VCE Hands-on Exercices

Lab installation:

VCE is installed in D:\ObeoDesigner-Community-W32bits

Run it by simply executing the shortcut : D:\VCEv4.exe

If you want to continue these exercises at home, you will have to install the VCE environment, following the instructions from here:

http://team.inria.fr/scale/software/vercors (and choose VCE-v4 download)

1. Check and Complete a small app

- The “Tutorial” project in the SummerSchool workspace is incomplete… Your first task is to explore it and complete the missing parts…
- The tutorial document is available for help.

1.1. State machines:

I have prepared a small example for the “Sum-up” state-machine of the tutorial, but it is incomplete. Set up the missing transition…

1.2. Attributes and interfaces:

Last_sum is a local attribute of this component. In the local Class1, we have declared a setter method for this attribute, but forgotten to declare a corresponding getter method. Add it.

Note: Attribute getter and setter do not need a state-machine; their behavioral model and their implementation are generated automatically.

1.3. Fractal-style attributes:

Management of local attributes could be specified using an “attribute controller” NF (non-functional) interface, rather than a functional service interface. Add such an interface, for the management of “last_sum” from outside the component. Name it “AT_last_sum”, declare its UML interface with getter and setter operations, and link the UML interface and the VCE interface.

1.4. Types:

Go to the VCE Type diagram of the Tutorial, and build a record type “Pair”, with two fields “l” and “r” of type argType.

Change the SumUp operation, and make it take a single argument of type Pair.

2. The BFT component

2.1. Go to the BFT project, open the VCE architecture diagram

2.2. Validate the diagram… what is wrong? Complete the necessary elements until the diagram is valid.

2.3. Add a new Server Interface named “Set F” to the BTF composite component.

2.4. Go to the UML Class diagram, add an UML Interface , containing an operation named SetF, with an input argument of type NatType

2.5. Attach this Interface to the “Set F” server interface.

2.6. Do the same (add a server interface, attach UML interface) for the Master Primitive component

2.7. Build the requested bindings… What is missing ?

2.8. Check the Diagram Validity.

3. Composite, multicast, matrix

3.1. In your workspace, create a new VCE project named “Composite”

Build a composite component, with :
Outside:

- 1 serveur interface SI
- 2 client interface CI1, CI2
- A number of control (NF) interfaces

Inside:

- 2 subcomponents
- One connected to SI
- Each connected to one client interface
- One binding between them

Check its validity and produce the AD

3.2. In your workspace, create a new VCE project named “Matrix”

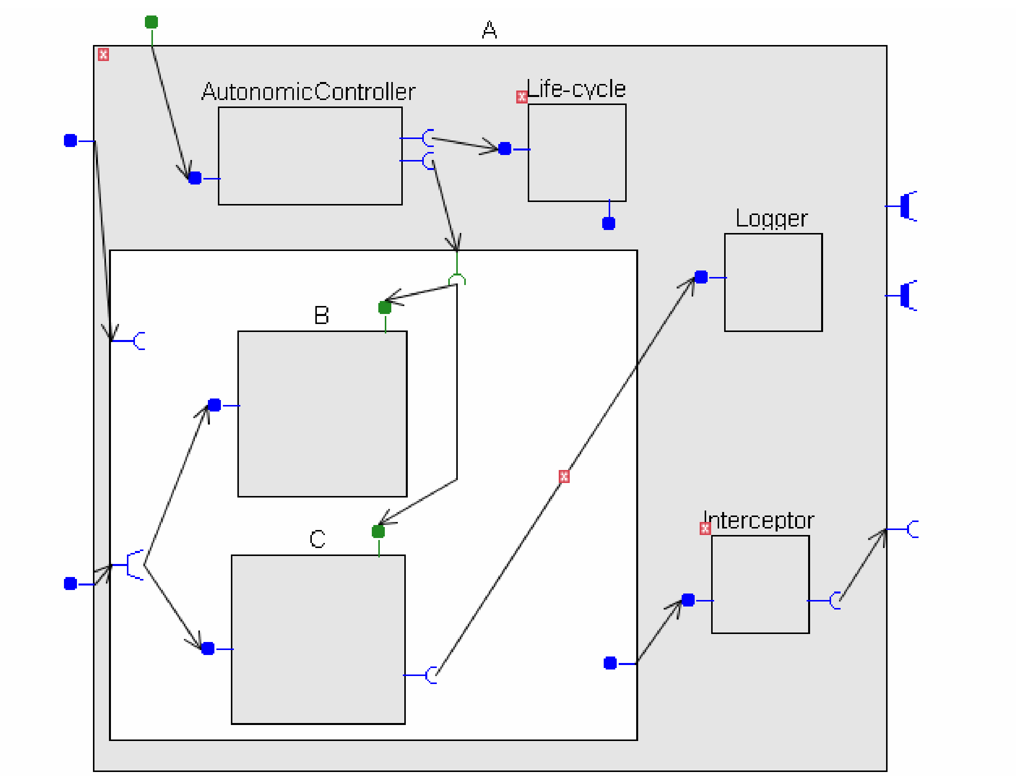
Build a composite component, with:

