Formal Modeling for UML/MARTE Concurrency Resources

Authors:

P.Peñil H.Posadas E.Villar

Index

- Motivation
- Design Flow
- UML/MARTE Methodology
- Conclusions
- Future Work

Motivation

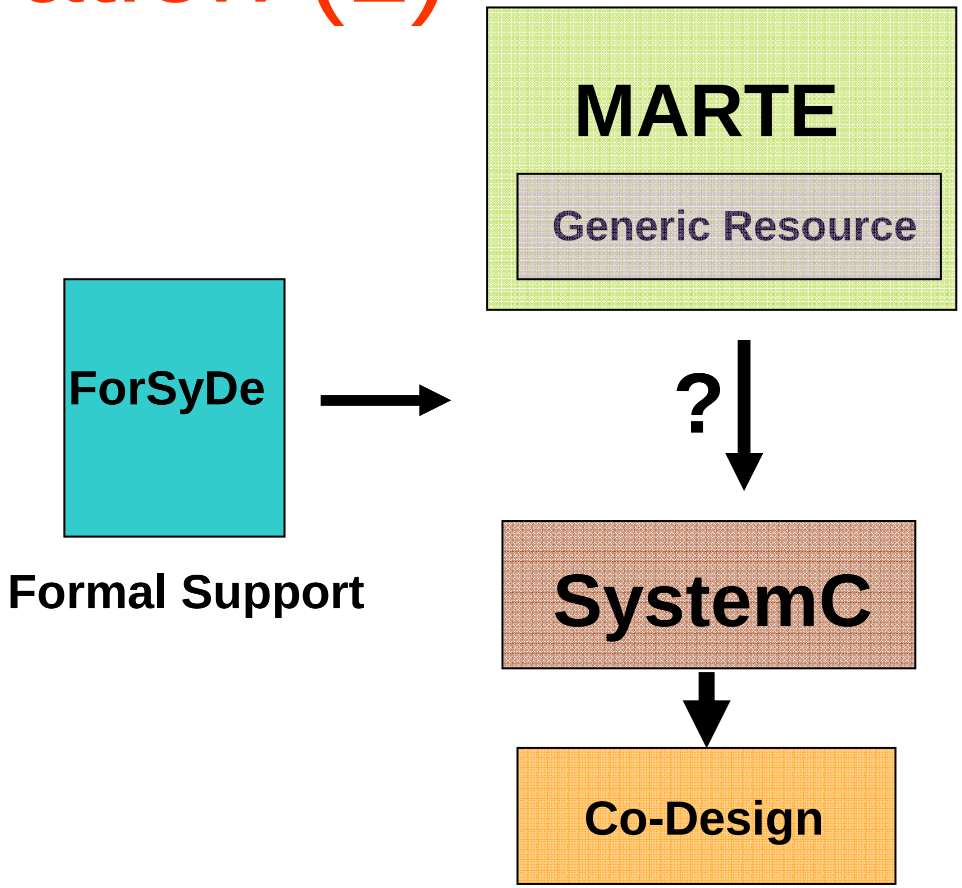
Motivation (1)

MARTE provides semantics to UML

Select a subset of MARTE

Relate UML/MARTE to SystemC

SystemC enables a link to Co-Design



Motivation (2)

- **Massive Concurrency**
  - Data Dependencies
  - Relations
- **Characteristic of the interactions**
  - Formal Semantics
  - Univocal Description
- **Models of Computation & Communication (MoCCs)**
  - Behaviors Semantics Heterogeneity

Contribution

UML/MARTE Models

Structural Modeling

Behavioral Modeling

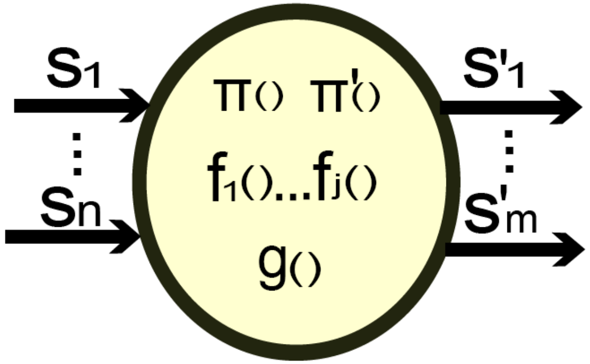
Abstraction

ForSyDe Formal Metamodel

Formal System Design

- ForSyDe formal metamodel
  - *Process*
  - *Signals*
    - *Separation Communication-Computation*
  - *MoCC generic characteristics*
    - Untimed MoCCs
    - Synchronous MoCCs
    - Timed MoCCs

