Modeling and testing
SDL, SDL-RT & TTCN overview

Emmanuel Gaudin

Emmanuel.gaudin@pragmadev.com

PragmaDev

- French software editor based in Paris
- Founded in 2001 by engineers having a strong background on modeling languages
- Dedicated to the development of modeling and testing tools for event driven systems
- Company is participating to:
    - European projects such as PRESTO on Modeling and Test subjects with Thales and INRIA
    - French "Competiveness Cluster" Systematic
    - Common lab with National Nuclear Research Center CEA
    - Dual Innovative Project with French Army

References

Aero/ Defense

Automotive

BlueSolutions by Bolloré

Telecoms

Semi-conductor

- Free university program : more than 800 licenses used over the world
- Distribution Network covering North America, Asia and Europe…

Trends in software developments

Complexity

“ 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.”

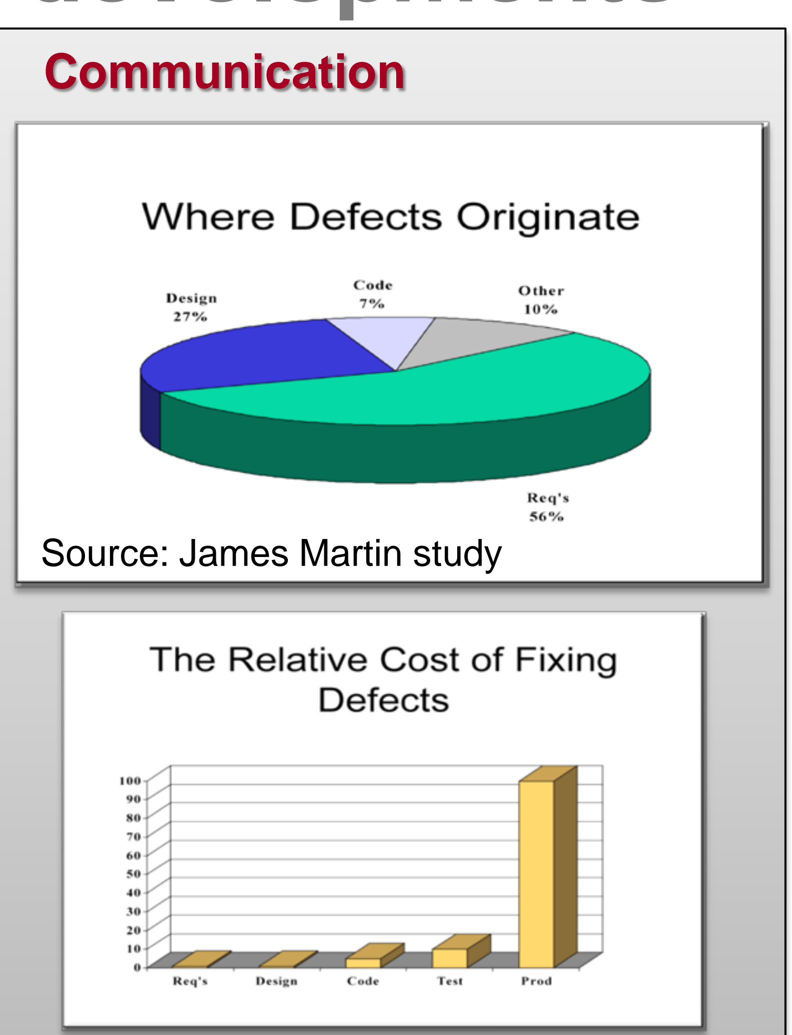
Wind River *Market Survey of Device Software Testing Trends and Quality Concerns in the Embedded Industry* – June 2010.

Maintainability

“Software development costs only comprise a portion of the total cost of software ownership. However, the development **process** itself has a **significant impact** on total cost of ownership.”

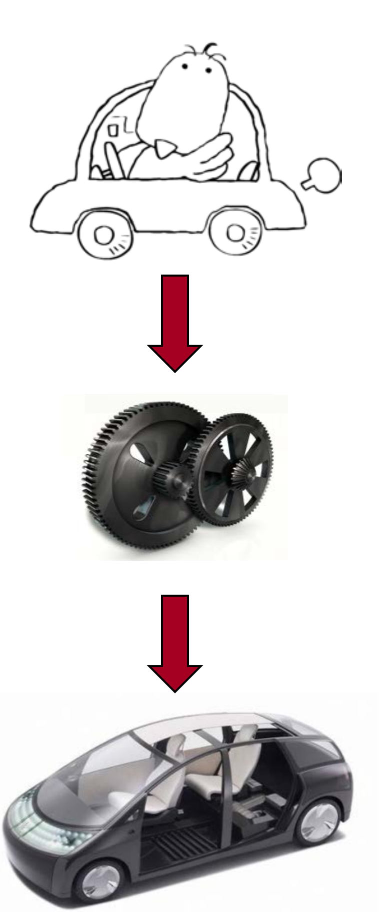
Total Cost of Ownership: Development is (only) Job One by Daniel D. Galorath - June 2008.

Communication



Modeling: an appropriate solution

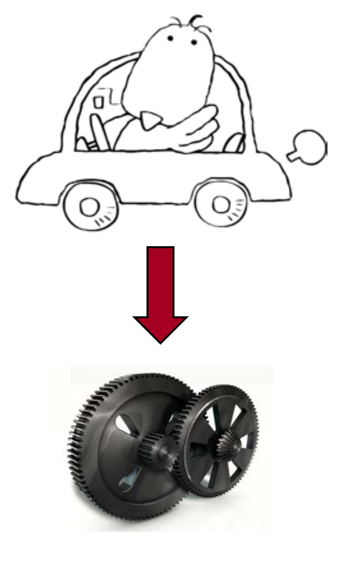
- To concentrate on What rather than How,
- Modeling is about Communication and Documentation,
- Legibility of large systems gets critical. Would you build a house without drawing detailed plans ?
  - Get control over productivity,
  - Improve quality,
  - Reduce development time.

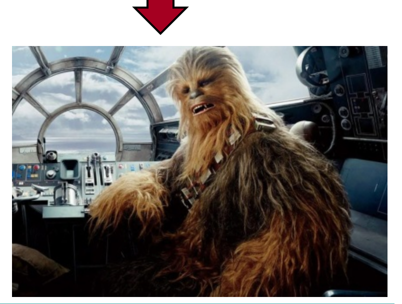


Modeling: an appropriate solution

- To concentrate on What rather than How,
- Modeling 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,
  - Reduce development time.





Existing modeling languages

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

Modeling languages positioning

	SysML	UML	MARTE	AADL	SDL	LUSTRE
Requirements						
Analysis						
Architecture						
Specification						
Design						
	Generic	Generic	HW/SW	HW/SW	Async	Sync

Static verification

Dynamic verification

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

Languages positioning

Architecture	SART / AADL / UML / SysML				
Specification		LUSTRE	SDL	Matlab	
Design	ADA				C
	Generic	Sync	Async	Sync	Generic

Modeling semantic

A modeling language without semantic is useless

- Can not be verified
- Does not help communication

Modeling concepts

- Synchronous
- Asynchronous

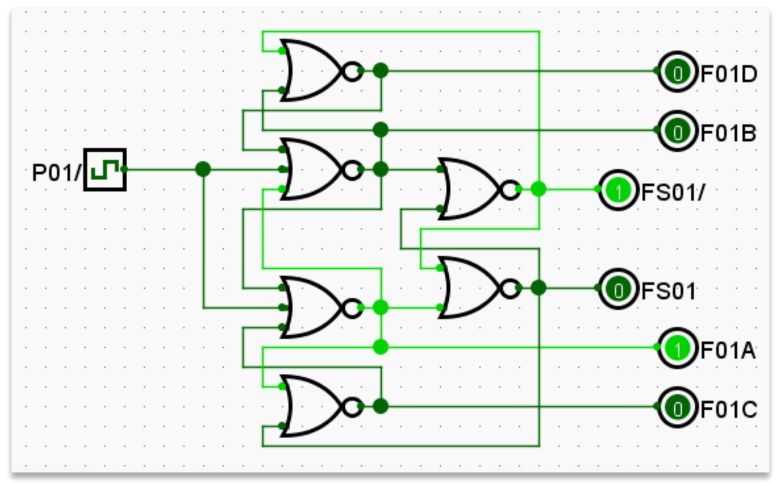
To describe any systems

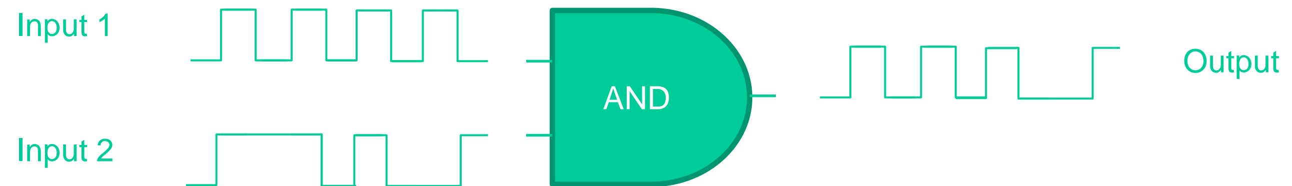
Synchronous models principle

- An input is the value read on a sensor at every tick of a clock. The period can vary from a few seconds to a few mS.
- The information is computed.
- An output is generated.
- Low level, close to the implementation.

A synchronous model

- Logical system
  - Binary sensor (on/off)
  - Logical operators





Synchronous model

- Continuous system
  - Complex sensor (temperature, gyroscope…)
  - Control law with differential equations

Testing of synchronous models

- Generate a value for all the inputs at each tick
  - A lot of information
  - Not legible
- Example of a train signaling system
  - The door closes after 10 seconds
  - Sample 10 times per second and we get 100 samples of the same information
  - Same redundancy for the output

Tick	Input 1	Input 2	Output
284	1	0	0
285	1	0	0
286	1	0	0
287	1	0	0
288	1	1	1
289	1	1	1
290	1	1	1
291	1	1	1
292	1	1	1
…			

Event driven models

- Based on messages
  - Telecommunications
  - High level of abstraction, GALS theory: Systems are Globally Asynchronous and Locally Synchronous
  - Closer to the requirements
- Parallel execution
  - Sequence of events
  - Complex parameters
  - Generates a lot of complexity

Architecture

Processes

Behavior

Testing event based models

- Also based on messages
- Legible
- The sequence of events is more important than the absolute time
- Message racing and the possible parameter values create a very important number of scenarios
- It is usually impossible to test all the possible cases

```
testcase simple()
  {
  port.send(m1)
  alt
    {
    []port.receive(m2)
      {
      setverdict(pass);
      }
    []port.receive(m4)
      {
      setverdict(fail);
      }
    }
  }
```

Event based testing applied to synchronous systems

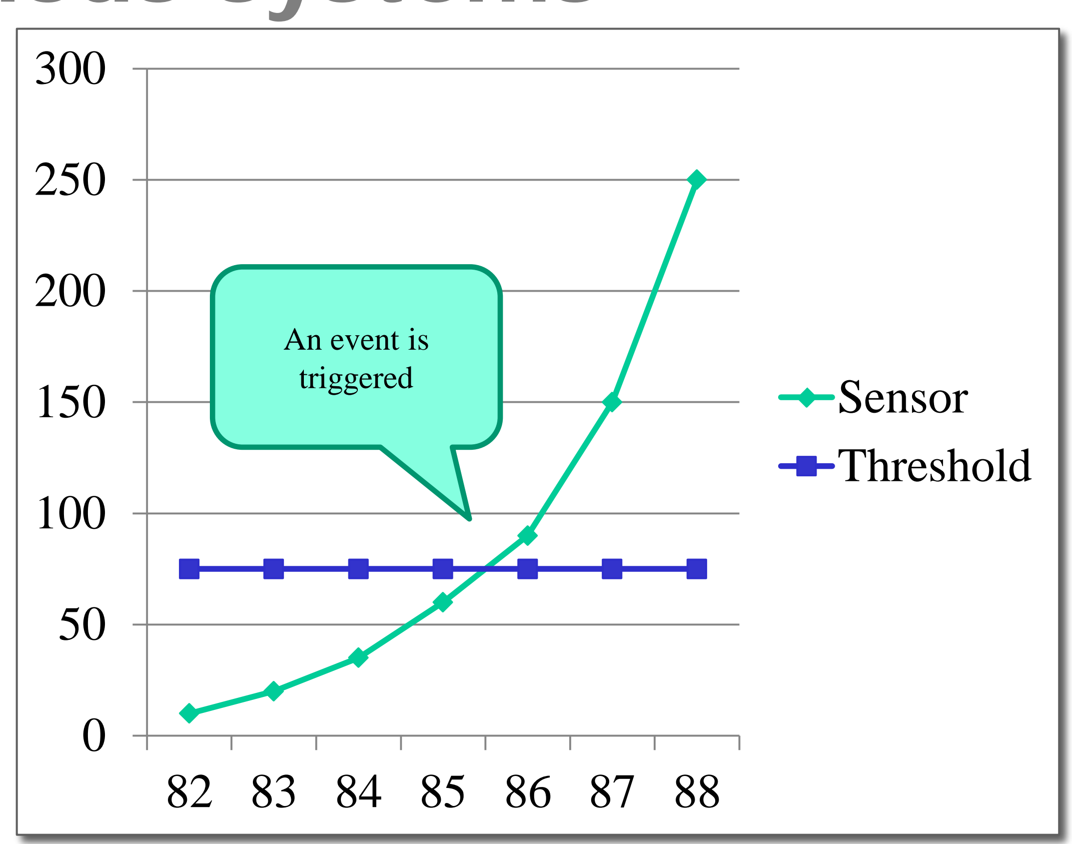
- A logical value triggers an event:
  - Example :

```
Sync   0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
Async  - - - - - - - - - - - - On- - - - - - - - - - - -
```

  - Equivalent to specify when the event arrives.
  - With an event driven approach only the sequence of events matter.

Event based testing applied to synchronous systems

- A value triggers an event:
  - Threshold.
  - Margin.