- One server interface, with an internal multicast interface
- 2 x 3 subcomponents representing matrix blocks, each linked to its left and upper neighbors

4. Validation:

4.1. Analyze this diagram (semantics, validation rules)

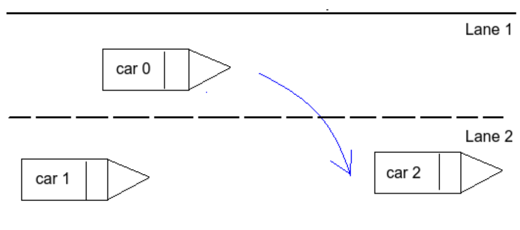
This can be simply on paper, on using VCE if you prefer…



5. If you have finished…:

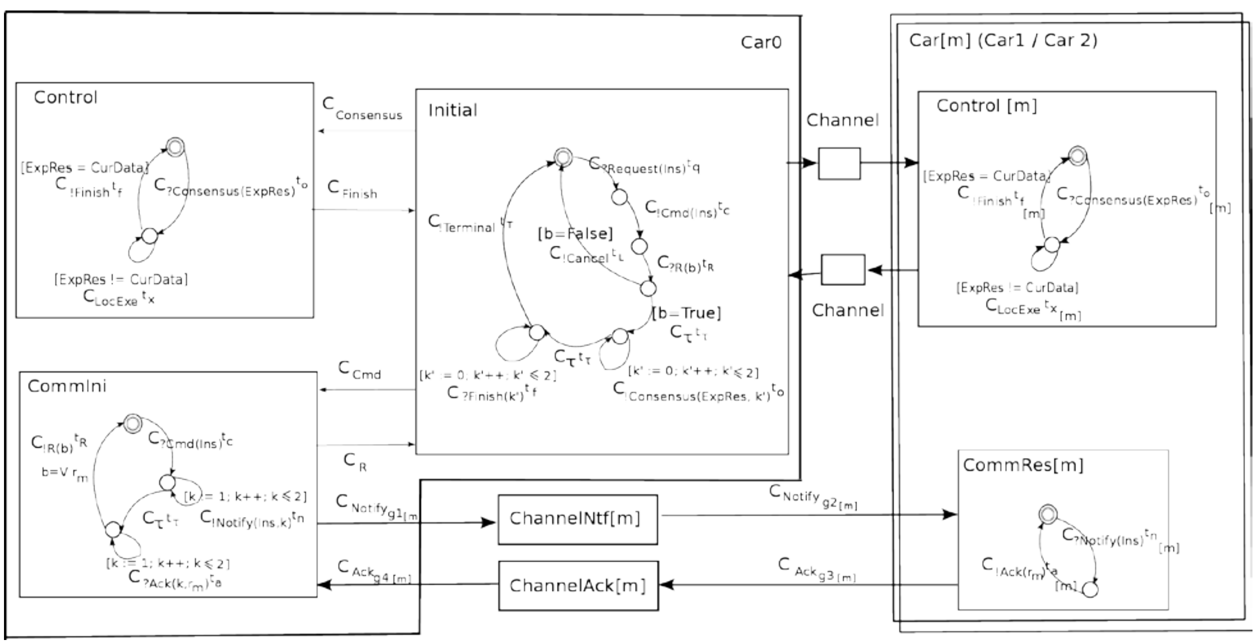
This one is part of last year summer-school lab exercises. You can keep it and terminate by yourself if you want !!!

