MDE for (large scale) safe distributed systems

Eric Madelaine
eric.madelaine@inria.fr

INRIA Sophia-Antipolis
University of Nice Sophia Antipolis

SCALE team

Context & people

SCALE research team at INRIA Sophia-Antipolis

https://team.inria.fr/scale/

4 years of collaboration with SEI@ECNU:

- Research (distributed systems, semantics, formal methods)
- Common PhD
- Master internships

Internship (master / PhD) in France – INRIA Sophia-Antipolis
contact me: eric.madelaine@inria.fr
or: mzhang@sei.ecnu.edu.cn; hbzhu@sei.ecnu.edu.cn;

Also:

International Master track “Ubiquitous Systems & Networks”
at University of Nice Sophia-Antipolis.

MDE for safe distributed systems

Thursday, July 16th , Afternoon(Course)

☐1) Introduction: academic and industrial examples: Distributed algorithms; distributed resiliant industrial infrastructure; QoS-aware elastic architecture for Cloud applications

☐2) GCM: a component-based programming model for safe distributed applications

☐3) VCE: a model-driven specification environment for GCM

Friday, July 17th , Morning: (Hands-on, Lab)

- Presentation and exercises with VCE:

☐ Tutorial and simple examples

Friday, July 17th , Afternoon: (Lab)

☐1) **Course**: VerCors theoretical background : behavioural models, model-checking, temporal logics, running the verification tools.

☐2) **Hands-on** with VCE and Vercors: continued…

Goals of the Course

1. Explore some features of component-based software
   1. Fractal/GCM component model
   2. GCM architecture and execution principles
2. VerCors: A software platform for (model-driven) specification and verification of GCM applications
3. Understand a non-trivial case-study in (model-checking) behavior verification

Lab sessions

1. Specification of distributed component architecture and behaviour with the VCE tools.

AGENDA (part 1)

1) Context and Motivation:
   - A short introduction to (safe) component-based applications
   - academic and industrial examples:
     - Distributed algorithms;
     - distributed resiliant industrial infrastructure;
     - QoS-aware elastic architecture for Cloud applications
2) Formal Methods and Component models:
   - Vocabulary and context:
     - specification, modeling, testing, verification…: Formal methods in the design flow of distributed/embedded systems
   - Fractal/GCM: a component-based programming model for safe distributed applications
3) VCE (VerCors Component Editor): a model-driven specification environment for GCM.

Distributed Components…

Specification and Verification for distributed applications

Components

Interfaces

Services

- **Distributed Systems**

Message passing

Grids

Clouds

MultiCores

Adaptation

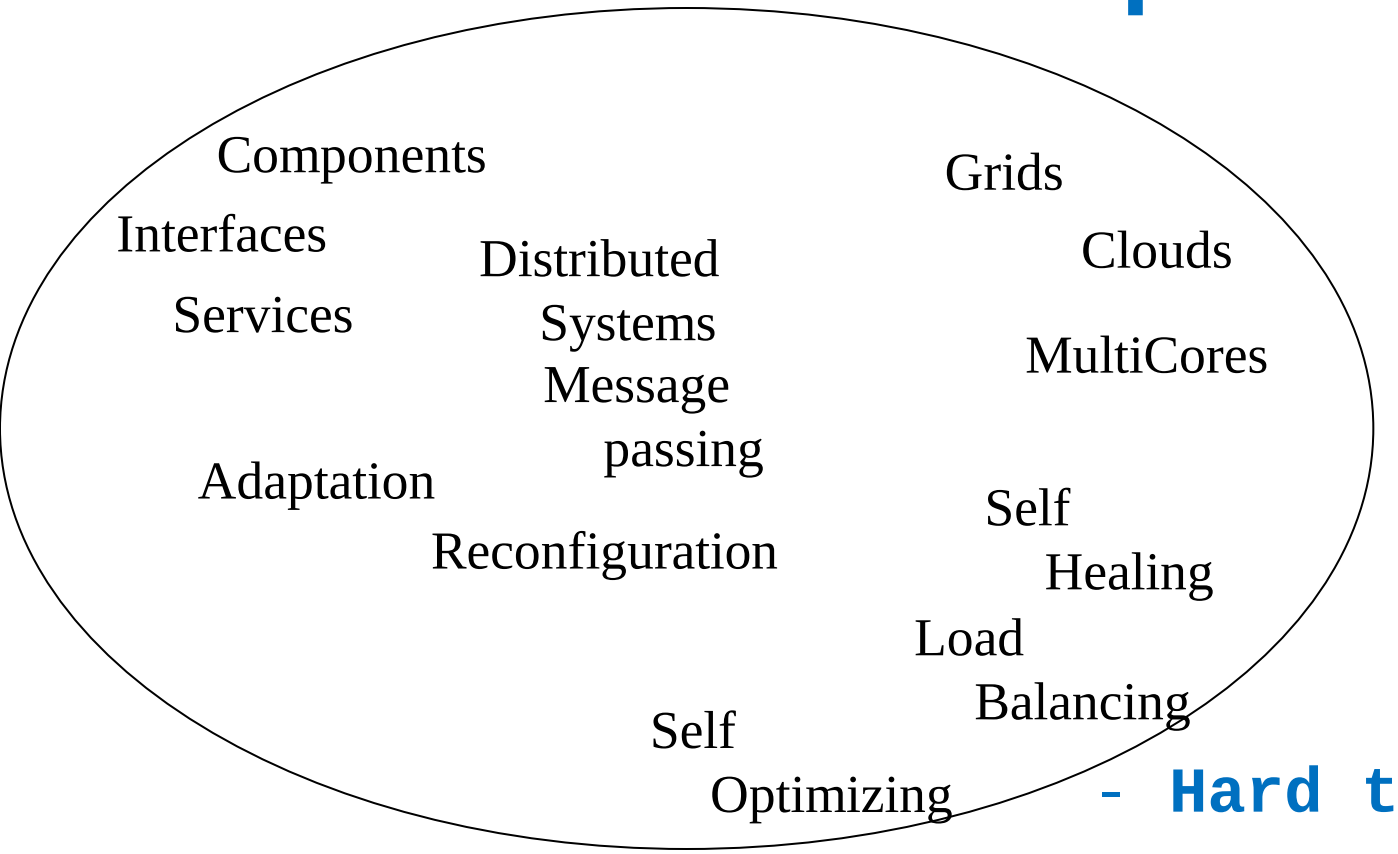
Reconfiguration

Self Healing

Load Balancing

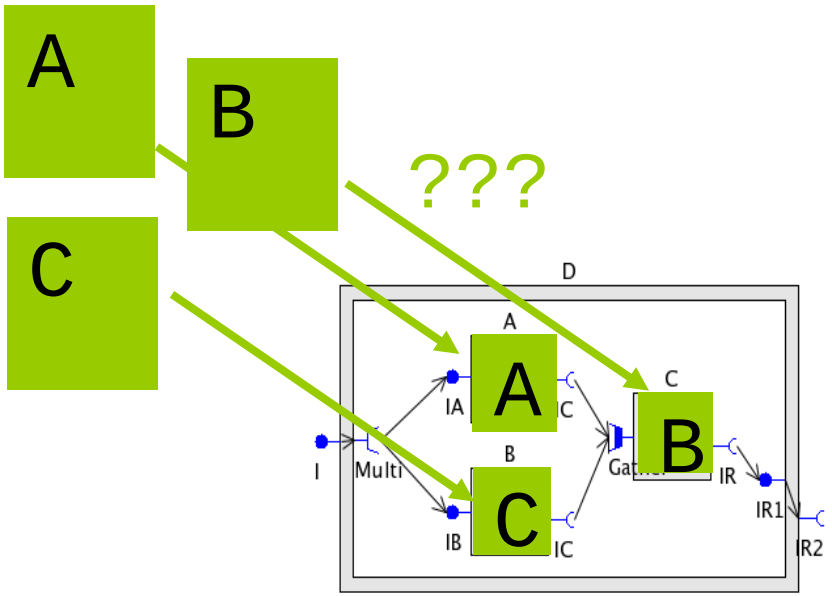
Self Optimizing

Distributed Components…

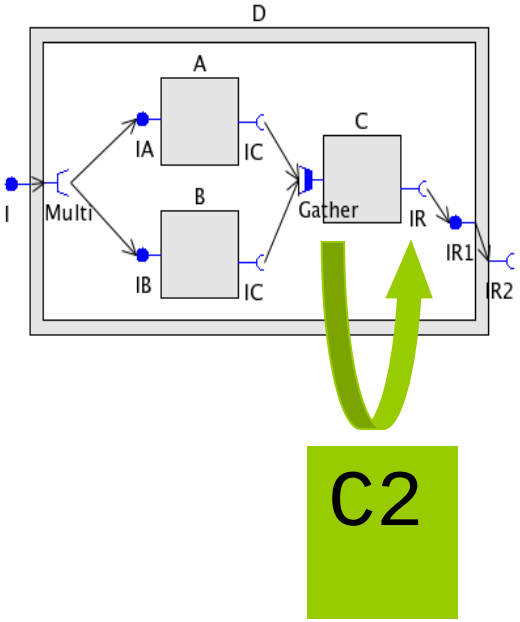


- **Hard to program ?**
- **Correct Behaviour ?**
- **Correct Assembly ?**

Do we need formal methods for developing component-based software ?

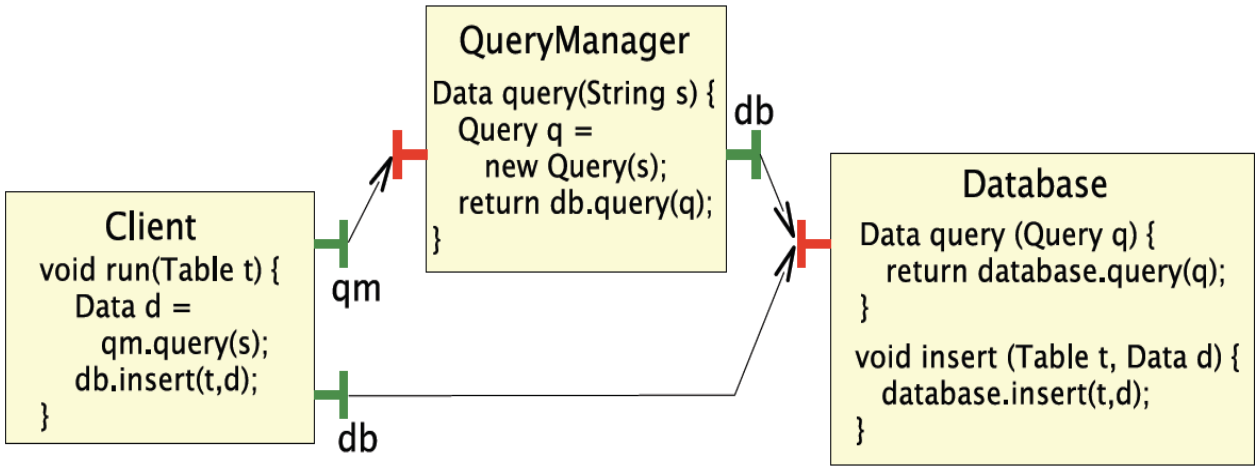


Safe COTS-based development
=>
Behaviour Specifications



Safe management for
complex systems
(e.g. replacement at runtime)

Is it more difficult for distributed asynchronous components ?



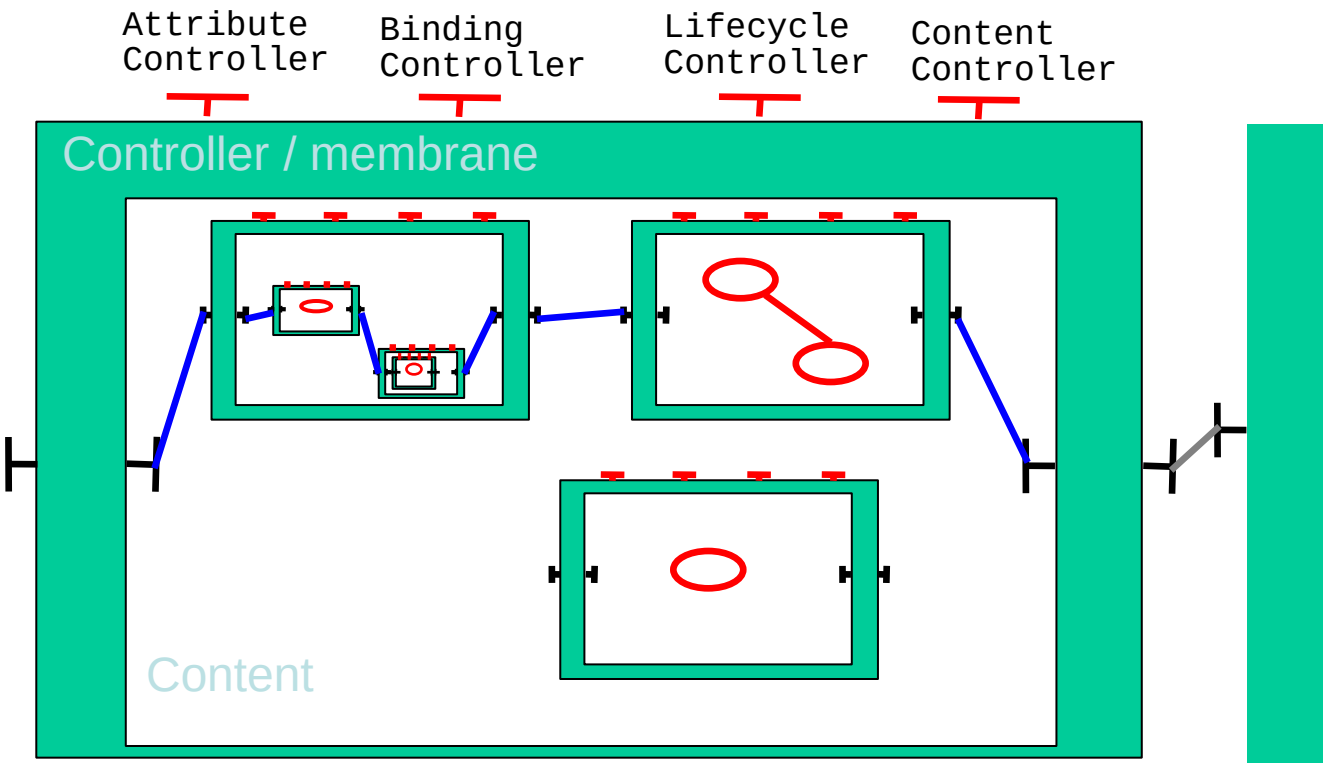
- Yes !
- Asynchrony creates race-conditions, dead-locks, etc.

AGENDA (part 1)

1) Context and Motivation:
   - A short introduction to (safe) component-based applications
   - academic and industrial examples:
     - Distributed algorithms;
     - distributed resiliant industrial infrastructure;
     - QoS-aware elastic architecture for Cloud applications
2) Formal Methods and Component models:
   - Vocabulary and context:
     - specification, modeling, testing, verification…: Formal methods in the design flow of distributed/embedded systems
   - Fractal/GCM: a component-based programming model for safe distributed applications
3) VCE (VerCors Component Editor): a model-driven specification environment for GCM.

The Fractal hierarchical model

- Server/Client Interfaces
- Hierarchy
- Separation of concern: functional / non-functional
- ADL
- Extensible



composites encapsulate primitives, primitives encapsulate code

The Grid Component Model (GCM):

A Grid&Cloud aware extension to Fractal

- Targetting Grids/Clouds requires to handle:
  - Scalability => hierarchy, parallelism
  - Volatility, heterogeneity => adaptation, dynamicity, autonomicity…
- Collective interfaces
  - Multicast, gathercast, gather-multicast, MxN parallel communications

GCM for parallel computing

- Non-functional concerns: Componentized membrane
  - Controllers as objects or Fractal/GCM components
  - Fractal extension for properly exposing the non-functional part, including non-functional client interfaces

GCM for autonomic computing

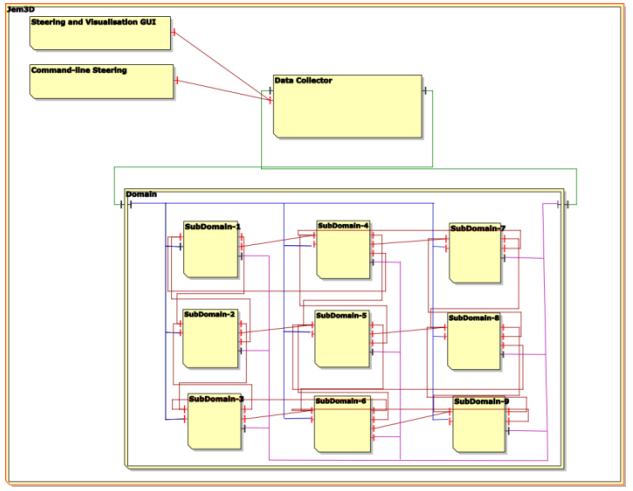
GCM for Parallel Computing

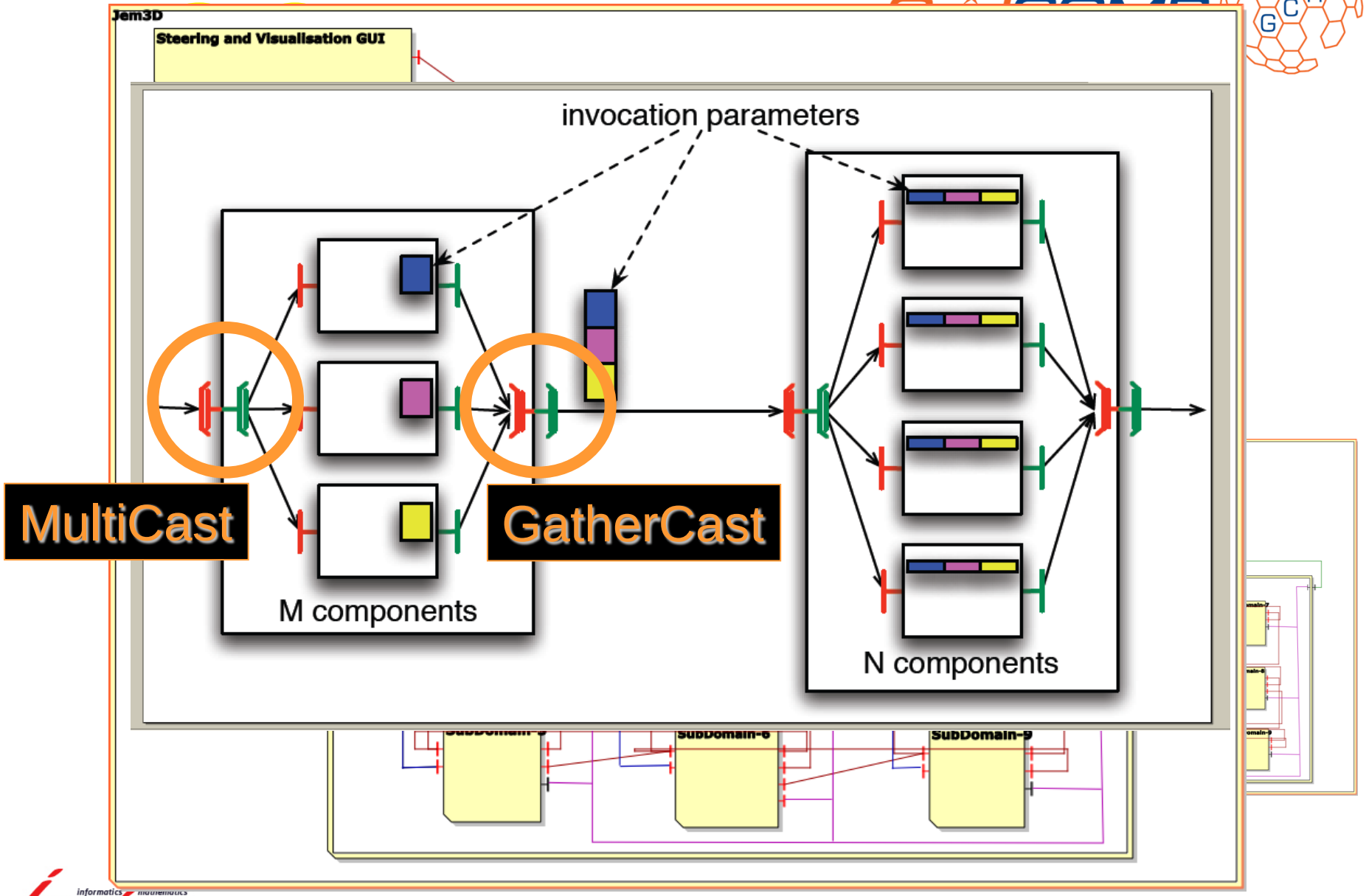
Scopes and Objectives:

- Cloud Applications/Services that Compose and Deploy
- No programming, No Scripting, …

Innovation:

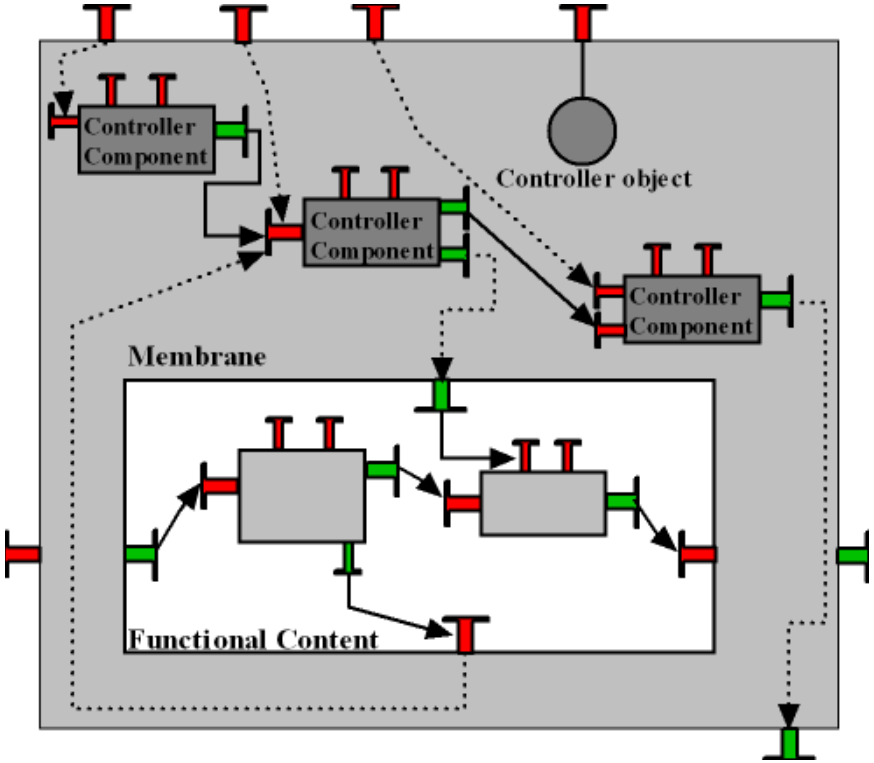
- Abstract Deployment
- Composite Components
- Multicast and GatherCast




Jem3D
Steering and Visualisation GUI
invocation parameters
MultiCast
GatherCast
M components
N components
SubDomain-6
SubDomain-9

GCM for autonomic computing

- Dynamic to Autonomic component-based system reconfiguration
  - Architecture of GCM membranes
  - How to plug autonomous strategies to drive all non-functional concerns



AGENDA (part 1)

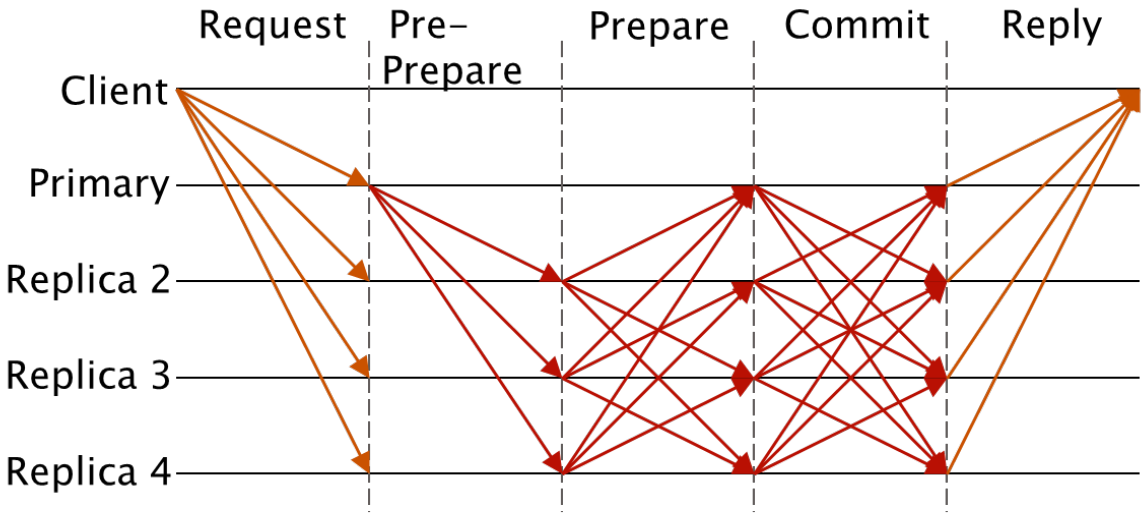
1) Context and Motivation:
   - A short introduction to (safe) component-based applications
   - academic and industrial examples:
     - Distributed algorithms;
     - distributed resiliant industrial infrastructure;
     - QoS-aware elastic architecture for Cloud applications
2) Formal Methods and Component models:
   - Vocabulary and context:
     - specification, modeling, testing, verification…: Formal methods in the design flow of distributed/embedded systems
   - Fractal/GCM: a component-based programming model for safe distributed applications
3) VCE (VerCors Component Editor): a model-driven specification environment for GCM.

