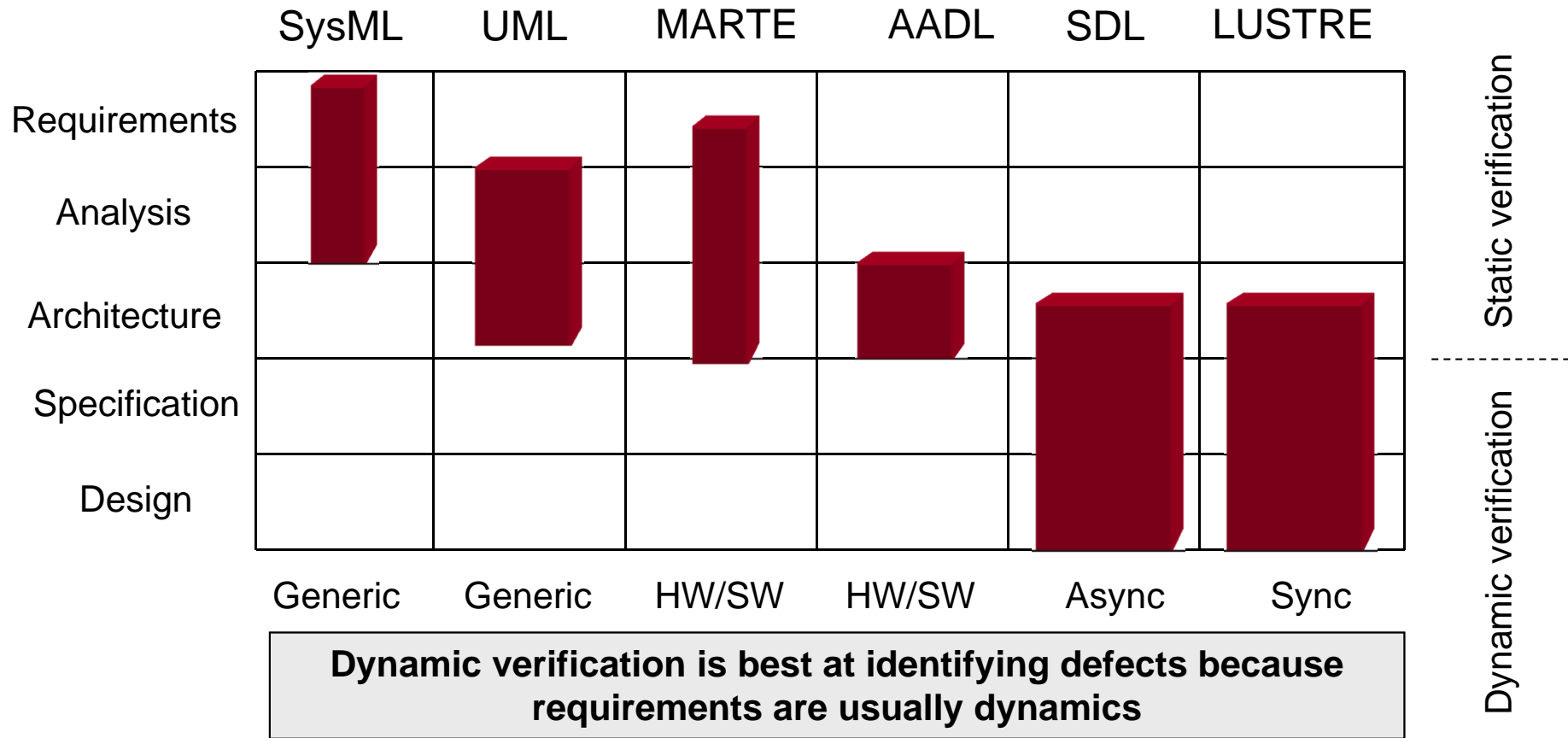
## Requirements for a good PIM

- The abstract model must be platform independant, 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

<b>SDL</b>	<p>Specification and Description Language is an ITU-T standard.</p> <ul style="list-style-type: none"> <li>• Event oriented,</li> <li>• Used by ETSI to standardize telecommunication protocols,</li> <li>• Formal (complete and non-ambiguous).</li> </ul>
<b>UML</b>	<p>Unified Modeling Language is an OMG standard.</p> <ul style="list-style-type: none"> <li>• Can be used to represent any type of systems,</li> <li>• Informal.</li> </ul>
<b>SysML</b>	System Modelling Language
<b>AADL</b>	Architecture Analysis Description Language
<b>MARTE profile</b>	Modeling and Analysis of Real-Time and Embedded systems
<b>Z.109</b>	UML profile based on SDL
<b>Lustre / Esterel</b>	Synchronous programming languages for the development of complex reactive systems
<b>MATLAB</b>	MAtrix LABoratory
<b>Autosar</b>	AUTomotive Open System Architecture
<b>SART</b>	Structured Analysis for Real Time (obsolete)

# Modelling languages positioning



## 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 PIM**.

## SDL: the figures

Years of experience allows to quantify gains of SDL usage.

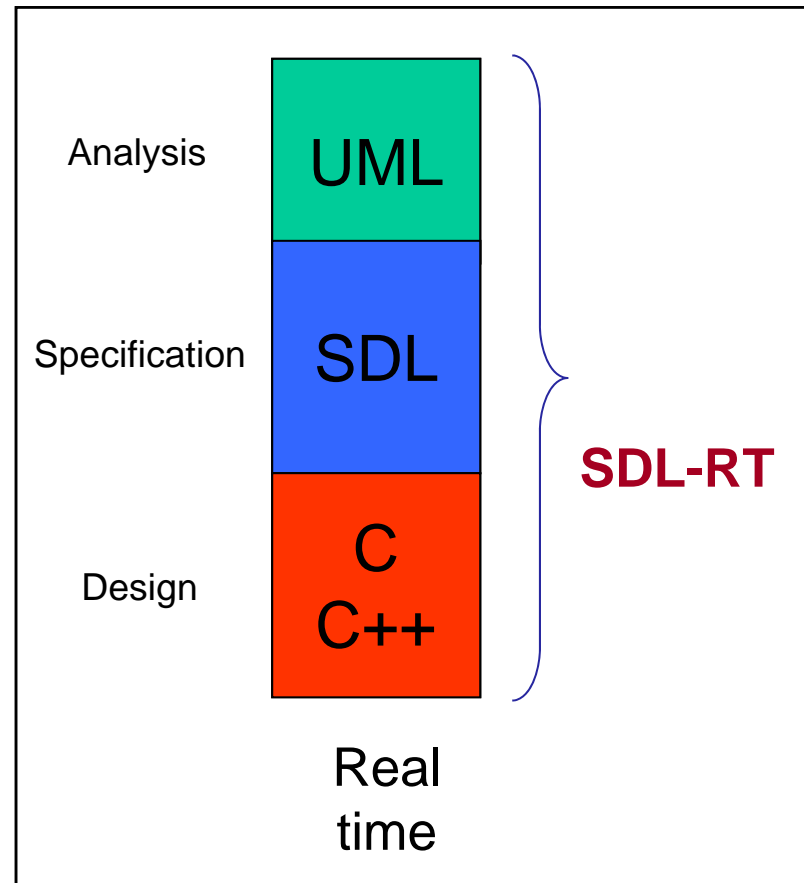
- C code: 35 to 50 mistakes per 1000 lines
- **SDL** code: **8** mistakes per 1000 lines
- Development time is globally reduced by **35%**
  - Reduced up to 50% in the left branch of the V cycle
  - Less gain on the right side of the V because of the gap with technical reality

## 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).





specification & description language - real time

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.