Process ForSyDe



$$p(s_1 \ldots s_n) = s'_1 \ldots s'_m$$

π partition function S_n

π' partition function S'_m

$g()$ next-state function

$f_1() \ldots f_j()$ output functions

Formal Notation

The partitions functions:

$$\pi(v_n, s_n) = \langle a_n(z) \rangle \qquad \pi'(v'_m, s'_m) = \langle a'_m(z) \rangle$$

where $a_n(z)$ is a subsignal of S_n

The function v_n gives the size of $a_n(z)$:

$$v_n(z) = \gamma(\omega_q) \qquad v'_m(z) = length(a'_m(z))$$

$$v_n(0) = length(a_n(0));\; v_n(1) = length(a_n(1))\ldots$$

The outputs are calculated:

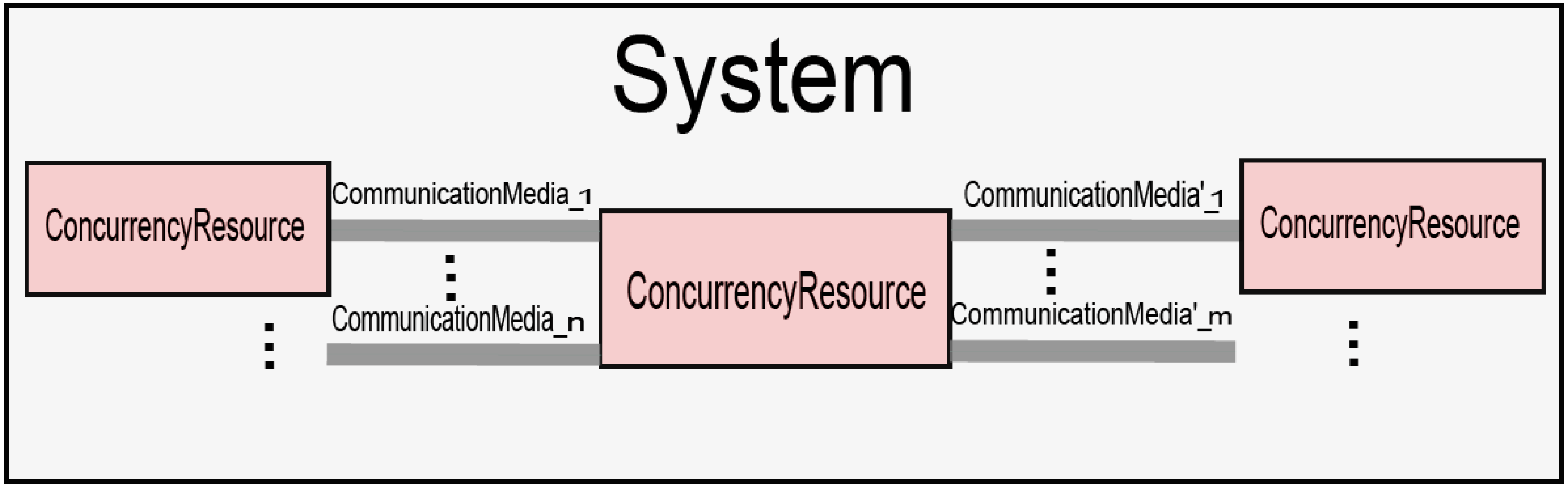
$$f_\alpha((a_1 \ldots a_n), \omega_q) = (a'_1 \ldots a'_m)$$

And the next internal state:

$$g((a_1 \ldots a_n), \omega_q) = \omega_{q+1}$$

UML/MARTE Methodology

Structural Modelling



<<ConcurrencyResource>>: models the concurrent computation

<<CommunicationMedia>>: models the communication

Communication Abstraction

ForSyDe
Signals

abstraction

<<CommunicationMedia>>

No insert any change in the data sequence:

- **No data losses or injections**
- **No data values changes**

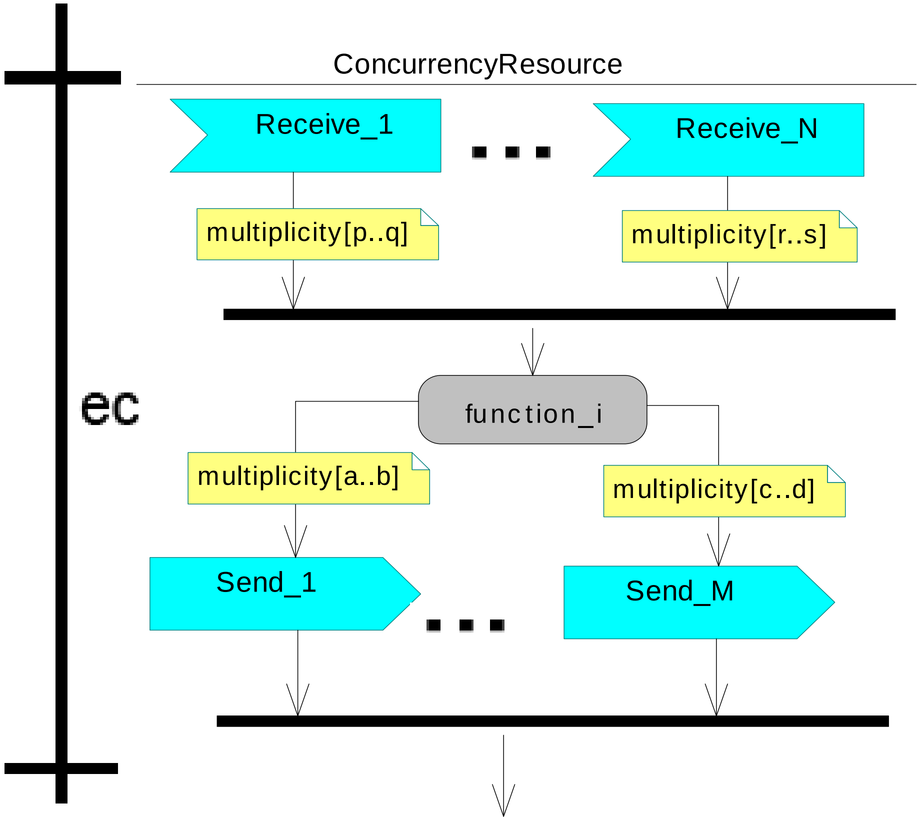
Behavioral Modeling

- Concurrent element described by a Finite State Machine
  - UML Finite State Machines
    - Explicit States
    - Modeling behavior State denoted by the label *do* by an UML Activity Diagram
  - UML Activity Diagram
    - Implicit States as a sequence of actions

Activity Diagram

- In each state
  - A set of inputs are received
  - These inputs are computed
  - The computation results are generated
  - The concurrency resource calculates its new internal state

Activity Diagram(2)



AccepEventActions

$$\omega_j = \begin{cases} P_j \\ D_j \end{cases}$$

SendObjectAction

Activity Diagram(3)

- *AcceptEventAction*
  - Call to a specific method with a specific data-receiving behavior
  - A new datum is available
- *SendObjectAction*
  - Call to a specific method with a specific data-sending behavior
  - A new datum is generated

Computation Abstraction

- The data received; $a_1 \cdots\ a_N$ ForSyDe subsignals
- The data send; $a'_1 \ldots a'_N$ ForSyDe subsignals
- The *function_i* action; f_α ForSyDe function
- The Concurrency Resource State:
  - *Pj;* segments of the behavioural modelling between two waiting states
  - *Dj;* internal values that characterizes the state
- *{Pj, Dj};* ω_j ForSyDe state
- The function *g()* calculates the new {Pj+1, Dj+1}

Computation Abstraction(2)