Example of distributed systems/algorithms

- Distributed Algorithms
  - BFT : A Fault tolerant algorithm
- Distributed Systems in Industry
  - Reconfigurable large/safe RFID management system
- Distributed Services in the Cloud
  - Architecture for QoS-aware services

Byzantine Fault Tolerant Systems

- **Byzantine systems:**
- “bad” guys can have any possible behaviour,
- everybody can turn bad, but only up to a fixed % of the processes.

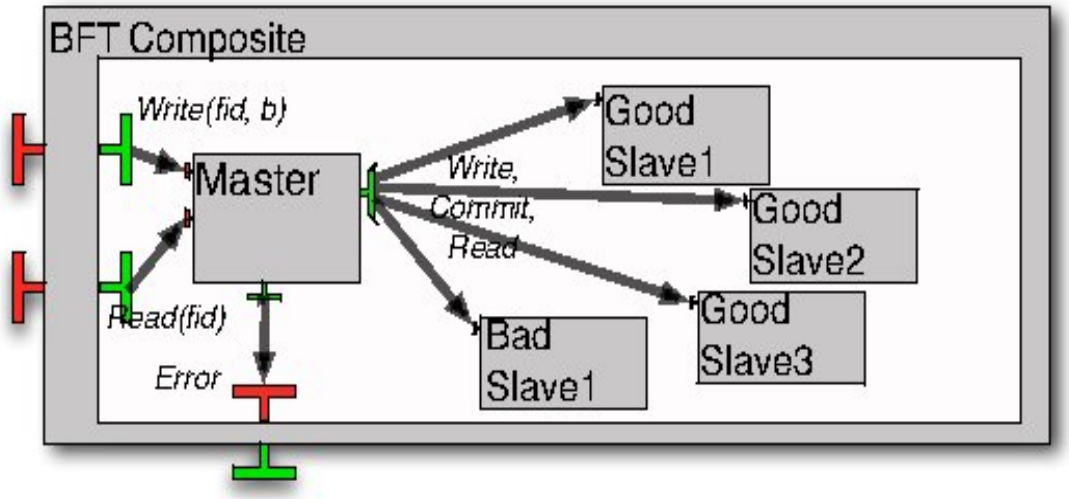


- **Very large Bibliography:**
- Algorithms
- Correctness proofs

Modelling a BFT application in GCM

- Constant « **f** » : number of faults the system can accomodate.
- 1 composite component with 2 external services Read/Write.
- The service requests are delegated to the Master.

VCE diagram



- **1** multicast interface sending write/read/commit requests to all slaves.
- the salves reply asynchronously, the master only needs 2f+1 coherent answers to terminate

Modelling a BFT application in GCM

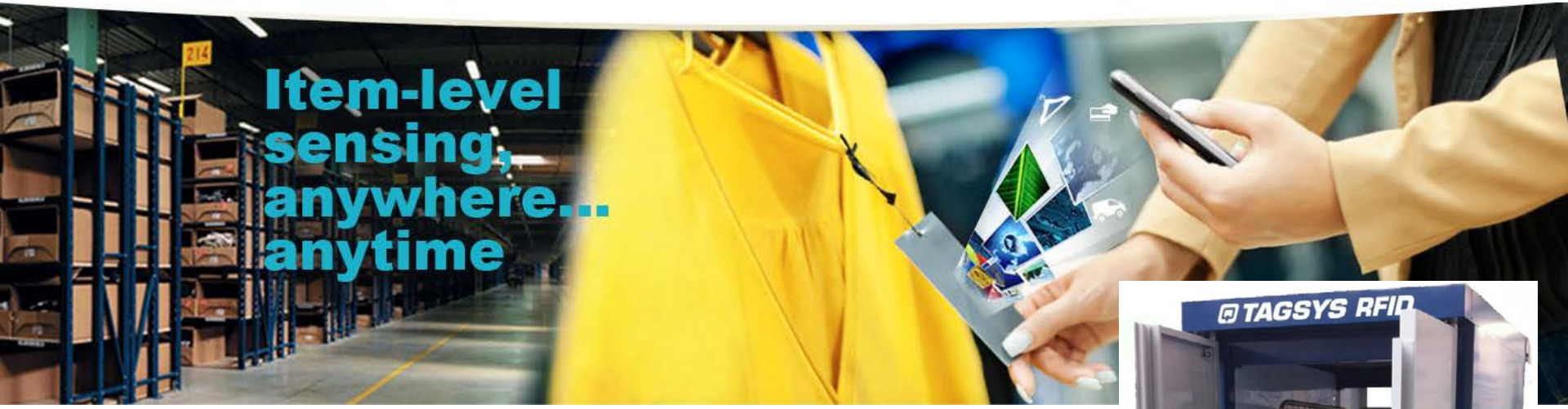
Challenges:

- Analysis of asynchronous group communication:
  complexity of the interleaving + mastering very large state-spaces

- Correctness = termination of service invocation + functional correctness:

  expression of logical properties / temporal logics

Industrial use-case: Rfid management systems



Collaborative project between INRIA and several academic & industrial partners,
led by TAGSYS-RFID (french SME)

http://www.spinnaker-rfid.com/

Targets widespread adoption of RFID-based systems
Mainly for retail and healthcare sectors

RFID Management systems

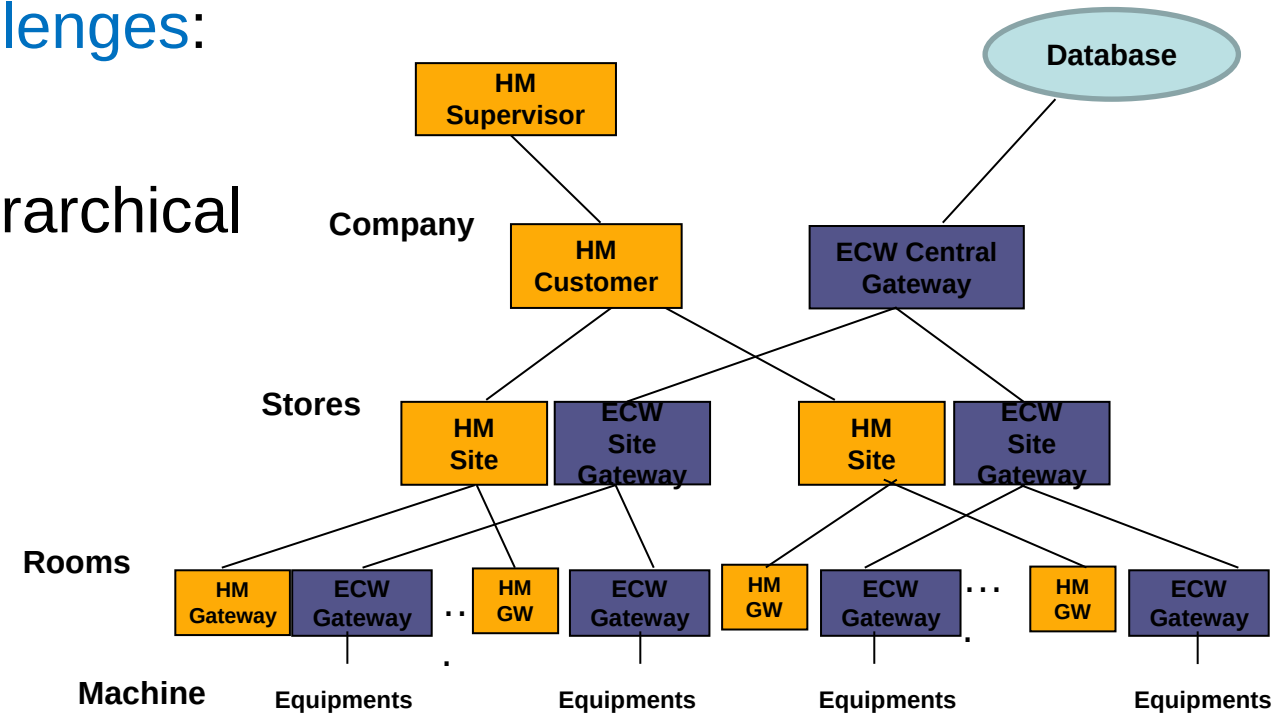
Inventory in stores, deliveries, laundries…

Location of people, objects, etc…

- ***Individual identification of large numbers of tags***
- ***Large geographically distributed infrastructure***
- ***Large flow of data, with safety guarantees***
- ***Dynamic topology.***

RFID Management systems

- Many technological challenges (radiowaves, tags, antennas, energy, …)
- Software challenges:
  - ❖ Large / hierarchical
  - ❖ Dynamic
  - ❖ Safety



RFID Management systems

- Existing infrastructure:
  - Distributed component-based software

- Questions:
  - ❖ How to scale up (from dozens to hundreds of sites)?
  - ❖ How to deploy automatically ?
  - ❖ How to reconfigure without stopping the whole service?
  - ❖ Traceability guarantees for the event storage database.

N. Gaspar, L. Henrio, E. Madelaine,
« Formally Reasoning on a Reconfigurable Component-Based System --- A Case Study for the Industrial World »,
FACS'13, Nanchang, China, oct. 2013.

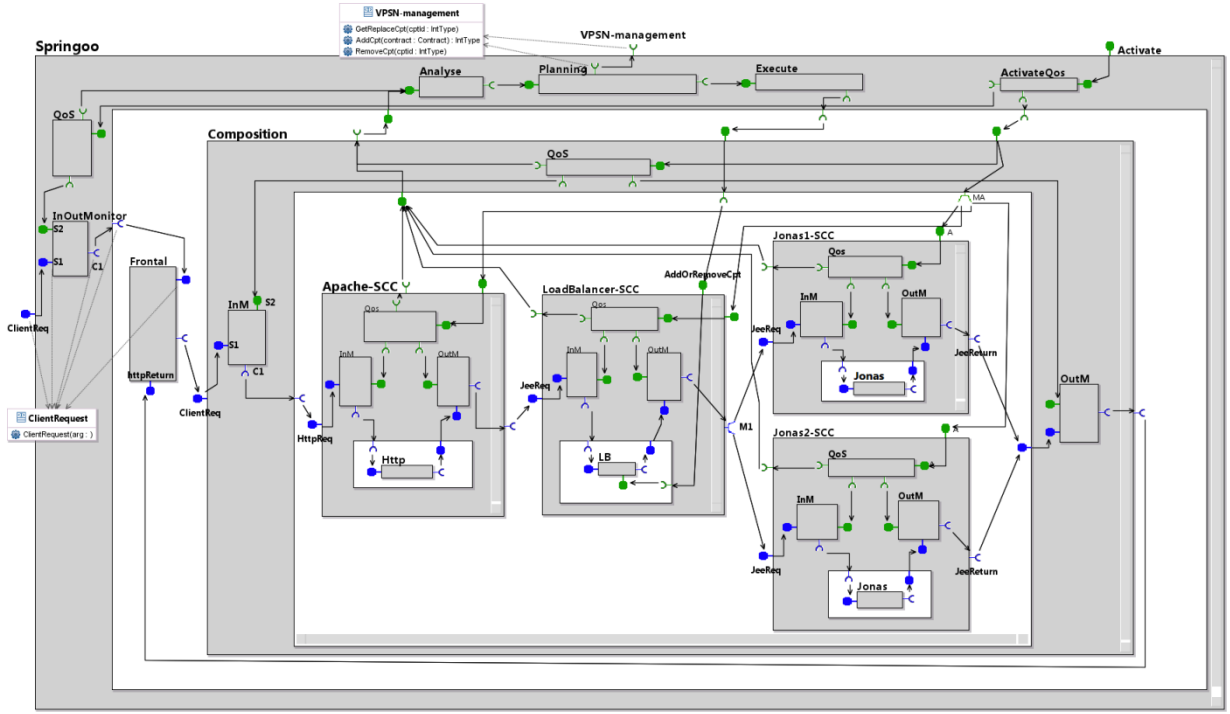
Providing Quality of Service (QoS) for complex Cloud services

- Service-based large applications on the Cloud
- SLA = contract between user and provider
- Shared resources

Predict / monitor performances ?

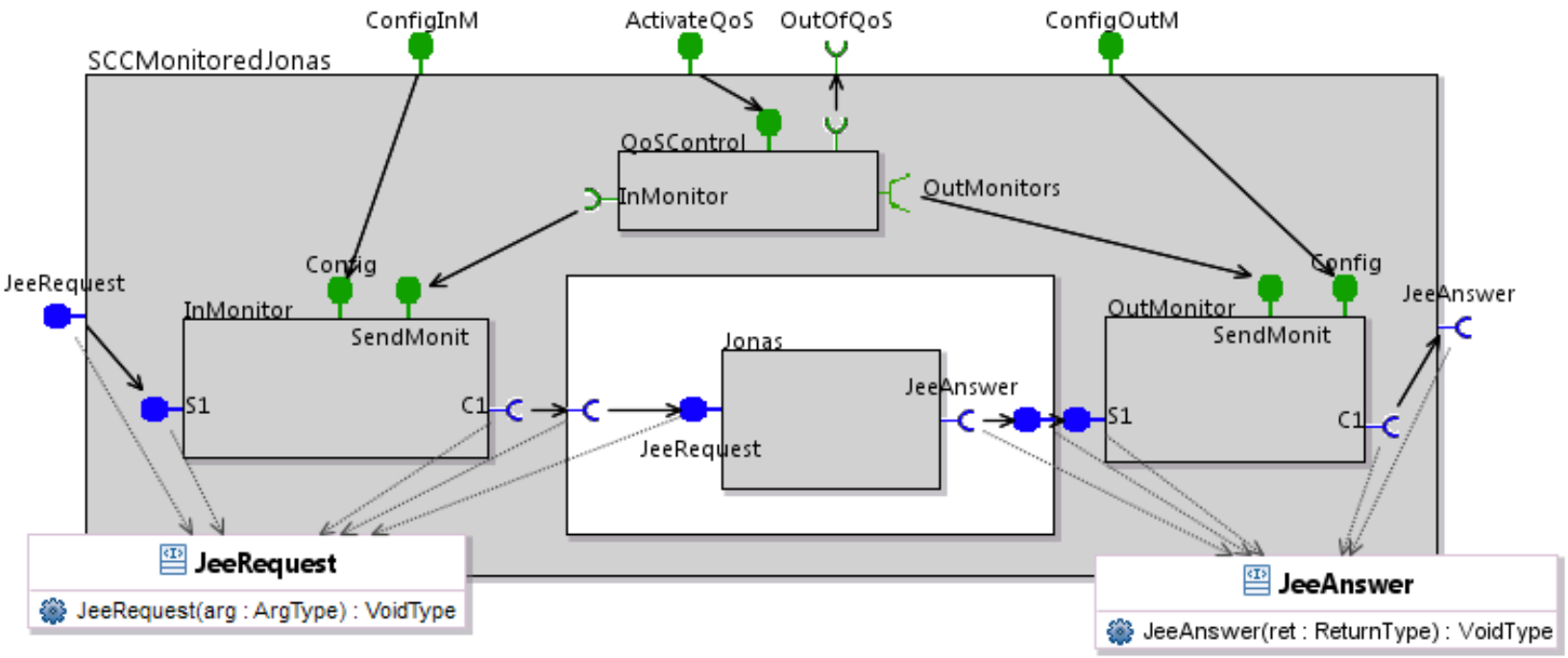
Adapt configuration and resources to changing demands or environment

MDE helping the application architect

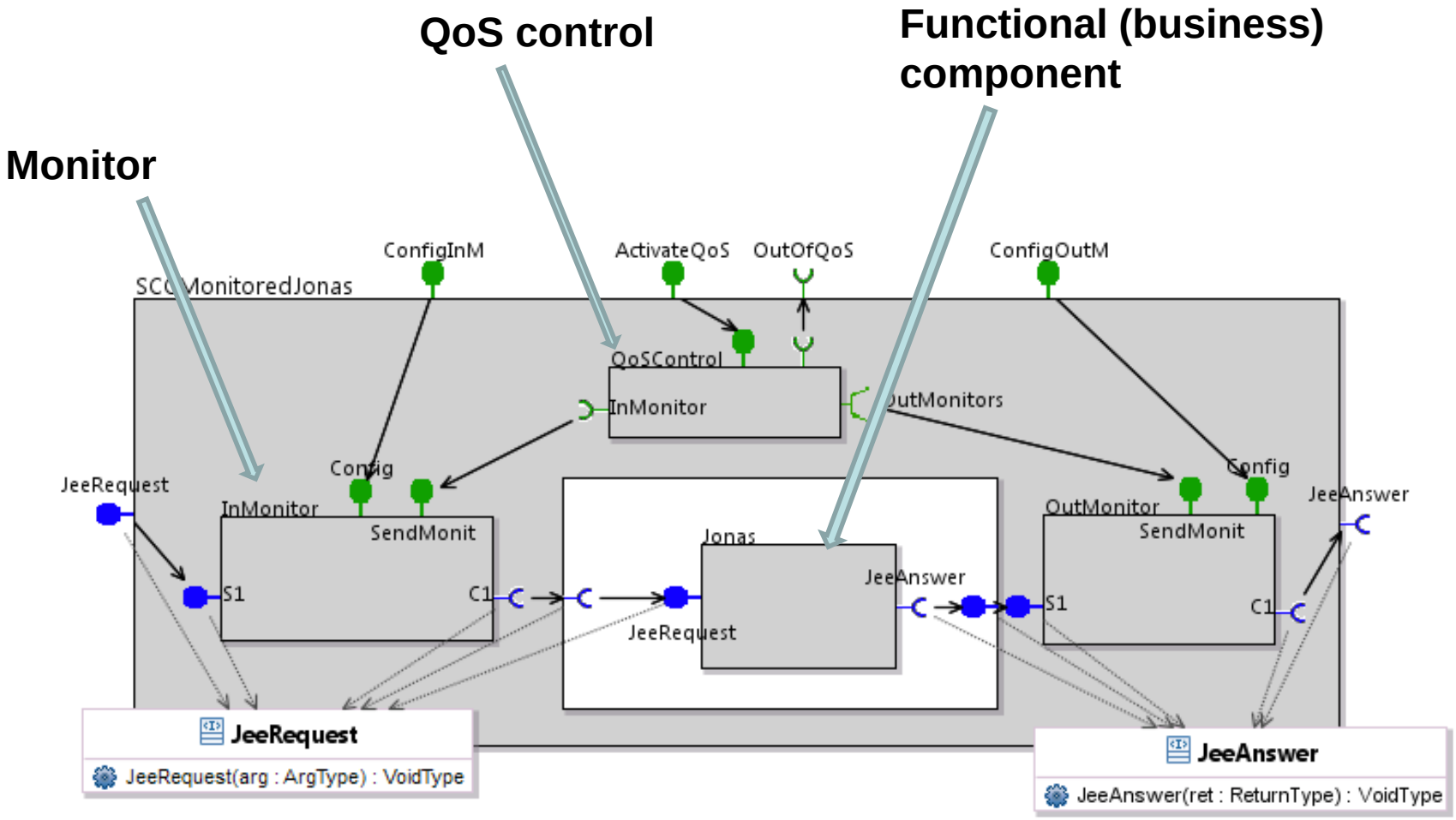


Component Template for QoS

- Context: french OpenCloudware project (leader Orange R&D)



Service Controlled-Component (SCC)



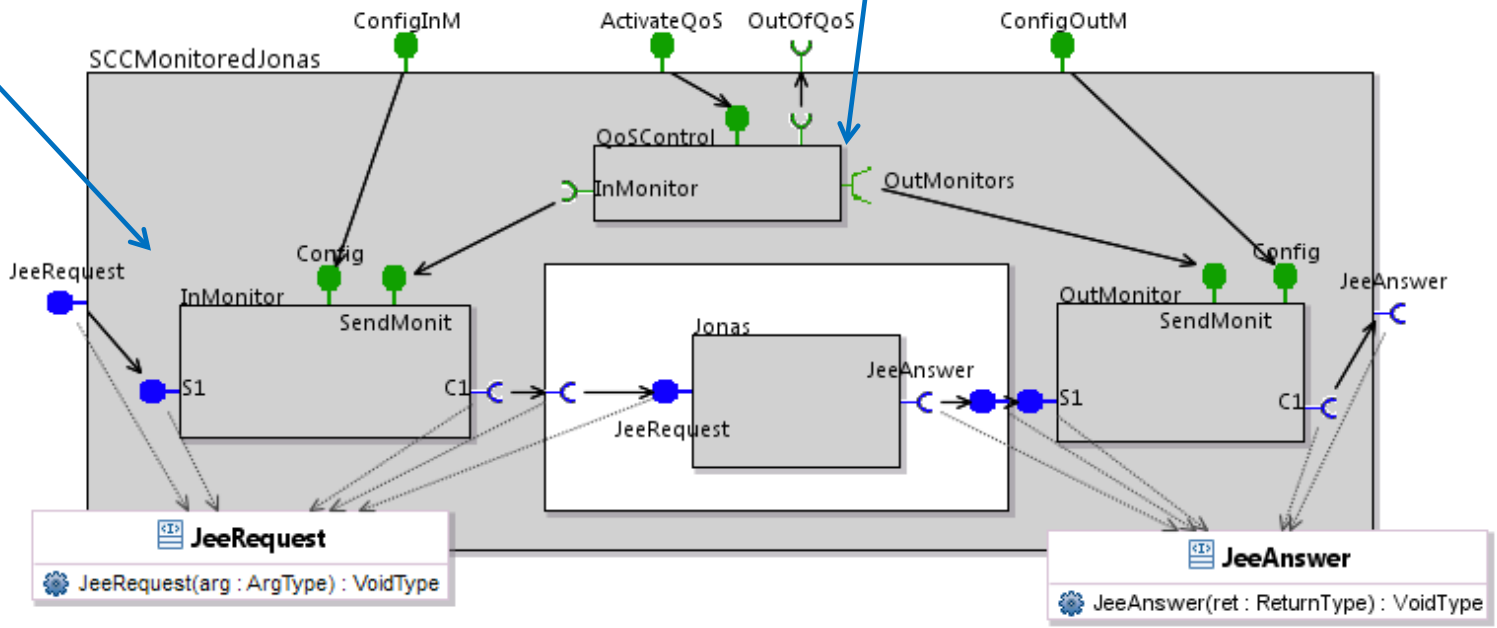
Generic SCC Template

Activation and configuration interfaces

QoS-Control queries the monitors, and signals In-contract / Out-contract on the NF interface