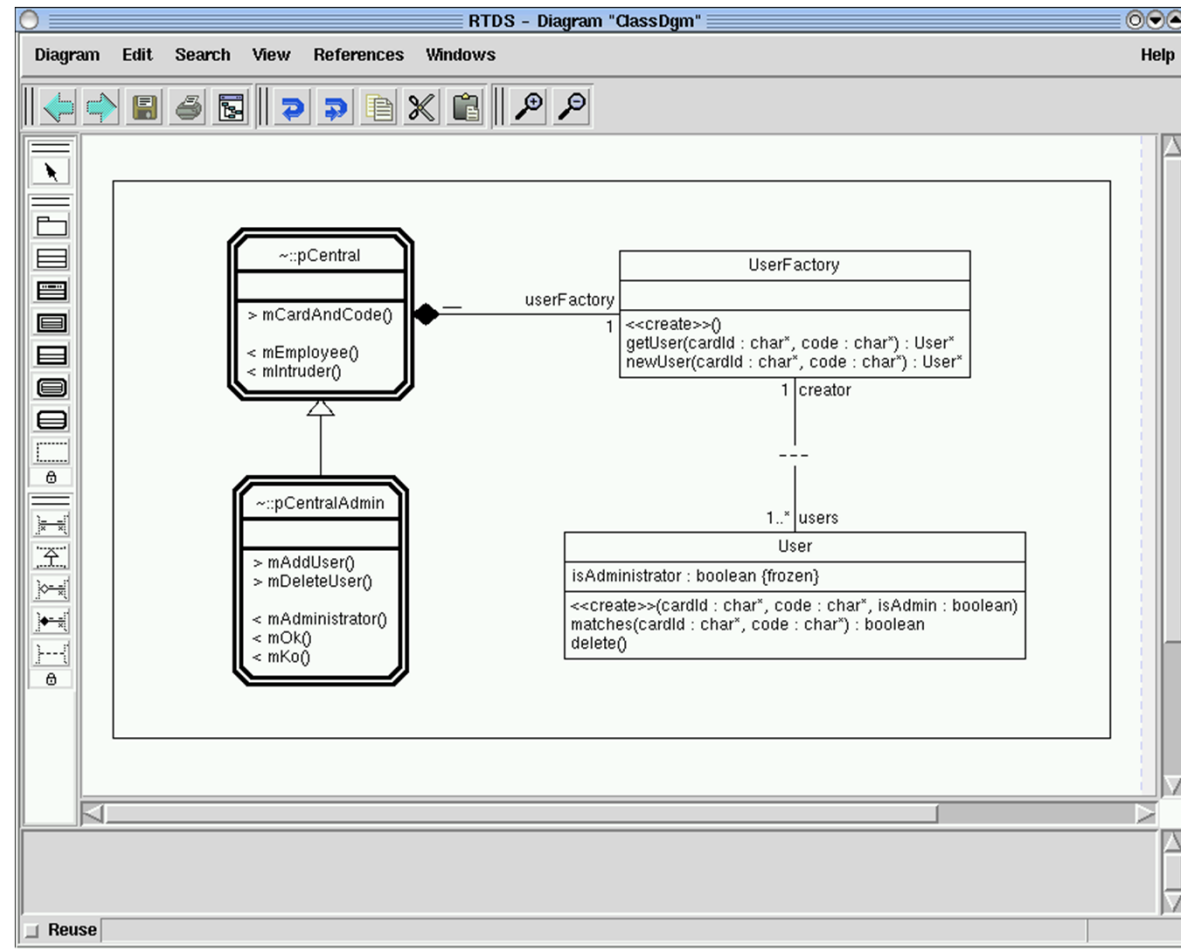


---

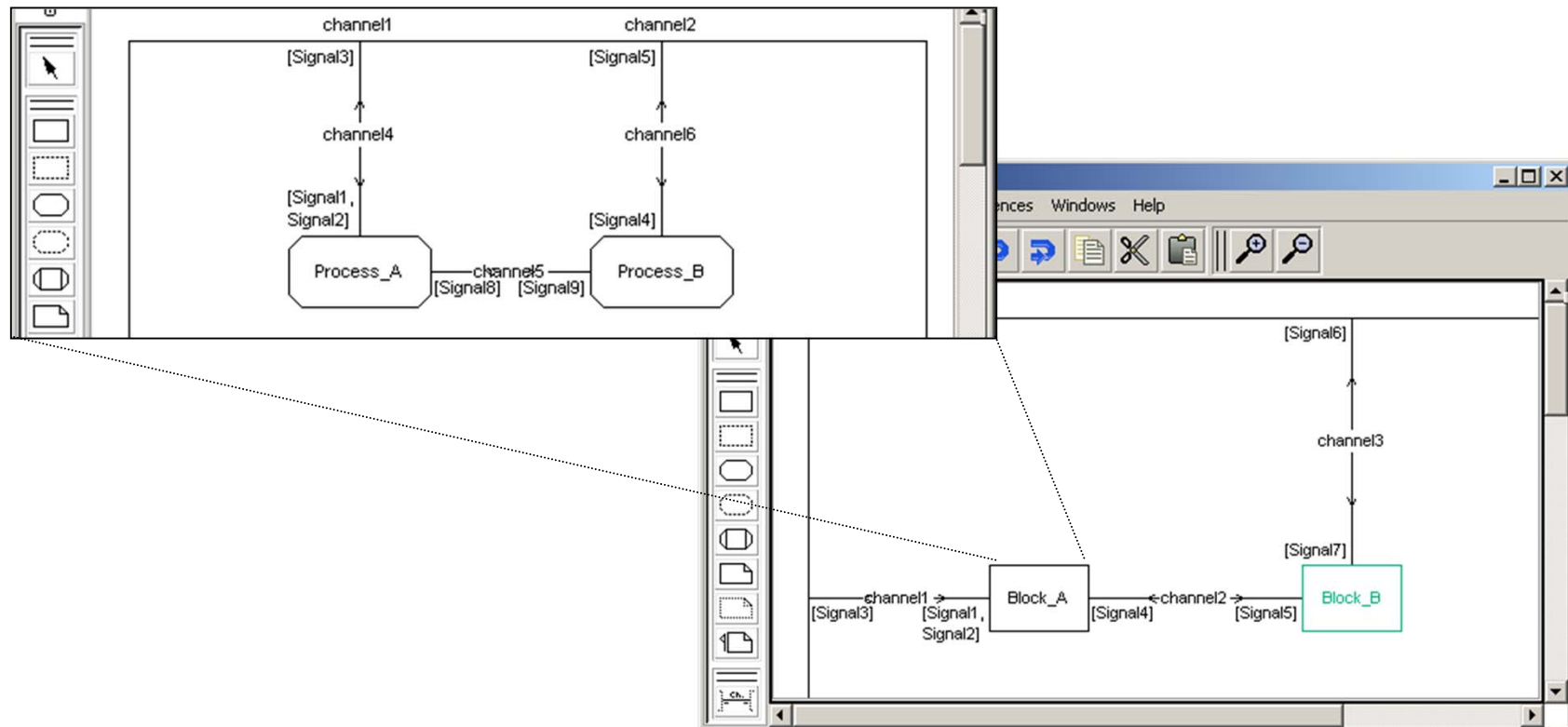
**real time developer studio**

# Views: Class diagram

Relations  
between static  
classes (C++)  
and dynamic  
classes (SDL)