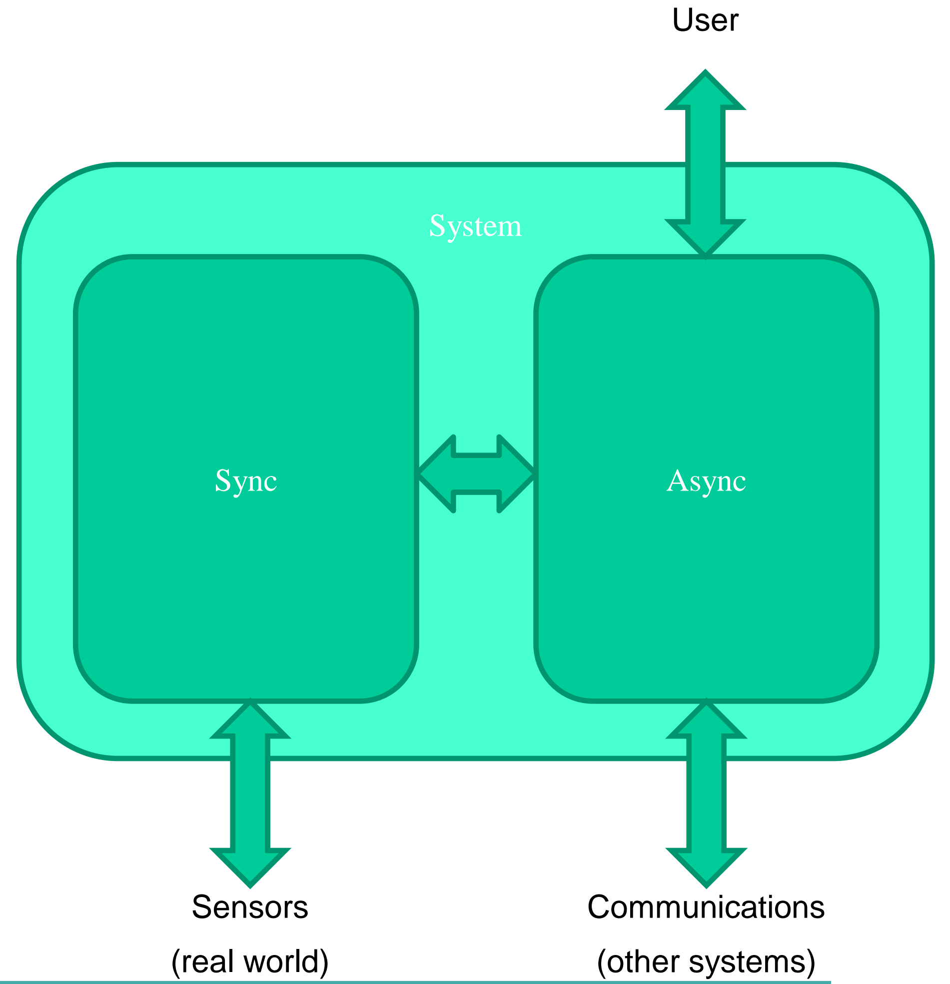


High Level Requirements

- Often based on event
- Usually textual but more and more models
- Conformance testing is close to the requirements
- Conformance testing is best described with an event based notation
- It is possible to link to synchronous implementations

Cyber Physical Systems (CPS)

- Combined approach
- Requirements are event base
- Event based testing will work for both sides
- Event based notation is the best suited



FMI

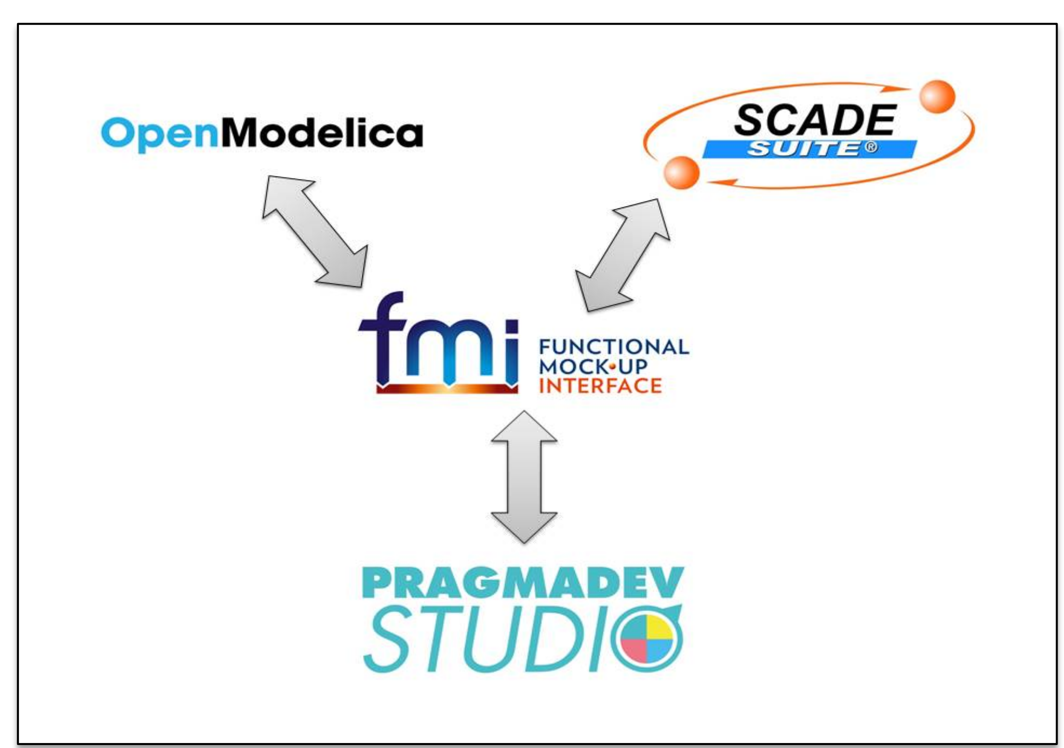
- Functional Mock-up Interface (FMI)
- Connects Functional Mock-up Units (FMU)
- Initially made to connect synchronous models
- The clock is part of the interface
- Similarly to the test, that can be used as a link between both approaches

FMI

The higher the level of abstraction the more appropriate is an event driven description.

CPS combine event drive and clock driven models.

PragmaDev Studio support FMI V2 for CPS.

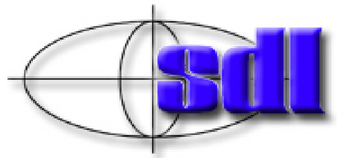


Half way conclusion

- Systems are heterogeneous (sync & async).
- Event driven approach is the best because it is:
  - Legible.
  - Closer to the requirements.
  - Can test any type of system (sync, async, and CPS).

What is SDL

- Specification & Description Language.
- ITU (International Telecommunication Union) standard.
- Was used by the ETSI to standardize telecommunication protocols.
- Event oriented.
- Graphical.
- Formal: complete & non-ambiguous; allows to fully describe the system.
- Object-oriented.
- Companion language: MSC, Message Sequence Chart



Why use SDL

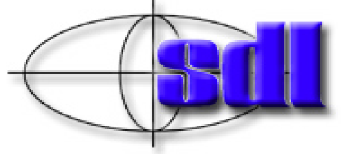
- SDL graphical abstraction (architecture, communication, behavior) brings a lot to development teams.
- SDL being formal, it is possible to simulate the system behavior on host with graphical debugging facilities.
- SDL being formal, full code generation is possible.
- Several model checking techniques allow to validate the design, or to generate tests.
- SDL being object oriented, software components are reusable (ETSI telecommunication protocol standards fully use object orientation).

UML: a brief presentation

- UML (Unified Modeling Language) standardized by the OMG (Object Management Group).
  - Can be used to represent any type of systems;
  - Graphical;
  - Used at a pretty high level of abstraction;
  - Not formal, i.e. another language is necessary to describe in detail (C, C++, Java, SDL);
  - Very object oriented.
- **Poor semantic**
- No real-time specificity in UML:
  - UML has no graphical representation for classical real time concepts such as: tasks, semaphores, messages, timers…
  - UML is adapted to C++ for static data representation;
  - Deployment diagram perfect for distributed systems;
  - In practice UML models are not synchronized with the design.

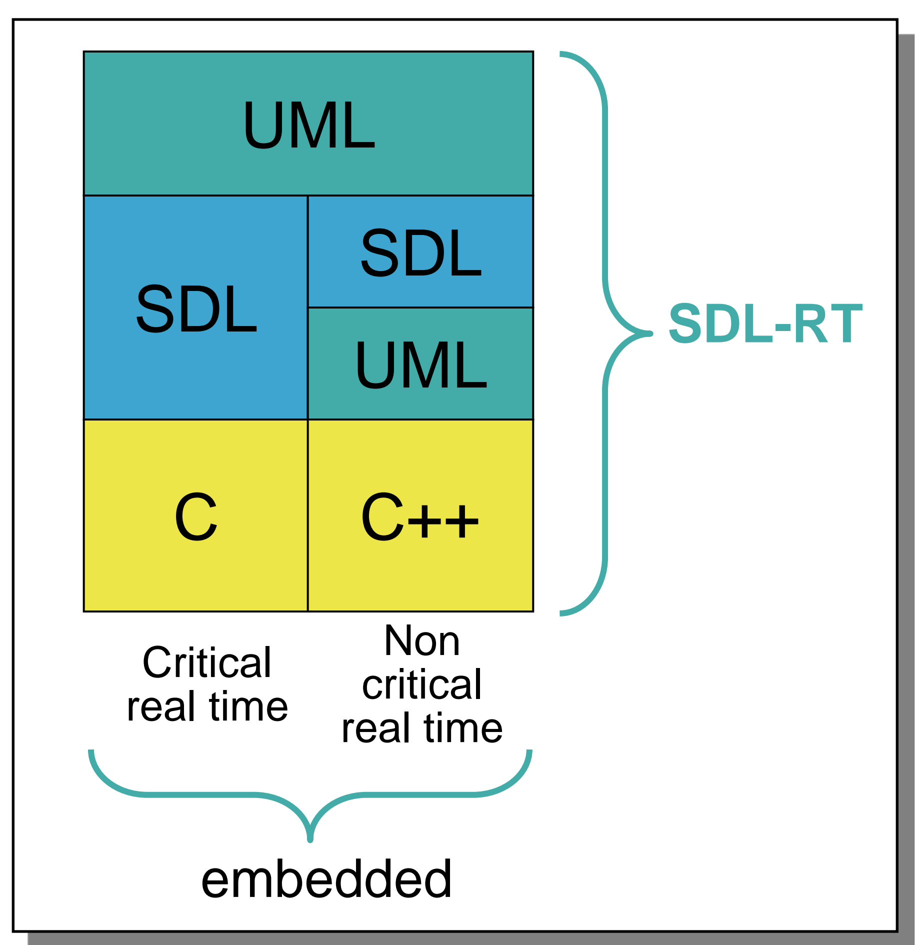
What is SDL-RT (1)

- Specification & Description Language, Real Time.
- Combination of languages:
  - SDL for architecture & dynamic aspects.
  - UML for high-level specification & static aspects.
  - C/C++ for data types & behavior.
- Standardized by the ITU in SDL 2010, allowing C as a alternative language for data & behavior in SDL.
- Can be considered as a UML 2 real-time profile.



What is SDL-RT (2)

- 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.
- Described static aspects in UML with C++ implementation.
- Remove SDL concepts having no practical implementation.
- Extend SDL to deal with uncovered real time concepts (interrupts, semaphores).



Why use SDL-RT

- Keeps the graphical representation and the level of abstraction of SDL and UML.
- Same graphical debugging facilities as with SDL.
- Easier to integrate with software modules providing C APIs: RTOS, drivers, legacy code, …
- Lower-level or classical real time concepts are available: pointers, semaphores, …
- Generated code is more legible than with a higher-level language.
- Available from http://www.sdl-rt.org for free.
- Based on a standardized textual format (XML).

ASN.1

- Abstract Syntax Notation 1.
- Standardized by the ITU.
- Describes only data types and values; for actual handling of these types & values, another language is needed.
- Interfaces natively with SDL (Z.105 / Z.107) & TTCN.
- Several software packages exist allowing to manipulate ASN.1 data in C & C++.
- Standardized platform-independent encoding rules, allowing data exchange in heterogeneous systems.

MSC: dynamic view

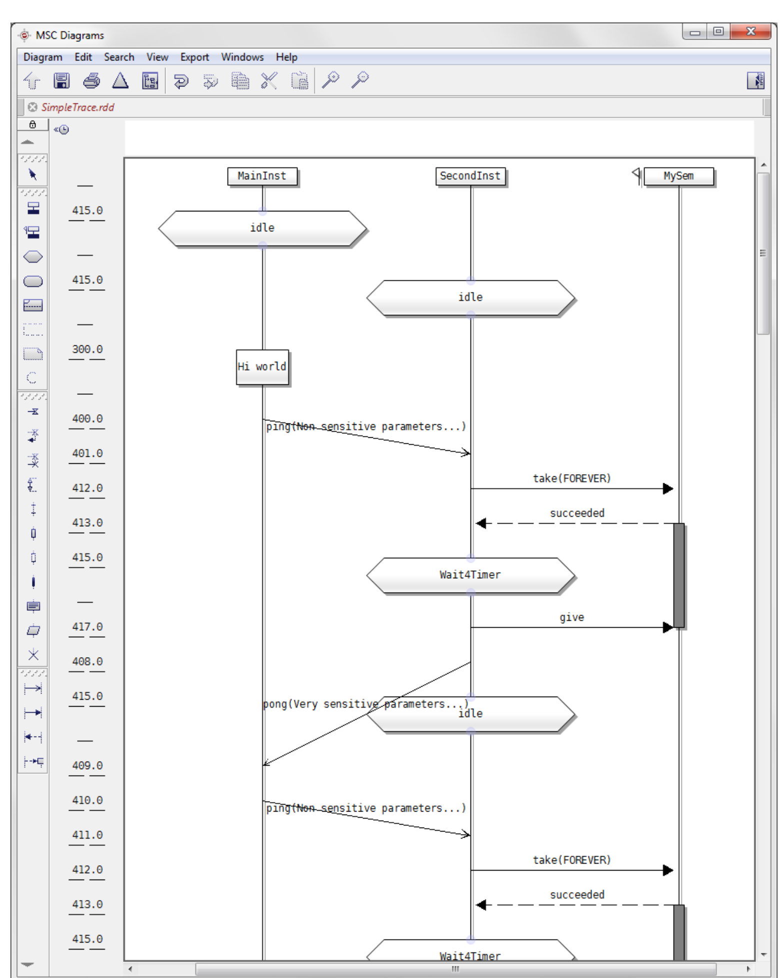
ITU (International Telecommunication Union) standard.