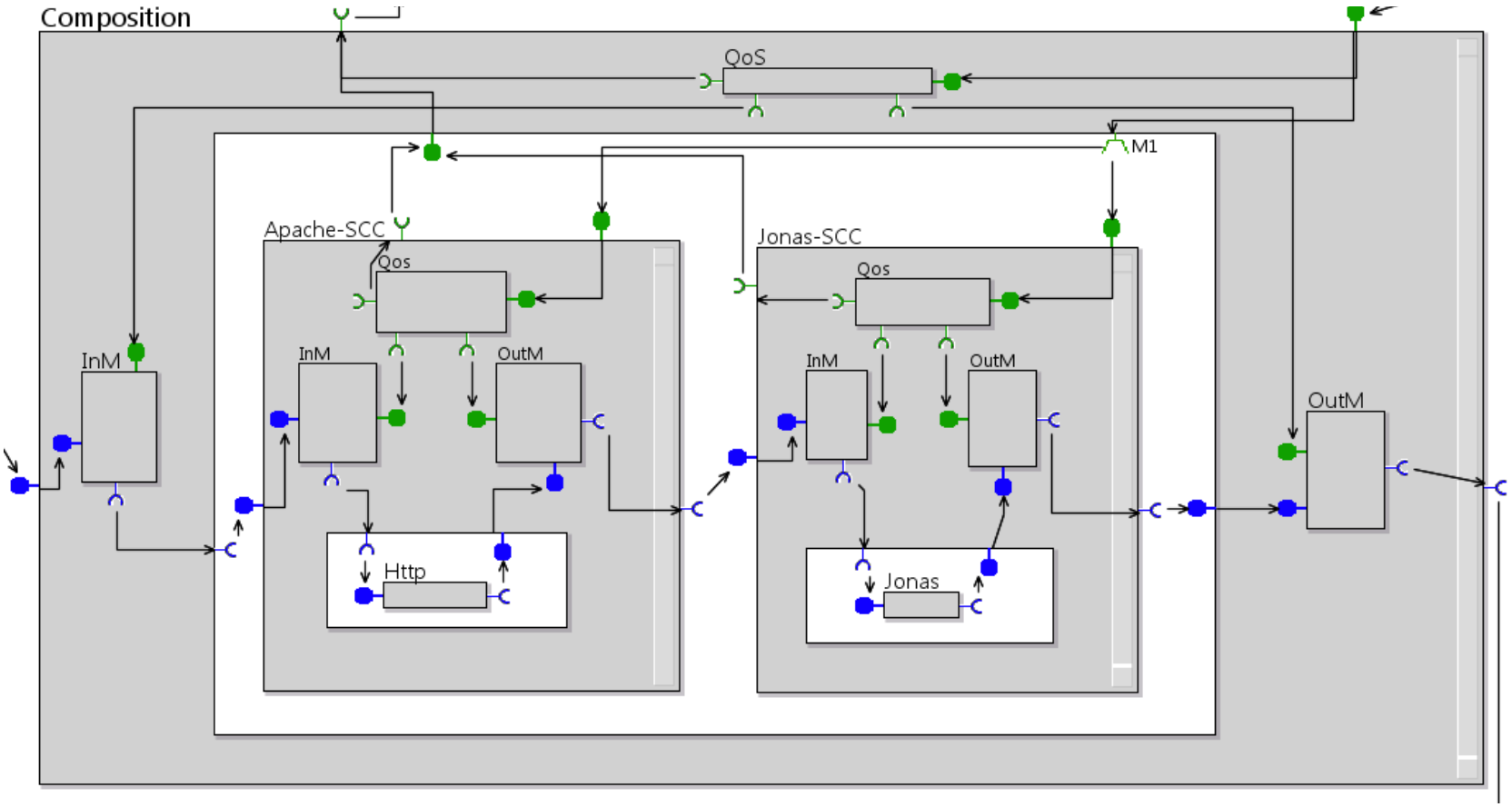
The monitor component is an **interceptor**

The interfaces are instantiated for specific request names and types



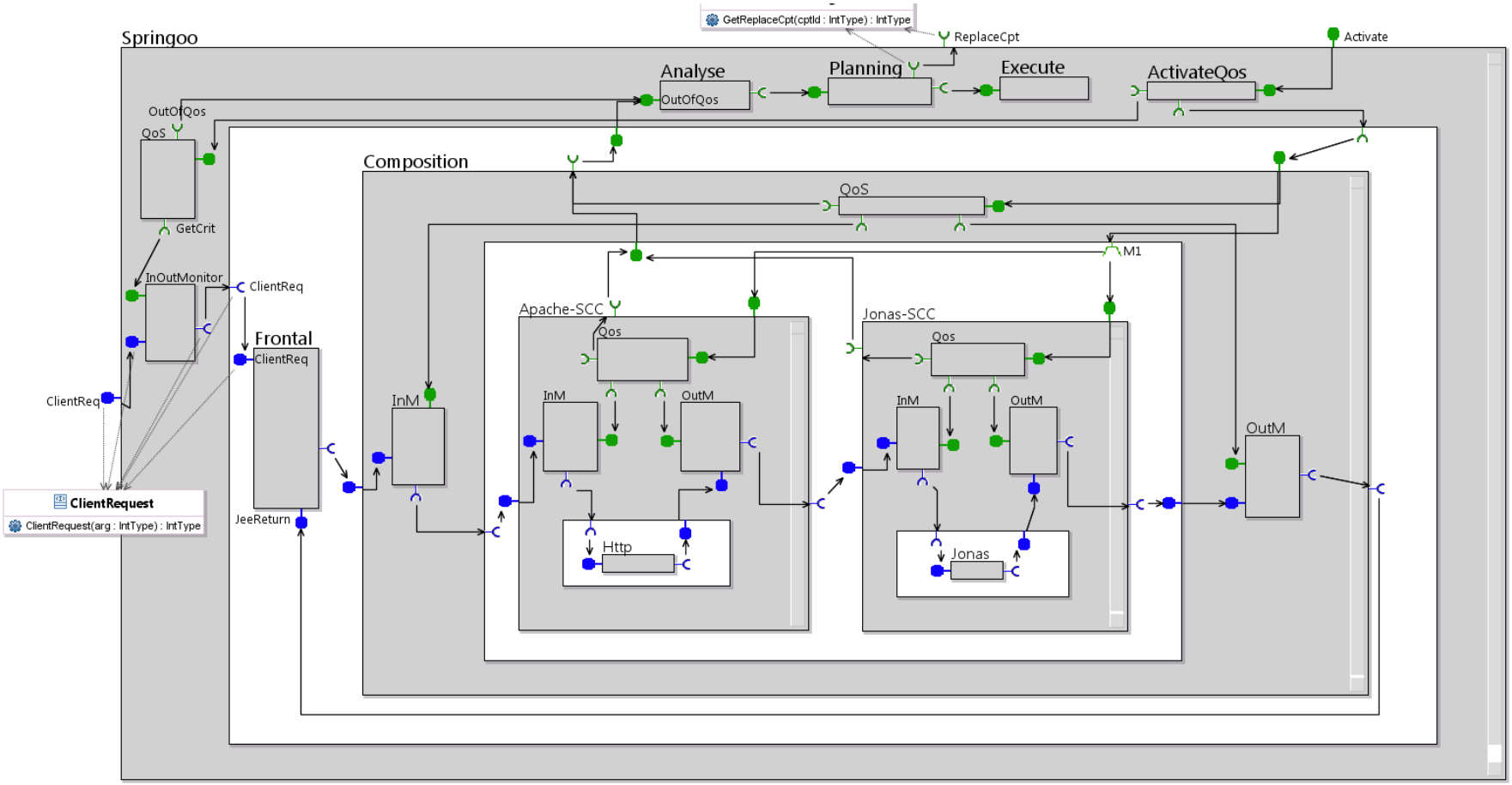
Service Composition

SCC components are composed within GCM composite,
In a distributed, and eventually hierarchical manner (architect choice).

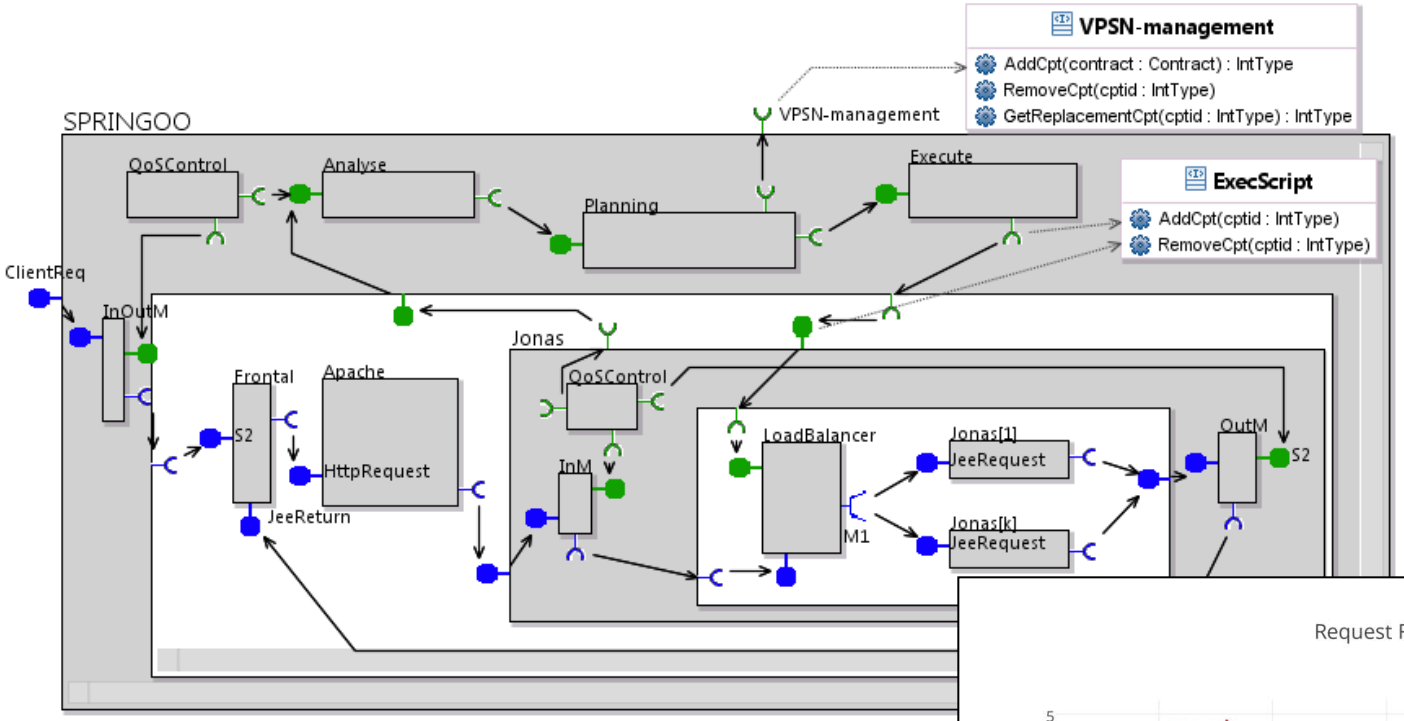


Autonomic management: the MAPE loop

Simply at the toplevel, or eventually distributed and hierarchical



Experiments



Simulation:

-Apache -> OutContract

=> Adjust an attribute value

-Jonas component too slow

=> Add another instance

AGENDA (part 1)

1) Context and Motivation:
   - A short introduction to (safe) component-based applications
   - academic and industrial examples:
     - Distributed algorithms;
     - distributed resiliant industrial infrastructure;
     - QoS-aware elastic architecture for Cloud applications
2) Formal Methods and Component models:
   - Vocabulary and context:
     - specification, modeling, testing, verification…: Formal methods in the design flow of distributed/embedded systems
   - Fractal/GCM: a component-based programming model for safe distributed applications
3) VCE (VerCors Component Editor): a model-driven specification environment for GCM.

Formal methods and the MDE approach

Formal methods :

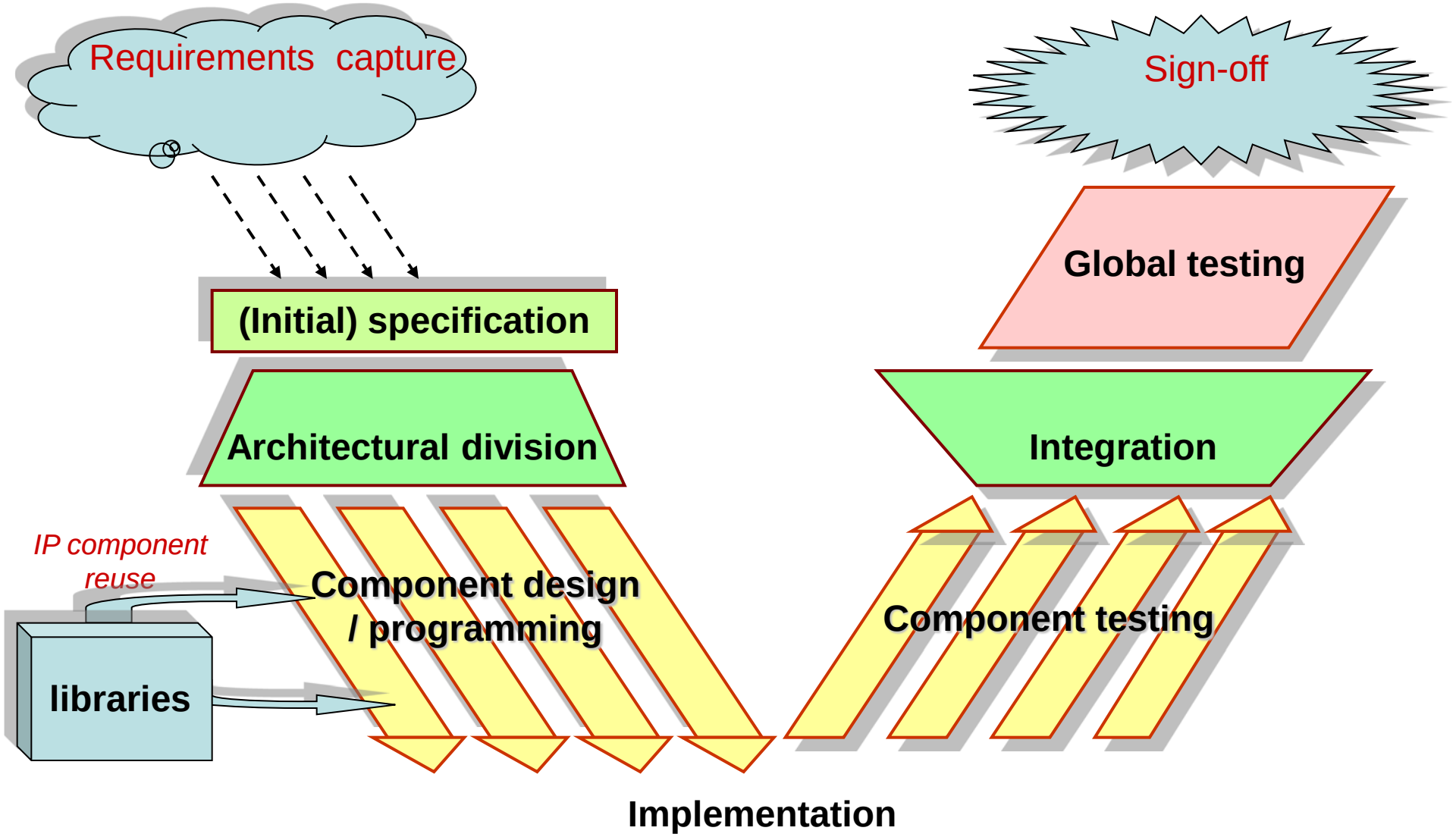
• Provide mathematical semantics to models so that their relation to implemented product can be asserted and proved :

- specification formalisms, (temporal) logics
- model checking, equivalence checking, theorem-proving
- certification, testing
- model-based test generation

• Modeling languages:

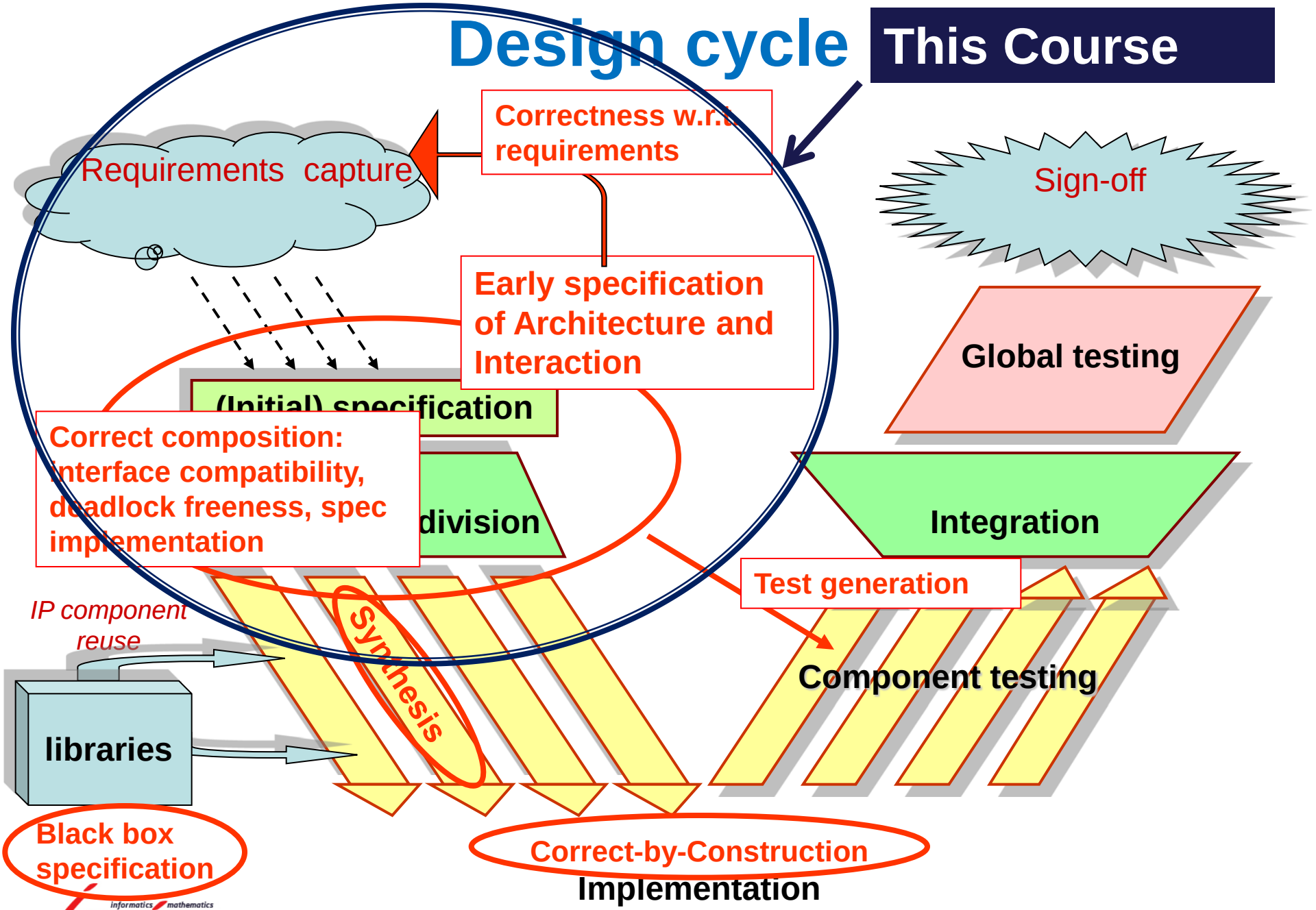
- UML and variants (State machines, StateCharts, SysML,…)
- Dedicated IDLs and ADLs for system decomposition (…)
- Assertion languages (annotations)

Design Cycle



Design cycle

This Course

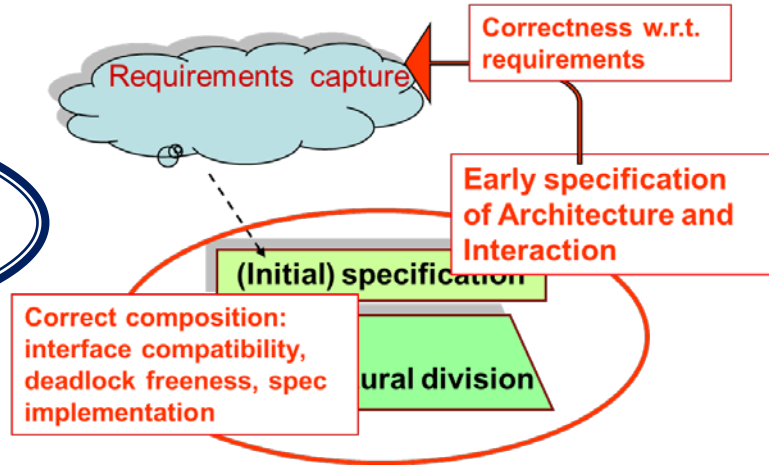


Modeling & Verifying Component Systems

- Goal:

⇒ Early specification of an Abstract view of the system:

Architecture, Behavior, Requirements.



⇒ Model-checking based verification:

Correct composition, Deadlock analysis, Correctness w.r.t. Requirements

Through finite model generation.

⇒ Implementation / code generation:

“Correct by construction” code, (partially) generated from models.

AGENDA (part 1)

1) Context and Motivation:
   - A short introduction to (safe) component-based applications
   - academic and industrial examples:
     - Distributed algorithms;
     - distributed resiliant industrial infrastructure;
     - QoS-aware elastic architecture for Cloud applications
2) Formal Methods and Component models:
   - Vocabulary and context:
     - specification, modeling, testing, verification…: Formal methods in the design flow of distributed/embedded systems
   - Fractal/GCM: a component-based programming model for safe distributed applications
3) VCE (VerCors Component Editor): a model-driven specification environment for GCM.

At the beginning there was…
The Fractal project

- Reflective software component model technology for the construction of highly adaptable, and reconfigurable distributed systems
  - A programming-language independent component model
  - A set of tools to support programming and assembling
  - Software industries needs (≠ object-orientation):
    Dependencies, assembly, packaging, deployment, configuration

- Open and adaptable/extensible

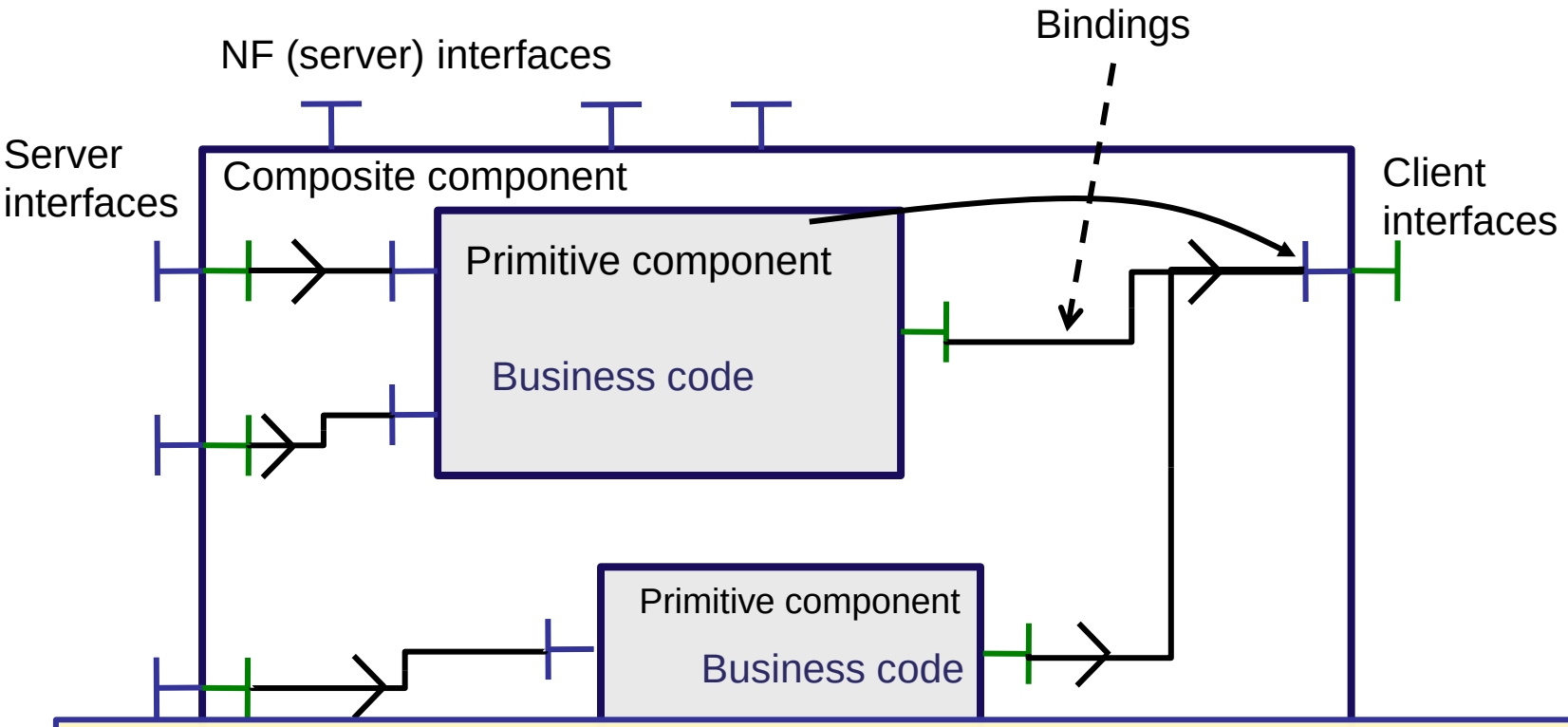
- Component [Szyperski, 2002]:
  "A component is a unit of composition with **contractually** specified **interfaces** and context **dependencies** only. A software component can be **deployed** independently and is subject to **composition** by third parties."