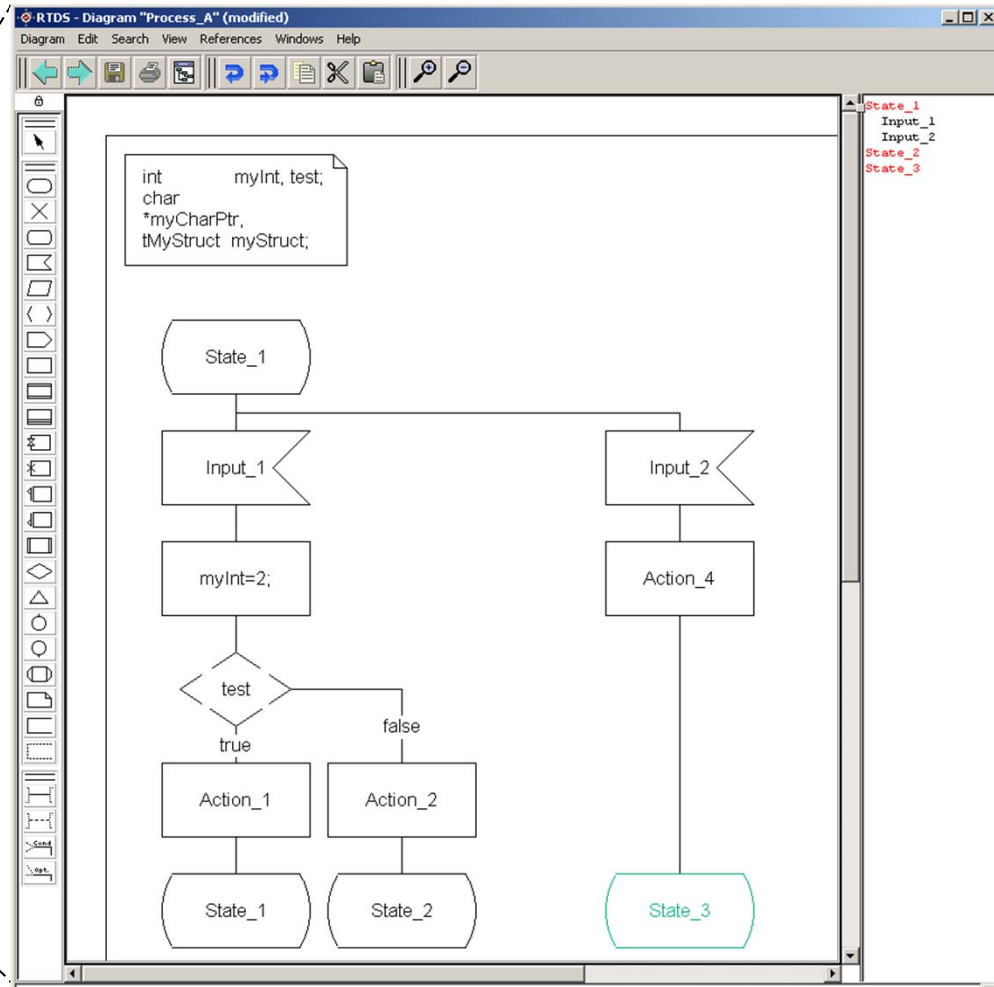
# Views: Architecture and communication



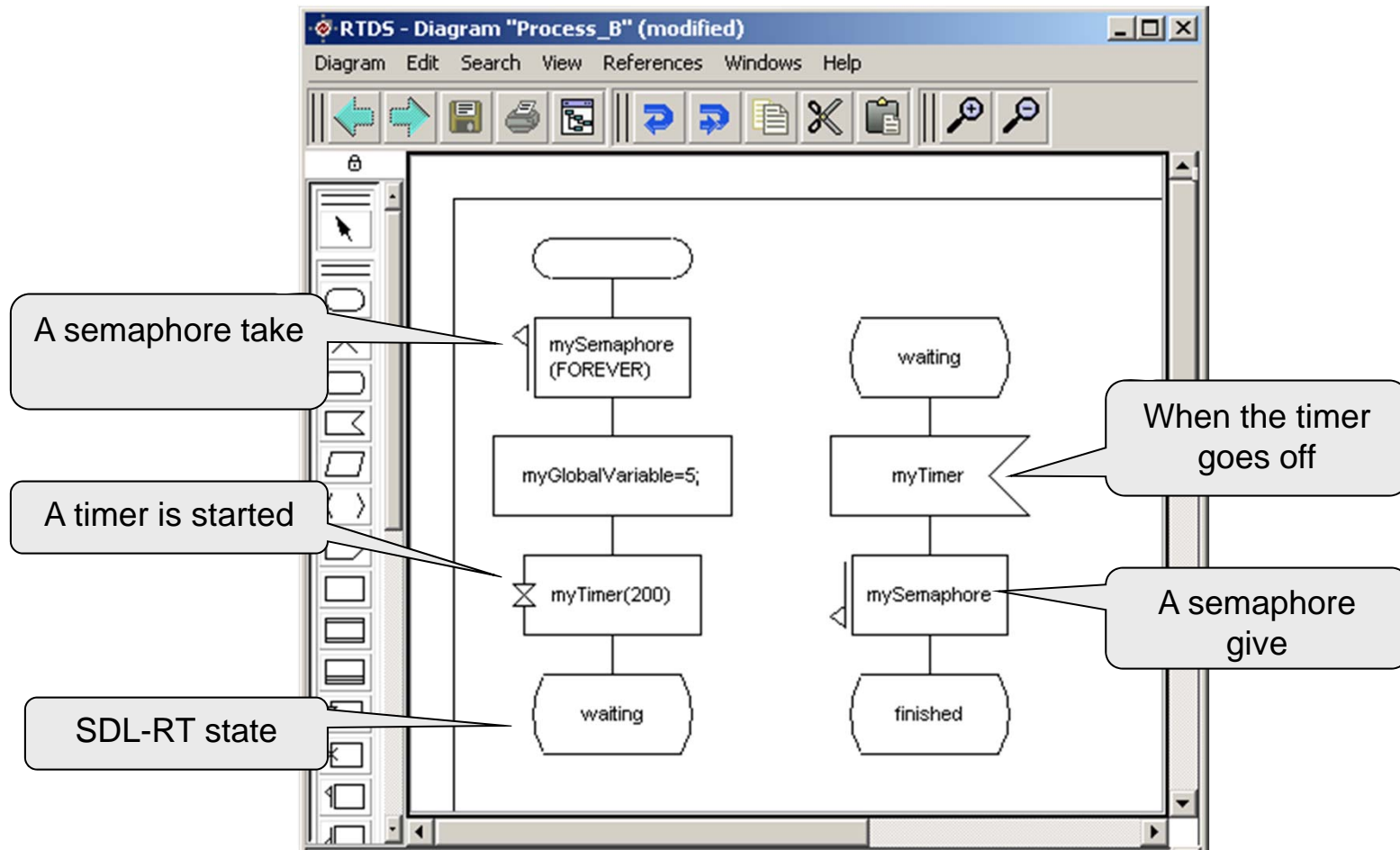
# Views: Behavior and Data

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

Process A

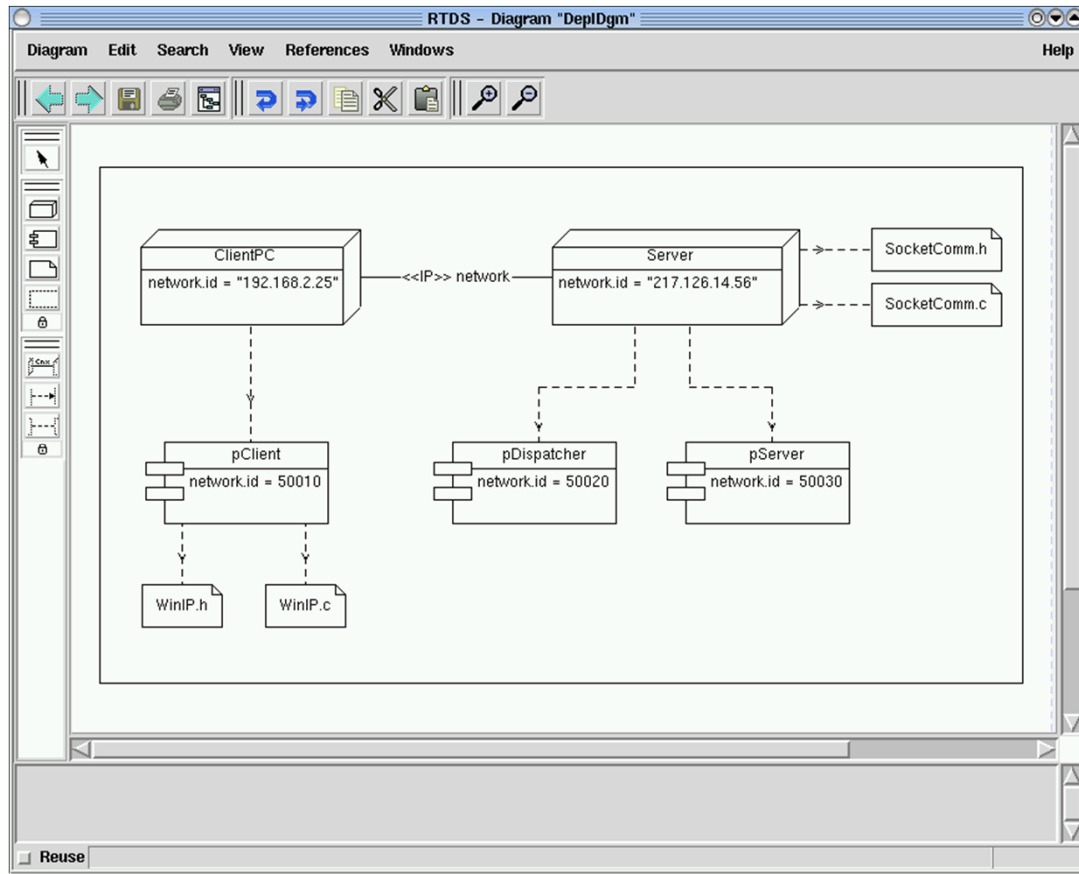


# Views: Operating system services



# Views: Distributed systems

Physical  
deployment



## Model views

- **Library of components**
- **System architecture**
- **Interface definitions**
- **Application deployment**
- **Real time concepts**
- **Key points in the design**

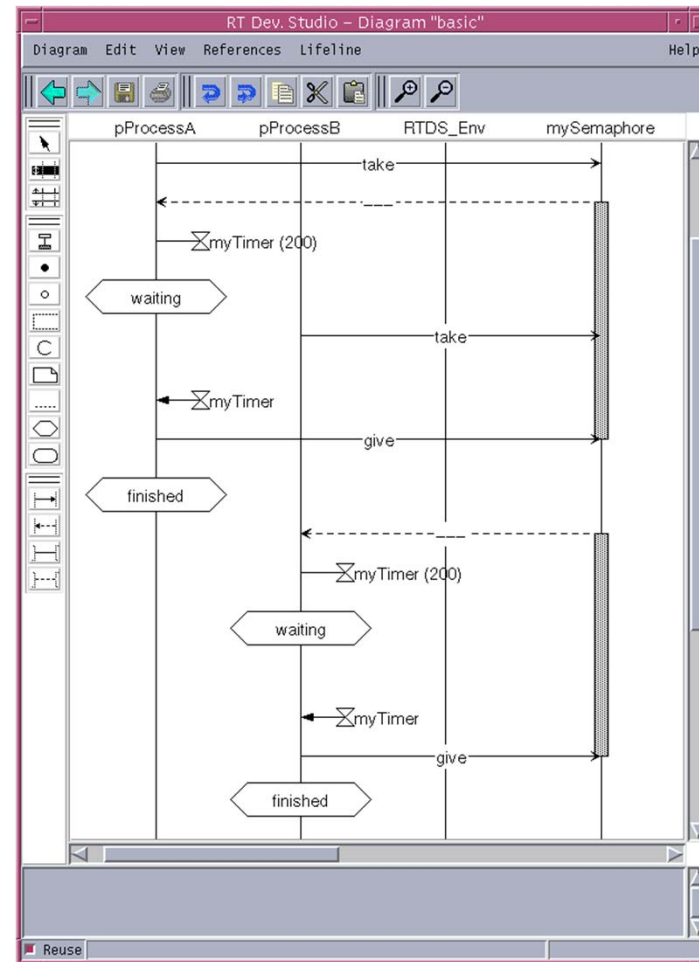
# MSC: dynamic view

## Message Sequence Chart

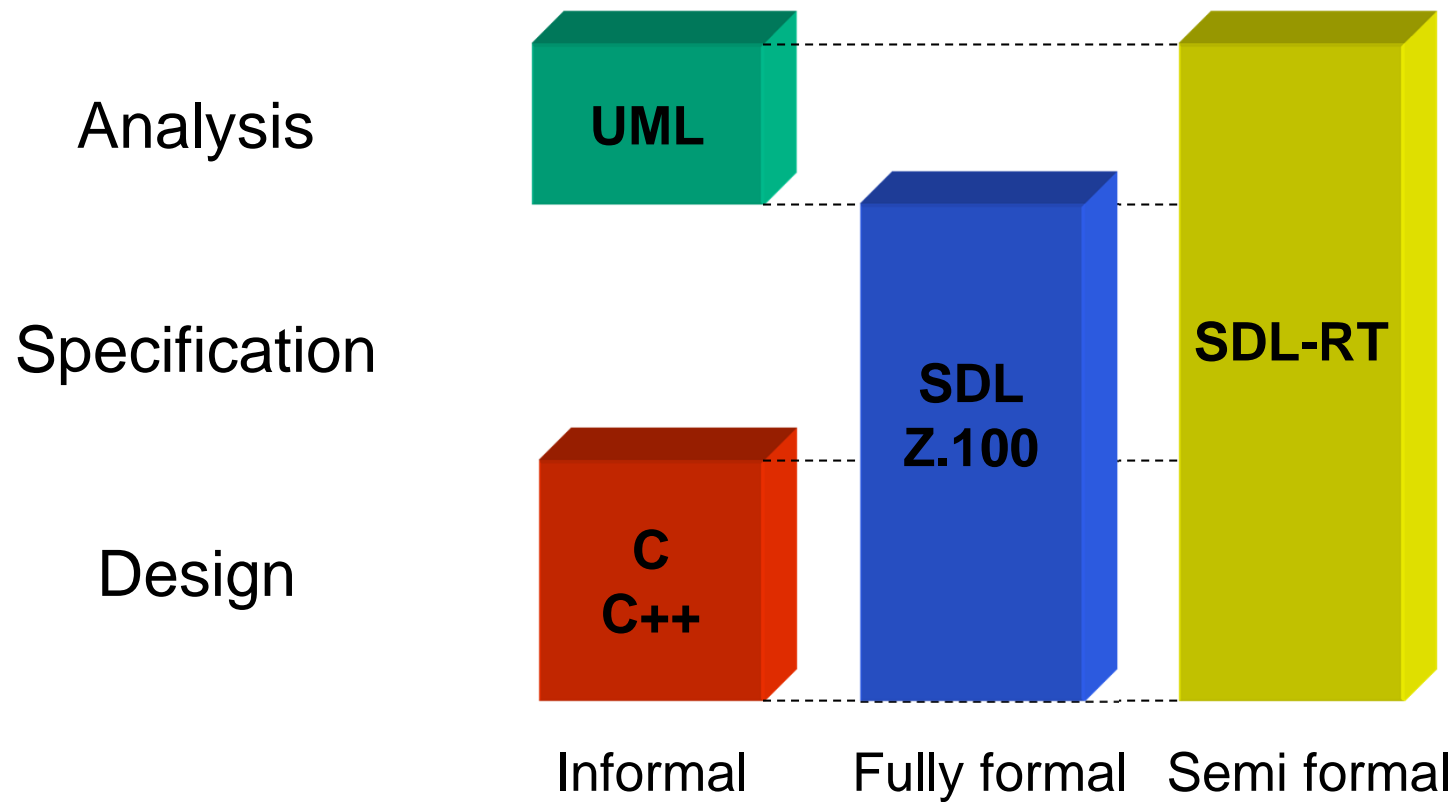
- Vertical lines represent a task, the environment or a semaphore,
- Arrows represent message exchanges, semaphore manipulations or timers.