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



SDL / SDL-RT: graphical representations

SIGNAL

SIGNAL

Interaction
(MSC)

Architecture / behavior
(SDL / SDL-RT system, blocks & processes)

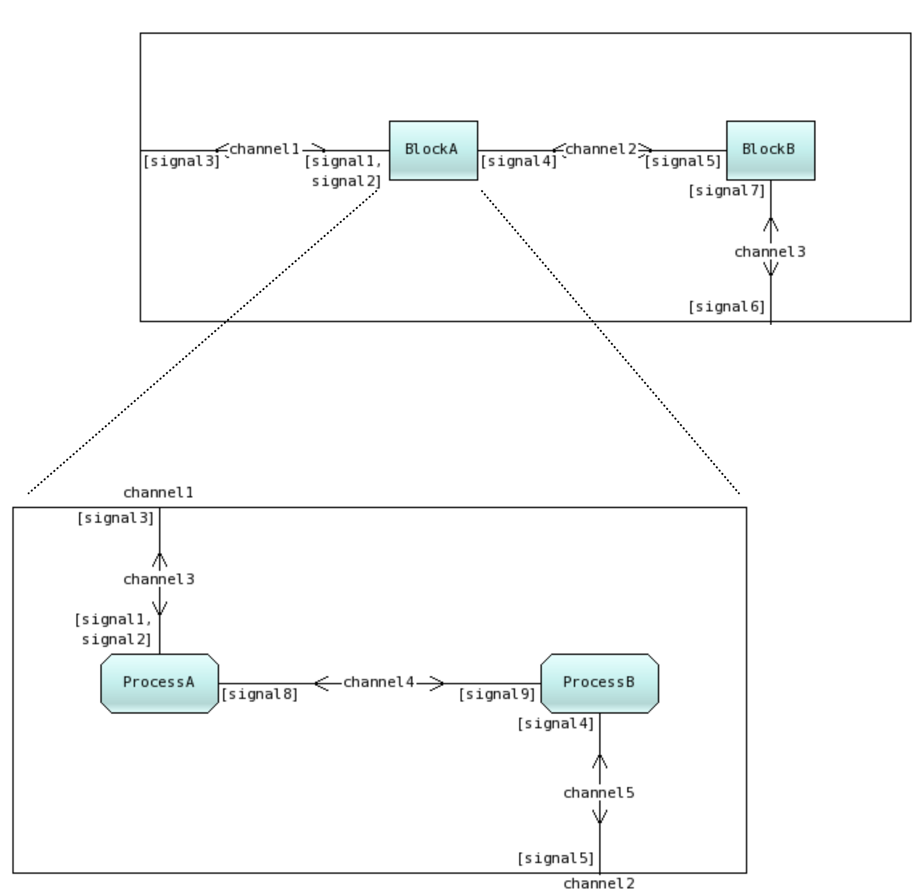
Hardware architecture
(UML deployment)

Development process

- Scenarios for system behavior described with MSCs
- External interface for system: messages with typed parameters
- Architecture based on agents: system, block, process, procedure
- Internal messages & communication channels
- Behavior: finite state machines:
    - SDL code
    - C / C++ code for SDL-RT

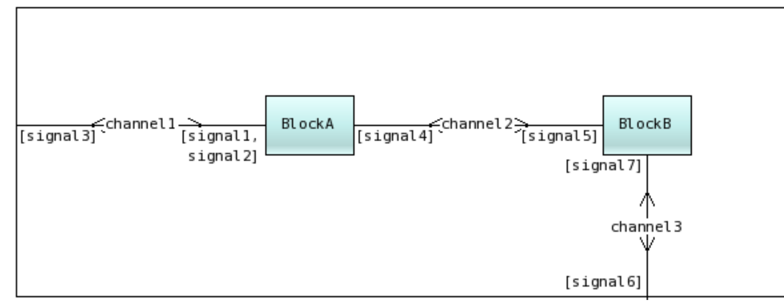
SDL / SDL-RT: architecture

Architecture
and
Communication



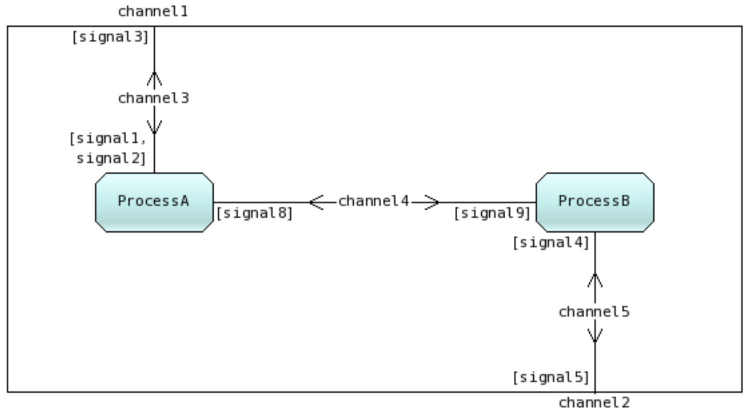
SDL / SDL-RT: architecture

- A block is a high level functional entity.
- It is used to organize and architecture the system.
- Block architecture should not change through the development process.
- Blocks are usually not implemented as an execution entities.



SDL / SDL-RT: architecture

- Lowest level of the architecture is the process.
- A process is an execution entity.
- Processes execute concurrently.
- It has an implicit message queue.
- Message based communication is asynchronous.



SDL / SDL-RT: communication

- Signal in SDL – message in SDL-RT: same concept.
- Message:
  - A message is defined by its name
  - A message can have 0 or several parameters
  - Parameters types are any SDL or C / C++ types
  - Messages can be gathered in lists
  - Messages can be defined at any level of the architecture

SDL-RT

```
MESSAGE my_first_msg, my_second_msg(char, int);
MESSAGE my_third_msg(MyStructType*);
MESSAGE_LIST my_msg_list = my_first_msg, my_second_msg;
MESSAGE_LIST another_msg_list = (my_msg_list), my_third_msg;
```

SDL

```
signal my_first_msg, my_second_msg(Character, Integer);
signal my_third_msg(MyStructType);
signallist my_msg_list = my_first_msg, my_second_msg;
signallist another_msg_list = (my_msg_list), my_third_msg;
```

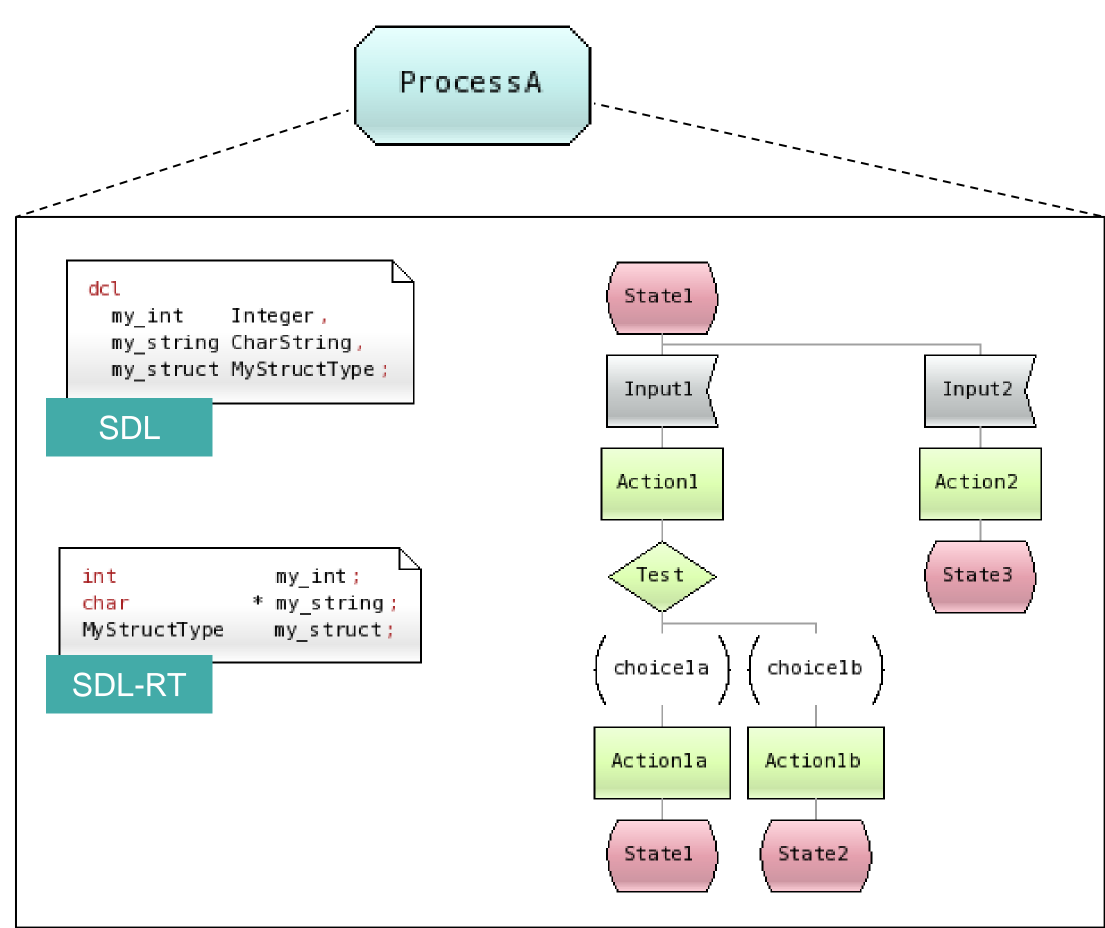
SDL-RT: synchronization

- Semaphore
  - A semaphore is a common resource.
  - It can be accessed by any process.
  - It can be defined in any agent.
- Unneeded in SDL since there is no global data.

```
<semaphore type> <semaphore name>( <option> { , <option> }* )
```

SDL / SDL-RT: behavior and data

- A process behavior is based on a graphical finite state machine
- Data types are either SDL, or C or C++ for SDL-RT



SDL-RT: data types

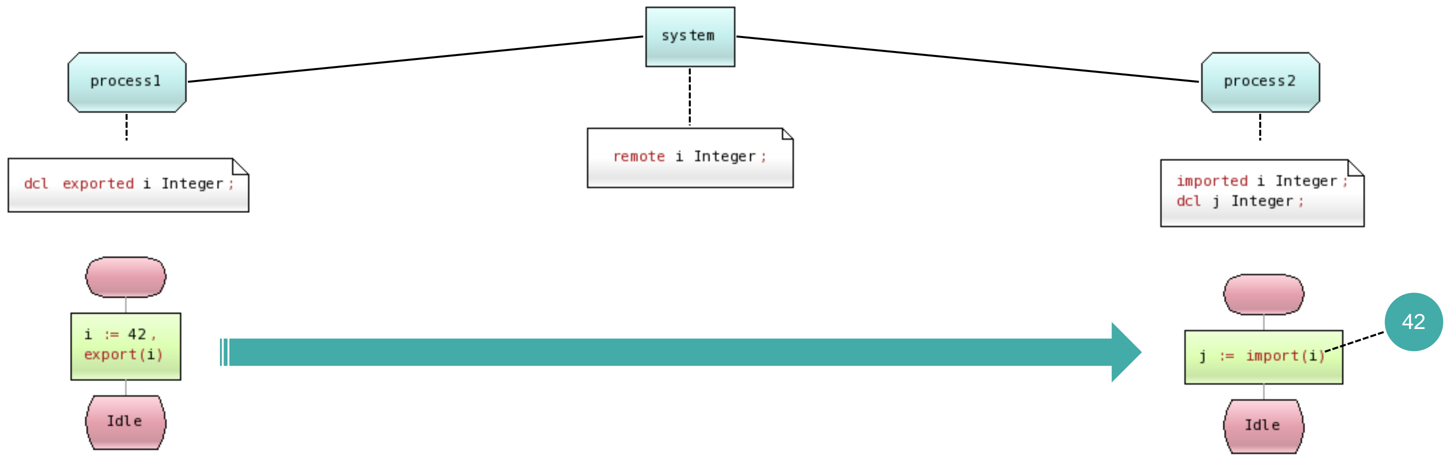
- Data types are C or C++, declared in .h files or in the agent diagram.
- Variables can be global to the whole system, declared at system level and defined in an external C file.
- Variables can be local to a process, defined in the process diagram.
- Variables can be declared in a block and will then be visible in all agents in the block. They will have to be defined in an external file and will actually be global.

SDL: data (1)

- Base types: Boolean, Integer, Natural, Real, Character, CharString, PID, Time, Duration.
- *Abstract* types: no need for `long` / `short`, `signed` / `unsigned`, …
- Complex data types:
  - Struct: similar to C `struct`; allows optional fields.
  - Choice: similar to C `union` but safer.
  - Array: associative array; *not* similar to C arrays.
  - String: ordered sequences.
  - Bag: multi-sets.

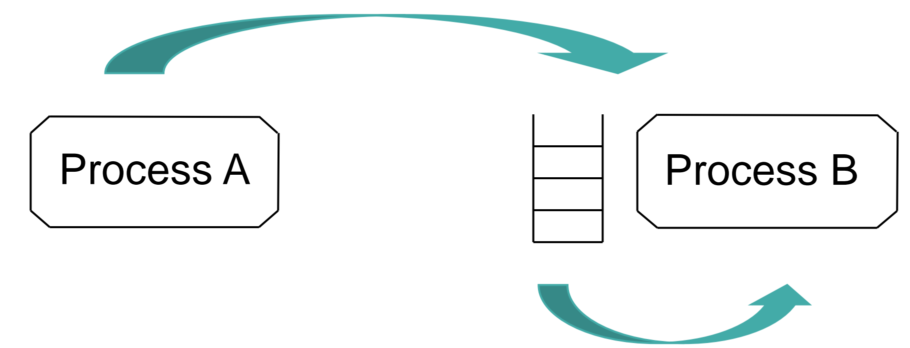
SDL: data (2)

- No variable at system or block level, only synonyms (constants).
- Variables in a parent agent are visible and modifiable in its child agents (e.g, a procedure can read and modify the variables of its parent process).
- Remote variables: allow an agent to make one of its variables visible in other parts of the system (not modifiable):



Process: message queue

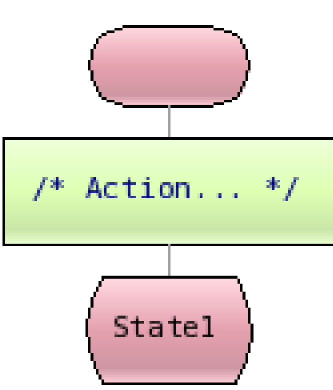
- A process has an implicit message queue
- The message queue is a FIFO
- Processes should not send messages to themselves; in some very special cases it might be a solution
- A timer going off is a message in the queue



Process: initial transition

Execution entry point of the process.