The Fractal component model

- Systems and middleware engineering
  - Generic enough to be applied to any other domain
  - Fine grain (opposed to EJB or CCM), close to a class model
  - Lightweight (low overhead on top of objects)
  - Independent from programming languages
  - Homogeneous vision of all layers (OS, middleware, services, applications)

- Usable as a component framework to build applications
  - with “standard” Fractal components
- Usable as a component framework framework
  - building different kinds of components
  - with minimum introspection and simple aggregation (à la COM)
  - with binding and lifecycle controllers (à la OSGi)
  - with a two-level hierarchy and bindings (à la SCA)
  - with persistence and transaction controllers (à la EJB)
  - with attribute controllers (à la MBean)

What are (GCM/Fractal) Components?



- Hierarchical
- Extensible
- Reflexive: runtime component management
- Separate functional / non-functional concerns

GCM: asynchronous model

Distributed components :

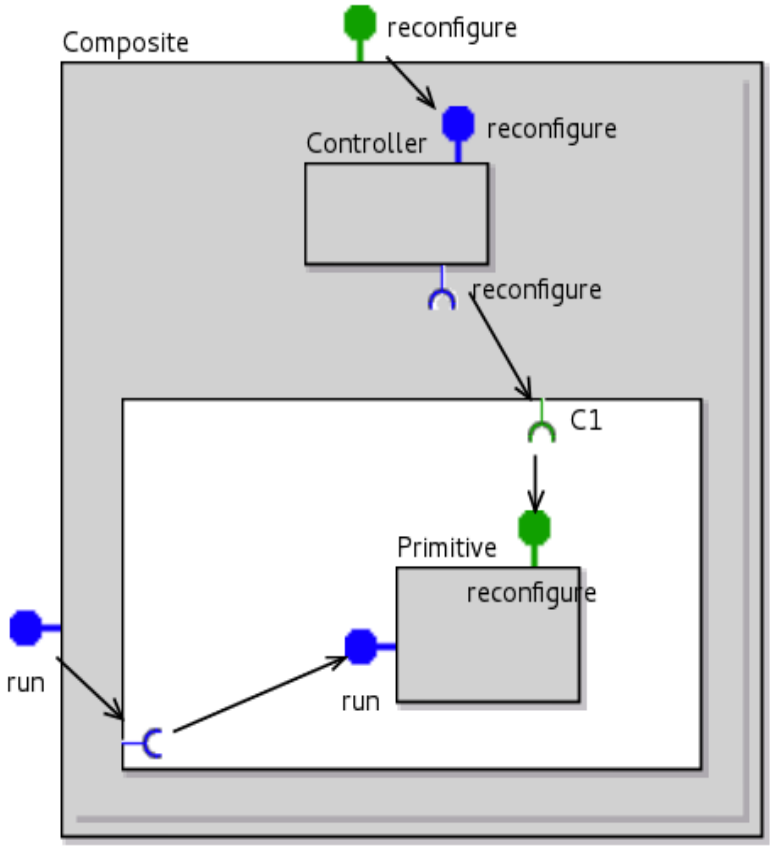
- ⇒ No shared memory
- ⇒ Communication = Remote Method Call
- ⇒ Physical infrastructure ≠ logical (virtual) architecture
- ⇒ Large scale Grid/Cloud computations:
  - ➢ Multicast and gathercast interfaces
- ⇒ Asynchrony of computation :

  Remote Calls are non-blocking

  Notion of Future Objects.

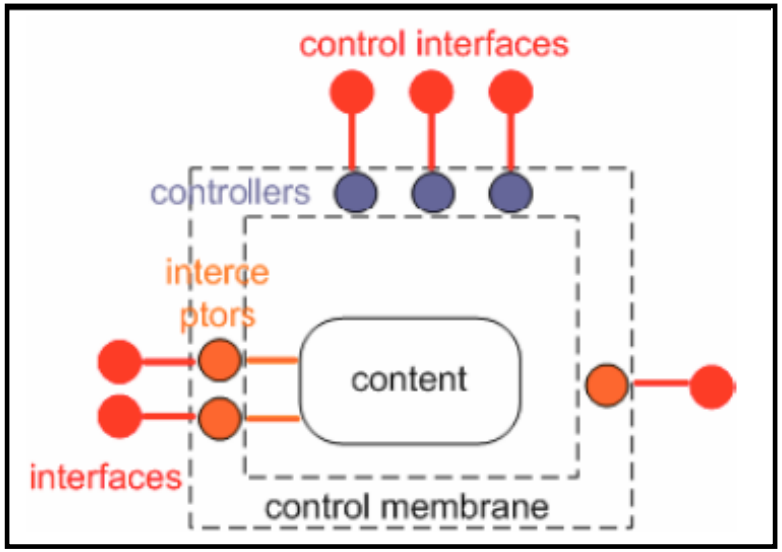
Separation of concerns in GCM architecture

- Content: responsible for business logic
- Membrane: responsible for control part
- Functional and non-functional interfaces
- Business logic and control part can be designed separately



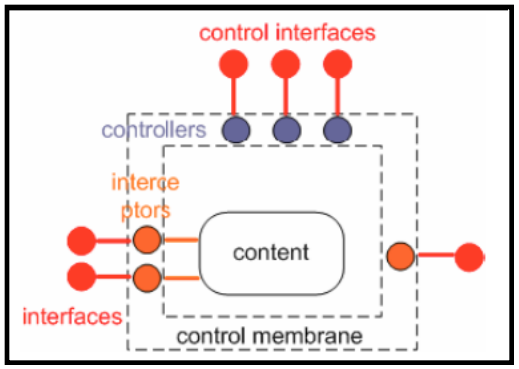
Fractal/GCM : controllers

- Control
  - Non functional (technical) properties
  - Implemented in the membrane
  - Made of a set of controllers
  - E.g. security, transaction, persistence, start/stop, naming, autonomicity
  - Controllers accessible through a control interface
  - Controllers and membranes are open



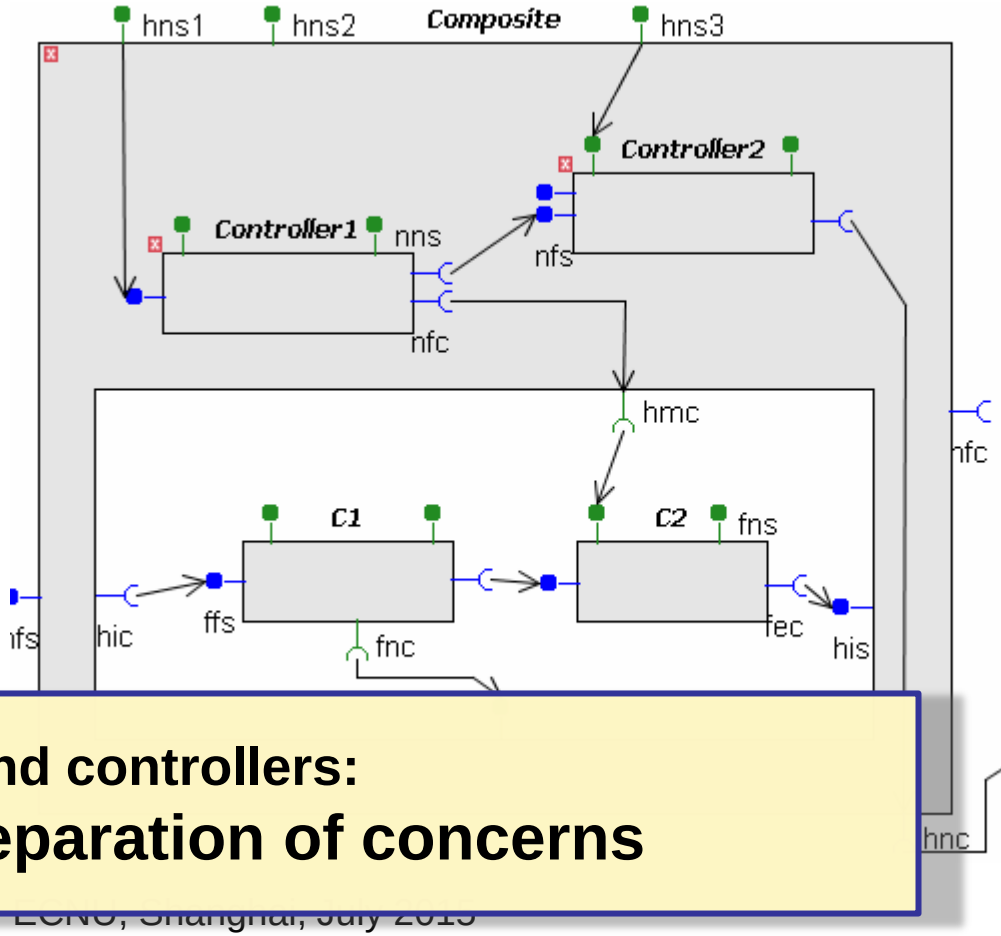
- Predefined :
  - Life-cycle
  - Binding controller
  - Attribute controller
  - Content controller

GCM: components for controllers



“Componentize” the membrane:

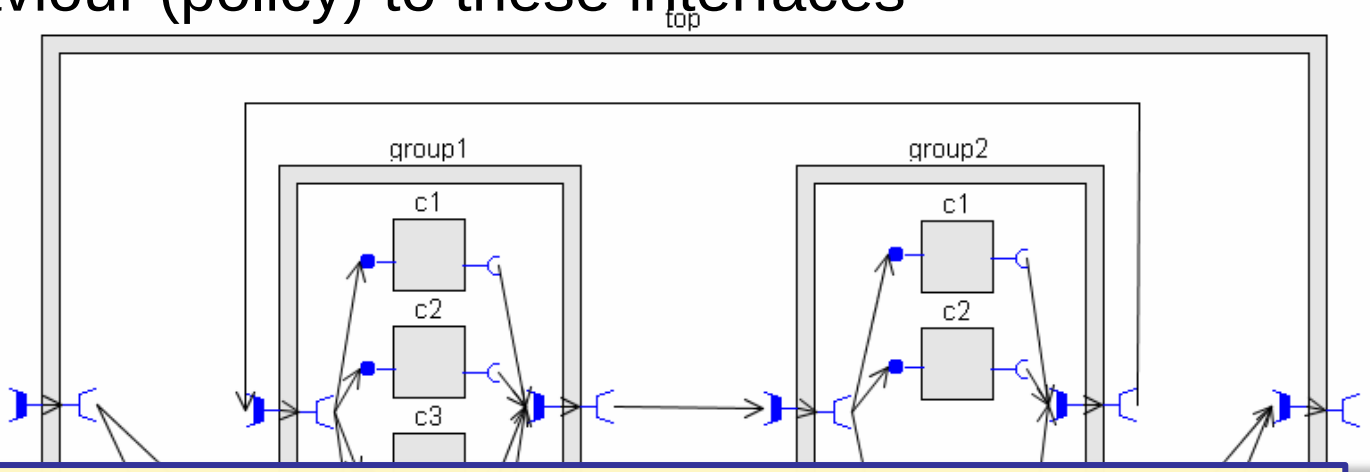
- Build controllers in a structured way
- Reuse of controller components
- Applications: control components for self-optimization, self-healing, self-configuring, interceptors for encryption, authentication



Membrane and controllers:
A way to improve Separation of concerns

GCM: NxM communication

- 1 to N = multicast / broadcast / scatter
- N to 1 bindings = gathercast
- Attach a behaviour (policy) to these interfaces



Group architecture and communication policies:

Allows for (high performance) very large scale applications,

With easy and portable architecture specification

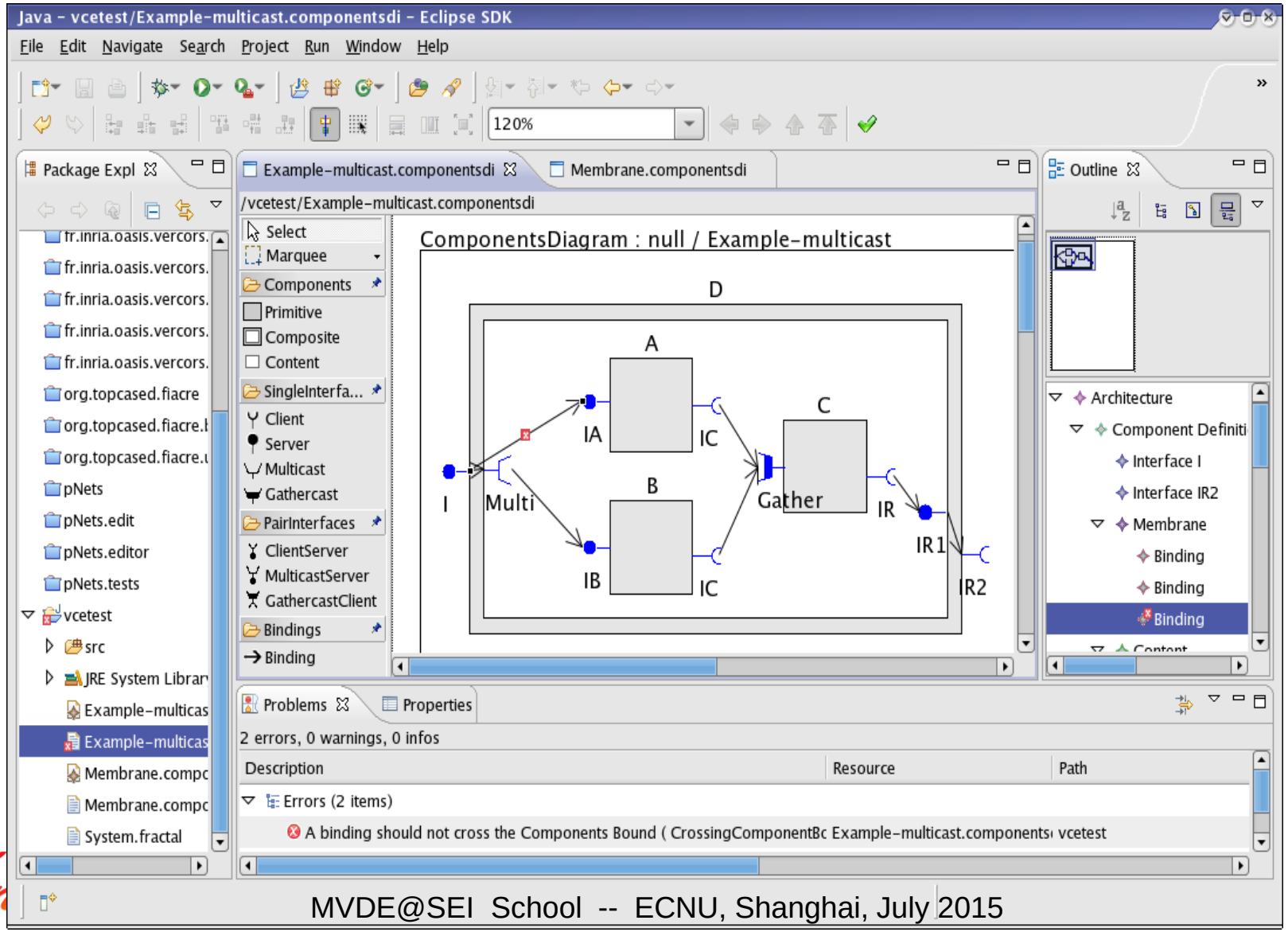
Summary

1. Developing large-scale distributed applications is not so easy…
2. Formalisms and concepts :
   a) Component-based architecture provides structure, composition, reuse, separation of concerns
   b) GCM addresses: distribution and asynchronous communication; large-scale parallel computation; dyamic reconfiguration; monitoring, autonomicity.
3. Tools ?

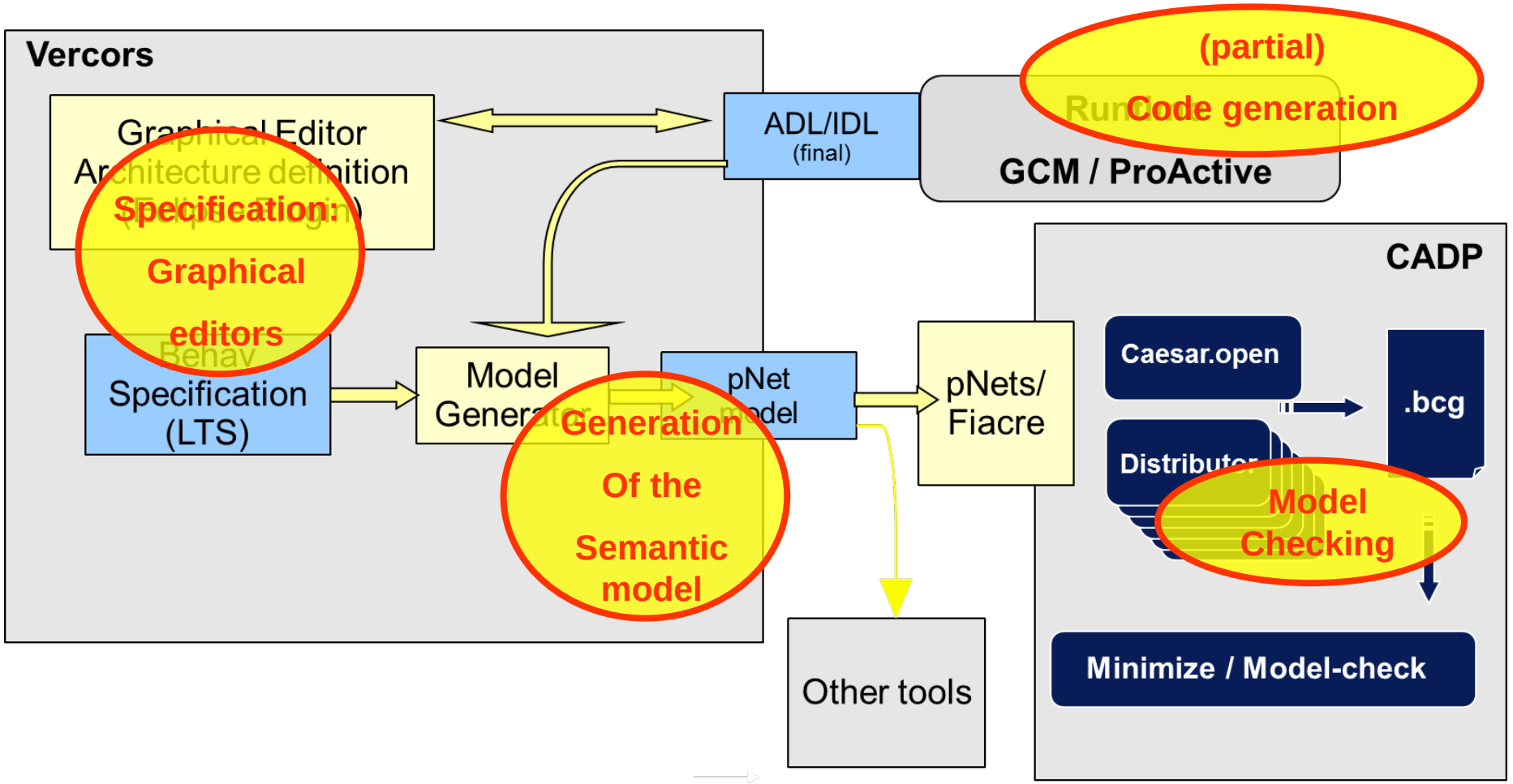
AGENDA (part 1)

1) Context and Motivation:
   - A short introduction to (safe) component-based applications
   - academic and industrial examples:
     - Distributed algorithms;
     - distributed resiliant industrial infrastructure;
     - QoS-aware elastic architecture for Cloud applications
2) Formal Methods and Component models:
   - Vocabulary and context:
     - specification, modeling, testing, verification…: Formal methods in the design flow of distributed/embedded systems
   - Fractal/GCM: a component-based programming model for safe distributed applications
3) VCE (VerCors Component Editor): a model-driven specification environment for GCM.

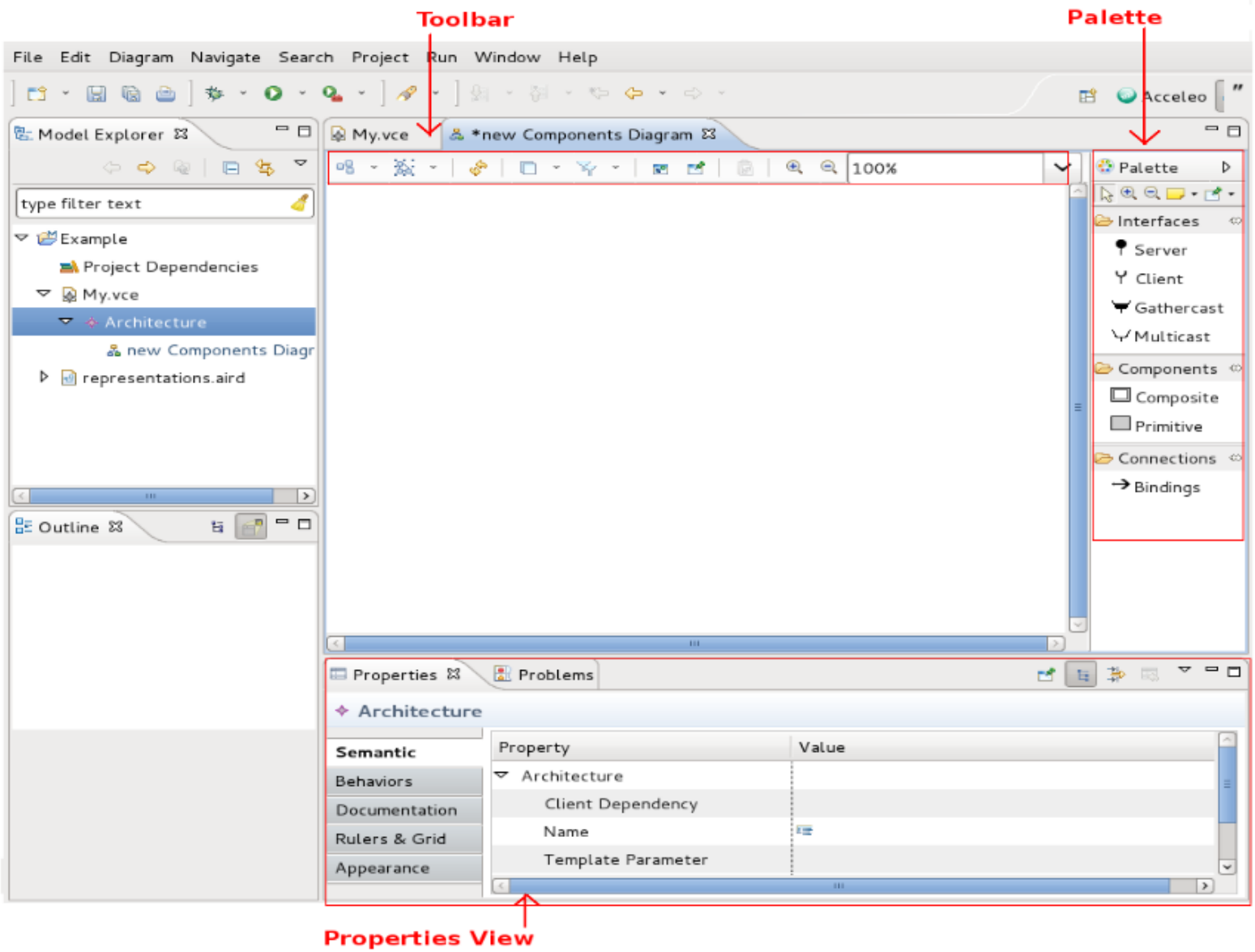
GCM high level specifications: The VCE tool



