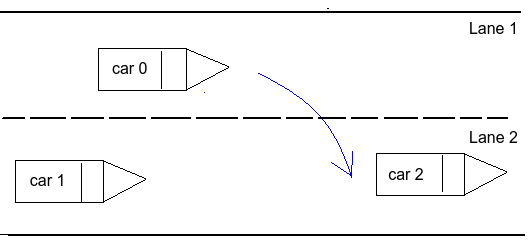
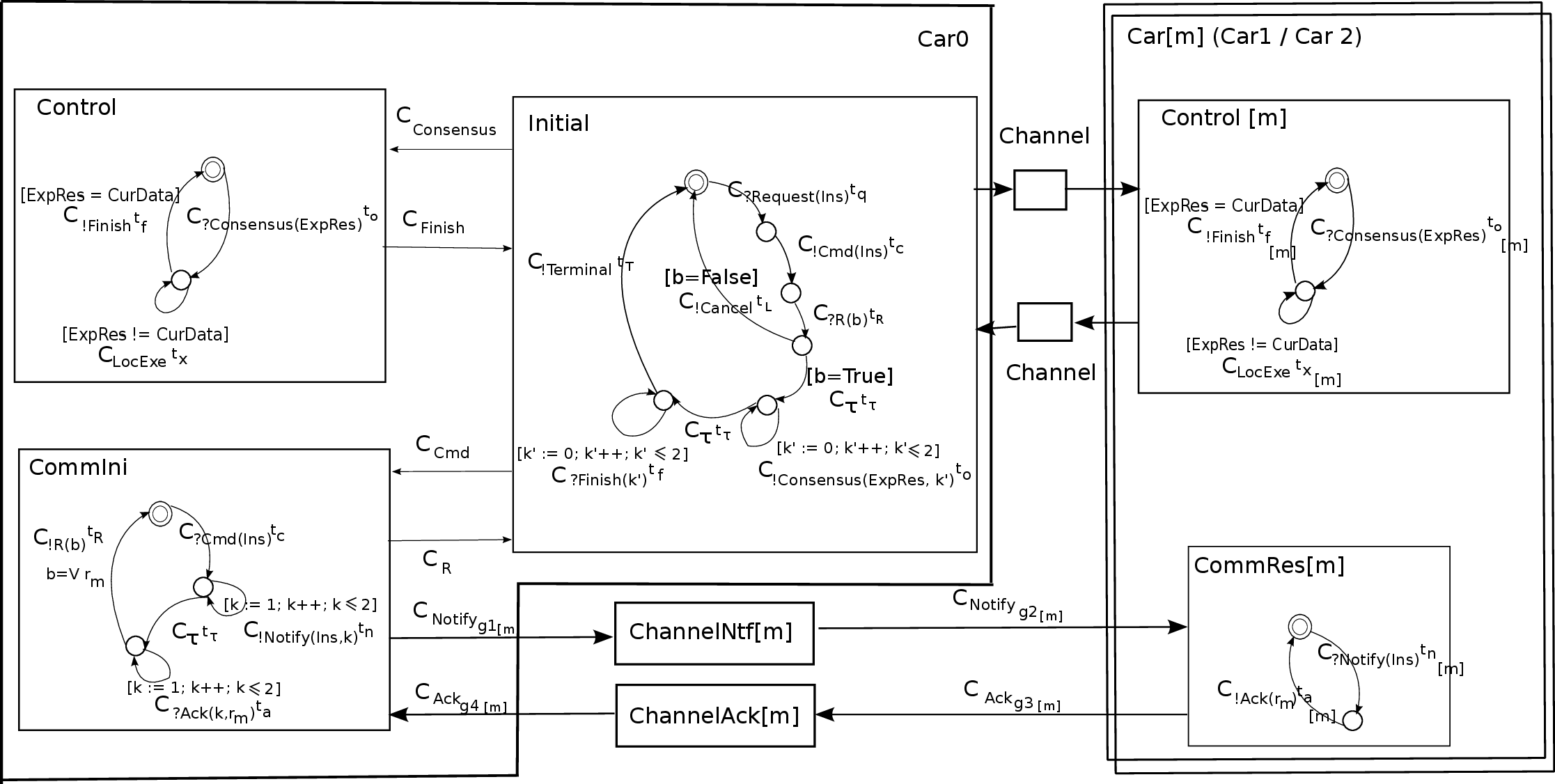
**VCE Hands-on Exercices**

Intelligent Cars use-case



1. System Architecture

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

1. Specific timed model transition language

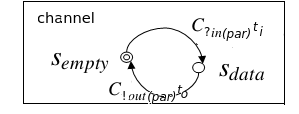
This use-case was originally built for a timed version of the semantic formalism.

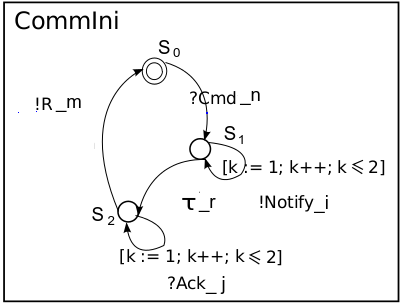
But in the exercises today, only consider the untimed fragment, by dropping the clocks and the time variables.

E.g. in the CommIni component:

C ?Cmd(Ins) t C 🡺 ?Cmd(Ins)

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

1. Exercise 1: Architecture
   1. Build a VCE architecture diagram, for the Car0 component, with its 3 subcomponents:
      1. Only the architecture (components, interfaces, bindings) in this first step.
      2. Respect the interface names.
      3. Add a service interface accepting messages from the car driver. Name it “Driver”.
   2. Check the diagram validity.
2. Exercise 2: Channels
   1. Channels here are primitive components with a specific behavior template:
   2. 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.
3. Exercise 3: Channel behaviors
   1. 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.
   2. Attach a state-machine specifying the behavior of the service method “In”.
   3. Use xtext to create the label of the “C.Out” transition of this machine.
4. Exercise 4: 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.

If time left (otherwise, you can do it back home !):

1. Exercise 5 : Composite, multicast, 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 neighbour

1. Exercise 6 : Validation

Analyse this diagram (semantics, validation rules)