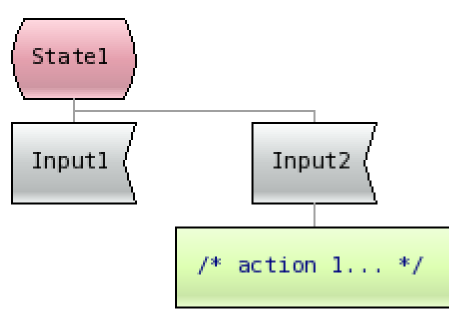
- Message output
- Task creation
- Timer start

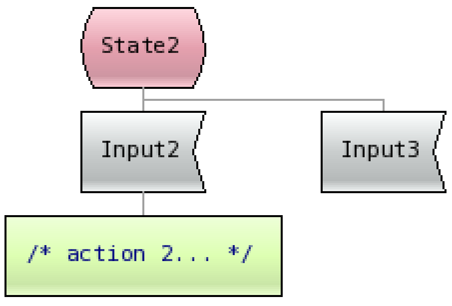


Process: state

A state means the same trigger will generate a different behavior. A trigger can be:

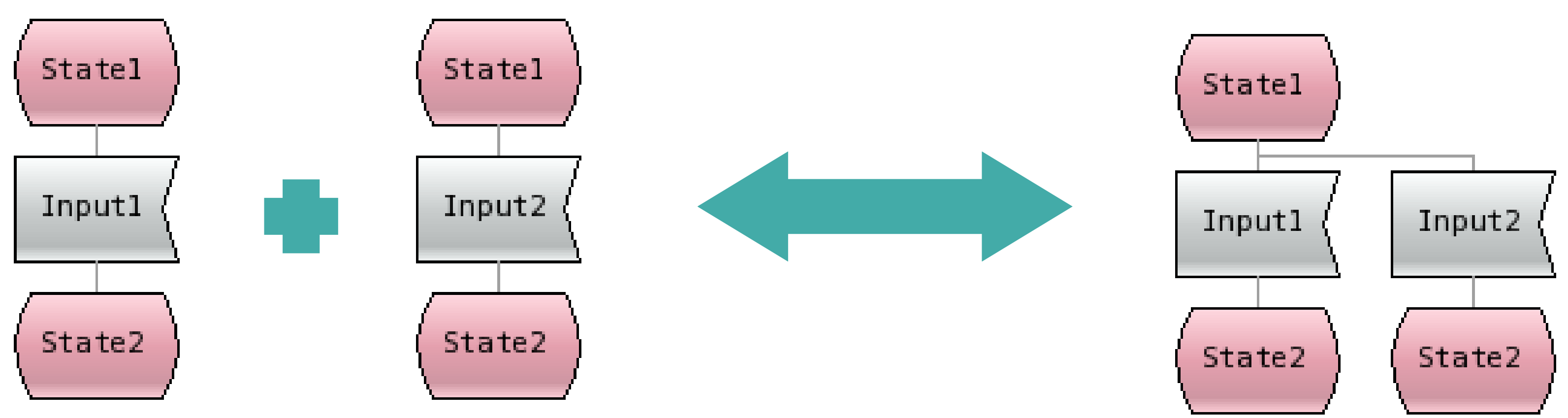
- Message input
  Read from the message queue; with or without parameters.
- Continuous signal
  A continuous signal is a condition to check when reaching the state. Continuous signals have associated priorities.





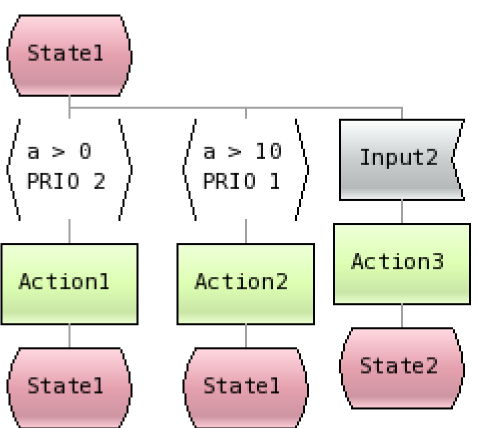
Process: state

- The same state can be described with several symbols
- At the end of a transition, the process goes to a next state. The same state symbol is used.



Process: continuous signal

- Continuous signals have priorities: `PRIO` keyword in SDL-RT; `priority` keyword in SDL.
- In SDL, continuous signals are evaluated only if the message queue is empty.
- In SDL-RT, continuous signals are evaluated before checking the message queue.

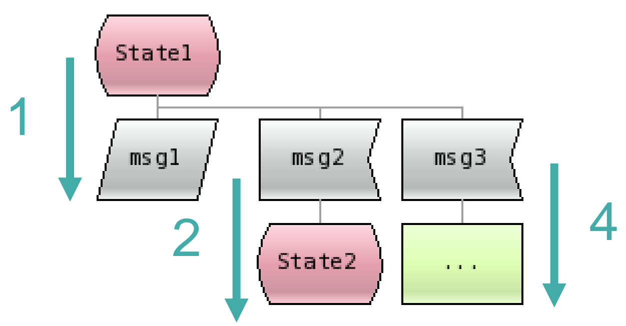


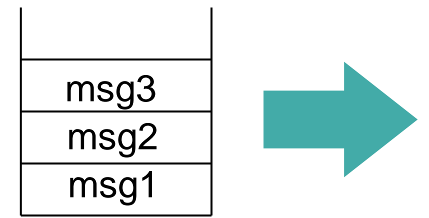
If the process gets into State1 with a =15 and Input2 is in the queue, the resulting behavior is:

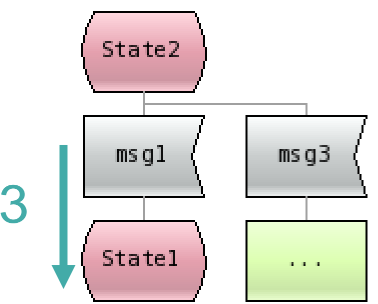
- Action2
- Action1
- Action3

Process: message save

- Messages received while unexpected are thrown away. It is considered normal behavior.
- A message can be saved to be treated when the finite state machine reaches a new state. The first saved message will be treated before the other messages.

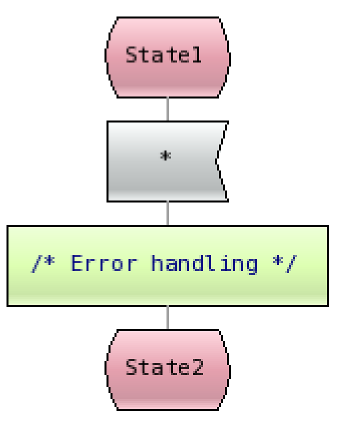




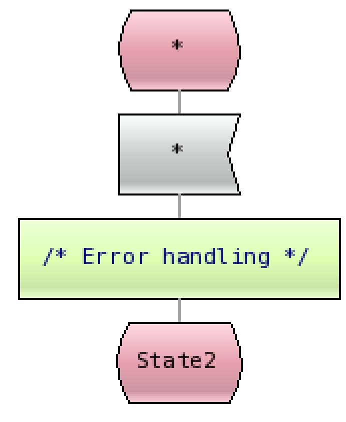


Process: unexpected messages

- Unexpected messages can be treated with the ‘*’ message representing a default behavior.
- All cases can be treated in the ‘*’ state.

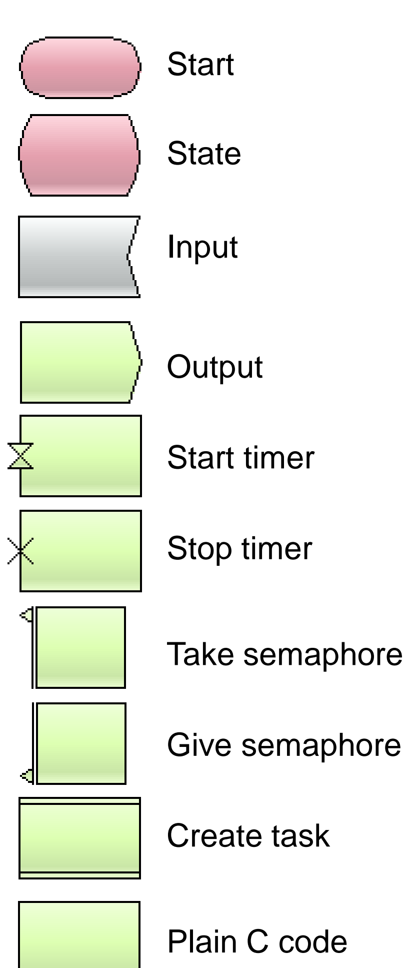


All unexpected messages in state will go through the same error handling code



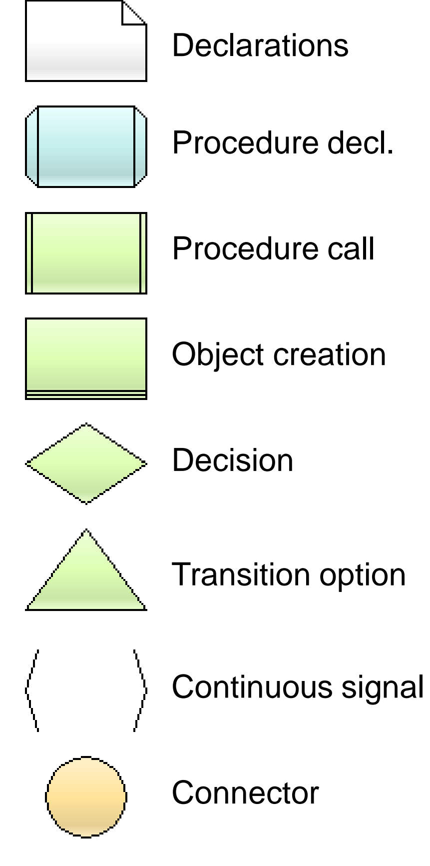
All unexpected messages whatever the state is will go through the same error handling code

Process: transitions



Actions that can be done in a transition are:

- Send out a message
- Start a timer
- Cancel a timer
- Take a semaphore (SDL-RT)
- Give a semaphore (SDL-RT)
- Create a task
- Call a procedure
- Execute any SDL or C / C++ code
- Test an expression
- Connect to another branch of code

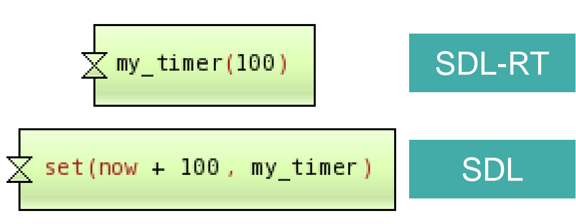


Process: message output

- Messages are sent:
  - To a named process:
    TO_NAME <name> (SDL-RT) / TO <name> (SDL)
  - To a process identifier:
    TO_ID <pid> (SDL-RT) / TO <pid> (SDL)
  - To the environment:
    TO_ENV <macro name> (SDL-RT) / TO ENV (SDL)
  - VIA <gate or channel name>
- Special keywords for <pid>:
  SENDER, PARENT, OFFSPRING, SELF
- Parameters are copied (shallow copy in SDL-RT, deep copy in SDL)

```
message [ (<parameters>) ]
[ <destination> ]
```

Process: timers



my_timer is started
for 100 time units

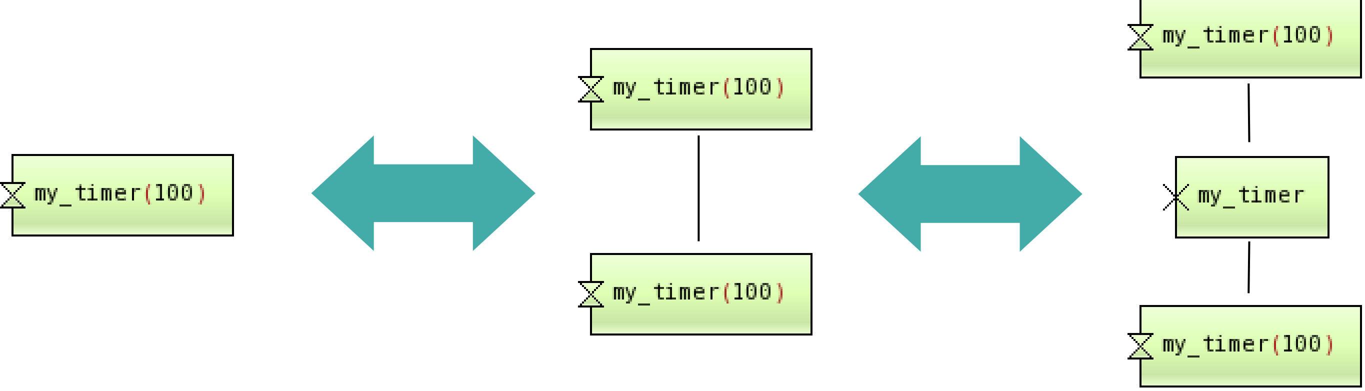
- Timers are not declared in SDL-RT, can be in SDL but don't need to.
- When a timer goes off it becomes a message in the queue like any other message. Messages already present will be treated first.
- If the timer is cancelled while the corresponding message is already in the queue; the message will be (virtually) removed from the queue.
- Timer is identified by its name.
- The timer message will have the same name.
- Time unit is the system tick in SDL-RT and irrelevant in SDL.

my_timer is cancelled

my_timer is received

Process: timers

- The timer message can only be received by the process that started it.
- The timer goes off only once. There is no concept of repeated or cyclic timer.
- A timer can only be started once. If restarted, the previous one is cancelled. Typical usage: checking a response has been received within a given amount of time.



Process: timer state

- A classical timer usage is to make sure to get an answer within a certain time frame.
- The new timer state simplifies the start and cancellation of timers associated to as ingle message.

mRequest
SET(NOW+5, requestTimer)
idle
mAnswer
requestTimer
requestTimer
error
done

mRequest
idle
mAnswer
state timer now+5
done
error

Process: semaphores (SDL-RT)

- Semaphore handling is RTOS specific.
- Binary, Counting, and Mutex are available if the RTOS supports them.
- Sometimes binary are mapped to counting semaphores. RTOS integration pages should be checked.
- SDL-RT semaphores are identified by their names. PragmaDev Studio provides C macros to manipulate semaphores with their ids.
- SDL-RT semaphores are automatically created at startup. PragmaDev Studio provides C macros to dynamically create semaphores.

Semaphore take

Semaphore give

Process: semaphores (SDL-RT)

- Taking a semaphore is a blocking action if the semaphore is not available.
- Timeout values can be provided. After that time the transition resumes. Timeout is expressed in time unit (usually ticks). SDL-RT defines FOREVER and NO_WAIT keywords.
- The return value indicates if the take was successful or not. Possible values are RTDS_OK and RTDS_TIMEOUT.

```
status = my_semaphore(100)
```

if status == RTDS_OK

  Take succeeded

else if status == RTDS_TIMEOUT

  Take failed

SDL / SDL-RT: object orientation

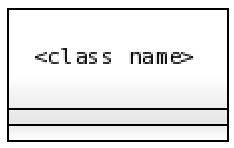
Object-orientation is available:

- For agents: block classes & process classes can be created and instantiated in a system.
- For passive objects: in SDL-RT, passive classes can be created and their instances used in processes and procedures. They will be implemented as C++ classes.

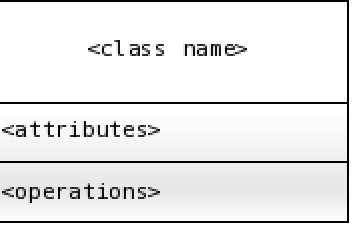
In both cases, “classic” OO concepts are available: specialization, composition, association, …

SDL / SDL-RT: class concept

A class is the descriptor for a set of objects with similar structure, behavior, and relationships.