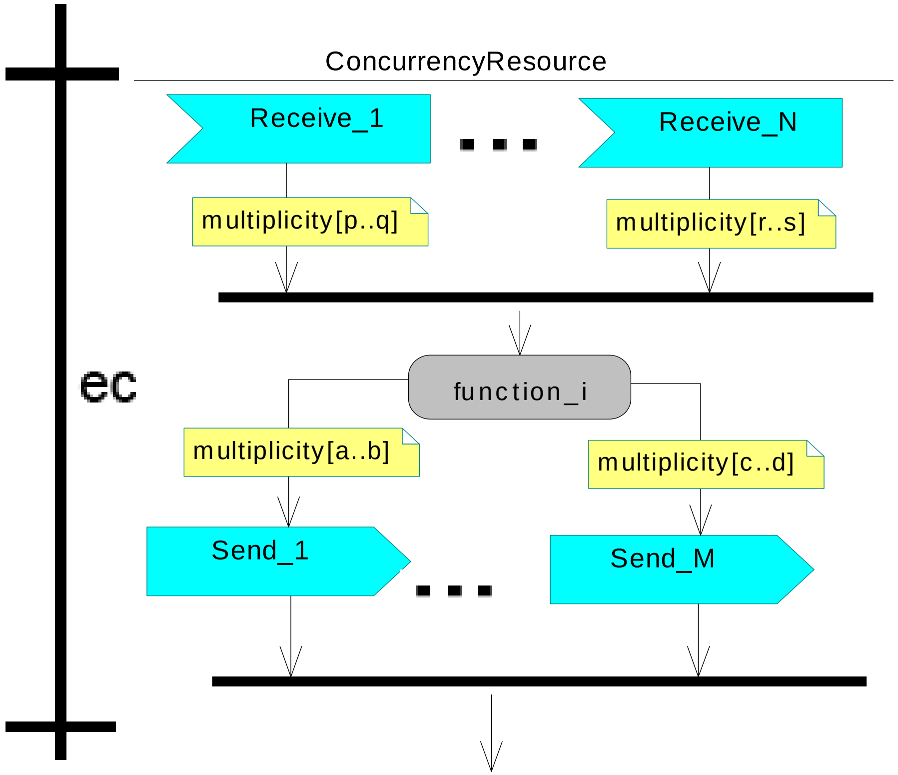
- The multiplicity values are abstracted as:

$$\nu_1(z) = \gamma(\omega_j) = \begin{cases} p \\ \dots \\ q \end{cases} \qquad \nu_n(z) = \gamma(\omega_j) = \begin{cases} r \\ \dots \\ s \end{cases}$$

$$lenght(f_j(a_1 \dots a_N), \omega_j) = \begin{cases} \nu'_1(z) = length(a'_1) \begin{cases} a \\ \dots \\ b \end{cases} \\ \dots \\ \nu'_M(z) = length(a'_M) \begin{cases} c \\ \dots \\ d \end{cases} \end{cases}$$