Model-checking applications: The Vercors platform

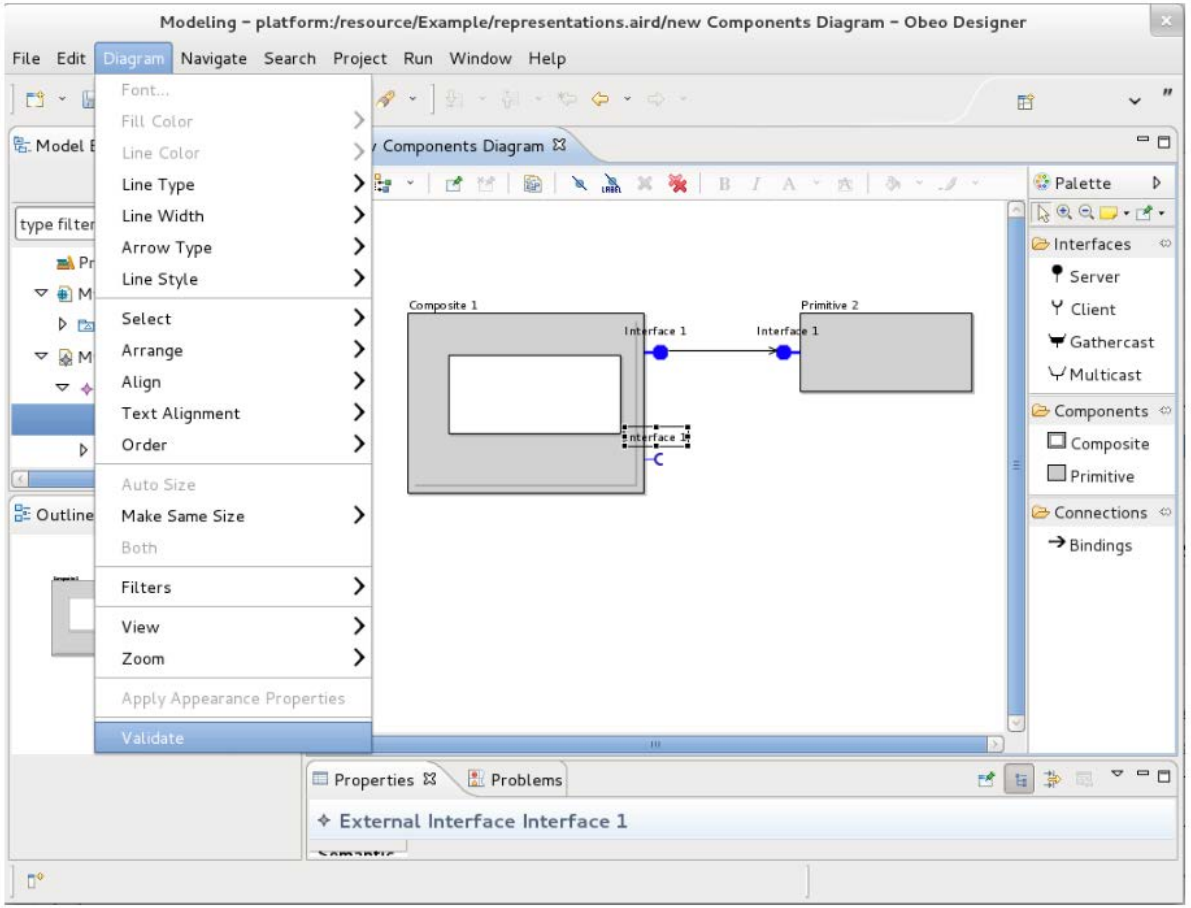


An Eclipse / Sirius environment



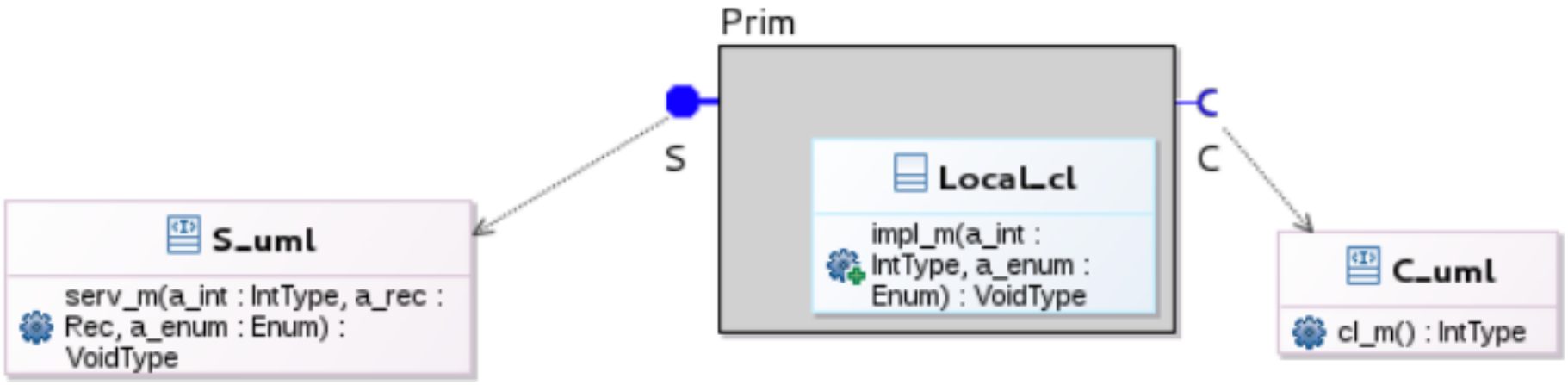
The graphical formalisms: (1) Architecture

- GCM full ADL, with full componentized membrane, and multicast/gathercast



The graphical formalisms: (2) Interfaces

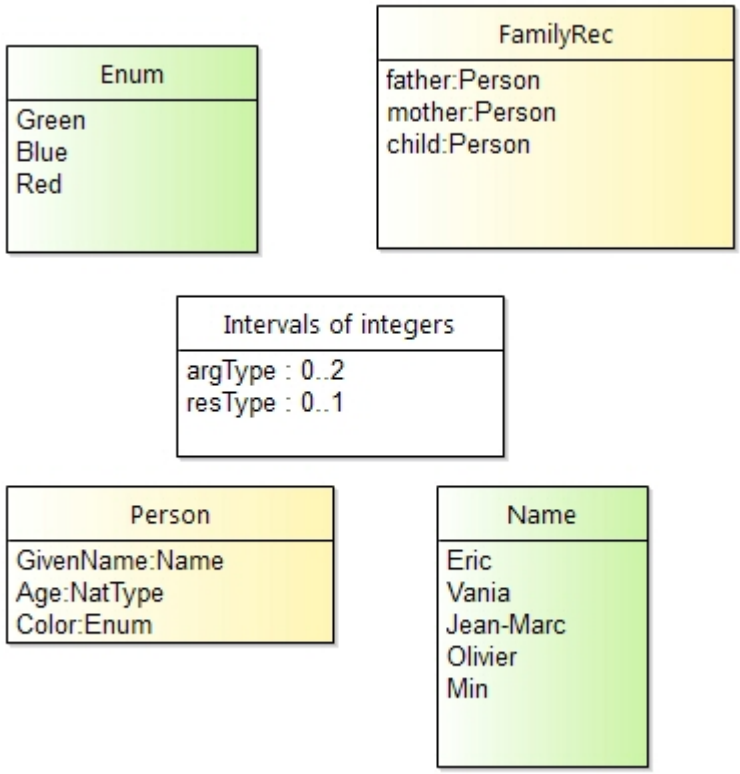
- UML class diagram, for interface signatures, and primitive component implementation class



The graphical formalisms: (3) data-types

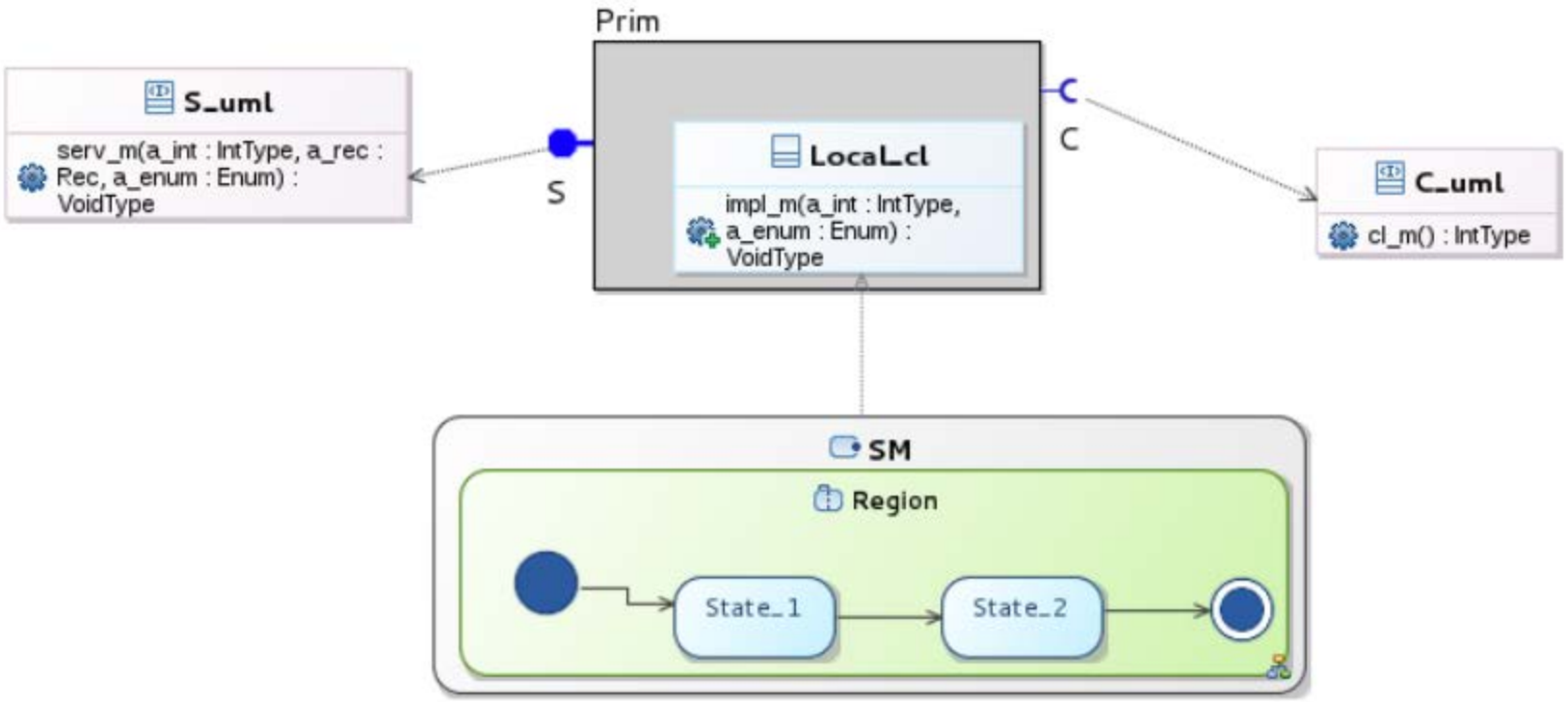
- First order types:
  - ✓ Boolean
  - ✓ Integer and interval
  - ✓ Enumeration
  - ✓ Record
  - ✓ Array (not in this version)

- Rational:
  - Can be easily abstracted into finite domains for model-checking purposes



The graphical formalisms: (4) behaviors

- UML state machines



The graphical formalisms: (5) action language

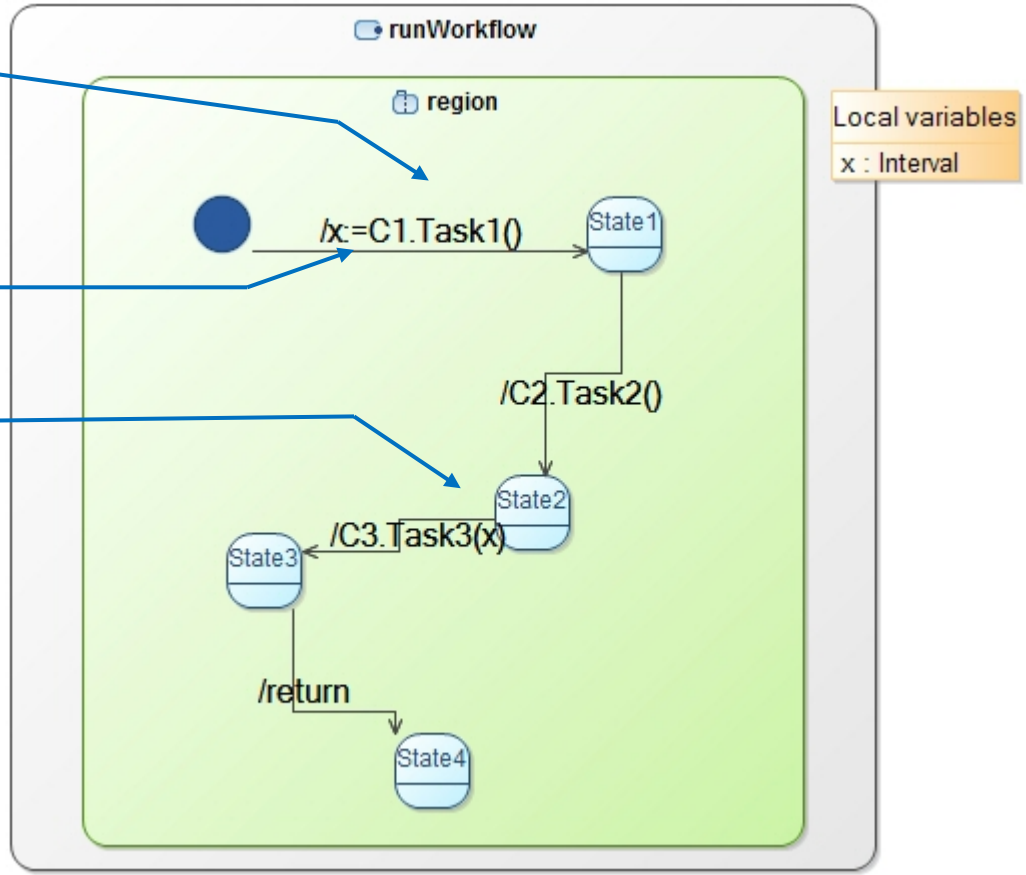
- The labels of transitions in State-machines express the interaction behavior between components.
- Goals:
  - model generation: easily mapped to Labelled Transition Systems with data (pLTS),
  - code generation:
    - Mapped to Java code executable within the GCM/ProActive middleware
    - Generated code is guaranteed to respect the proved (model-checked) properties

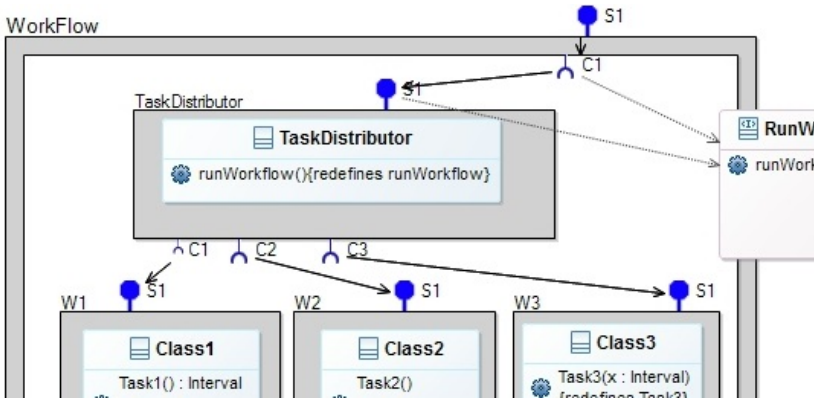
The graphical formalisms: (5) action language

Call method Task1 on client interface C1

Assign result to local variable x

Pass x as a parameter





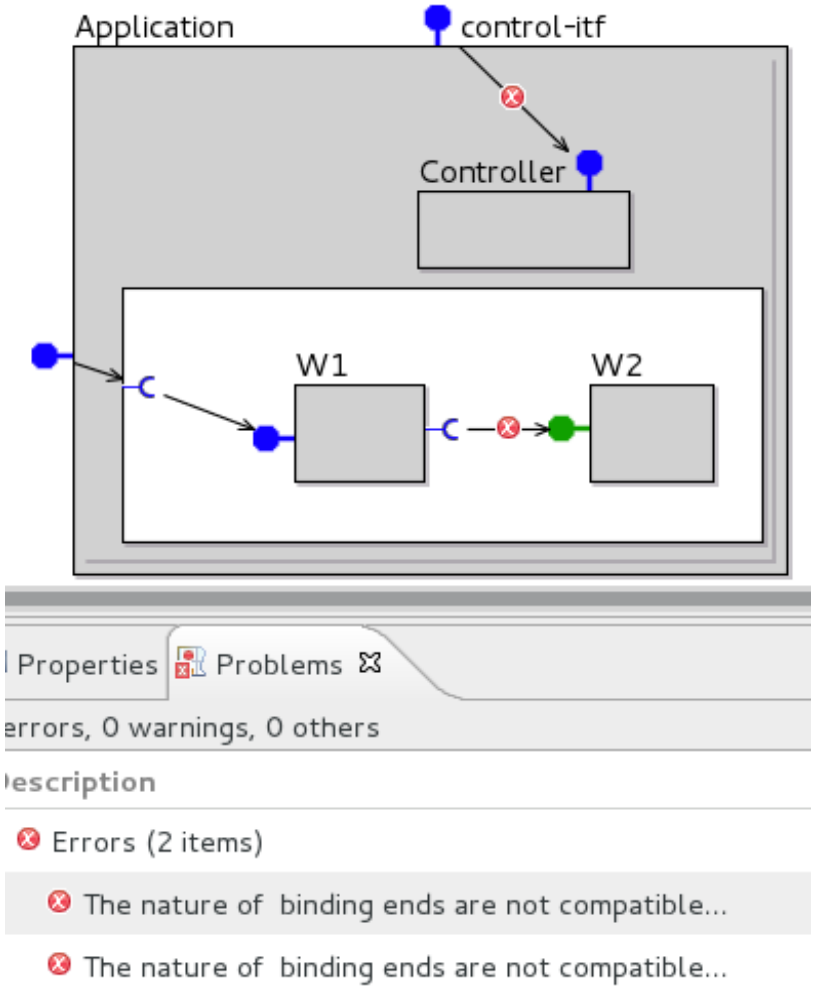
Validity check, code & model generation

- Semantic validity rules:
  - Structural (components, bindings, interface roles)
  - Typing (interface compatibility)
  - Behavioral (variables/methods well-definedness)

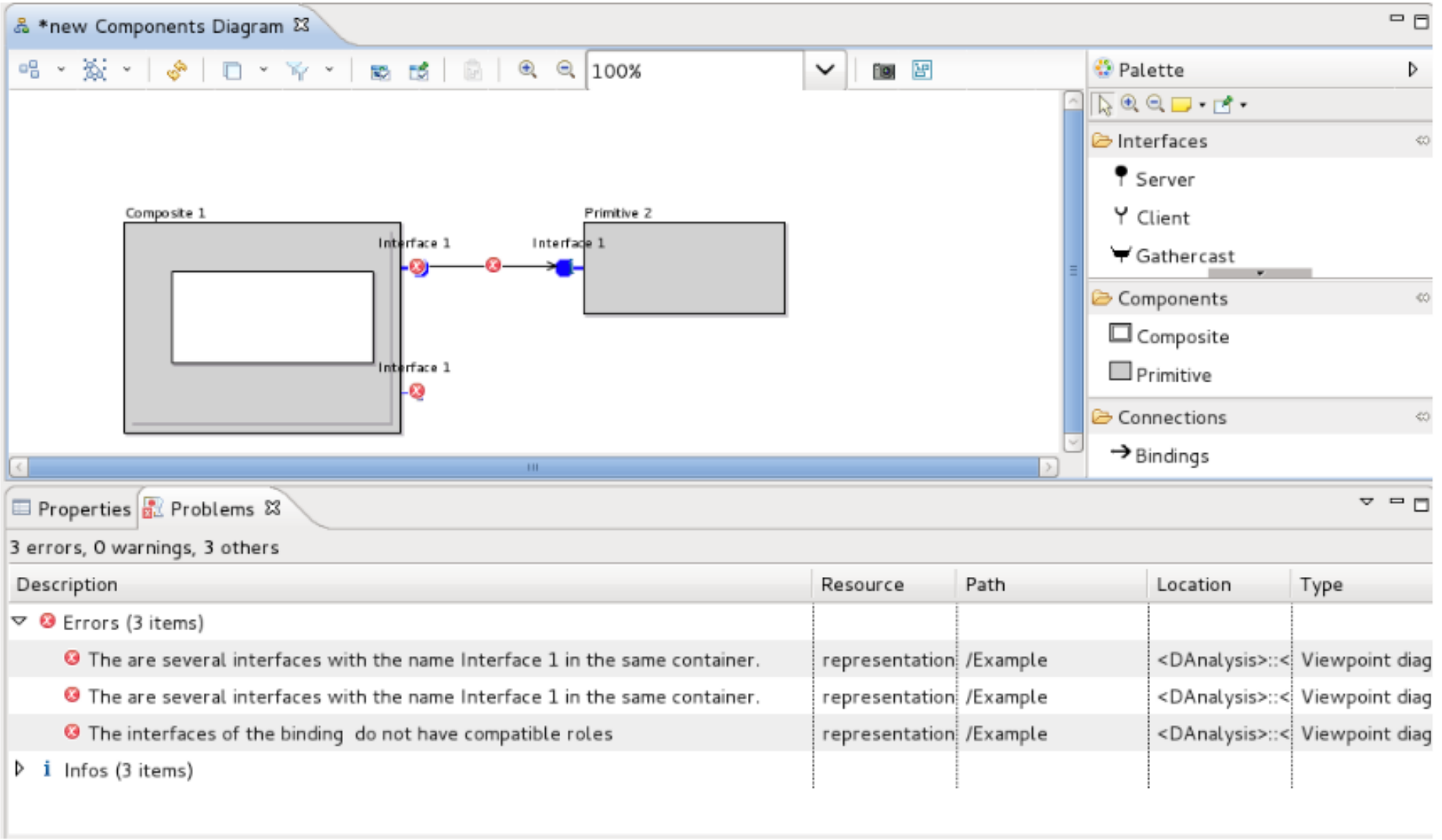
- Guarantees generation of correct code:
  - ADL file (for GCM/ProActive component factory)
  - Behavior model (for model-checking)

Static validation in VCE

- Check all architecture validity the constraints
- Use Acceleo, OCL and Java Services
- Inform user about the violation of constraints



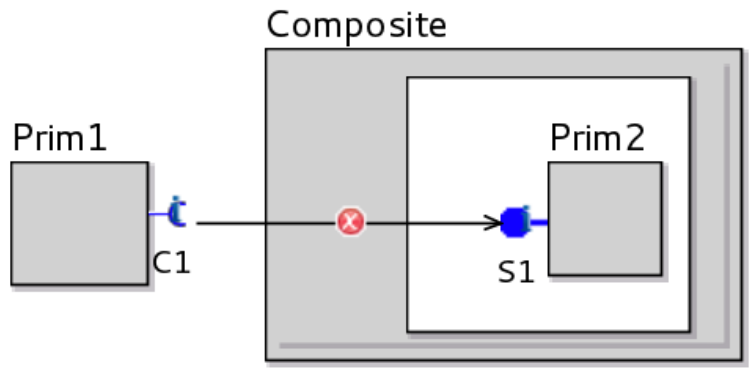
Static validation in VCE



Static validation rules (1)

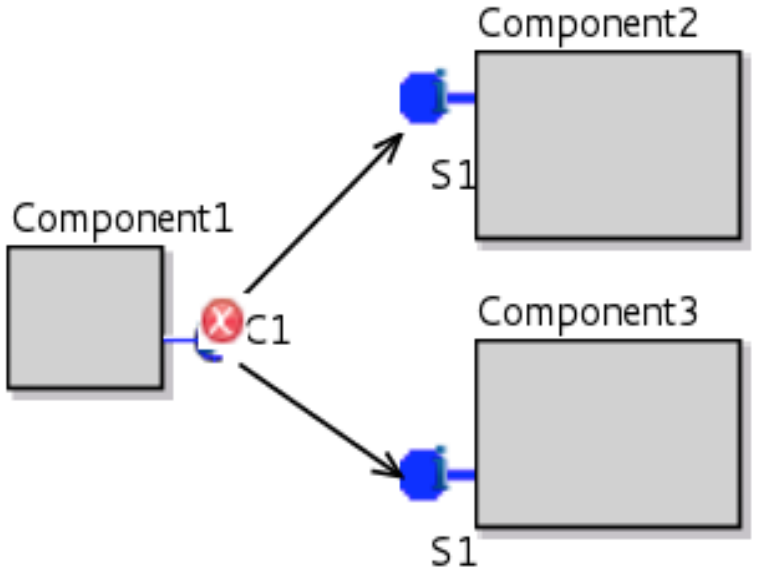
Component encapsulation

Bindings do not cross the boundaries of the components



Deterministic communications

Each client interface is connected to at most one server interface



Static validation rules (2)