Class symbol with details hidden

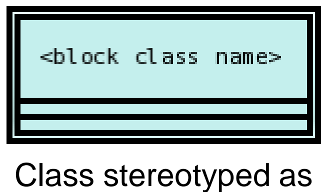


Class symbol full representation

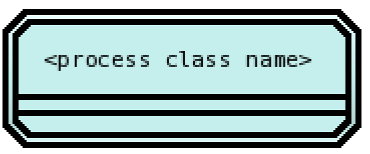
The outer border of the class symbol indicates what kind of class it is: passive, active, block class, process class, …

SDL / SDL-RT: active classes

An instance of an active class owns a thread of control and may initiate control activity. An instance of a passive class holds data, but does not initiate control.

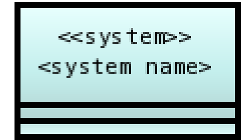


Class stereotyped as a block class



Class stereotyped as a process class

Agents in the system are also classes and can be represented as such. Unneeded in SDL, useful in SDL-RT to associate passive classes to them:



Class stereotyped as a system

Class stereotyped as a block

Class stereotyped as a process

SDL / SDL-RT: classes & instances

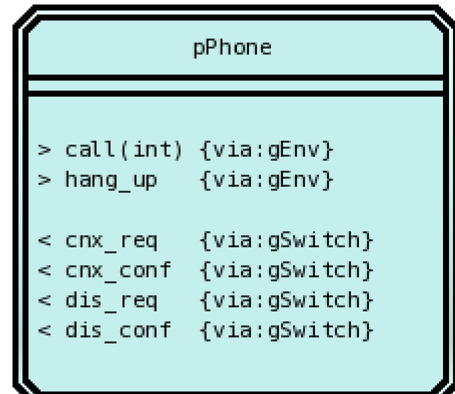
- Processes & blocks support multi-instantiation, so they're actually classes, that do not support specialization.
- Data is always encapsulated: variables in processes cannot be seen from the outside, communication can only be done via messages.
- For actual agent classes, object orientation applies to:
  - Declarations;
  - Transitions;
  - Connectors.

SDL-RT: active class interface

Attributes for active classes are for documentation only (e.g: local variables in class). Operations for active classes are the incoming or outgoing asynchronous messages for the class. For block & process classes, they define the class's interface.

Syntax: `<message way> <message name>[(<param. types>)] [{via:<gate name>}]`

`<message way>` is '>' for incoming messages & '<' for outgoing messages.



Process class pPhone can receive messages:

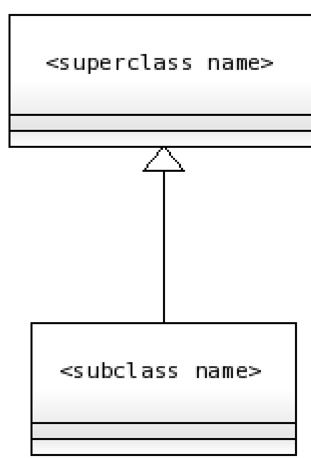
- `call`, & hang_up through gate gEnv;
- `cnx_req`, `cnx_conf`, `dis_req` & `dis_conf` through gate `gSwitch`.

SDL / SDL-RT: class specialization

Specialization defines a ‘is a’ relationship between two classes. The most general class is called the superclass and the specialized class is called the subclass.

All instances of the subclass are also instances of the superclass.

For active classes, only block classes & process classes support specialization, not systems, blocks or processes.



SDL / SDL-RT: specialization example (1)

myGate
[msg1]

state1
msg1
action1
state2

MySuperClass

inherits MySuperClass;

state1
msg2
action2
state3

myGate
[msg2]

MyClass

=

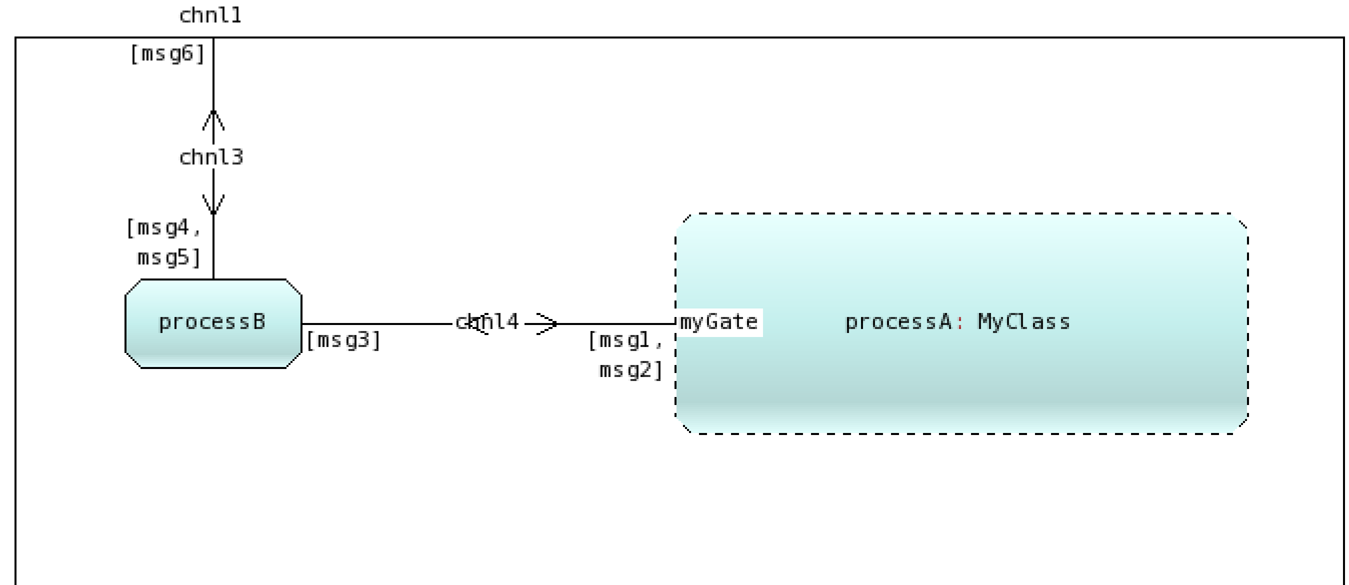
myGate
[msg1,
msg2]

state1
msg1
msg2
action1
action2
state1
state3

An instance of MyClass

SDL / SDL-RT: specialization example (2)

The class is instantiated in the system architecture.

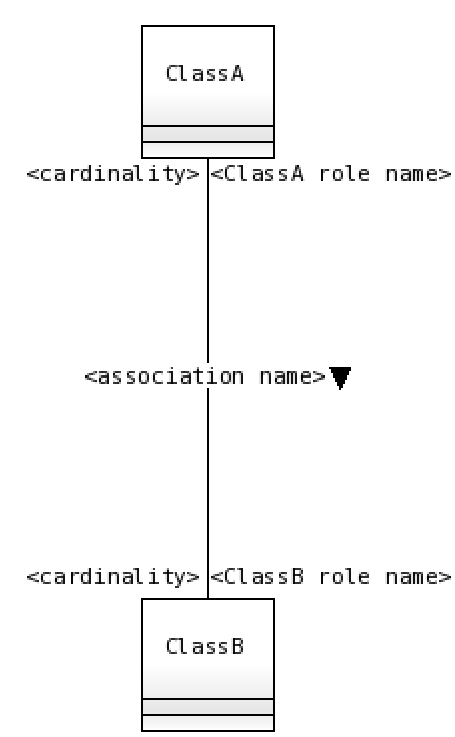


SDL-RT: specialization example

SDL-RT: class association

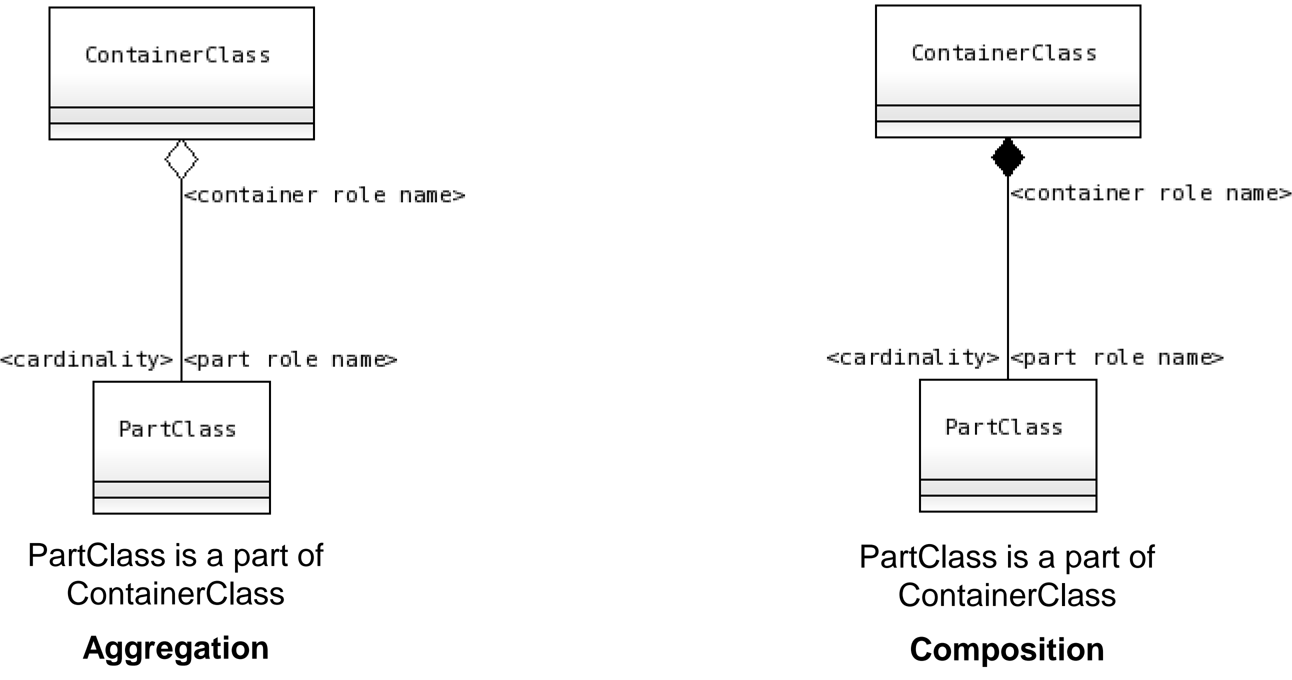
An association is a relationship between two classes. It enables objects to communicate with each other. The meaning of an association is defined by its name & read direction, or the role names on the associated classes.

The cardinality indicates how many objects are connected at the end of the association: 1, *, n, n..m



SDL-RT: class aggregation and composition

- Aggregation defines a ‘is a part of’ relationship between two classes.
- Composition is a strict form of aggregation, in which the existence of the parts depend on the existence of the container.



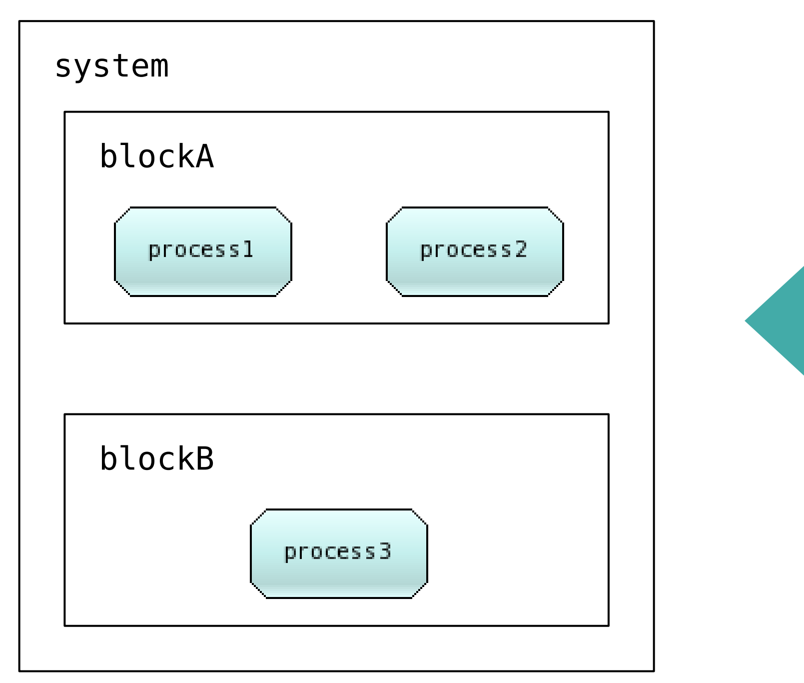
PartClass is a part of ContainerClass

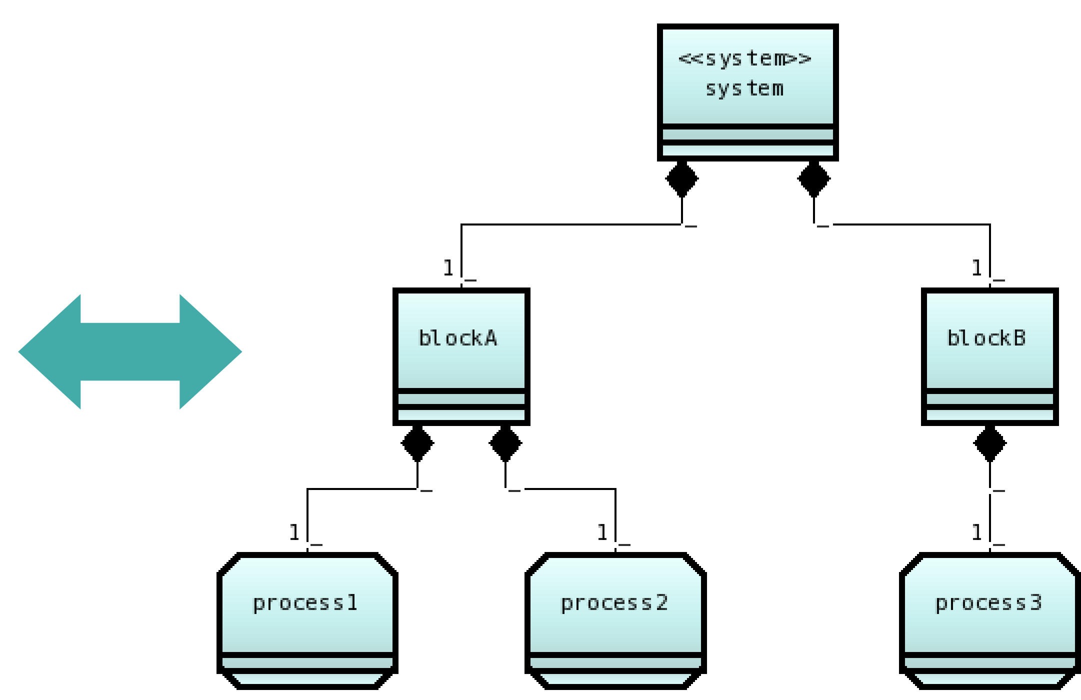
Aggregation

PartClass is a part of ContainerClass

Composition

SDL / SDL-RT architecture: implicit composition





PRAGMADEV *STUDIO*

SDL-RT: class diagram example

Relations between passive classes (C++) and active classes (SDL)