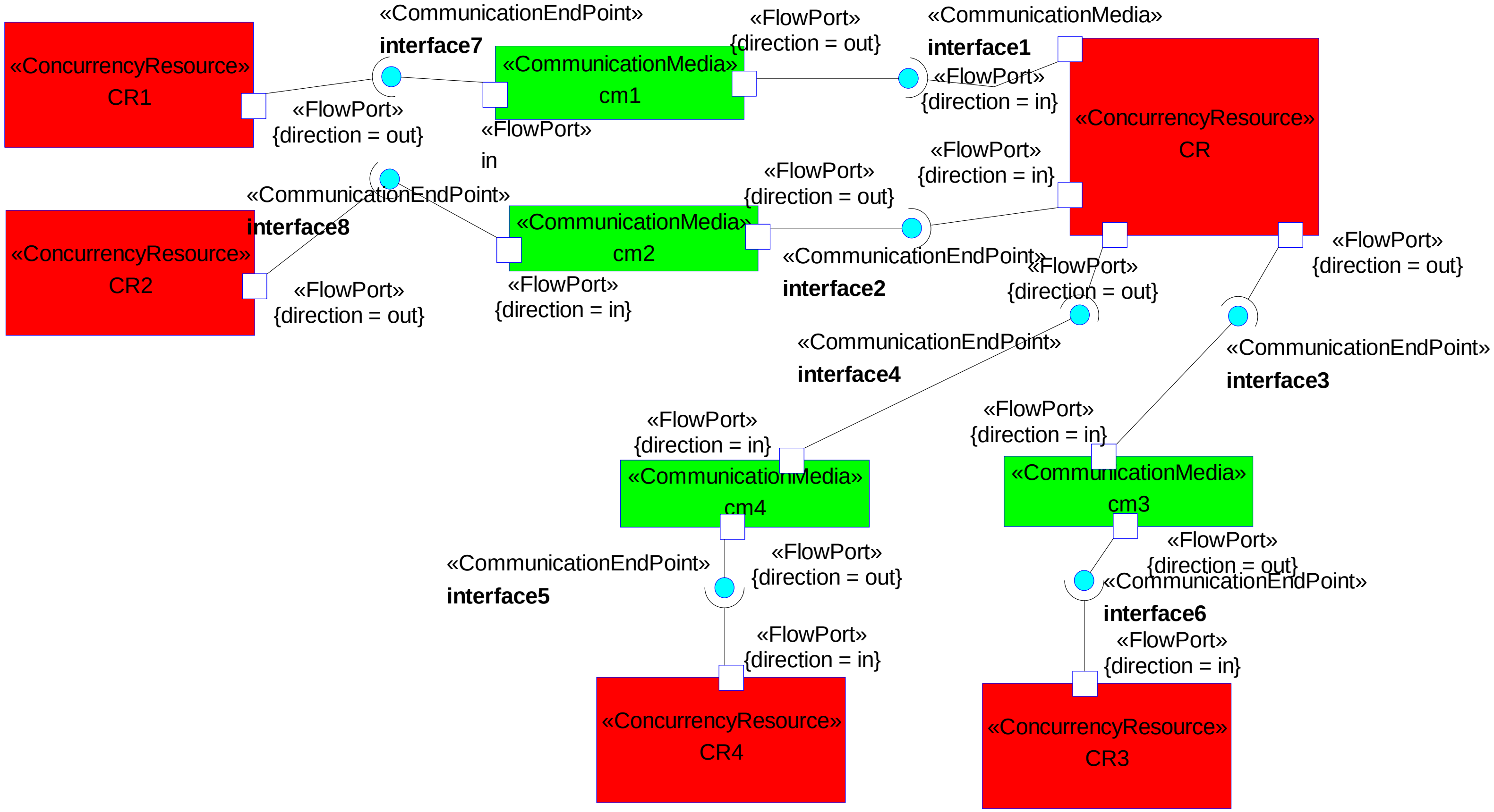
Computation Abstraction(3)

- The advance of time is a totally ordered set of *evaluation cycles* (**ec**).
- In each **ec** *“a process consumes inputs, computes its new internal state, and emits outputs”.*
- The interpretation of a **ec** depends on the time domain capture in the models.