Intelligent Cars use-case



a) Presentation of the use-case

Beware, these are NOT GCM diagrams, but low level specification. You will have to build the corresponding GCM diagrams.



Specific timed model transition language

This use-case was originally built for a timed version of the semantic formalism.

But in the exercises today, we only consider the untimed fragment, by dropping the clocks and the time variables.

E.g. in the CommIni component:

$C_{?Cmd(Ins)}\, t_C \rightarrow$?Cmd(Ins)

This is a GCM-RPC style here (= no return value) rather than a message oriented style

b) Architecture diagram

a. Build a VCE architecture diagram, for the Car0 component, with its 3 subcomponents:

   i. Only the architecture (components, interfaces, bindings) in this first step.

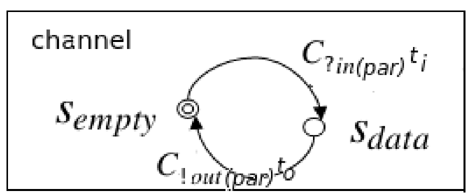
   ii. Respect the interface names.

   iii. Add a service interface accepting messages from the car driver. Name it “Driver”.

b. Check the diagram validity.

c) Channel components

a. Channels here are primitive components with a specific behavior template:



b. Draw a primitive component with interfaces S1 and C1. Build the UML class diagram of these interfaces, and of the implementation class for the method “In” of the service interface S1.

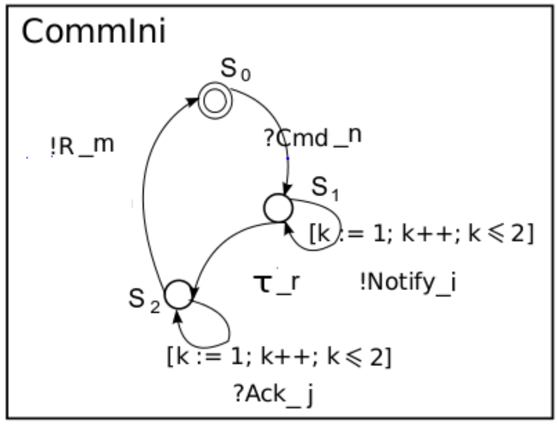
d) Channel behaviors

a. A channel repeatedly receives “In” requests on its service interface. The “In” method receives a parameter, calls the “Out” method on the client interface, then returns.

b. Attach a state-machine specifying the behavior of the service method “In”.

c. create the label of the “C.Out” transition of this machine.

e) CommIni component



This is more complicated…:

1. CommIni has 2 service interfaces (bound from Initial and from ChannelAck).
   - When receiving "Cmd(Ins)" from Initial, it sends a number of "Notify" on client Itf ChannelNotify, then wait.
   - When receiving "Ack(k,r)" from ChannelAck, it stores the corresponding "r_k". We suppose it receives them in order.
   - When all received, it computes the result and sends it on Itf "ToInitial"
2. The way to formalize this in GCM is with 2 service methods, plus a local "body" method describing the (statefull) behavior policy.
3. Build the class diagram for this impl. Class, then the State machines for the service methods and the body.