PragmaDev Studio – Diagrams

Classes

~::pCentral
> mCardAndCode()
< mEmployee()
< mIntruder()

~::pCentralAdmin
> mAddUser()
> mDeleteUser()
< mAdministrator()
< mOK()
< mKO()

user_factory
1

UserFactory
<<create>>()
get_user(card_id : char*, code : char*) : User*
new_user(card_id : char*, code : char*) : User*

1 creator
records ▼
1..* users

User
is_administrator : boolean {frozen}
<<create>>(card_id : char*, code : char*, is_admin : boolean)
matches(card_id : char*, code : char*) : boolean
delete()

UML / SDL-RT: Deployment diagram

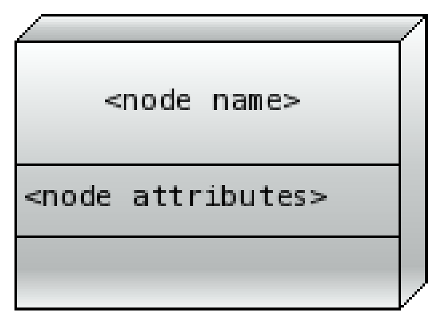
Physical deployment

PragmaDev Studio – Diagrams
depl

ClientPC
network.id = "192.168.2.25"
<<IP>> network
Server
network.id = "217.126.14.56"
SocketComm.h
SocketComm.c
pClient
network.id = 50010
pDispacther
network/id = 50020
pServer
network.id = 50030
WinIP.h
WinIP.c

Deployment diagram: node

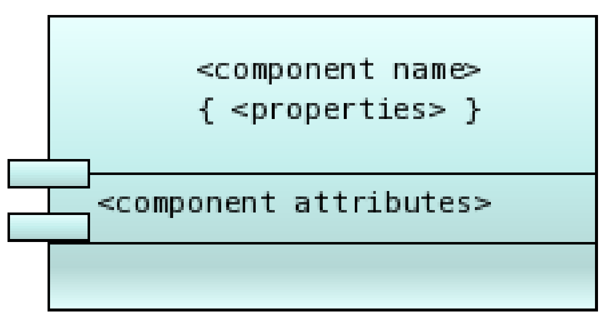
A node is a physical object that represents a processing resource.



Deployment diagram: node

A component represents a distributable piece of implementation of a system.
Two types of components:

Executable component

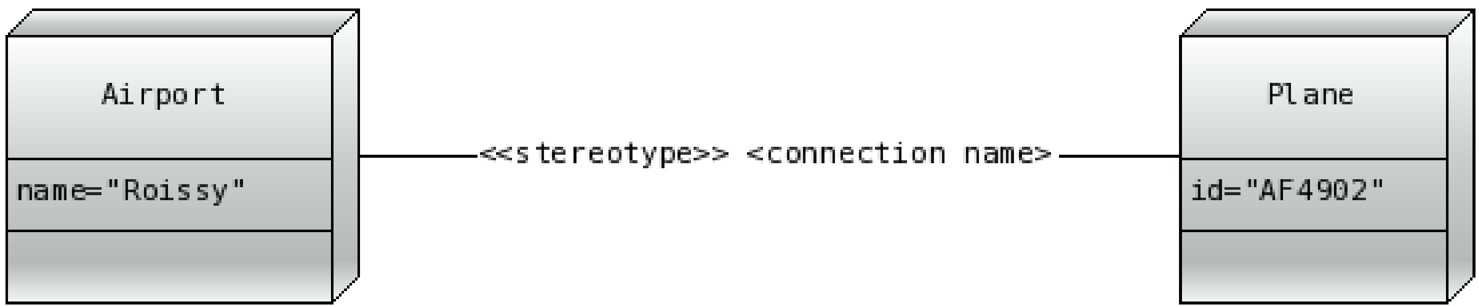


File component

Typically, <component name> = name of an agent in the architecture.
Properties used to define mapping between agents and RTOS tasks.

Deployment diagram: connection

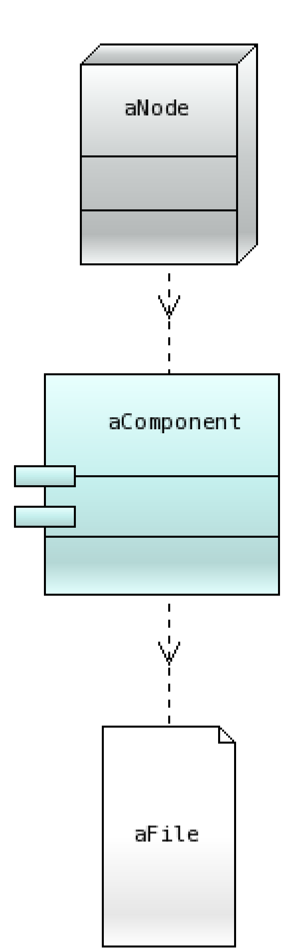
A connection is a physical link between two nodes or two executable components. It is defined by its name and stereotype.



Deployment diagram: dependency

Dependency between elements can be represented graphically.

- A dependency from a node to an executable component means the executable is running on the node.
- A dependency from a component to a file component means the component needs the file to be built.
- A dependency from a node to a file means that all the executable components running on the node need the file to be built.



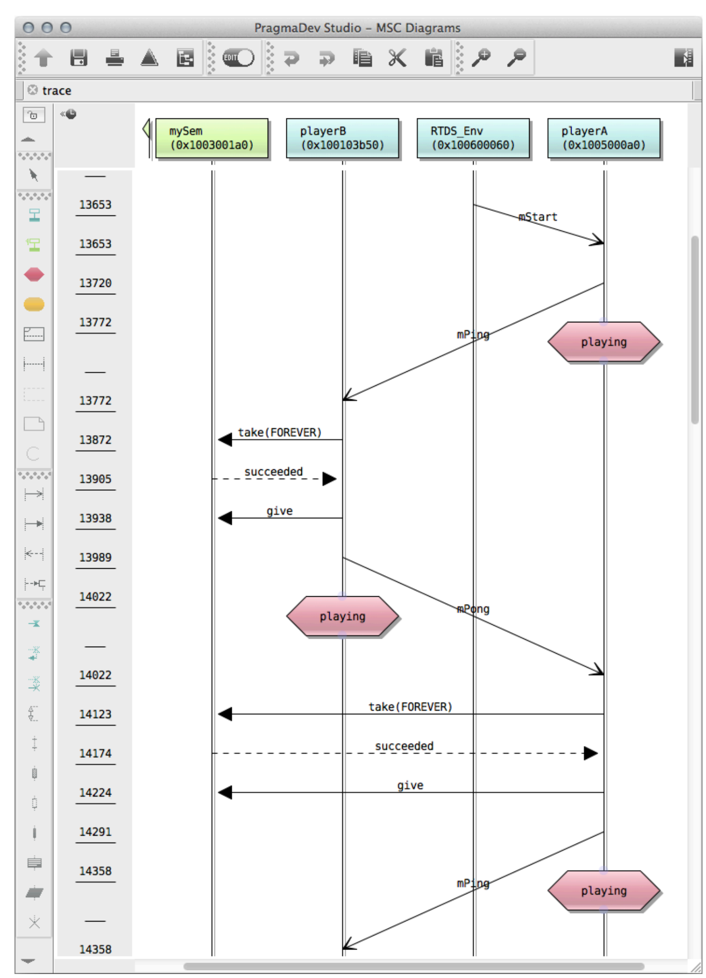
MSC: dynamic view

SDL-RT Message Sequence Chart

- Vertical lines represent a task, the environment, an object or a semaphore,
- Arrows with a stick arrowhead represent message exchanges.
- Arrows with a filled solid arrowhead represent synchronous operation calls, semaphore manipulations or timers.

Can be used:

- As specification
- As execution traces
- As properties (PSC)



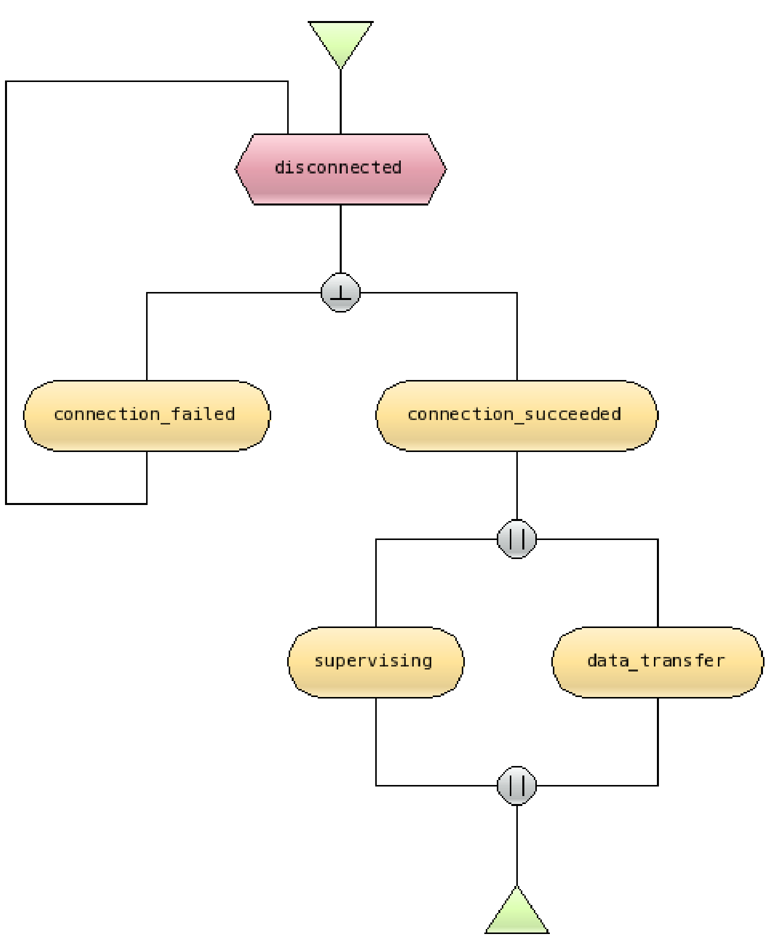
MSC: symbols

- Agent or passive object instance
- Semaphore
- Condition (global or instance state)
- MSC reference
- Inline expression: optional, alternative, parallel…
- Message
- Method call or semaphore operation
- Method return
- Dynamic instance or semaphore creation
- Timer started by instance
- Timer time-out for instance
- Timer cancelled by instance
- Relative time constraint
- Co-region: order of events is not significant
- Method segment (instance busy) or semaphore unavailable
- Suspended segment (instance waiting)
- Strict operator (PSC)
- Action symbol
- Message saved by instance
- Instance killed or semaphore destroyed

HMSC: dynamic overview

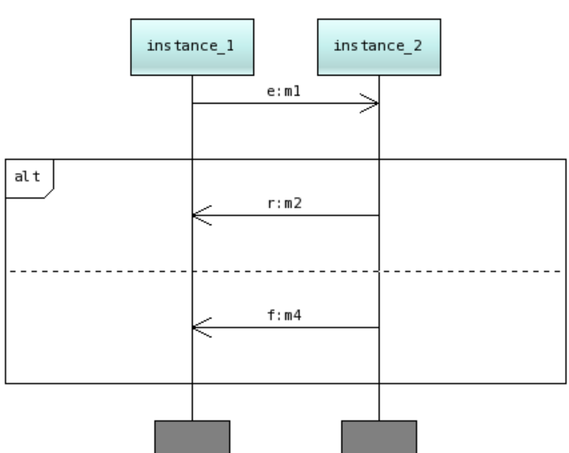
High level Message Sequence Chart

- Sequence of MSCs,
- Parallel independent execution of MSCs.



PSC: Property Sequence Chart

- Expresses requirements on events happening in the system in an if / then sequence.
- Positive or negative properties: set of condition events followed by a set of required or forbidden events.
- Functional and non-functional properties (timing).
- Support for inline expressions: optional, alternative, …
- Support for complex properties via chain constraints.



PSC: Property Sequence Chart

What is TTCN-3

- Testing and Test Control Notation V3.
- Standardized test language.
- Developed and maintained by the ETSI.
- ITU-T Z.161
- Use in several domains : telecommunication, transportation, financial application...

TTCN: Module

- The module is the top level obligatory container for TTCN-3 code.
- A module contains a declaration part and an optional control part.
- It is possible to import a complete module or just a module part (group).

```
module myModuleName {
        //All TTCN-3 code
}
```

TTCN: Data types

- Basic : integer, boolean, float, charstring
- Structured : enumerated, record, set, union
- List types : array, set of, record of
- ASN.1 (Abstract Syntax Notation 1) types supported
- Specific test types : verdicttype

TTCN: components

- Component is the execution entity.
- Test scenarios are executed on one or more components.
- Communication with other components or with the SUT is done through ports.

TTCN: components

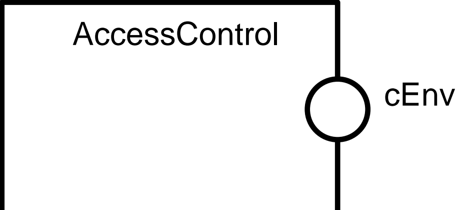
- Component is the execution entity.
- Test scenarios are executed on one or more components.
- Communication with other components or with the SUT is done through ports.

```
type component AccessControl {
        port port_cEnv cEnv;
}
```

TTCN: components

- Component is the execution entity.
- Test scenarios are executed on one or more components.
- Communication with other components or with the SUT is done through ports.

```
type component AccessControl {
        port port_cEnv cEnv;
}
```



TTCN: Ports

- Ports are the interface of the components
- Messages are sent and received through ports
- FIFO
- Ports are mapped to others ports

TTCN: Ports

- Ports are the interface of the components
- Messages are sent and received through ports
- FIFO
- Ports are mapped to others ports

```
type port port_cEnv message {





}
```

TTCN: Ports

- Ports are the interface of the components
- Messages are sent and received through ports
- FIFO
- Ports are mapped to others ports

```
type port port_cEnv message {
        out card;
        out key;



}
```

TTCN: Ports

- Ports are the interface of the components
- Messages are sent and received through ports
- FIFO
- Ports are mapped to others ports

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

TTCN: Test System Interface

- The TSI is the interface between the test and the SUT.
- It is a component which contains only ports.
- A mapping must be done between the test components and the TSI.