Can be used:

- As specification
- Execution traces



## RTDS: supported languages



# RTDS: supported languages

## **Informal** modelling for requirements: UML

- Edition
- C++ stubs generation



## **Semi-formal** modelling for design: SDL-RT

- Edition
- Syntactic et semantics checking
- Code generation
- Graphical debugging



## **Fully formal** modelling for specification: SDL Z.100

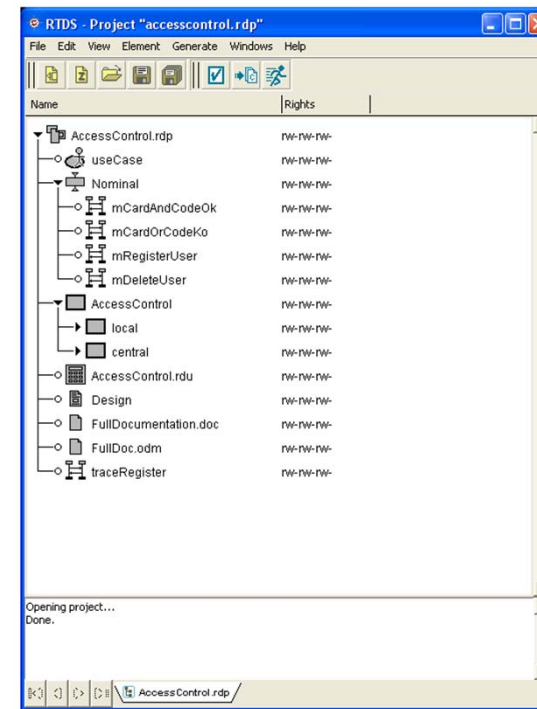
- Edition
- Syntactic et semantics checking
- Simulation
- Verification
- Code generation
- Graphical debugging
- Test



# Tools: the Project manager

It is the « hub » of the toolset:

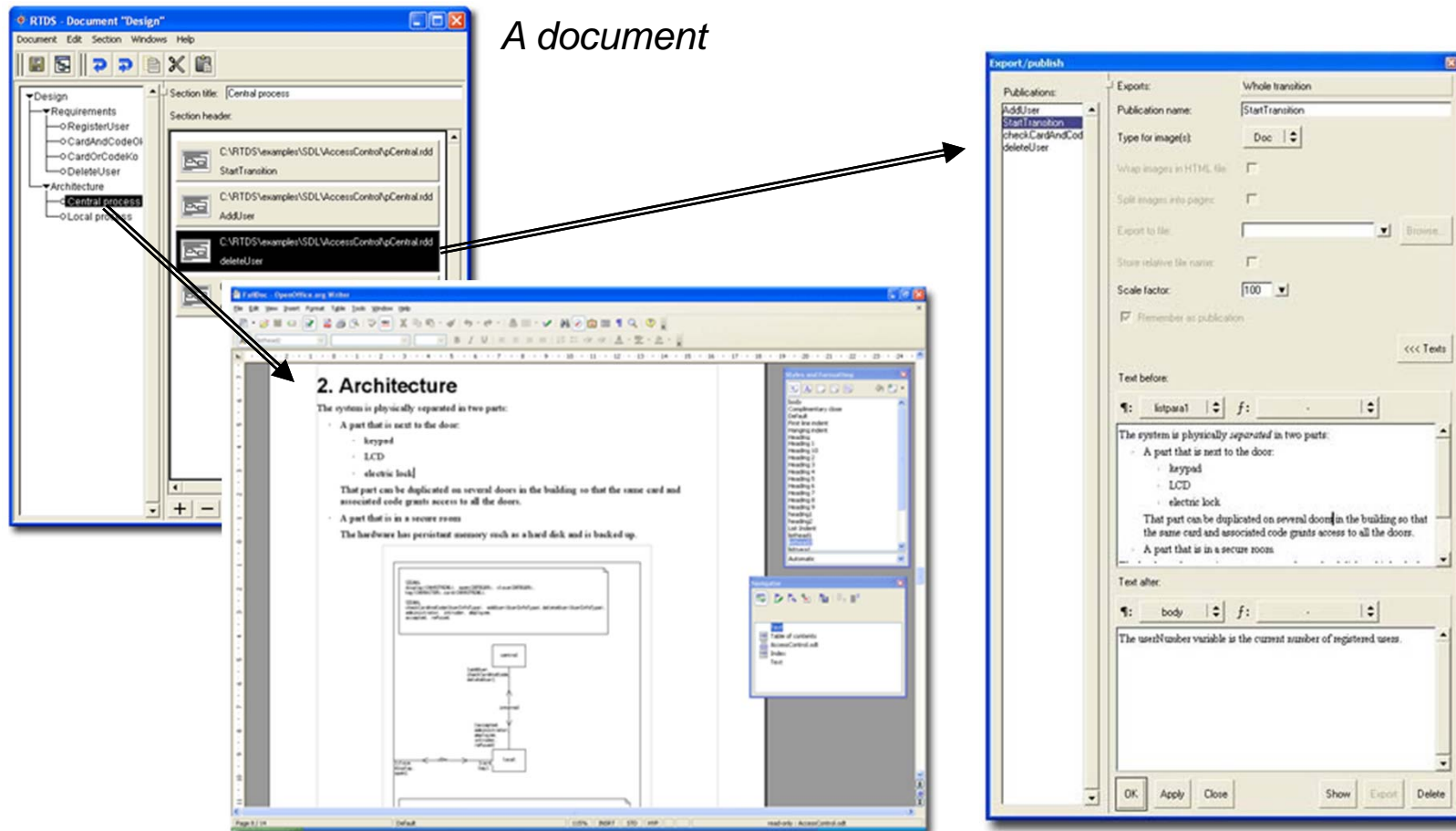
- File organization (UML, SDL, SDL-RT, C, C++, H, traces and others) with packages,
- Calls the editors, the debugger, and the code generator,
- Handles code generation profiles.