Separation of concerns

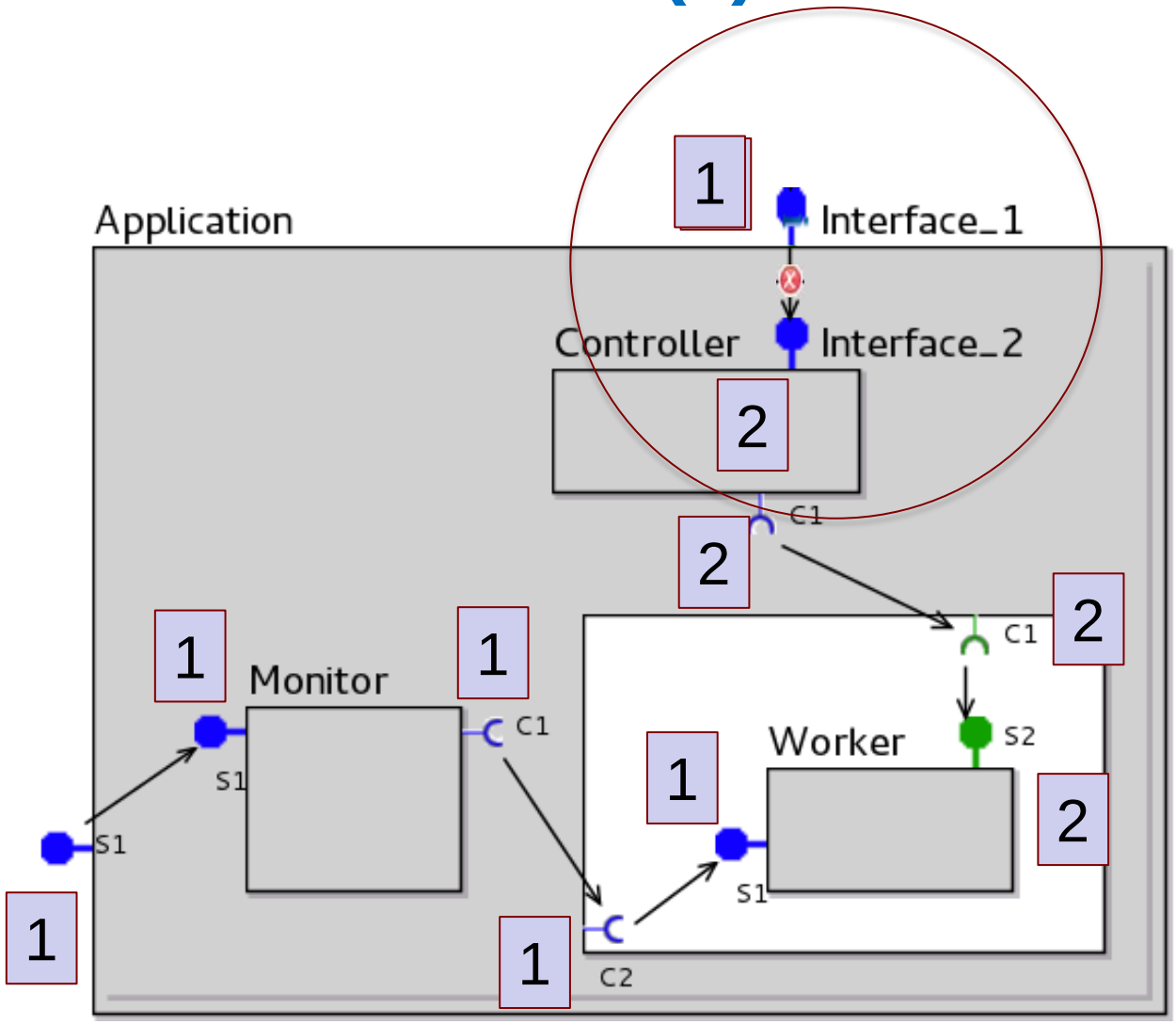
The interfaces connected by a binding should have compatible control levels

- CL of a functional interface = 1
- CL of a non-functional interface = 2
- CL is increased by 1 for interfaces of controllers

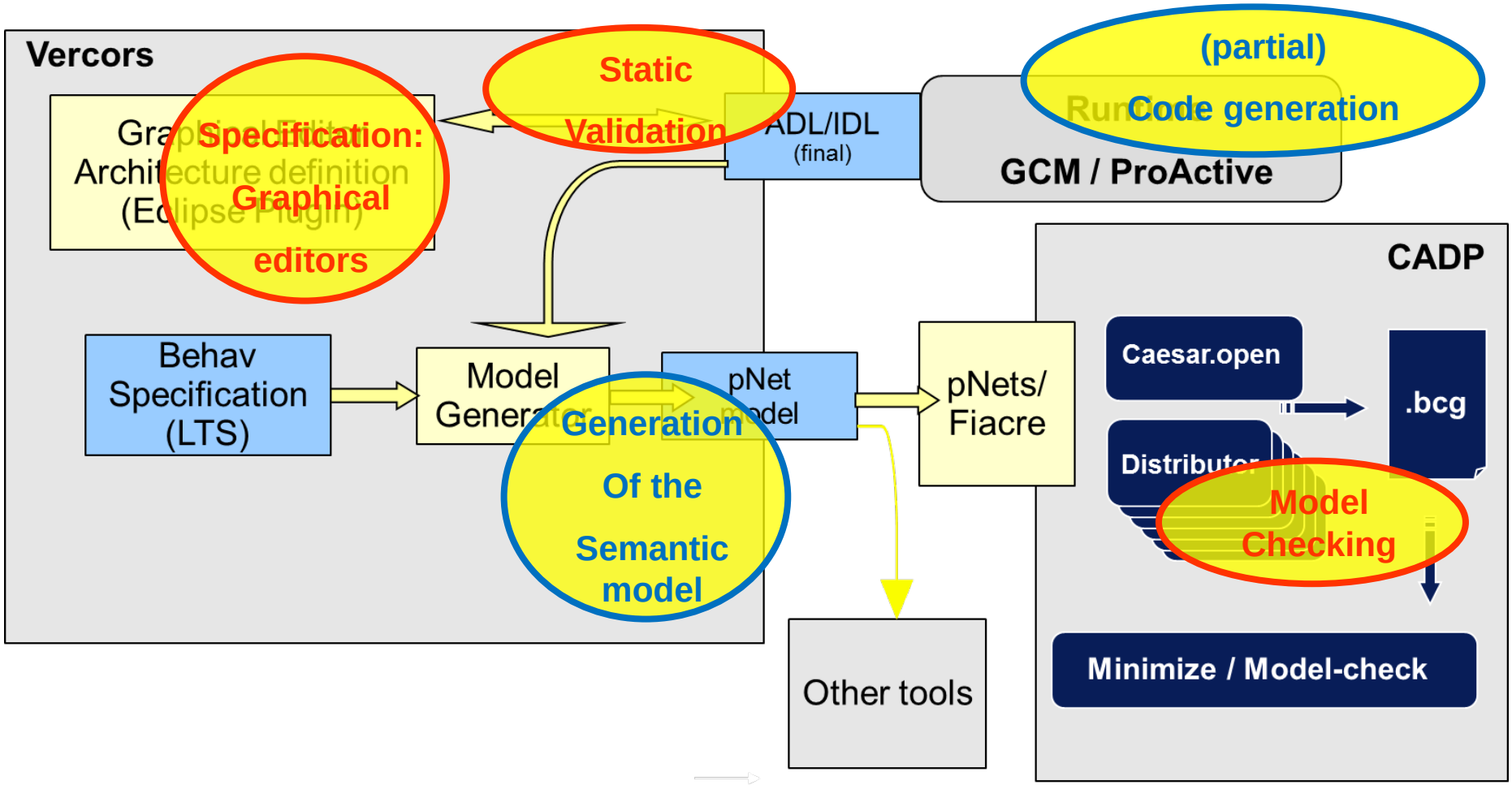
- Compatible CLs: either both = 1, or both >1

Static validation rules (3)

- CL of a functional interface = 1
- CL of a non-functional interface = 2
- CL is increased by 1 for interfaces of controllers
- Compatible CL: either = 1, or >1



How far have we gone yet ?



MDE for safe distributed systems

Thursday, July 16th , Afternoon(Course)

☐1) Introduction: academic and industrial examples: Distributed algorithms; distributed resiliant industrial infrastructure; QoS-aware elastic architecture for Cloud applications

☐2) GCM: a component-based programming model for safe distributed applications

☐3) VCE: a model-driven specification environment for GCM

Friday, July 17th , Morning: (Hands-on, Lab)

- Presentation and exercises with VCE:

☐ Tutorial and simple examples

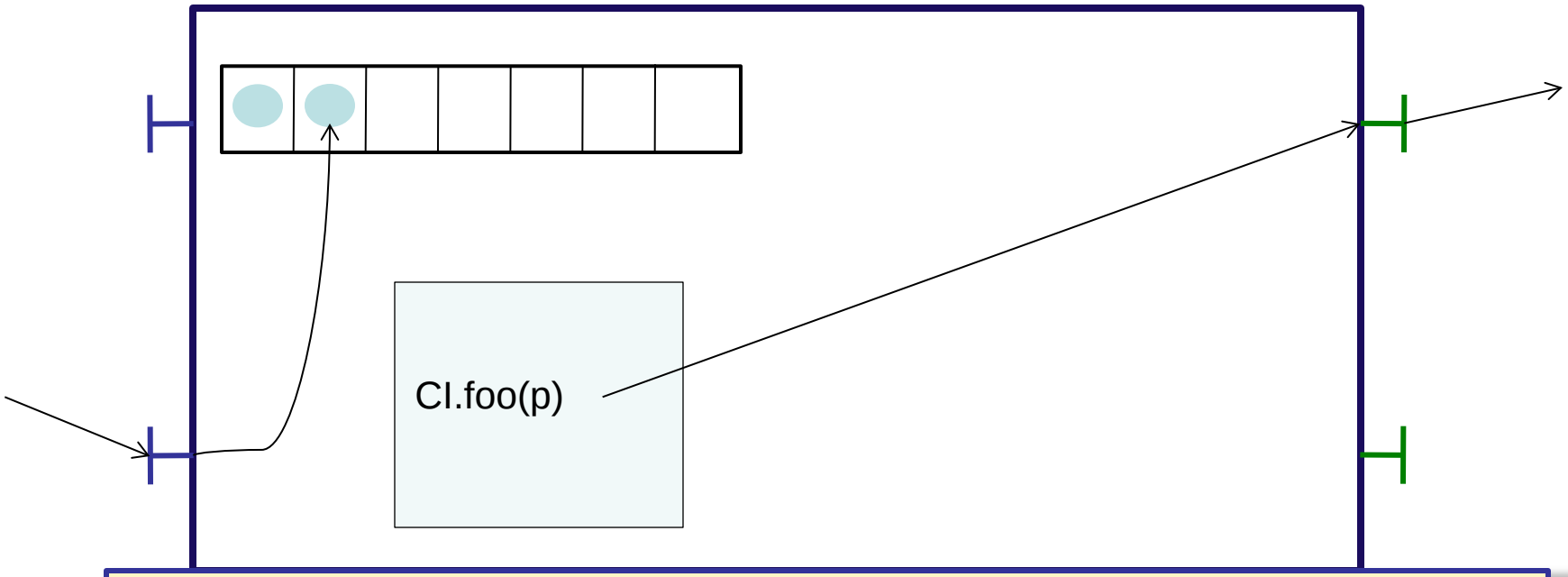
Friday, July 17th , Afternoon: (Lab)

☐1) **Course**: VerCors theoretical background : behavioural models, model-checking, temporal logics, running the verification tools.

☐2) **Hands-on** with VCE and Vercors: continued…

A Glimpse at GCM operational semantics

A Primitive GCM Component

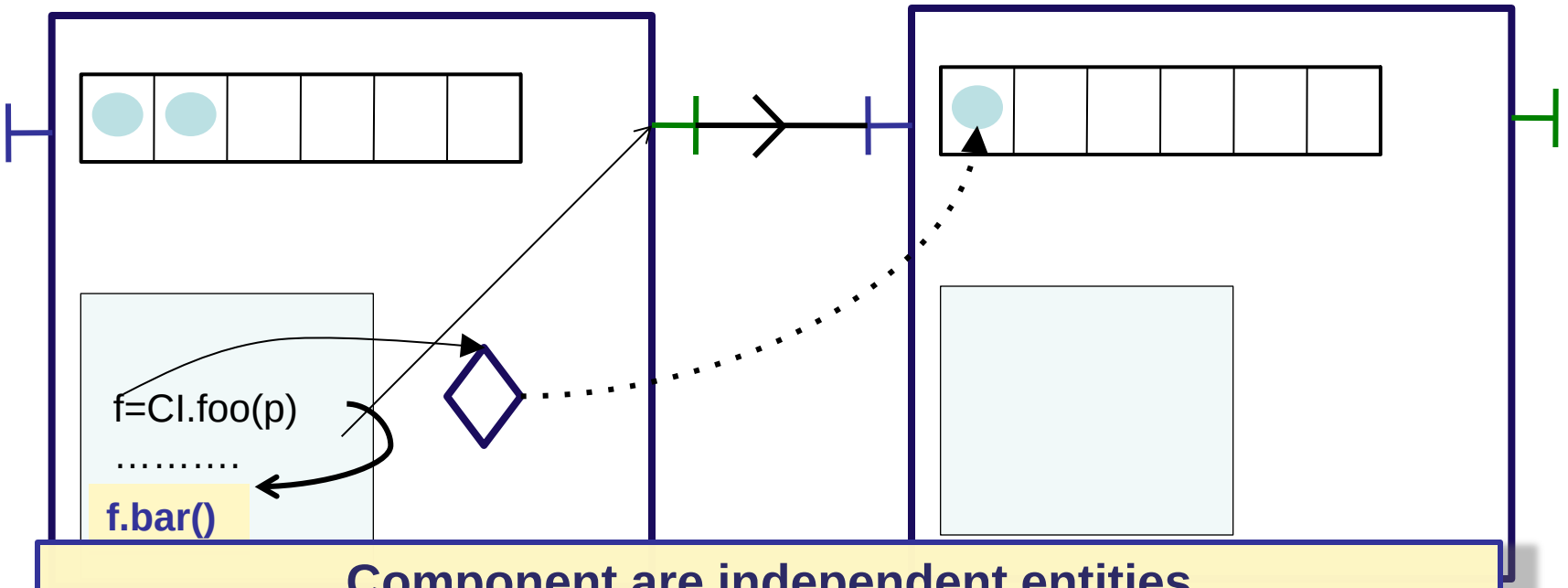


Primitive components communicating by *asynchronous* remote method invocations on interfaces (*requests*)

➔ **Components abstract away distribution and *concurrency***

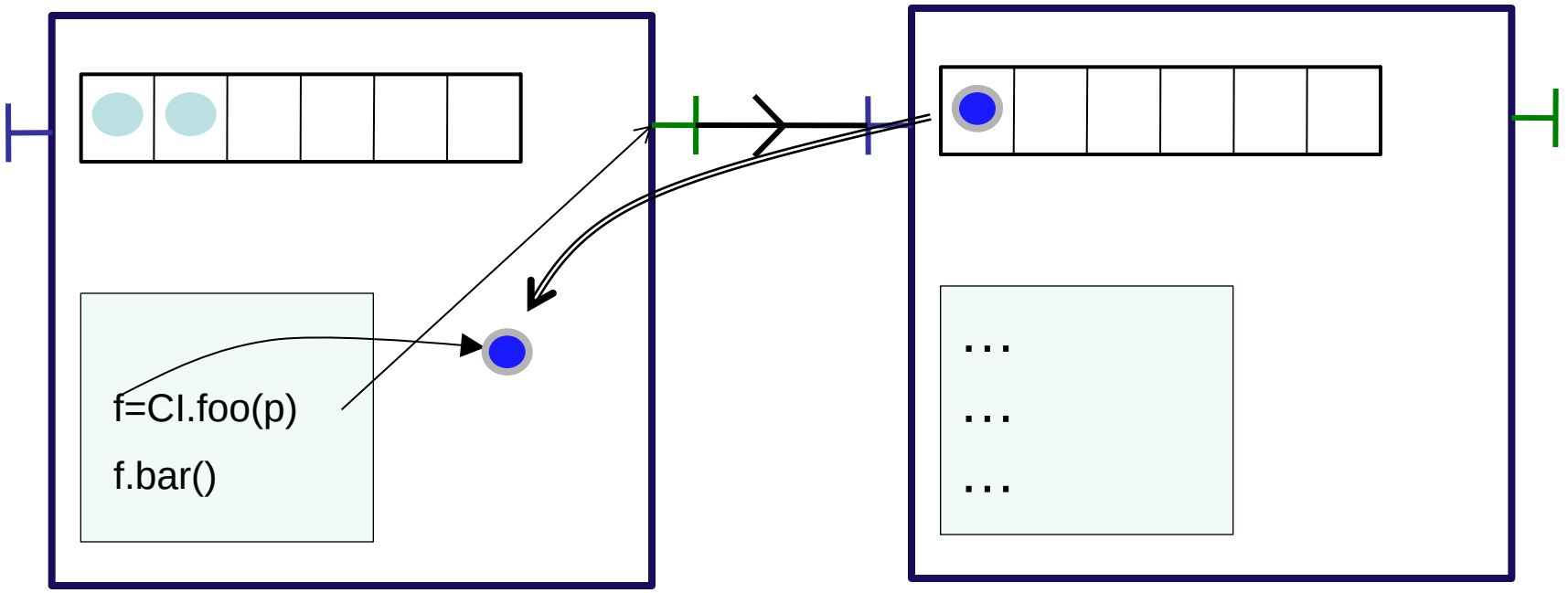
in GCM/ProActive components are mono-threaded
➔ **simplifies concurrency** but can create **deadlocks**

Futures for Components



Component are independent entities
(threads are isolated in a component)
+
Asynchronous method invocations with results
↓
Futures are necessary

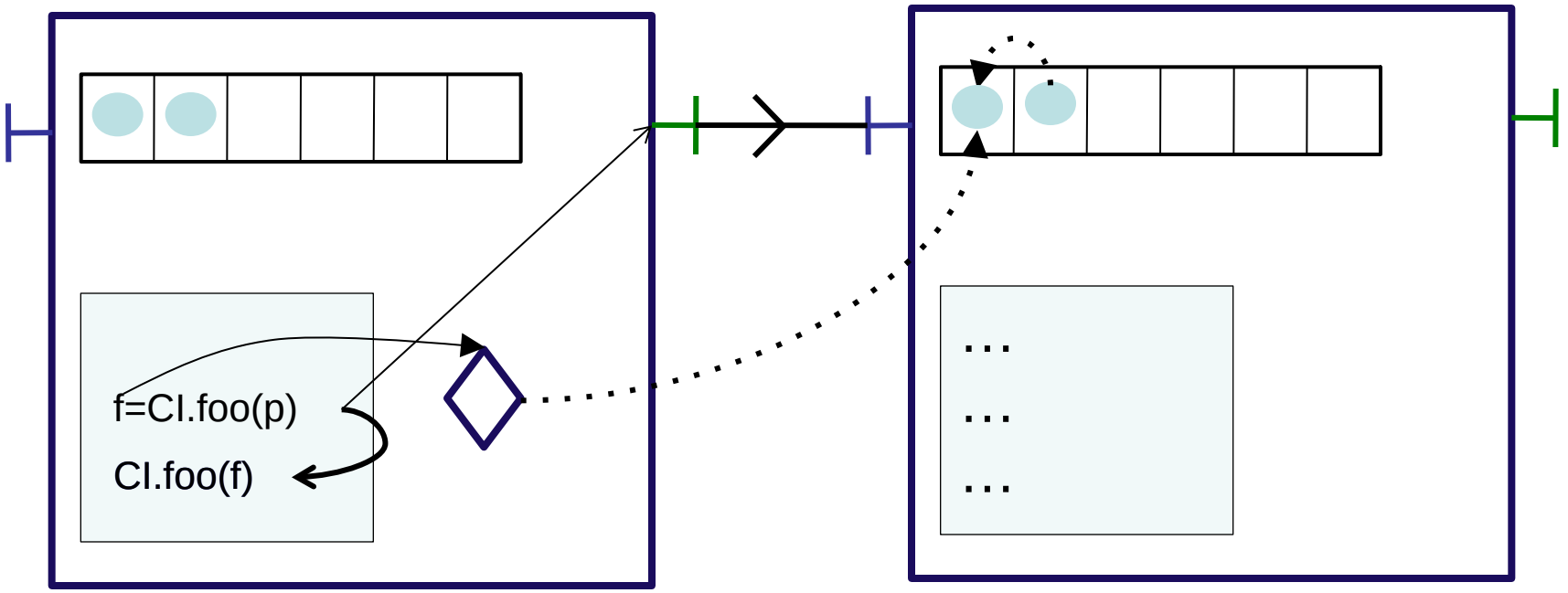
Replies



What is a Future?

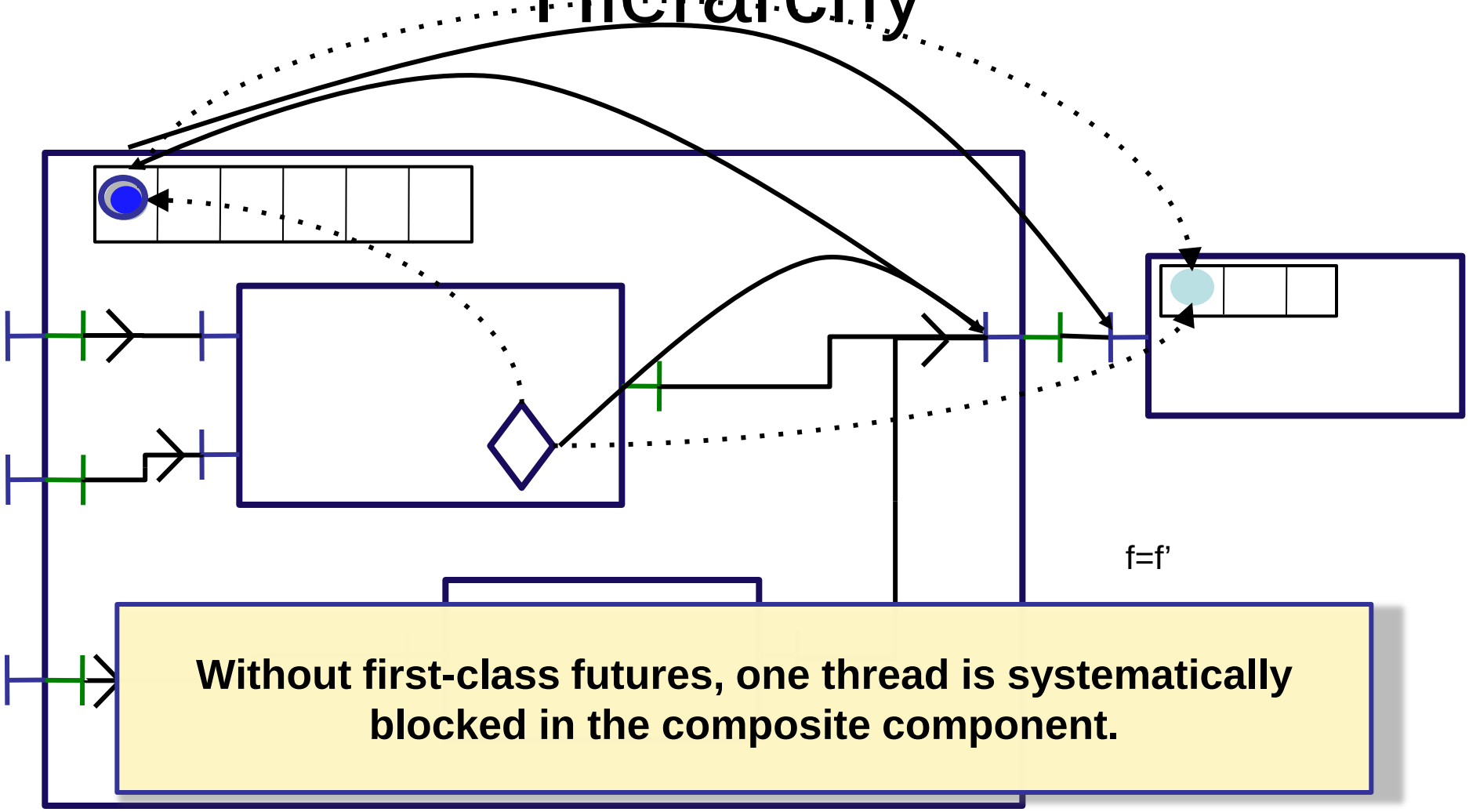
- Future = Placeholder for an awaited result
- Creation of a future
  - How and when are futures created?
    - Implicit creation = automatic upon asynchronous invocation (on a remote object)
    - Explicit creation = there is a *future* construct
- Manipulation and access
  - How to manipulate the futures?
    - Explicit access = *get* operation (+ *future* type)
    - Implicit (transparent) access = any variable can contain a future