TTCN: Testcase

- The testcase is the behavior description of the test. It describes how the SUT will be stimulated and what are the expected reactions.
- Composed of :
  - one or several test scenarios executed on one or several components.
  - one TSI

TTCN: Testcase start

- Testcases are started in the control part.

- Two components will be implicitly created when the testcase starts, the TSI and the main test component (MTC).

TTCN: Testcase start

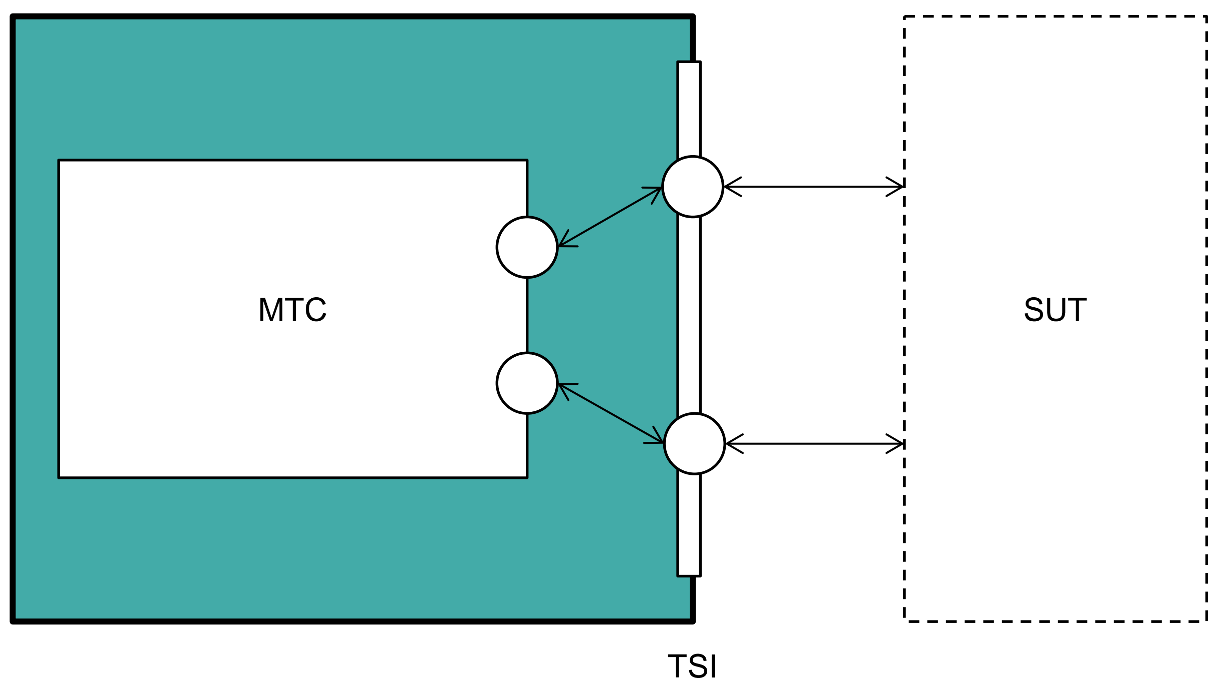
- Testcases are started in the control part.

```
control{
        execute(testcase1());
}
```

- Two components will be implicitly created when the testcase starts, the TSI and the main test component (MTC).

TTCN: single component testcase

- By default, MTC and TSI are based on the same component type. An implicit mapping is done in that case.



TTCN: Multi components testcase

- It is possible to have several components for one testcase, in this case, there is one MTC and several PTCs.
- When a testcase is started, only the MTC is started, then the MTC creates and starts PTCs.

TTCN: Multi components testcase

- It is possible to have several components for one testcase, in this case, there is one MTC and several PTCs.
- When a testcase is started, only the MTC is started, then the MTC creates and starts PTCs.

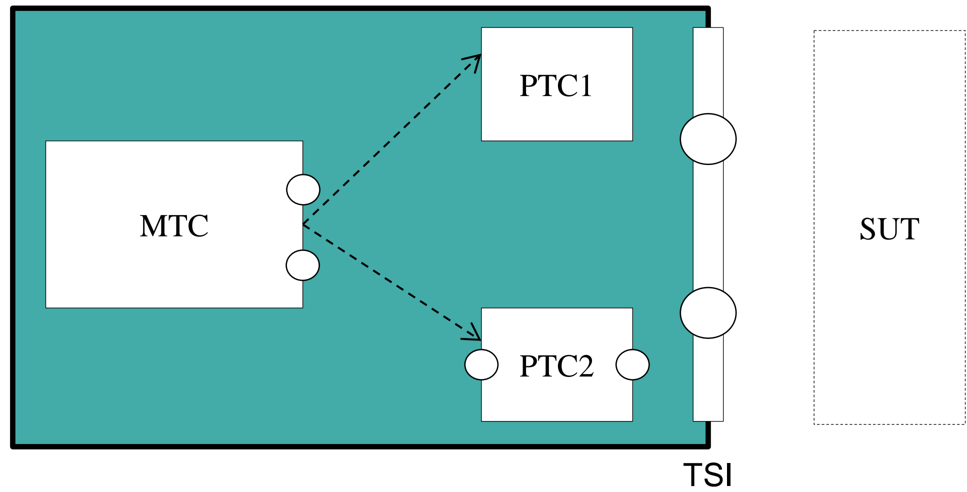
```
Var componentTypeName PTCName;
PTCName = componentTypeName.create()
```

TTCN: Multi components testcase

- It is possible to have several components for one testcase, in this case, there is one MTC and several PTCs.
- When a testcase is started, only the MTC is started, then the MTC creates and starts PTCs.

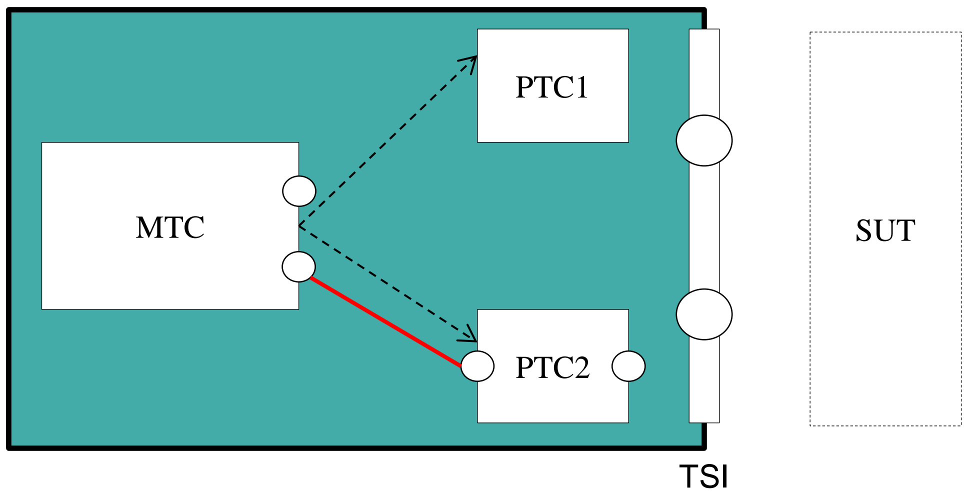
```
Var componentTypeName PTCName;
PTCName = componentTypeName.create()
PTCName.start(functionName)
```

TTCN: Multi components testcase



In this case, a mapping between the ports of the components and the TSI has to be done.

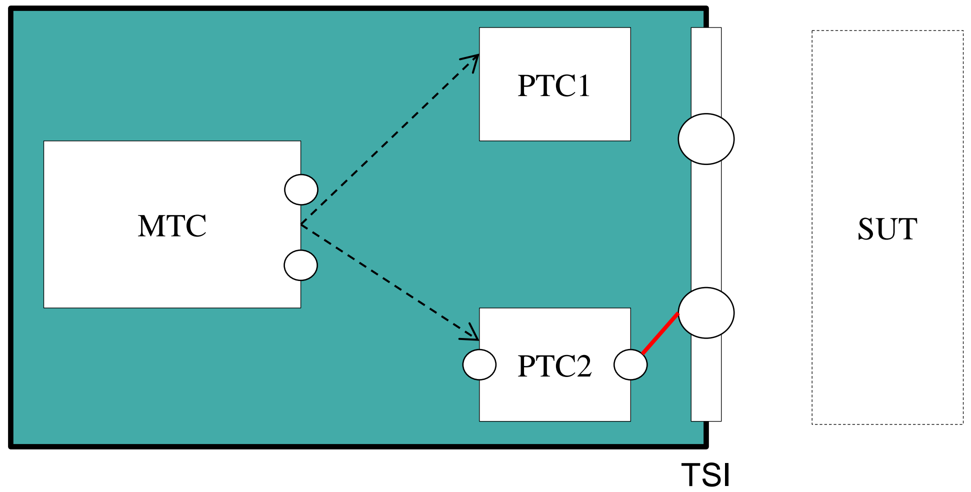
TTCN: Multi components testcase



In this case, a mapping between the ports of the components and the TSI has to be done.

```
connect(mtc:p2,PTC2:p1);
```

TTCN: Multi components testcase



In this case, a mapping between the ports of the components and the TSI has to be done.

```
connect(mtc:p2,PTC2:p1);
map(system:p2,PTC2:p2);
```

TTCN: Communication

- Two types of communication:
  - Synchronous (procedure based)
  - Asynchronous (messages)
- Communication is done via ports.
- Message is declared as record

TTCN: Communication

- Two types of communication:
  - Synchronous (procedure based)
  - Asynchronous (messages)
- Communication is done via ports.
- Message is declared as record

```
type record Request {
```

TTCN: Communication

- Two types of communication:
  - Synchronous (procedure based)
  - Asynchronous (messages)
- Communication is done via ports.
- Message is declared as record

```
type record Request {
    integer param1,
```

TTCN: Communication

- Two types of communication:
  - Synchronous (procedure based)
  - Asynchronous (messages)
- Communication is done via ports.
- Message is declared as record

```
type record Request {
    integer param1,
    charstring param2
}
```

TTCN: Template

- Defines one or more values of a specific type.
- Used to easily send messages or match received messages.
- Parameterized to re-usability

TTCN: Template

- Defines one or more values of a specific type.
- Used to easily send messages or match received messages.
- Parameterized to re-usability

```
type record displayMessage {
        charstring param1
}
```

TTCN: Template

- Defines one or more values of a specific type.
- Used to easily send messages or match received messages.
- Parameterized to re-usability

```
type record displayMessage {
        charstring param1
}

template displayMessage EnterCard:= {
        param1 := “Enter card“
};
```

TTCN: Message based communication

- Two main operations :
  - send : to send a message via the specified port.
  - receive : to receive and check a message from a specified port. receive operation removes the top message of the associated port queue if this message satisfied the matching criteria.

TTCN: Message based communication

- Two main operations :
  - send : to send a message via the specified port.
  - receive : to receive and check a message from a specified port. receive operation removes the top message of the associated port queue if this message satisfied the matching criteria.

```
p1.send(Request1)
```

TTCN: Message based communication

- Two main operations :
  - send : to send a message via the specified port.
  - receive : to receive and check a message from a specified port. receive operation removes the top message of the associated port queue if this message satisfied the matching criteria.

```
p1.send(Request1)
p1.receive(Answer1)
```

TTCN: Simple testcase

```
testcase testcase1() runs on sSystem {
        port.send(Request1)
        port.receive(Answer1)
        port.send(Request2)
        port.receive(Answer2)
}
```

- Sequence of send and receive operations.
- What if many receives are possible ?

TTCN: Alternative statement

- Several exclusive alternative can be possible during a testcase execution.
- Alternative branch are based on variable evaluation or message reception.

TTCN: Alternative statement

- Several exclusive alternative can be possible during a testcase execution.
- Alternative branch are based on variable evaluation or message reception.

```
alt {
  [] ch.receive(correctAnswer) {
```

TTCN: Alternative statement

- Several exclusive alternative can be possible during a testcase execution.
- Alternative branch are based on variable evaluation or message reception.

```
alt {
  [] ch.receive(correctAnswer) {
    // action 1
  }
```

TTCN: Alternative statement

- Several exclusive alternative can be possible during a testcase execution.
- Alternative branch are based on variable evaluation or message reception.

```
alt {
  [] ch.receive(correctAnswer) {
    // action 1
  }
  [] ch.receive(otherAnswer) {
```

TTCN: Alternative statement

- Several exclusive alternative can be possible during a testcase execution.
- Alternative branch are based on variable evaluation or message reception.

```
alt {
  [] ch.receive(correctAnswer) {
    // action 1
  }
  [] ch.receive(otherAnswer) {
    // action 2
  }
}
```

TTCN: Snapshot

All the operations of an alternative have to be tested with the same information. Snapshot is an implicit save of all the relevant information of the testcase before an alternative statement.

If no alternative branch is verified, a new snapshot is taken.

TTCN: Altstep

- As a function but for alternative.
- To structure alternative behavior.
- Can be activated by default in a testcase.

```
altstep fail_alt() runs on AccessControl {
  []cEnv.receive(WrongMessage) {
    setverdict(fail);
  };
}
```

TTCN: Verdict

- Result of a testcase execution.
- 5 verdict value:

  none > pass > inconc > fail > error
- Each component has its own local verdict. The verdict of a testcase is the worst verdict off all its components.

TTCN: Verdict

- Result of a testcase execution.
- 5 verdict value:

  none > pass > inconc > fail > error
- Each component has its own local verdict. The verdict of a testcase is the worst verdict off all its components.

```
setverdict(pass)
```

TTCN: Verdict

- Result of a testcase execution.
- 5 verdict value:

  none > pass > inconc > fail > error

- Each component has its own local verdict. The verdict of a testcase is the worst verdict off all its components.

```
setverdict(pass)
var verdicttype v1;
v1 := getverdict;          //get verdict in v1
```

TTCN: SDL interfacing

Interfacing made via names: system, messages, channels.

```
module tests
  {
  // Messages
  type record m1 { integer param1 };
  type record m2 { integer param1, charstring param2 };
  type record m3 { charstring param1 };
  type enumerated m4 { e_m4 };

  // Ports types
  type port port_ch1 message
    {
    out m1;
    out m2;
    in  m3;
    };

  type port port_ch2 message
    {
    in  m4;
    };

  // Component for system interface
  type component sys
    {
    port port_ch1 ch1;
    port port_ch2 ch2;
    };
  }
```

sys

signal m1(Integer), m2(Integer, CharString), m3(CharString), m4;

[m3] ch1 [m1, m2] blk1

[] internal []

[m4] ch2 [] blk2

TTCN: SDL interfacing

- SUT defined by the name of the component type used as the system interface. Can be an SDL system or a block.
- SDL messages mapped to TTCN records.
- No message parameter name in SDL ⇒ generated names in TTCN (param1, param2, …).
- No empty records in TTCN ⇒ messages with no parameters mapped to enumerated type with a single possible value.
- Channels mapped to ports with the same name. Port type name can be anything.
- Outgoing messages in SDL are incoming messages in TTCN, and vice-versa.
- Only the messages sent to / received from the environment in SDL need to be described in TTCN. Internal messages cannot be received or sent.