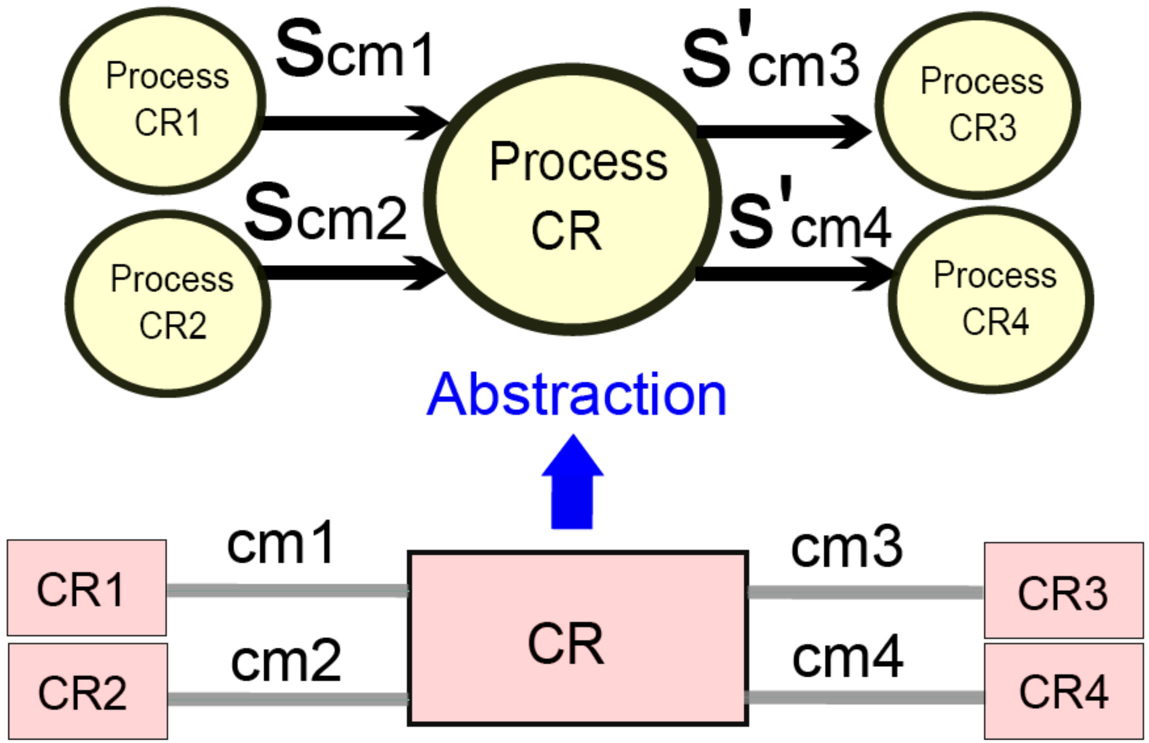


Example

Example(1)

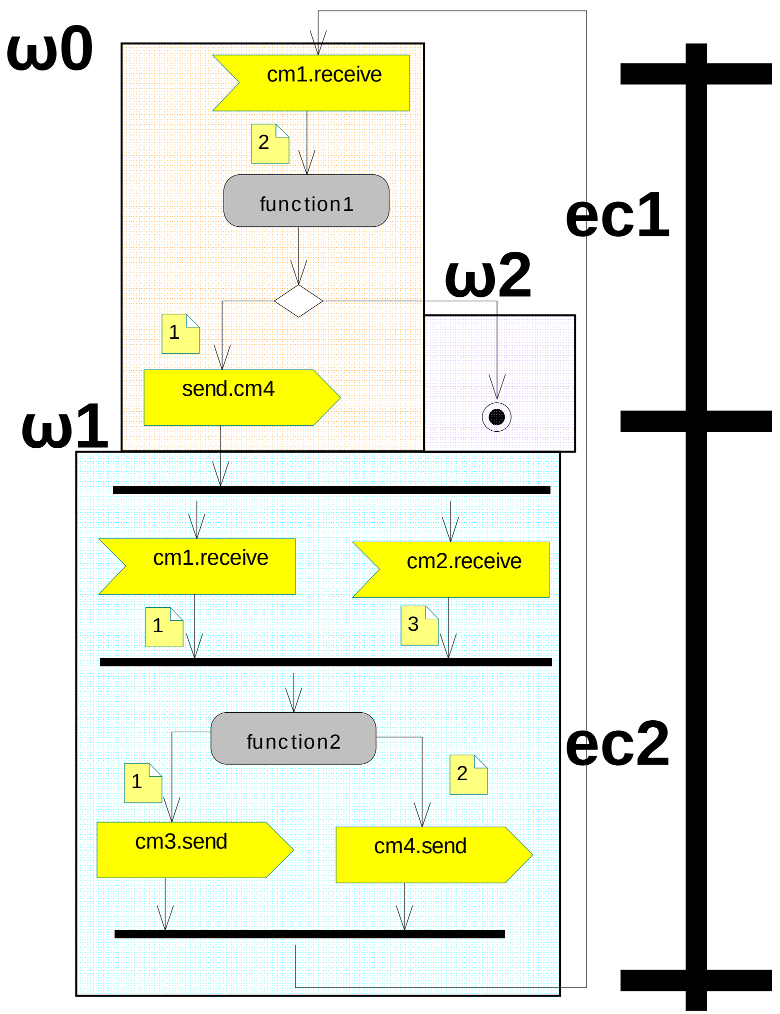

«ConcurrencyResource»
CR1
«CommunicationEndPoint»
interface7
«FlowPort»
{direction = out}
«CommunicationMedia»
cm1
«FlowPort»
in
«FlowPort»
{direction = out}
«CommunicationMedia»
interface1
«FlowPort»
{direction = in}
«ConcurrencyResource»
CR
«FlowPort»
{direction = in}
«CommunicationEndPoint»
interface8
«ConcurrencyResource»
CR2
«FlowPort»
{direction = out}
«CommunicationMedia»
cm2
«FlowPort»
{direction = in}
«FlowPort»
{direction = out}
«CommunicationEndPoint»
interface2
«FlowPort»
{direction = out}
«FlowPort»
{direction = out}
«CommunicationEndPoint»
interface4
«CommunicationEndPoint»
interface3
«FlowPort»
{direction = in}
«CommunicationMedia»
cm4
«FlowPort»
{direction = in}
«CommunicationMedia»
cm3
«CommunicationEndPoint»
interface5
«FlowPort»
{direction = out}
«FlowPort»
{direction = out}
«CommunicationEndPoint»
interface6
«FlowPort»
{direction = in}
«FlowPort»
{direction = in}
«ConcurrencyResource»
CR4
«ConcurrencyResource»
CR3