# 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

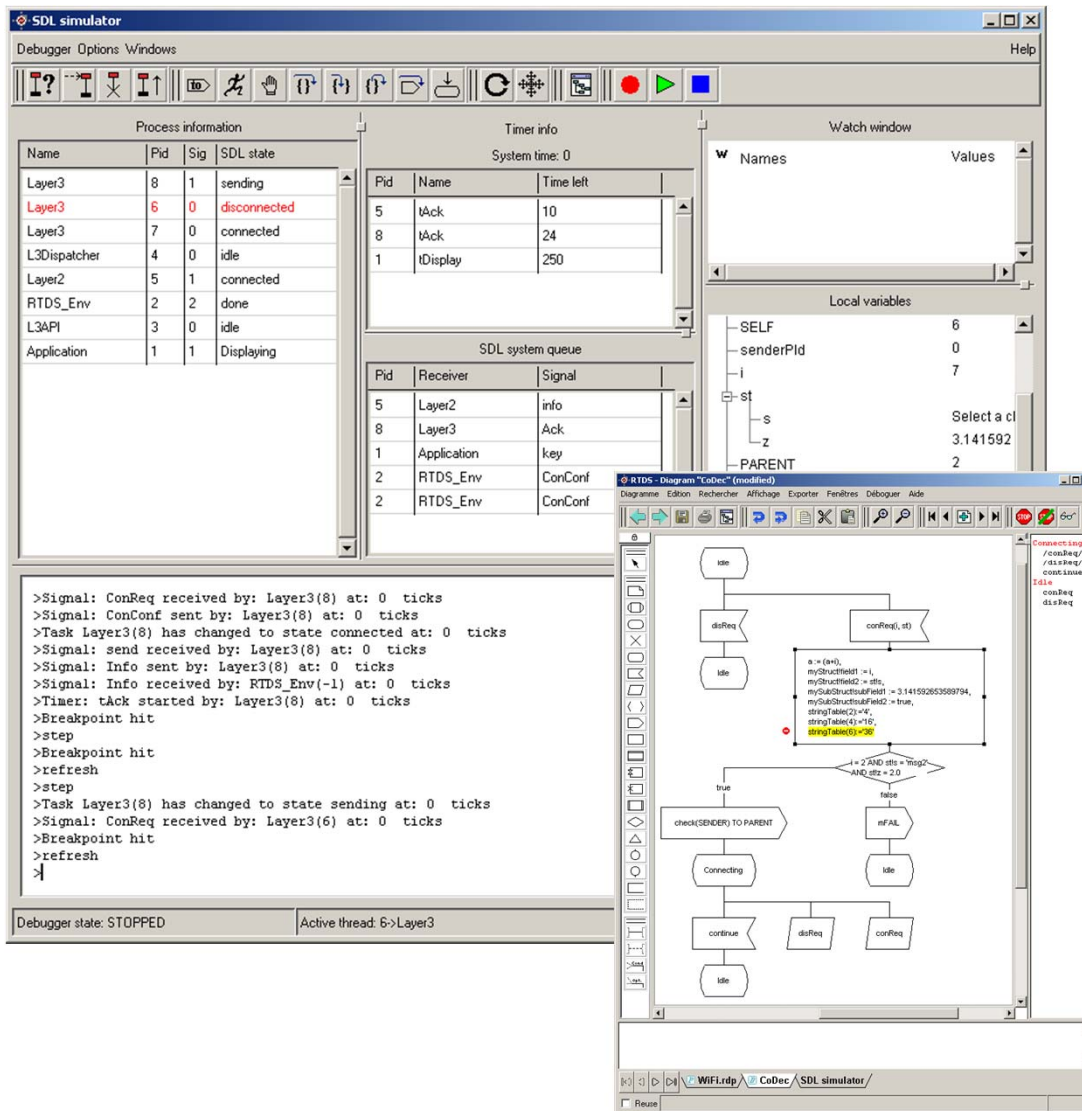
# Documentation generation



*A document*

*The generated documentation*

*A publication*

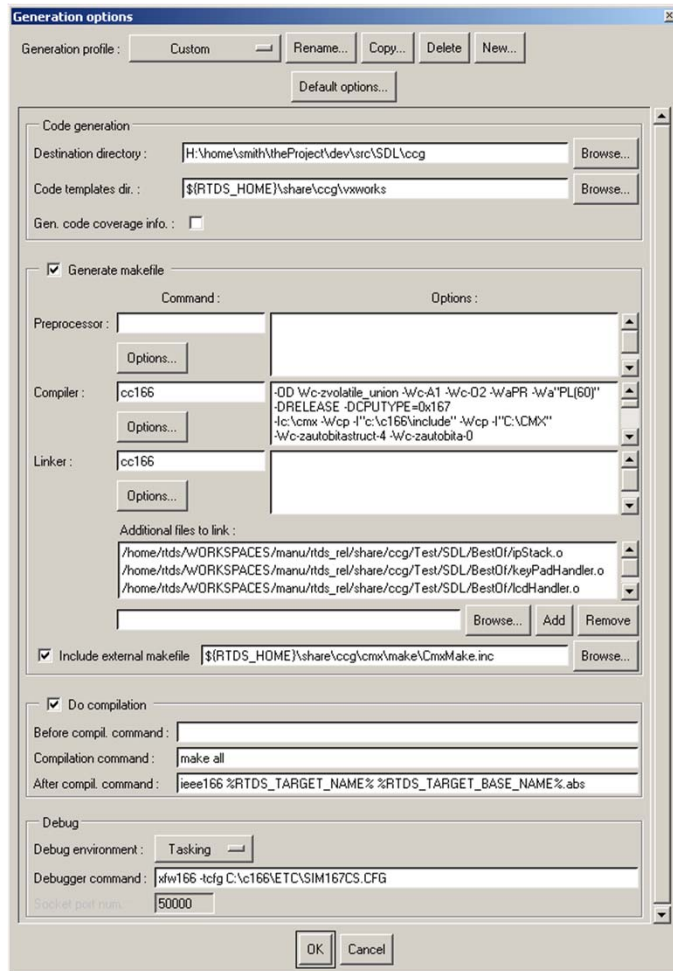


# Model simulator

A graphical *debugger* for fully formal models based on the model semantic

- Set breakpoints and step in the diagrams,
- Externally defined or interactive operator calls,
- Dynamic traces,
- Connecting an external tool is possible through a socket.

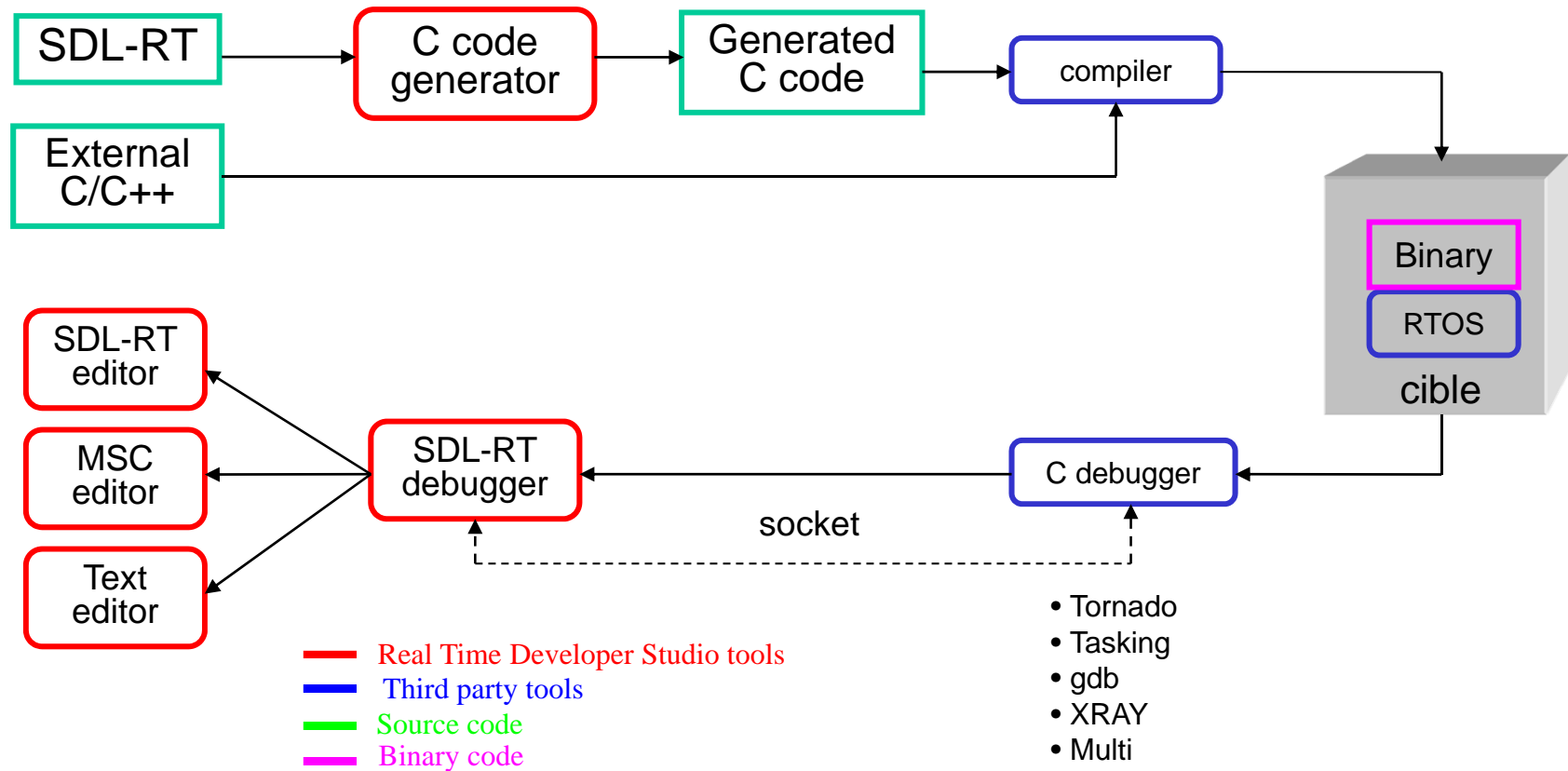
# Code generation



- 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.

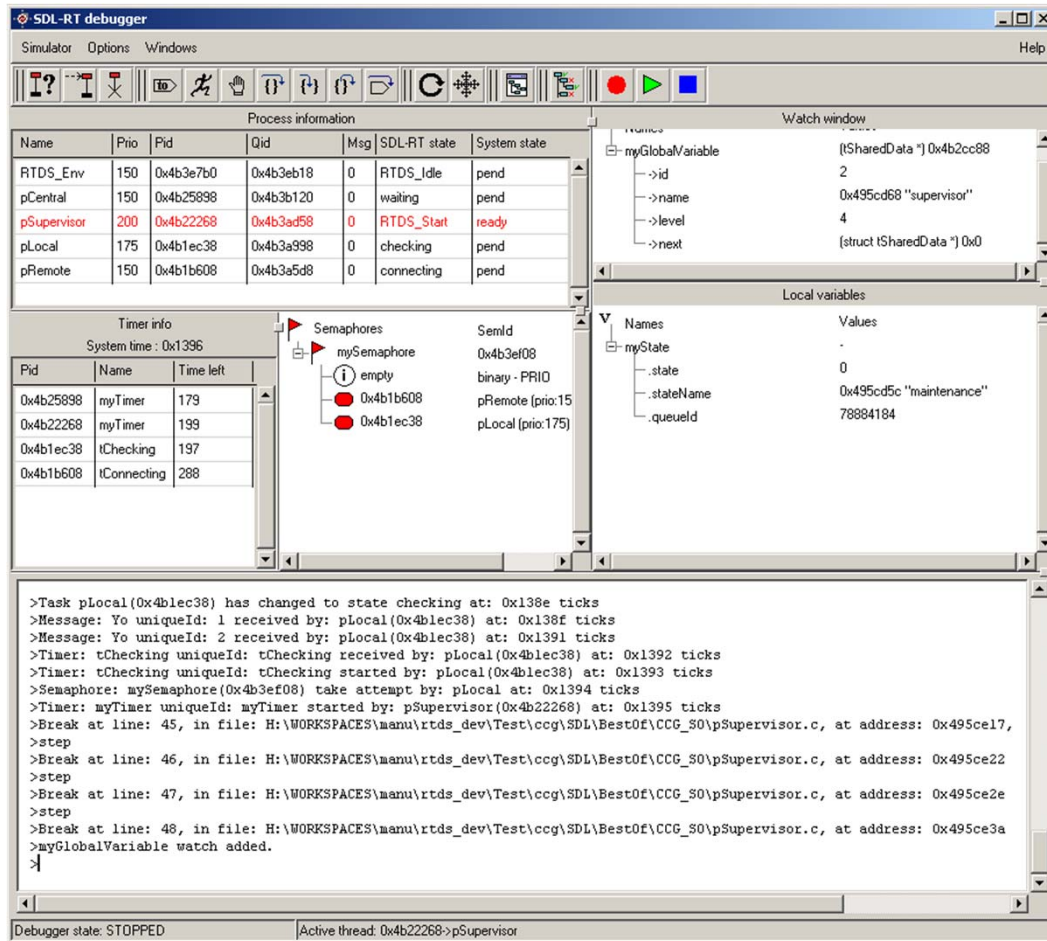


# Model debugger

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.