Conclusion: concepts

- Synchronous
  - Low level
    - Logical
    - Continuous
  - Easier to verify
- Event driven
  - Higher level
  - Can embed synchronous
  - High level of complexity

Conclusion: technologies

- SysML: any system (OMG)
- UML: any object oriented organisation (OMG)

Poor semantic

- SDL: executable event driven systems (ITU-T Z.100)
- SDL-RT: real time embedded application (ITU-T Z.104)
- ASN.1: abstract data types (ISO, IEC, ITU)
- MSC: scenario (ITU-T Z.120)
- HMSC: scenario (ITU-T Z.120)
- PSC: property
- TTCN-3: testing (ITU-T, ETSI)

Strong event driven semantic

Technical possibilities

- Because based on formally defined languages
  - Functional verification
  - Automatic non regression testing
  - Property verification
  - Deployment simulation
  - Performance optimization

Property verification

- Exhaustive simulation:
  - Verimag
  - LAAS (FIACRE)
- Context Description Language
  - ENSTA (OBP)
- Symbolic resolution:
  - PragmaList: joint lab with French National Nuclear Research center CEA
- Property verification on traces

Benefits:

- **System validation**
- **Robustness**

Model Based Testing

Test generated from:

- Execution traces
  - Simulation
  - Real execution
- Test objective (property in exhaustive simulation)
- Code coverage

Benefits:

- **Easier to write**
- **Easier to maintain**

Deployment simulator

Benefits:

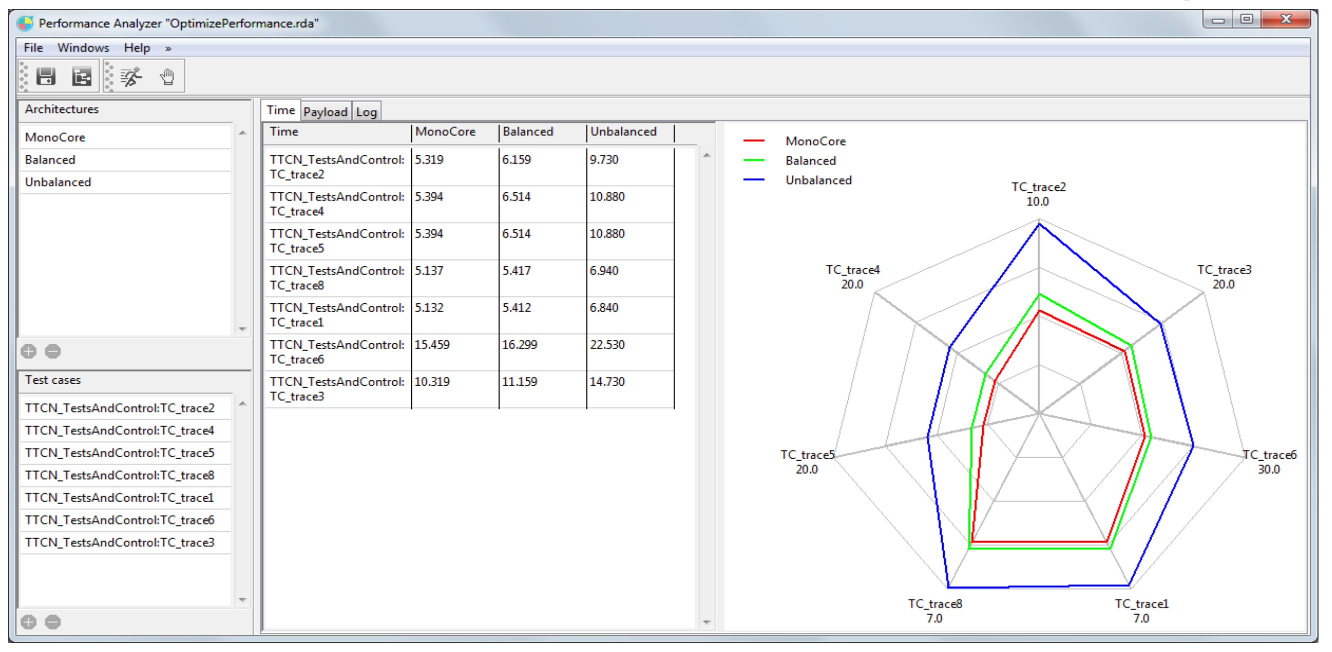
- **Simulate configuration that can not be tested on the field**

	A	B	C	D	E
1	TIME	192.168.1.1:50000	192.168.1.2:50000	192.168.1.3:50000	192.168.1.4:50000
2	1000		mStart(100)		
3	2000			mStart(100)	
4	3000				mStart(100)

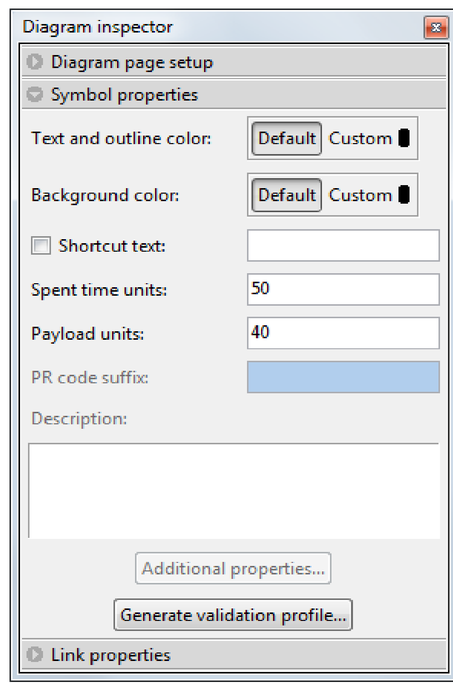
Deploy a substantial number of instances and verify the system of system behaves correctly:

- Deployment diagram
- Environment scenario
- Live and post-mortem simulation traces

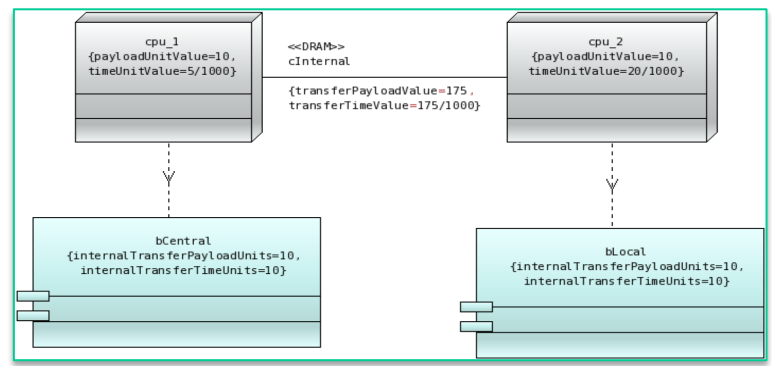
Performance analyzer



Time	MonoCore	Balanced	Unbalanced
TTCN_TestsAndControl: TC_trace2	5.319	6.159	9.730
TTCN_TestsAndControl: TC_trace4	5.394	6.514	10.880
TTCN_TestsAndControl: TC_trace5	5.394	6.514	10.880
TTCN_TestsAndControl: TC_trace8	5.137	5.417	6.940
TTCN_TestsAndControl: TC_trace1	5.132	5.412	6.840
TTCN_TestsAndControl: TC_trace6	15.459	16.299	22.530
TTCN_TestsAndControl: TC_trace3	10.319	11.159	14.730



Automatically execute test cases on different architectures defined with a deployment diagram.



Benefits:

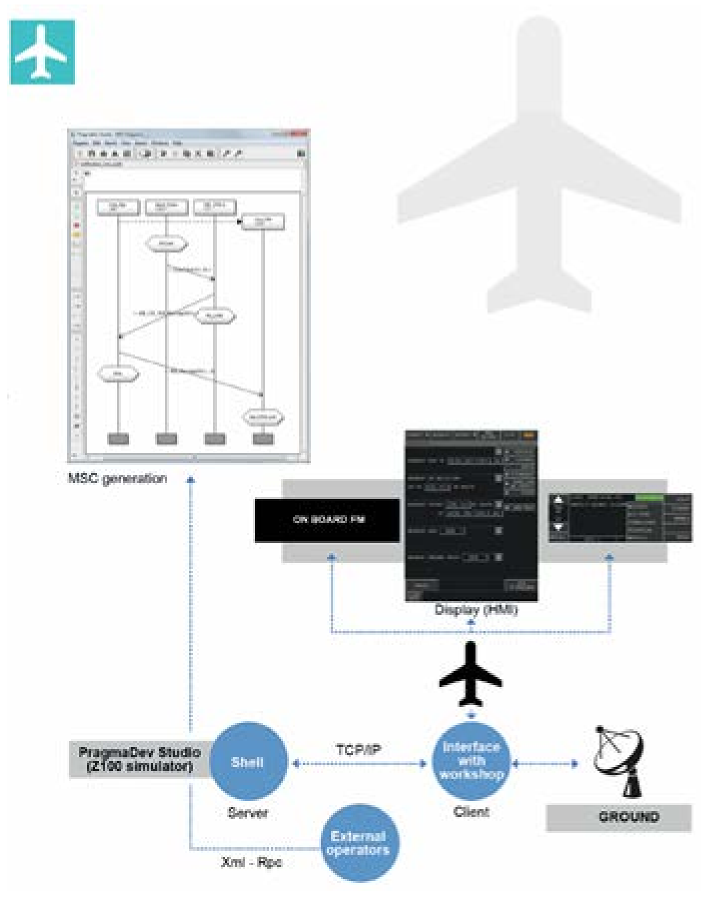
- **Find the best architecture**

4 integrated tools

Real use cases

ATC (Air Traffic Control)

- Simulation
- Connects to cockpit interface
- Certified code generation
- Embedded in all Airbus planes



Real use cases

Renault production line management software designed with PragmaDev.

The Système d'Information de Pilotage de la Tôlerie (SIPTOL) application manages the manufacturing of vehicles in real time, asynchronously handling the various workshops on the production line. Build orders are received by the software, converted into actions, and sent to the robots of the various workshops. SIPTOL is a critical software because if it ever stops, the whole line is stopped. As an example, a one minute stop is one vehicle lost.

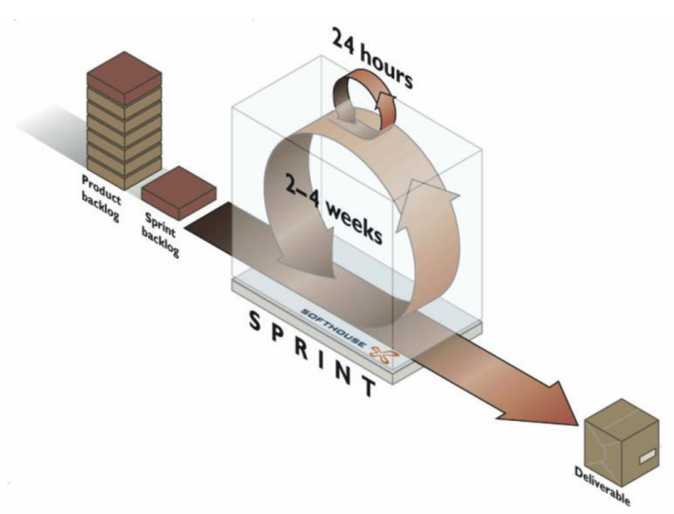
RENAULT

Real use cases

Telecom infrastructure

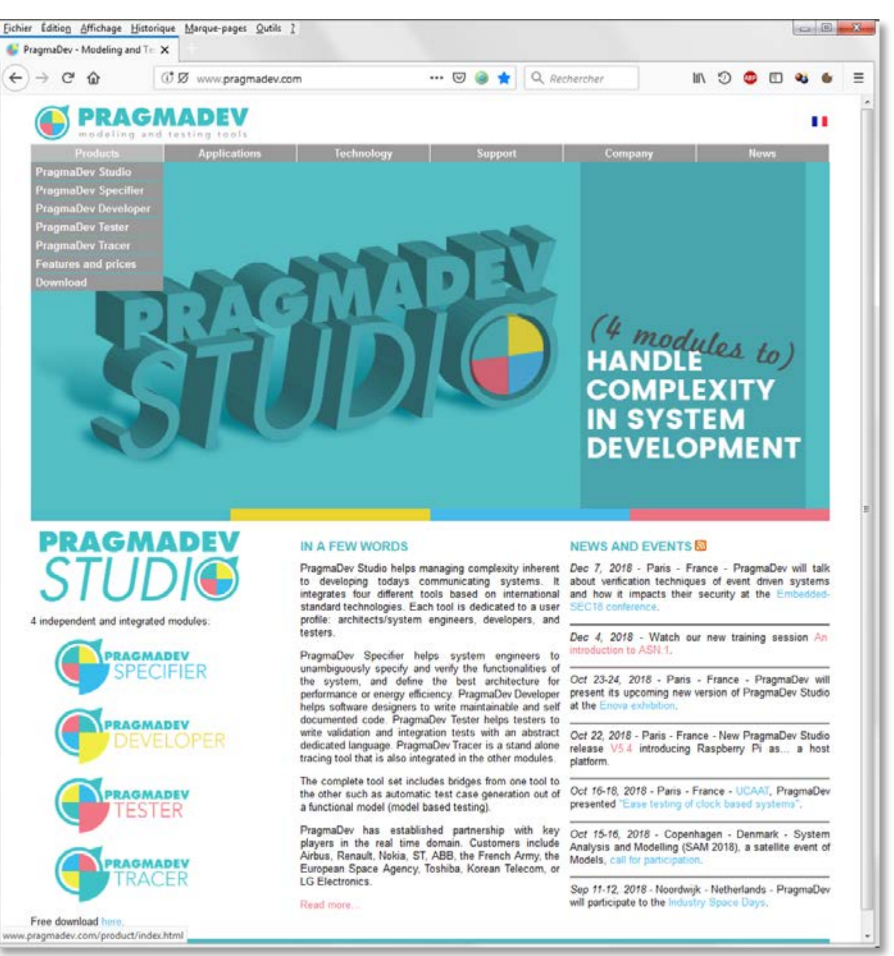
Alcatel-Lucent portfolio is very broad and covers among other things the network infrastructure equipment based on 3G and LTE technologies. PragmaDev is used in several of them.

Alcatel-Lucent uses agile methods and follows a continuous integration process.



Try it out

- Free for small projects
- Working and simple examples



What did you understand ?

kahoot.it