Example(2)



Process
CR1
Process
CR2
Scm1
Scm2
Process
CR
S'cm3
S'cm4
Process
CR3
Process
CR4
Abstraction
CR1
CR2
cm1
cm2
CR
cm3
cm4
CR3
CR4

CR

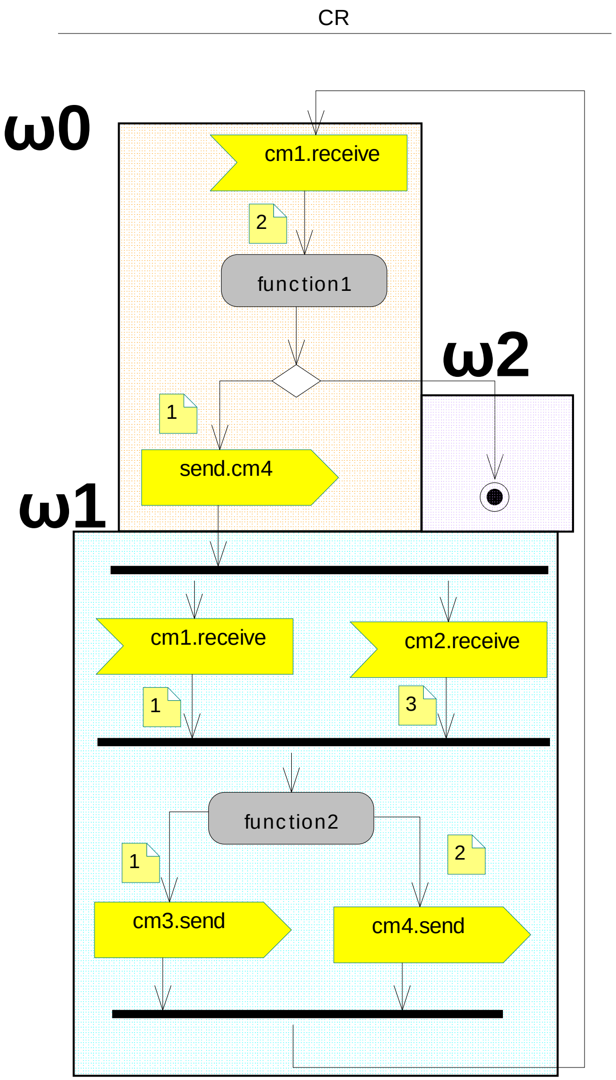


data inputs $a_{cm1}(z), a_{cm2}(z)$

data outputs $a'_{cm3}(z), a'_{cm4}(z)$

$$g(\omega_0, a_{cm1}) = \begin{cases} \omega_1 \\ \omega_2 \end{cases}$$

$$g(\omega_1, (a_{cm1}, a_{cm2})) = \omega_0$$



Example(4)

In the state ω_0:

$$v_{a_{cm1}}(0) = \gamma(\omega_0) = 2$$

$$v_{a_{cm4}}(0) = length(function1(a_{cm1}(0), \omega_0)) =$$

$$= lenght(a'_{cm4}(0)) = 1$$

In the state ω_1:

$$v_{a_{cm1}}(1) = \gamma(\omega_1) = 1$$

$$v_{a_{cm2}}(0) = \gamma(\omega_1) = 3$$

$$v_{a_{cm3}}(0) = length(function2((a_{cm1}(1), a_{cm2}(0)), \omega_1)) =$$

$$= lenght(a'_{cm3}(0)) = 1$$

$$v_{a_{cm4}}(1) = length(function2((a_{cm1}(1), a_{cm2}(0)), \omega_1)) =$$

$$= lenght(a'_{cm4}(1)) = 2$$

Conclusions

- The need of a formalism to UML/MARTE models that supports the generation of executable specifications (SystemC)
- ForSyDe can provides this formal support