The screenshot displays the SDL-RT debugger interface with the following components:

- Process information table:**

Name	Prio	Pid	Qid	Msg	SDL-RT state	System state
RTDS_Env	150	0x4b3e7b0	0x4b3eb18	0	RTDS_Idle	pend
pCentral	150	0x4b25898	0x4b3b120	0	waiting	pend
pSupervisor	200	0x4b22268	0x4b3ad58	0	RTDS_Start	ready
pLocal	175	0x4b1ec38	0x4b3a998	0	checking	pend
pRemote	150	0x4b1b608	0x4b3a5d8	0	connecting	pend
- Timer info table:**

Pid	Name	Time left
0x4b25898	myTimer	179
0x4b22268	myTimer	199
0x4b1ec38	tChecking	197
0x4b1b608	tConnecting	288
- Semaphores:** mySemaphore (SemId: 0x4b3ef08, binary - P/ID) with holders for pRemote (prio:15) and pLocal (prio:175).
- Watch window:** myGlobalVariable (0x4b2cc88) with fields: id (2), name (0x495cd68 "supervisor"), level (4), next ((struct tSharedData \*) 0x0).
- Local variables:** myState with fields: state (0), stateName (0x495cd5c "maintenance"), queueId (78884184).
- Log window:**

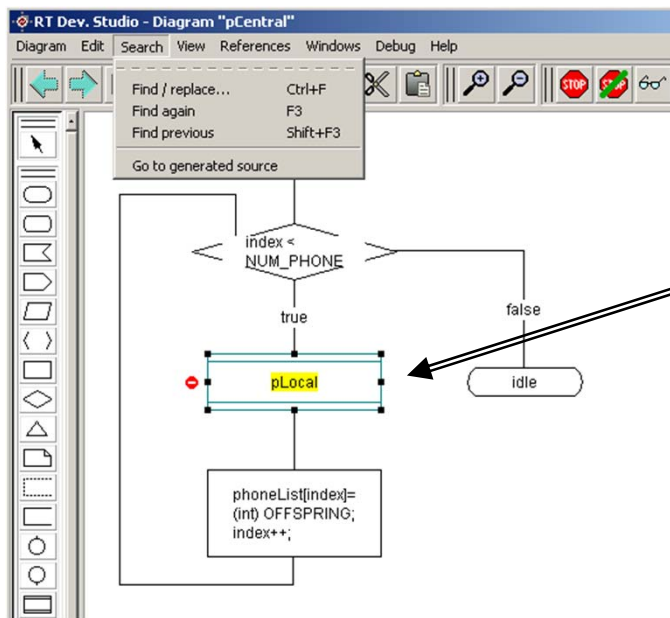
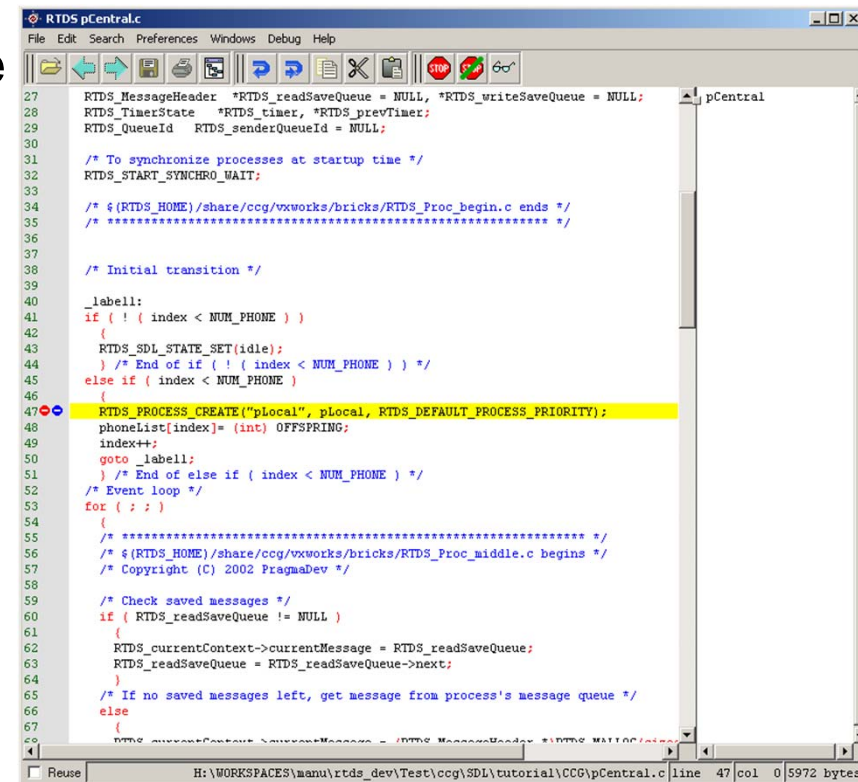
```

>Task pLocal(0x4b1ec38) has changed to state checking at: 0x138e ticks
>Message: Yo uniqueId: 1 received by: pLocal(0x4b1ec38) at: 0x138f ticks
>Message: Yo uniqueId: 2 received by: pLocal(0x4b1ec38) at: 0x1391 ticks
>Timer: tChecking uniqueId: tChecking received by: pLocal(0x4b1ec38) at: 0x1392 ticks
>Timer: tChecking uniqueId: tChecking started by: pLocal(0x4b1ec38) at: 0x1393 ticks
>Semaphore: mySemaphore(0x4b3ef08) take attempt by: pLocal at: 0x1394 ticks
>Timer: myTimer uniqueId: myTimer started by: pSupervisor(0x4b22268) at: 0x1395 ticks
>Break at line: 45, in file: H:\WORKSPACES\manu\rtdev\Test\ccg\SDL\BestOf\CCG_S0\pSupervisor.c, at address: 0x495ce17,
>step
>Break at line: 46, in file: H:\WORKSPACES\manu\rtdev\Test\ccg\SDL\BestOf\CCG_S0\pSupervisor.c, at address: 0x495ce22
>step
>Break at line: 47, in file: H:\WORKSPACES\manu\rtdev\Test\ccg\SDL\BestOf\CCG_S0\pSupervisor.c, at address: 0x495ce2e
>step
>Break at line: 48, in file: H:\WORKSPACES\manu\rtdev\Test\ccg\SDL\BestOf\CCG_S0\pSupervisor.c, at address: 0x495ce3a
>myGlobalVariable watch added.
>

```
- Debugger state:** STOPPED, Active thread: 0x4b22268->pSupervisor

# Debug features

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

```

RTDS pCentral.c
File Edit Search Preferences Windows Debug Help
27 RTDS_MessageHeader *RTDS_readSaveQueue = NULL, *RTDS_writeSaveQueue = NULL;
28 RTDS_TimerState *RTDS_timer, *RTDS_prevTimer;
29 RTDS_QueueId RTDS_senderQueueId = NULL;
30
31 /* To synchronize processes at startup time */
32 RTDS_START_SYNCHRO_WAIT;
33
34 /* $(RTDS_HOME)/share/ccg/vxworks/bricks/RTDS_Proc_begin.c ends */
35 /* ***** */
36
37
38 /* Initial transition */
39
40 _labell:
41 if ( ! ( index < NUM_PHONE ) )
42 {
43   RTDS_SDL_STATE_SET(idle);
44 } /* End of if ( ! ( index < NUM_PHONE ) ) */
45 else if ( index < NUM_PHONE )
46 {
47   RTDS_PROCESS_CREATE("pLocal", pLocal, RTDS_DEFAULT_PROCESS_PRIORITY);
48   phoneList[index]= (int) OFFSPRING;
49   index++;
50   goto _labell;
51 } /* End of else if ( index < NUM_PHONE ) */
52 /* Event loop */
53 for ( ; ; )
54 {
55   /* ***** */
56   /* $(RTDS_HOME)/share/ccg/vxworks/bricks/RTDS_Proc_middle.c begins */
57   /* Copyright (C) 2002 PragmaDev */
58
59   /* Check saved messages */
60   if ( RTDS_readSaveQueue != NULL )
61   {
62     RTDS_currentContext->currentMessage = RTDS_readSaveQueue;
63     RTDS_readSaveQueue = RTDS_readSaveQueue->next;
64   }
65   /* If no saved messages left, get message from process's message queue */
66   else
67   {
68     RTDS_currentContext->currentMessage = RTDS_MessageHeader *RTDS_MessageHeader;
69   }
70 }

```

# Graphical traces

Execution traces:

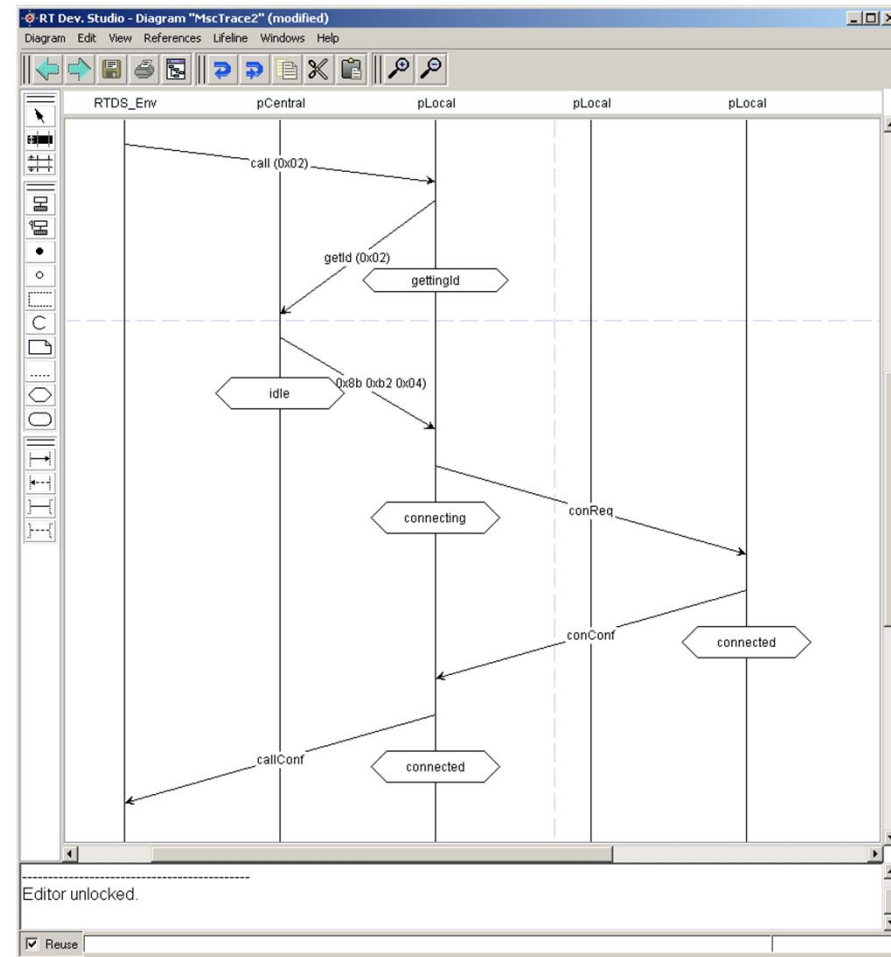
- States,
- Events,
- Semaphores,
- Timers.