First-class Futures



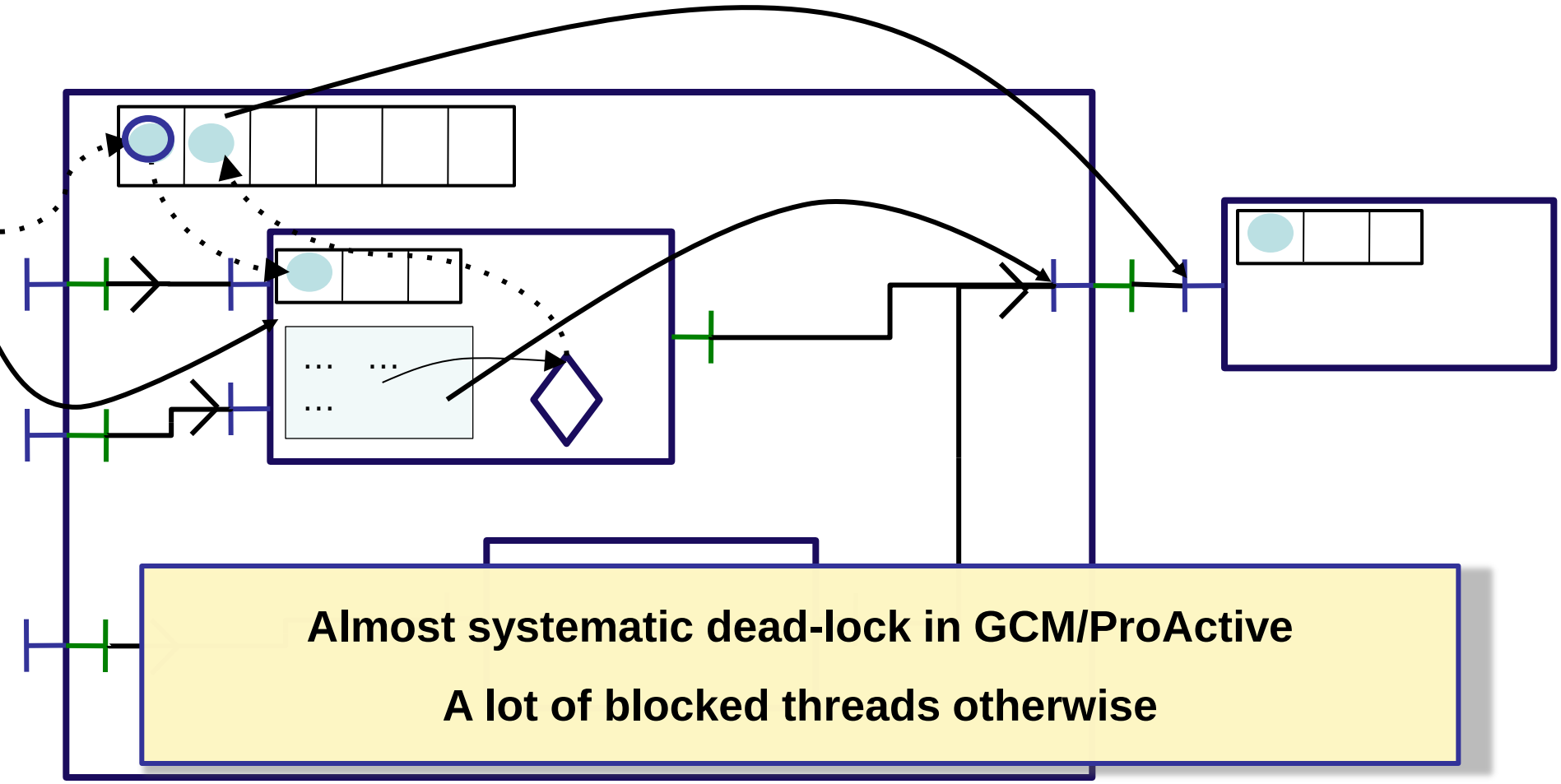
- **Only strict operations are blocking (access to a future)**
- **Communicating a future is not a strict operation**

First-class Futures and Hierarchy



Without first-class futures, one thread is systematically blocked in the composite component.

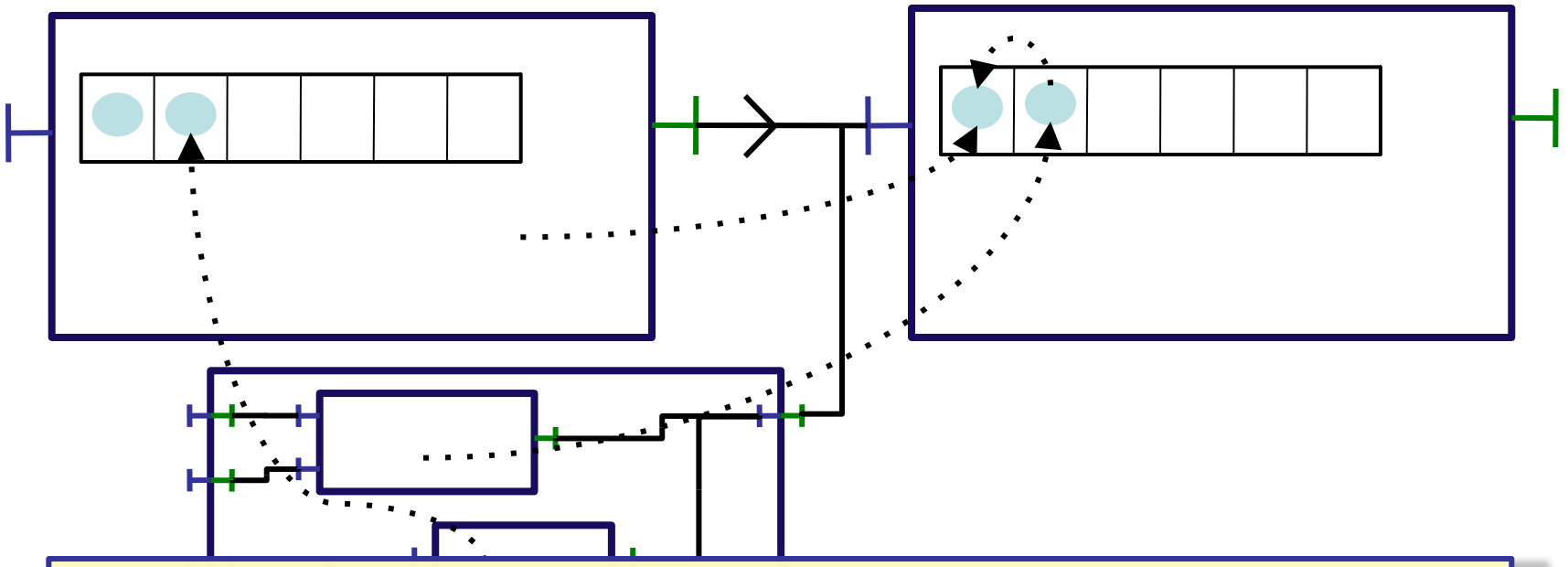
First-class Futures and Hierarchy



Almost systematic dead-lock in GCM/ProActive

A lot of blocked threads otherwise

Reply Strategies



In ASP / ProActive, the result is insensitive to the order of replies (shown for ASP-calculus)

experiments with different strategies (lazy, eager)

Summary

- Primitive components contain the business code
- Primitive components act as the unit of distribution and concurrency (each thread is isolated in a component)
- Communication is performed on interfaces and follows component bindings
- Futures allow communication to be asynchronous requests
- **Futures are transparent can lead to optimisations and are a convenient programming abstraction but …**

MDE for safe distributed systems

Thursday, July 16th , Afternoon(Course)

☐1) Introduction: academic and industrial examples: Distributed algorithms; distributed resiliant industrial infrastructure; QoS-aware elastic architecture for Cloud applications

☐2) GCM: a component-based programming model for safe distributed applications

☐3) VCE: a model-driven specification environment for GCM

Friday, July 17th , Morning: (Hands-on, Lab)

- Presentation and exercises with VCE:

☐ Tutorial and simple examples

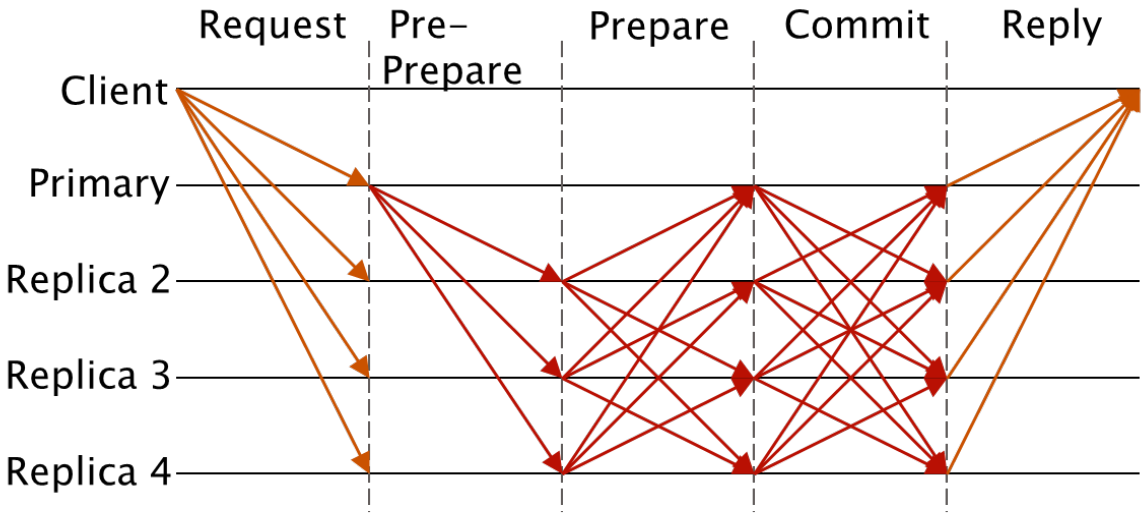
Friday, July 17th , Afternoon: (Lab)

☐1) **Course**: VerCors theoretical background : behavioural models, model-checking, temporal logics, running the verification tools.

☐2) **Hands-on** with VCE and Vercors: continued…

Byzantine Fault Tolerant Systems

- **Byzantine systems:**
- “bad” guys can have any possible behaviour,
- everybody can turn bad, but only up to a fixed % of the processes.

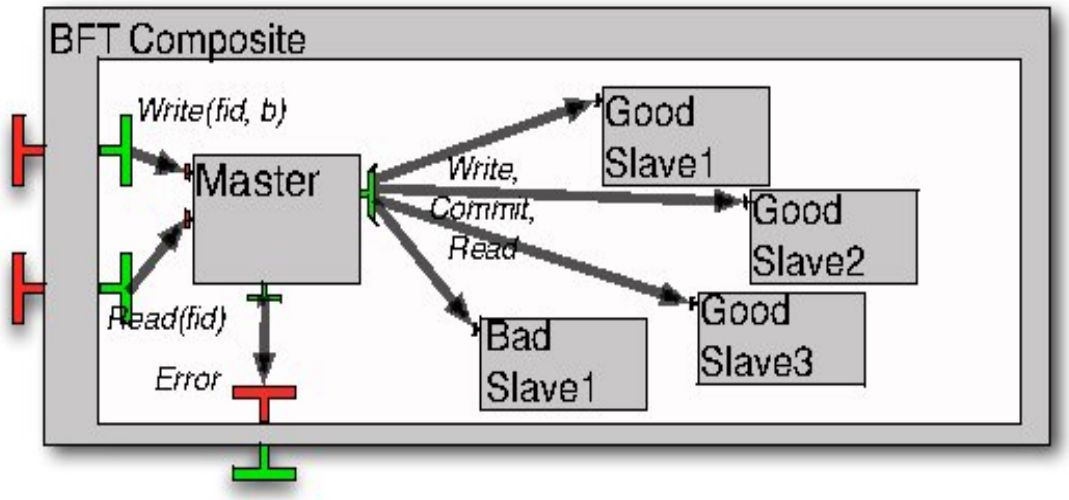


- **Very large Bibliography:**
- Algorithms
- Correctness proofs

Modelling a BFT application in GCM

- 1 composite component with 2 external services Read/Write.
- The service requests are delegated to the Master.

VCE diagram



- 1 multicast interface sending write/read/commit requests to all slaves.
- the salves reply asynchronously, the master only needs 2f+1 coherent answers to terminate

Our simplification hypothesis

1. **The master is reliable:** this simplifies the 3-phases commit protocol, and avoid the consensus research phase.
2. **The underlying middleware ensures safe communications:** faulty components only respond to their own requests, and communication order is preserved.
3. **To tolerate f faults** we use 3f+1 slaves, and require 2f+1 agreeing answers, as in the usual BFT algorithms.

Requirements (= logical properties)

1- Reachability(*):

- The Read service can terminate

∀ fid:nat among {0...2}. ∃ b:bool. <true* . {!R_Read !fid !b}> true

- Is the BFT hypothesis respected by the model ?

< true* . 'Error (NotBFT)'> true

2- Termination:

– After receiving a Q_Write(f,x) request, it is (fairly) inevitable that the Write services terminates with a R_Write(f) answer, or an Error is raised.

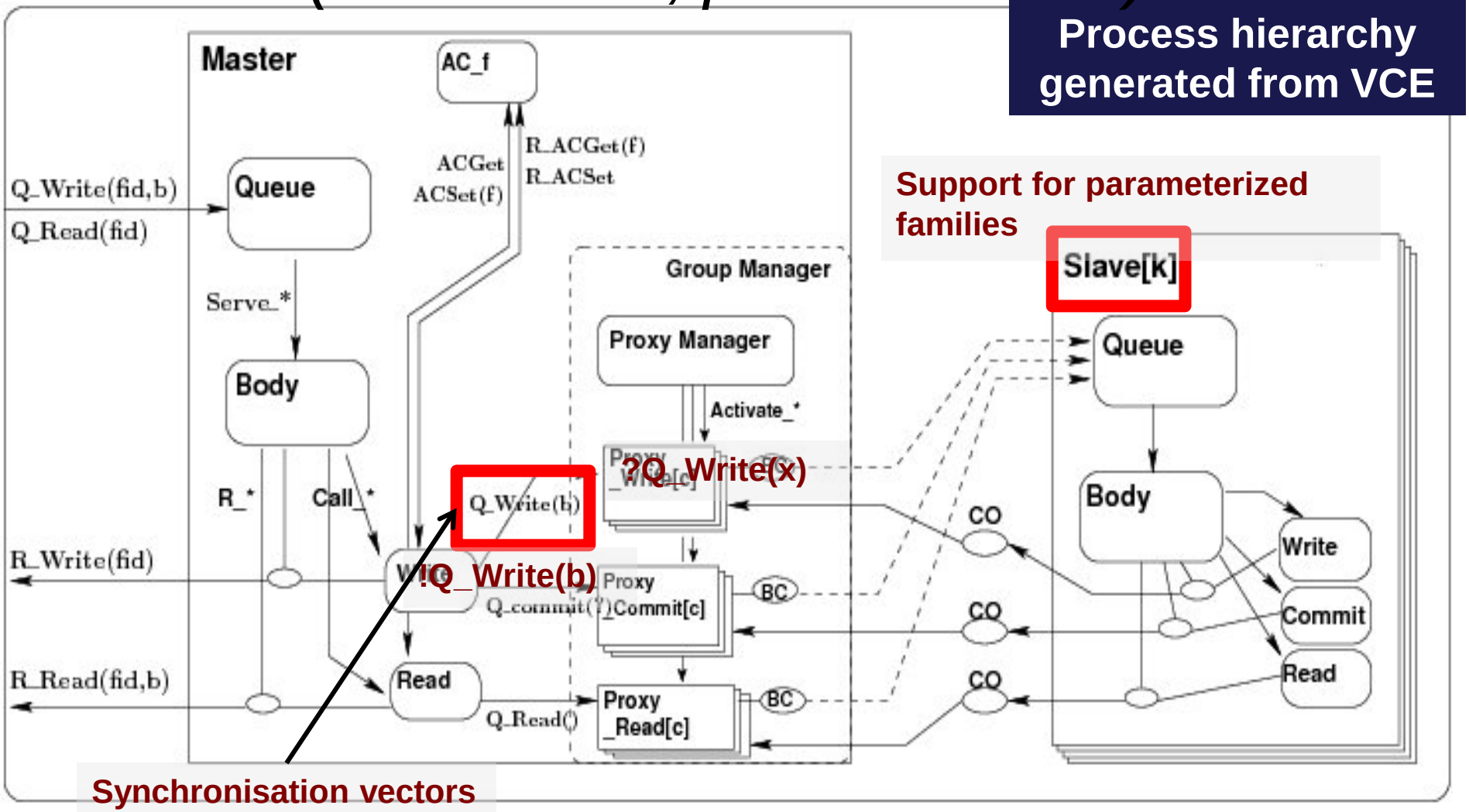
3- Functional correctness:

– After receiving a ?Q_Write(f1,x), and before the next ?Q_Write, a ?Q_Read requests raises a !R_Read(y) response, with y=x

(*) Model Checking Language (MCL), Mateescu et al, FM'08

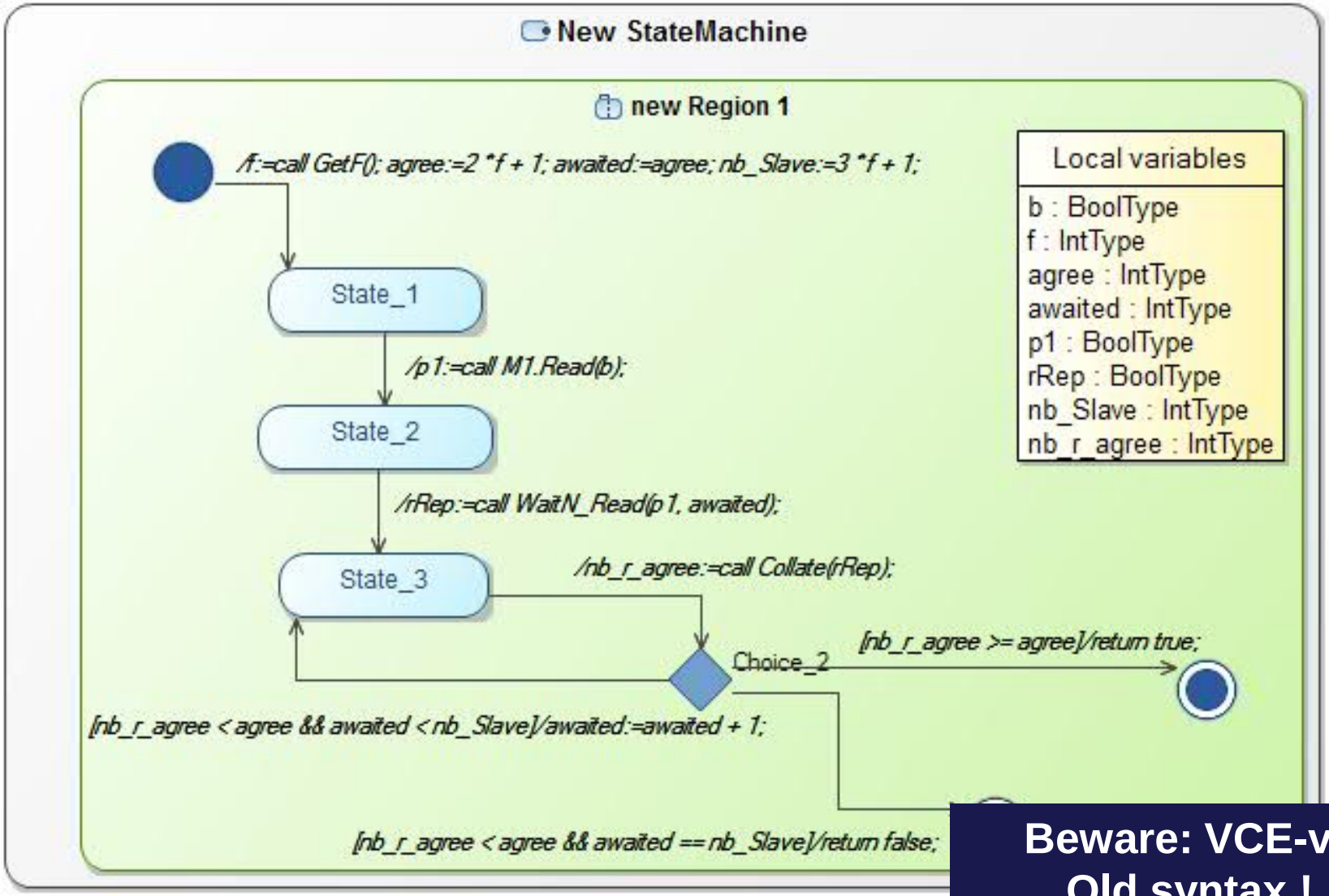
Semantic Model: network of processes (hierarchical, parameterized)

Process hierarchy generated from VCE



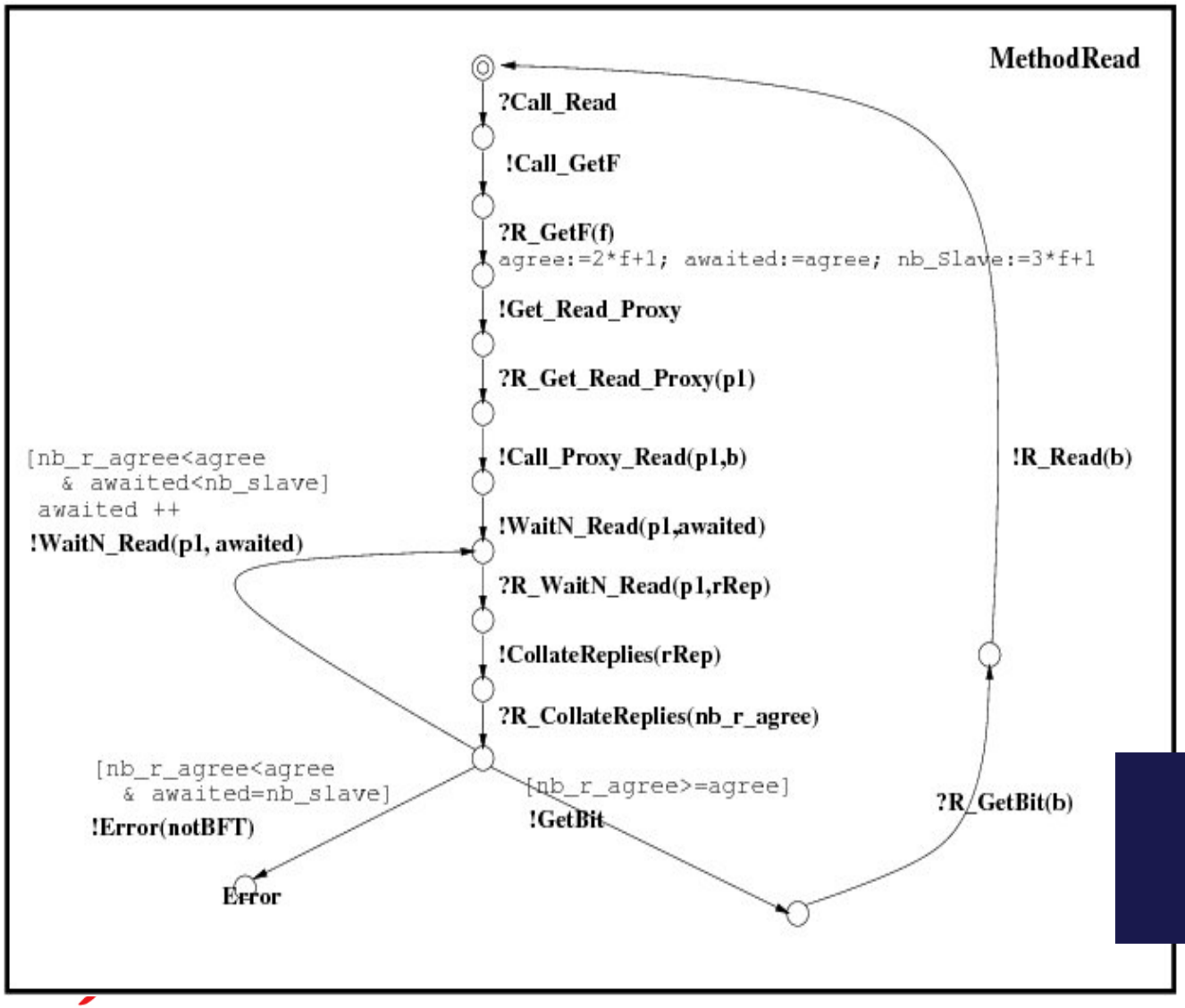
$$\langle -, -, -, -, -, (k \mapsto !Q\_m_n(f, arg), k' \mapsto ?Q\_m'_n(f, arg)) \rangle \rightarrow Q\_m_n(f, arg)$$

Semantics of State-machines ?