Trace level configuration

Display of system time

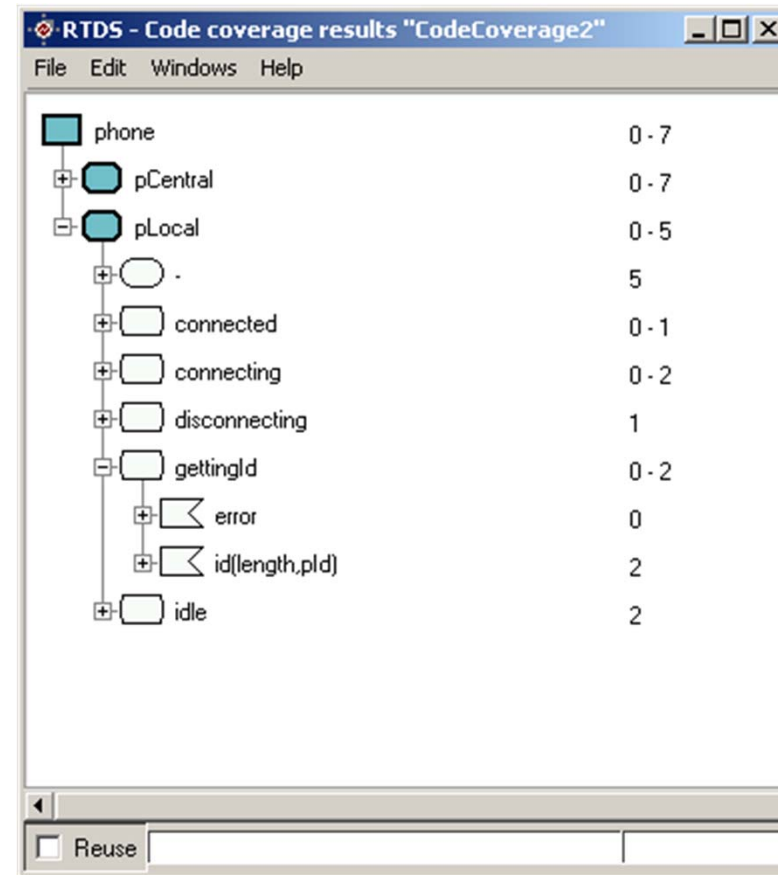
MSC Diff allows to check:

- Conformity,
- Non-regression.

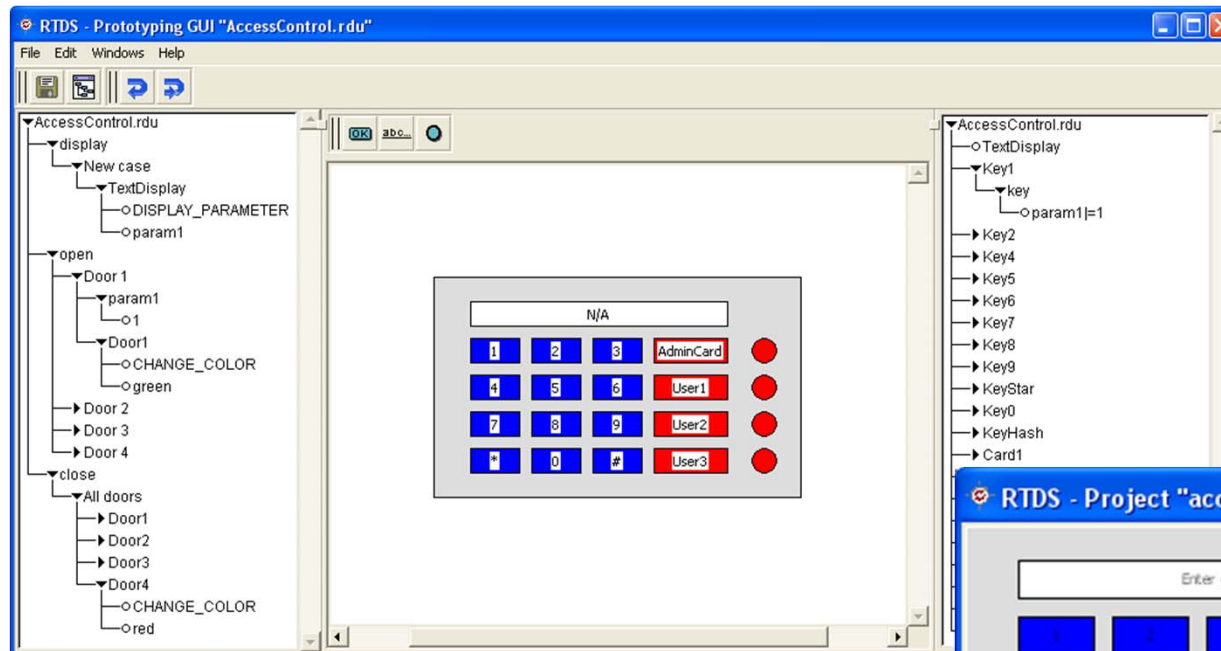


# Model coverage

- Graphical model coverage analysis
- Merge feature

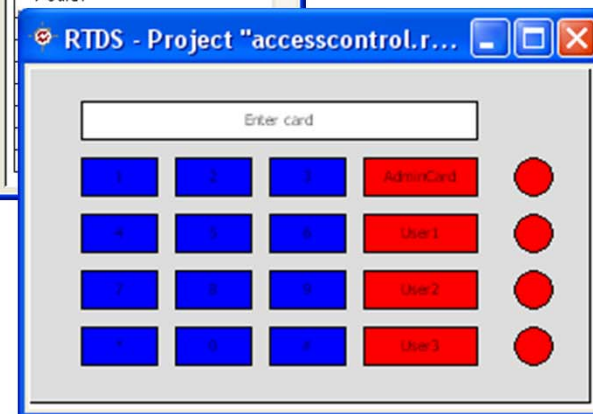


# Prototyping interface

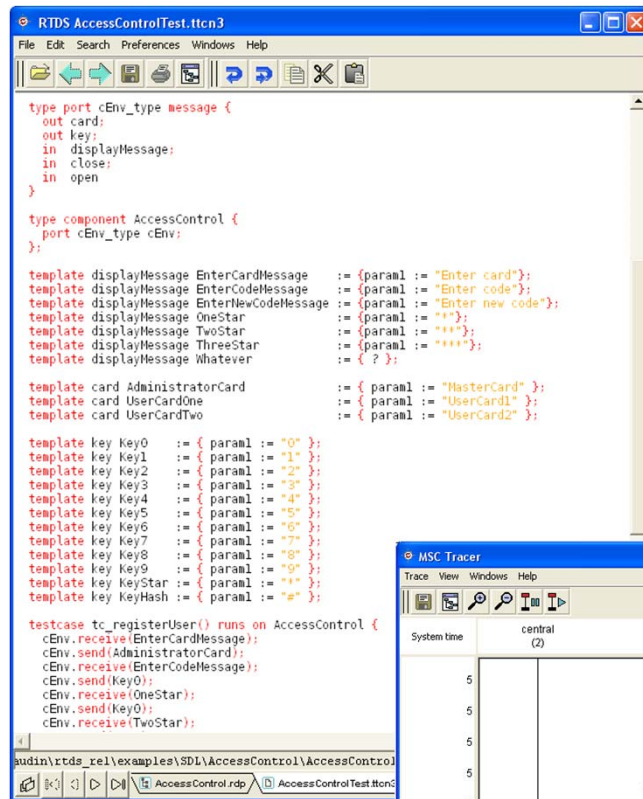


➤ Knows about the model inputs and outputs.

➤ Connects automatically to the simulator or the debugger.



# Model based testing



```

type port cEnv_type message {
  out card;
  out key;
  in displayMessage;
  in close;
  in open
}

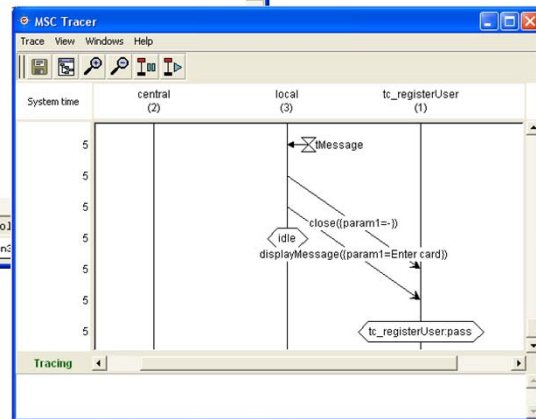
type component AccessControl {
  port cEnv_type cEnv;
};

template displayMessage EnterCardMessage := { param1 := "Enter card";
template displayMessage EnterCodeMessage := { param1 := "Enter code";
template displayMessage EnterNewCodeMessage := { param1 := "Enter new code";
template displayMessage OneStar := { param1 := "+";
template displayMessage TwoStar := { param1 := "+";
template displayMessage ThreeStar := { param1 := "+";
template displayMessage Whatever := { param1 := "?";

template card AdministratorCard := { param1 := "MasterCard";
template card UserCardOne := { param1 := "UserCard1";
template card UserCardTwo := { param1 := "UserCard2";

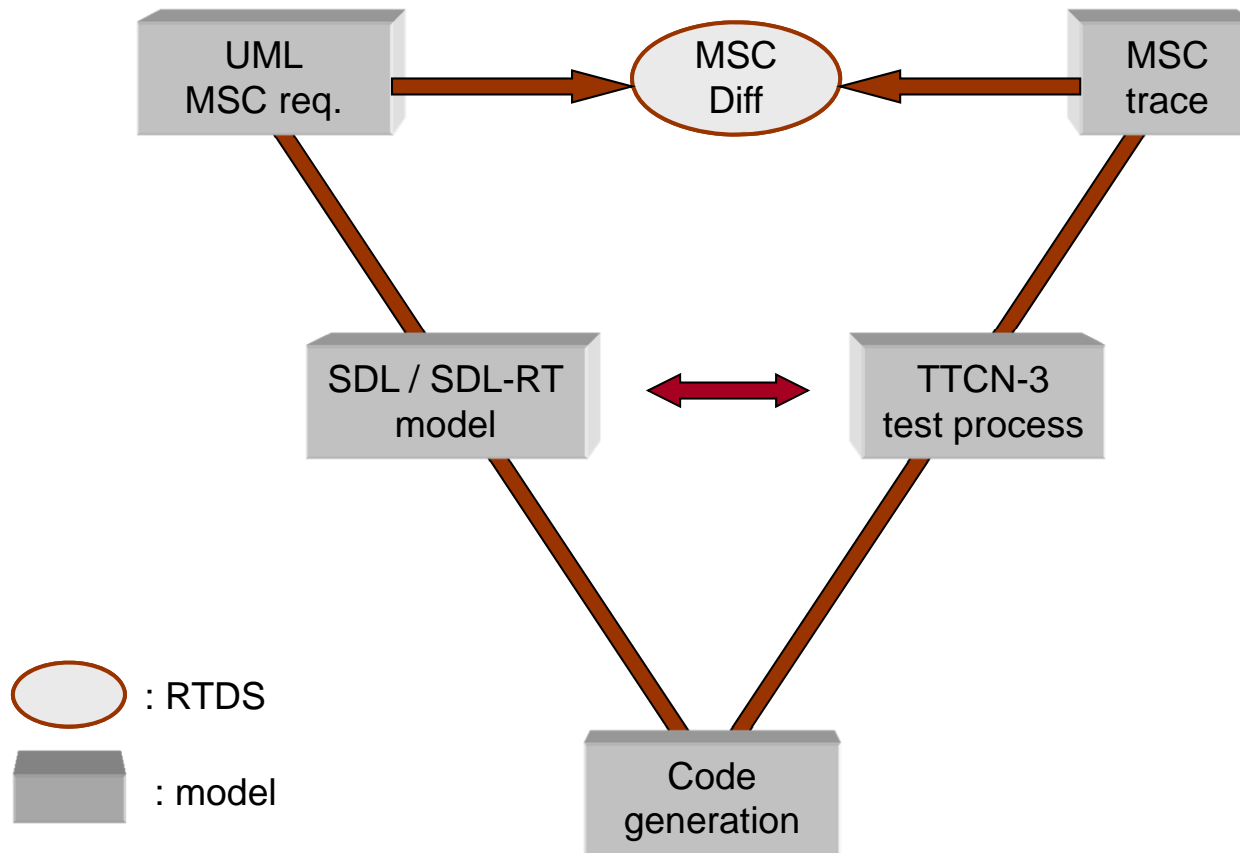
template key Key0 := { param1 := "0";
template key Key1 := { param1 := "1";
template key Key2 := { param1 := "2";
template key Key3 := { param1 := "3";
template key Key4 := { param1 := "4";
template key Key5 := { param1 := "5";
template key Key6 := { param1 := "6";
template key Key7 := { param1 := "7";
template key Key8 := { param1 := "8";
template key Key9 := { param1 := "9";
template key KeyStar := { param1 := "+";
template key KeyHash := { param1 := "#";

testcase tc_registerUser() runs on AccessControl {
  cEnv.receive(EnterCardMessage);
  cEnv.send(AdministratorCard);
  cEnv.receive(EnterCodeMessage);
  cEnv.send(Key0);
  cEnv.receive(OneStar);
  cEnv.send(Key0);
  cEnv.receive(TwoStar);
  
```



- Based on TTCN-3 international standard:
  - Data types definitions,
  - Templates definitions,
  - Test cases,
  - Execution control.
- Connects automatically to the Simulator:
  - Breakpoints in the model or in the test suite,
  - Verdict displayed in the trace.

# Methodology



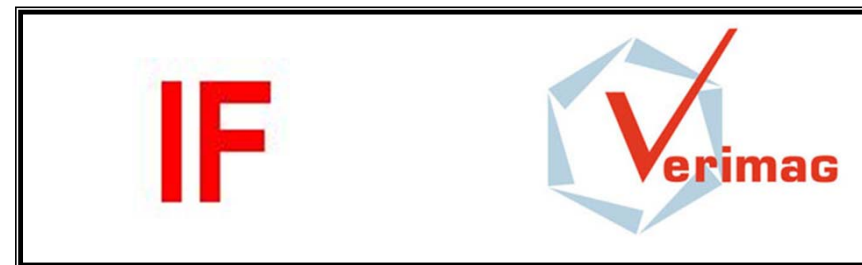
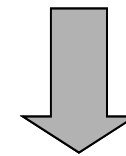
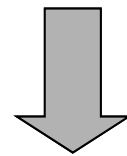
---

**real time developer studio**

# Model checking

Exoticus

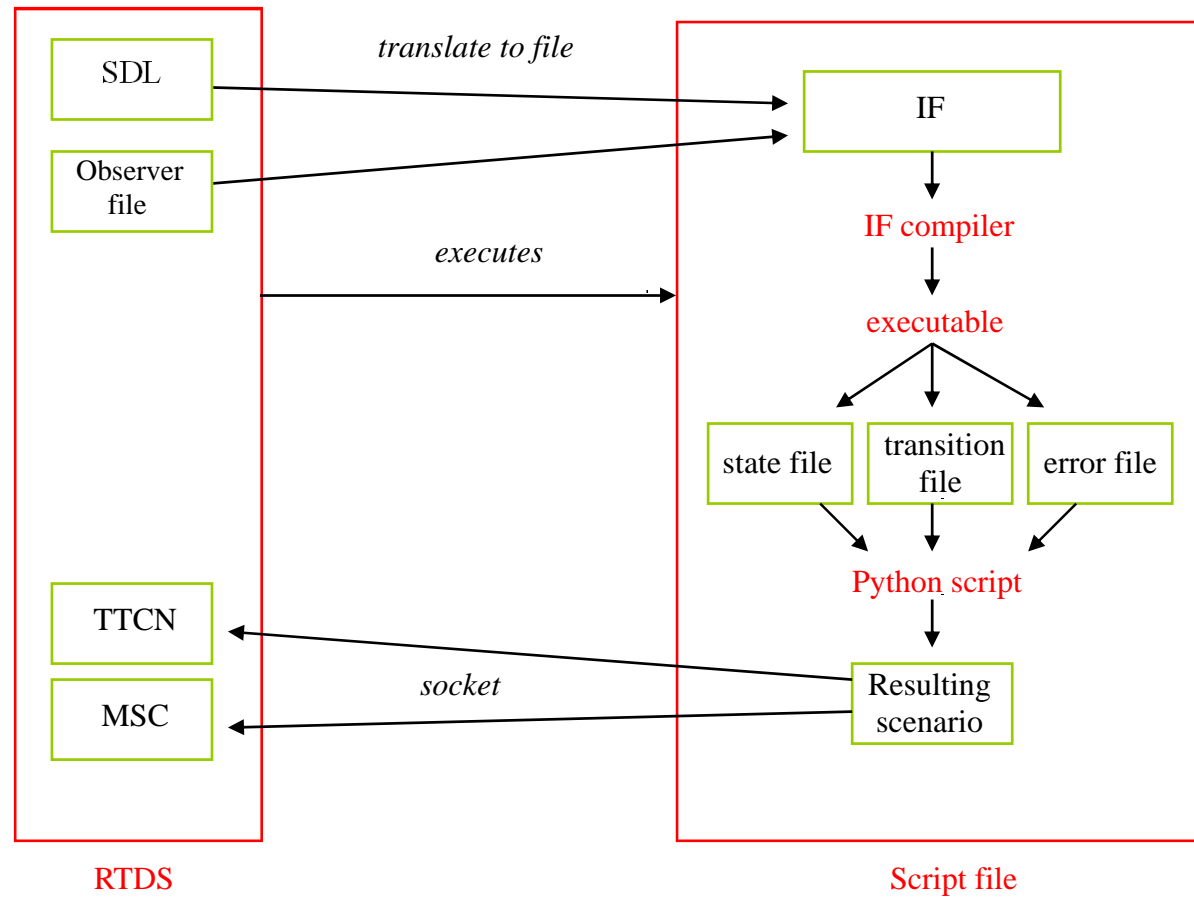
- Partnership with Verimag on IF technology.
  - Exhaustive simulation,
  - Observers,
  - Test generation.
- RTDS feature
  - Export to IF,
  - Execute a script
  - Generate an MSC feedback.



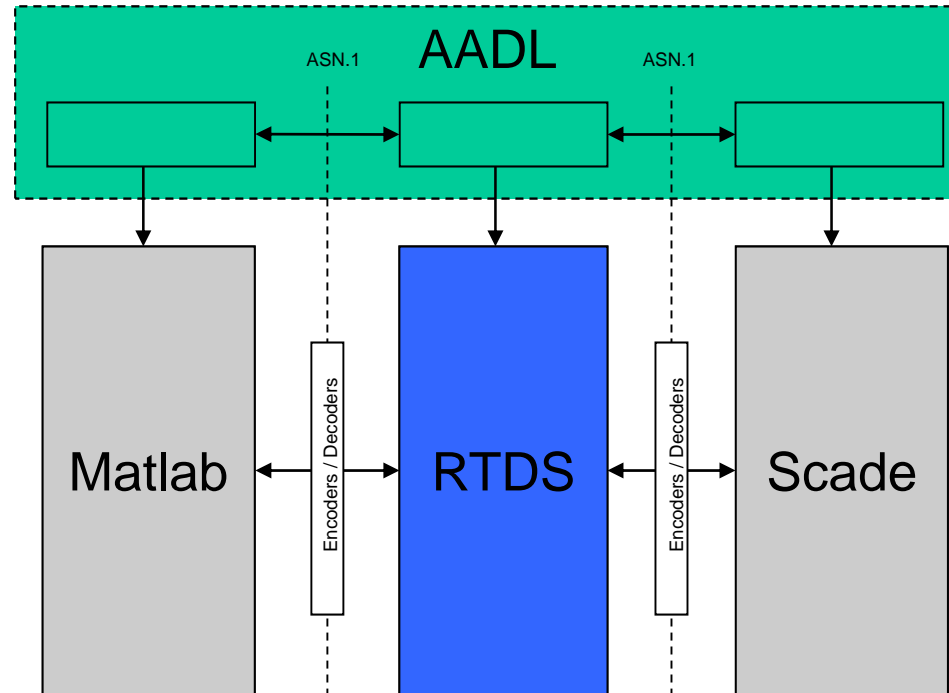
---

**real time developer studio**

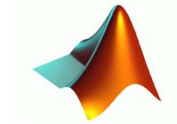
# Implementation



# ESA Taste framework integration



Continuous



Event driven



Synchronous



**real time developer studio**

# 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 Reqify 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.