Beware: VCE-v3
Old syntax !

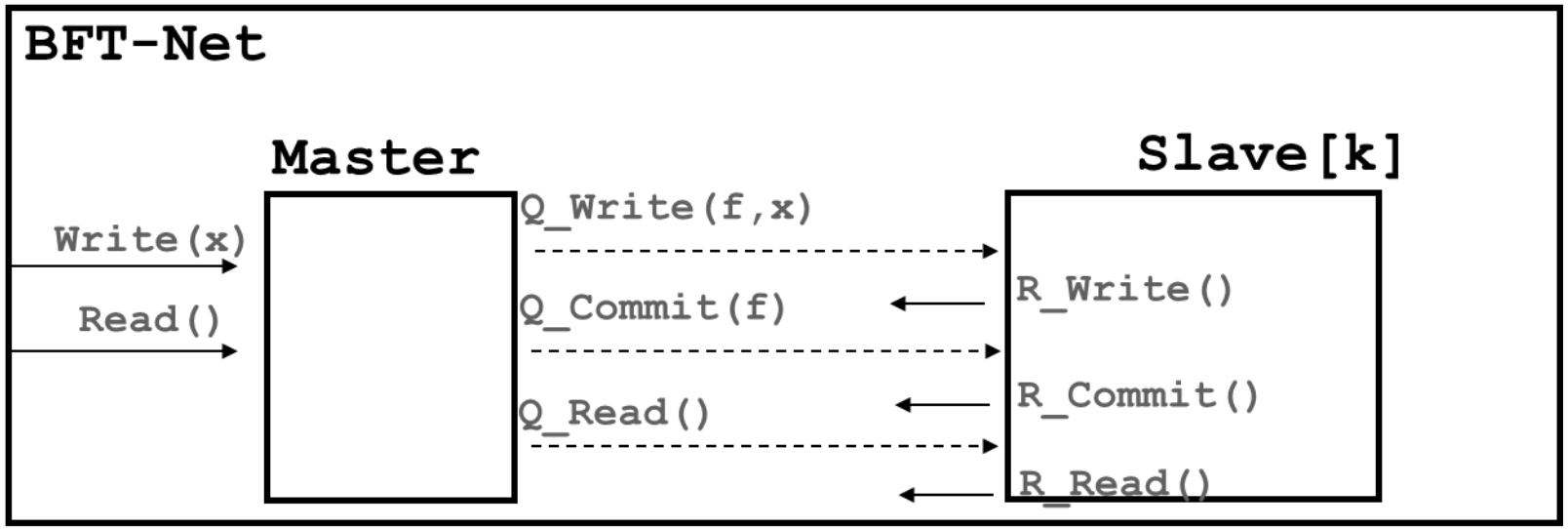
Basic Processes : parameterized LTSs



- **Labelled transition systems, with:**
- **Value passing**
- **Local variables**
- **Guards….**

pLTS generated from State Machines

Combining processes: generalized parallel operator



BFT-Net : < Master, Slave_1,…,Slave_n > k ∈ [1:n]

with synchronisation vectors :

<?Write(x), - , …, - > => ?Write(x)

<!Q_Write(f,x), ?Q_Write(f,x) , …, ?Q_Write(f,x) > => Q_Write(f,x)

∀k <?R_Write(f,k), - , …, !R_Write(f), …, - >

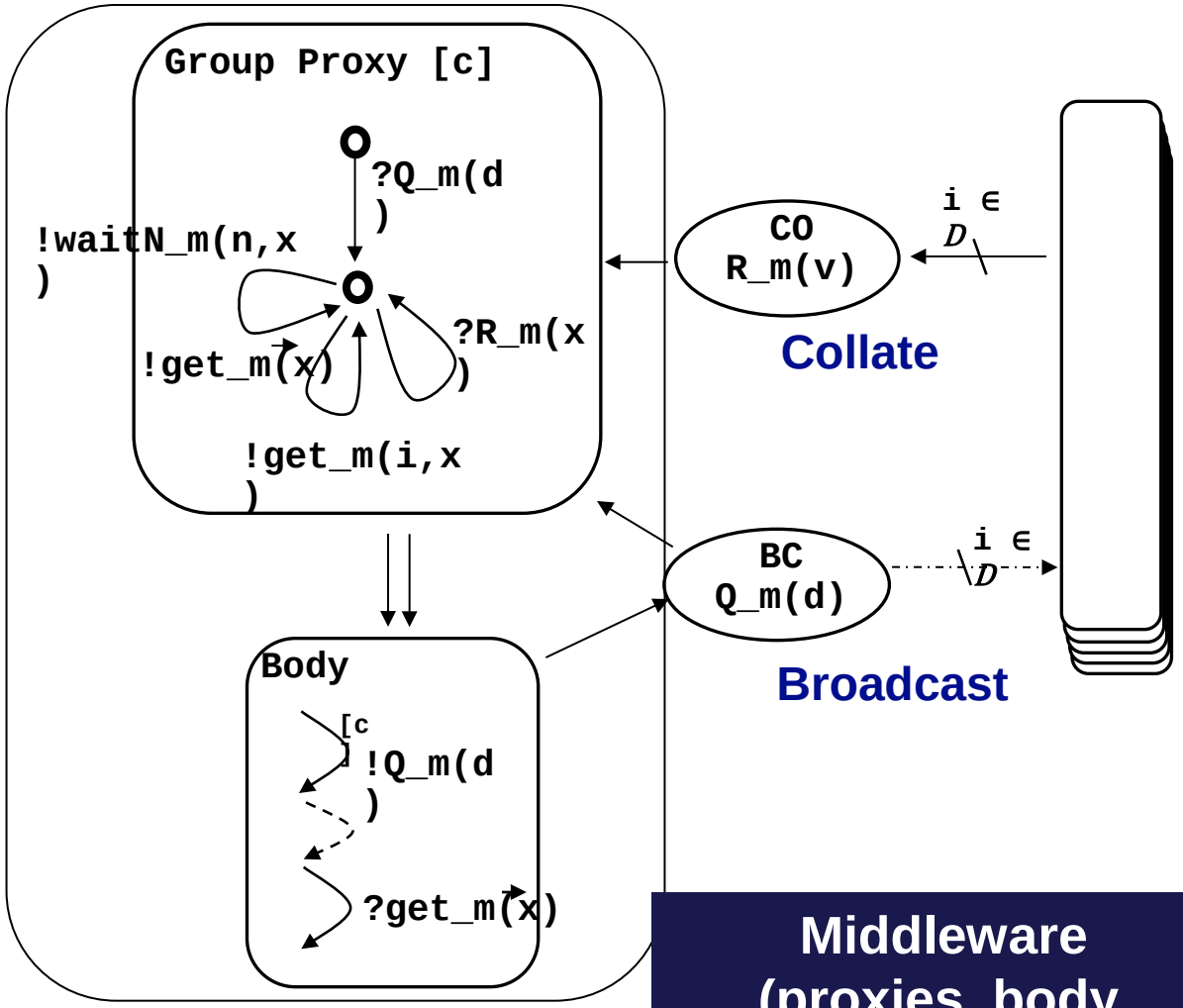
EXP format (product of automata) generated by VCE

Building pNet models (3)

- **Proxies for Asynchronous group requests**

- **manage the return of results, with flexible policies:**

- **Vector of results**
- **First N results**
- **Individual results**



Middleware (proxies, body, queues) generated by VCE

Generation of state-space

Taming state-space explosion:

Data abstraction (through abstract interpretation):

integers => small intervals

arrays … => reduce by symmetry arguments.

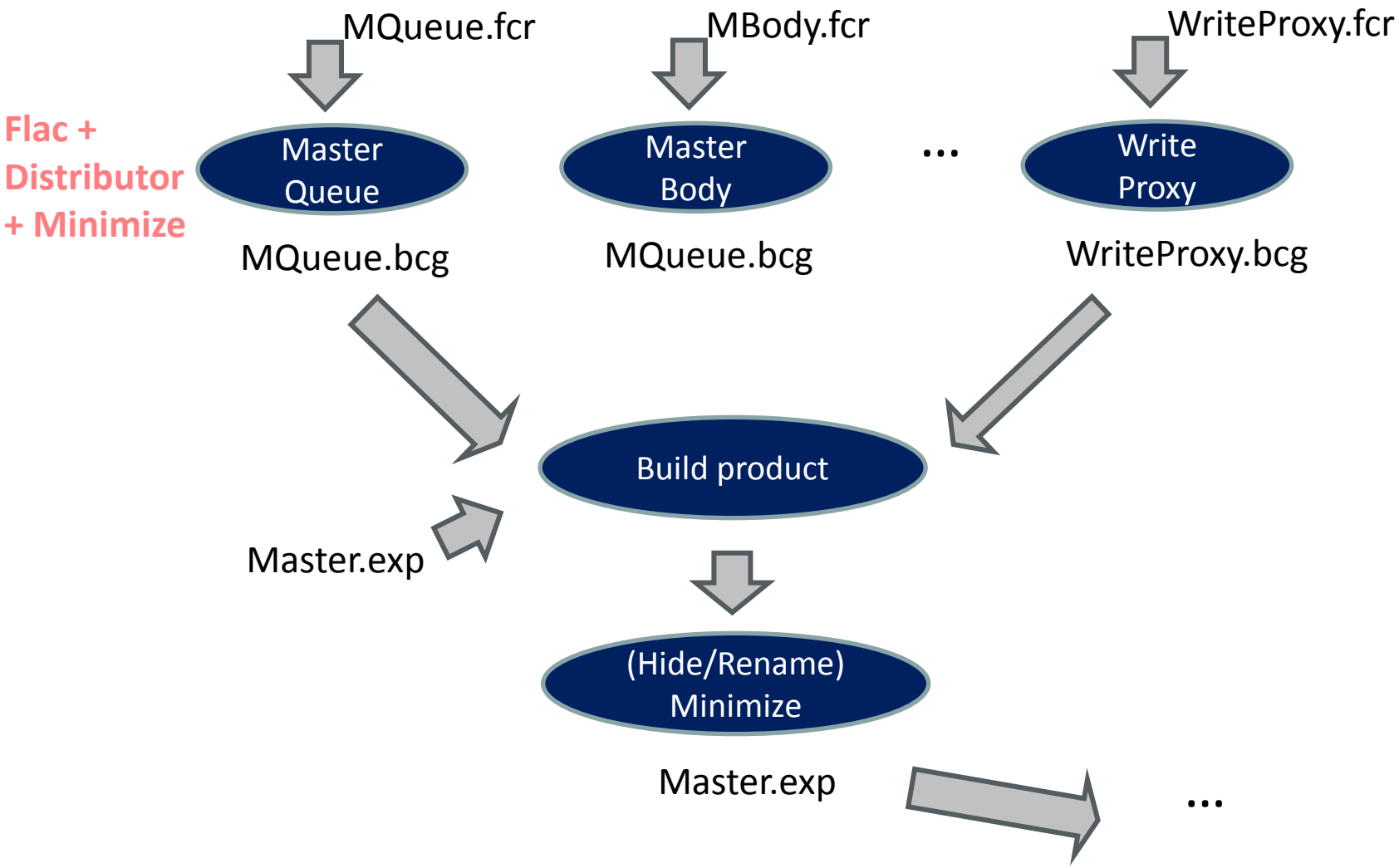
Partitioning and minimizing by bisimulation + context specification

Distributed verification engines,

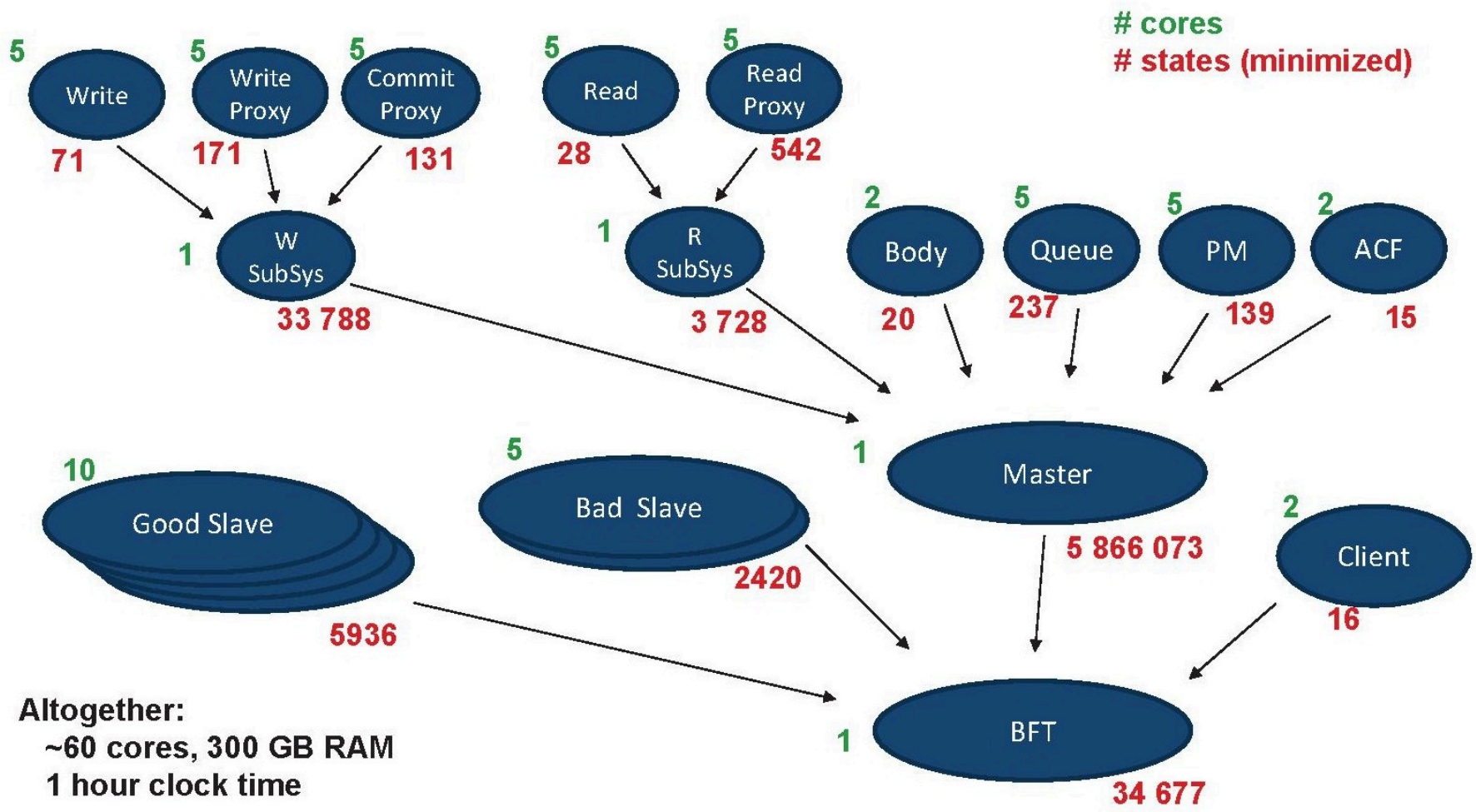
Only partially available (state-space generation, but no M.C.).

3 Tbytes of RAM ?

State-space generation workflow



State-space generation workflow



Distributed State generation

Abstract model: (for a fixed valuation of the parameters):
f=1, (=> 4 slaves), |data|= 2, |proxies|=3*3, |client requests|=3

Master queue size = 2

~100 cores, max 300 GB RAM

- **System parts sizes (states/transitions):**

Queue	Largest intermediate	Master	Good Slave	Global
237/3189	524/3107	5M/103M	5936/61K	34K/164K

Time
59’

Estimated brute force state spaces :

10^{18}	6.10^{3}	~ 10^{32}

All properties above have been successfully proven

Some Research Challenges

1. **Tooling :**
   - hide model-checker internal languages,
   - show results in user-level formalism
2. **Scale-up:**
   - better abstraction techniques for useful datatypes (e.g
   - Compositional & symbolic techniques,
   - minimization of the size of the semantic model
3. **Verifying dynamic distributed systems (GCM + R**
   - handle Life-cycle and Binding Controllers,
   - encode sub-component updates,
   - several orders of magnitude bigger.

1 Postdoc Subject…

1 Master Internship

1 PhD Subject

1 Master Internship

=> In the long term, combine model-checking & theorem-proving

Conclusion

1) Component frameworks: provide method, tools, middleware for programming large-scale applications

2) Vercors: an example of a modeling+verification framework for component-based applications

3) Model-checking distributed component systems: large use-cases – methods for mastering state-space explosion

http://team.inria.fr/scale/Vercors

+ Available master/PhD subjects at INRIA/Scale
+ International Master track at University of Nice Sophia-Antipolis

More References

Fractal & GCM:

- http://fractal.objectweb.org/ [doc tutorials]
- http://www-sop.inria.fr/members/Eric.Madelaine/Teaching/Ubinet2010/2006-GCM-GridsWork.ppt
- F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, C. Perez: *GCM: A Grid Extension to Fractal for Autonomous Distributed Components,* in Annals of Telecommunications, Vol. 64, nº1, jan 2009.
- pNet [def & semantics]: *http://hal.inria.fr/hal-00761073 + PDP-4Pad'15*

Overview:

E. Madelaine, *Specification, Model Generation, and Verification of Distributed Applications,* sept. 2011, URL: http://hal.inria.fr/index.php?halsid=o3253cd31tsjbo0bo40ogqas53&view_this_doc=tel-00625248

Vercors:

http://team.inria.fr/scale/Vercors/software/VCEv4 (download, examples)

Case-studies:

- R. Ameur-Boulifa, R. Halalai, L. Henrio, E. Madelaine, *Verifying Safety of Fault-Tolerant Distributed Components - Extended Version, RR INRIA #7717, sept 2011, URL: http://hal.inria.fr/inria-00621264/*
- Nuno FACS'13
- Foclasa'14

Thank you

谢谢

Merci

Papers, Use-cases, and Tools at :

http://team.inria.fr/scale/software/vercors

Hands-on Session

- Introduction: exercices from the tutorial example
  - Interfaces, attributes, types, state-machines
- BFT application
- Composites, Multicast, Matrix
- Static Semantics Validation
- Use-case: Intelligent cars system
  - Build a full application with its architectural and behavioral diagrams.

Installation

- VCE-v4 installed here for windows platforms. ***Alpha-version available only since 2015, June : Experimental !***
  - ***Please report any bugs/problems/badly documented features***
- Prerequisite: Java7 (jre or jdk) installed

Documents:

Tutorial

Exercices

Thanks a lot

to Siqi Li, and E.C.N.U. & S.E.I. support

BFT

1. Go to the BFT project, open the VCE architecture diagram
2. Validate the diagram… what is wrong?
3. Complete the necessary elements until the diagram is valid.
4. Add a new Server Interface named “Set F” to the BTF composite component.
5. Go to the UML Class diagram, add an UML Interface , containing an operation named SetF, with an input argument of type NatType
6. Attach this Interface to the “Set F” server interface.
7. Do the same (add a server interface, attach UML interface) for the Master Primitive component
8. Build the requested bindings… What is missing ?
9. Check the Diagram Validity.

Composite

- Create a new “Composite”project. 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 ADL

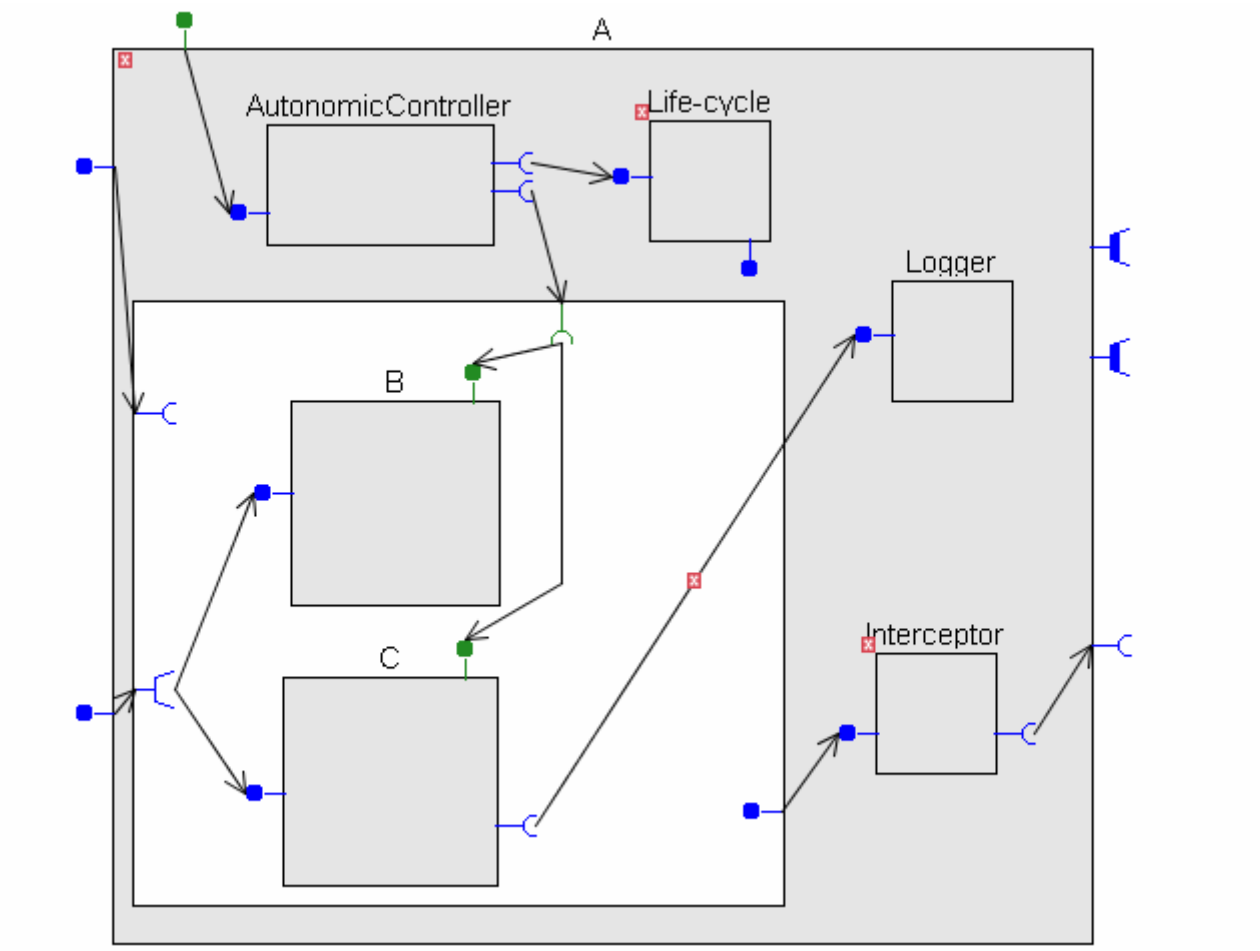
Composite, multicast, matrix

Build a composite component, with:

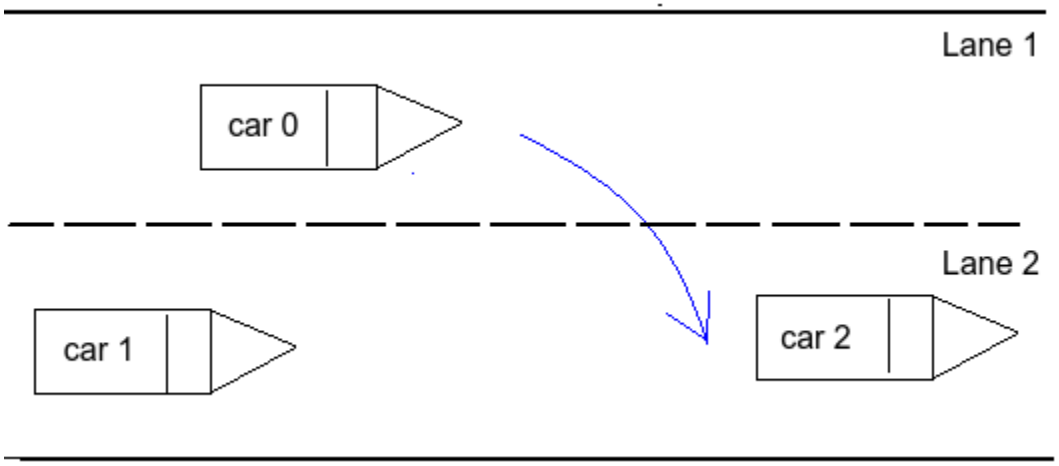
- One server interface, with an internal multicast interface
- 2 x 3 subcomponents representing matrix blocks, each linked to its left and up neighbours

Exercice

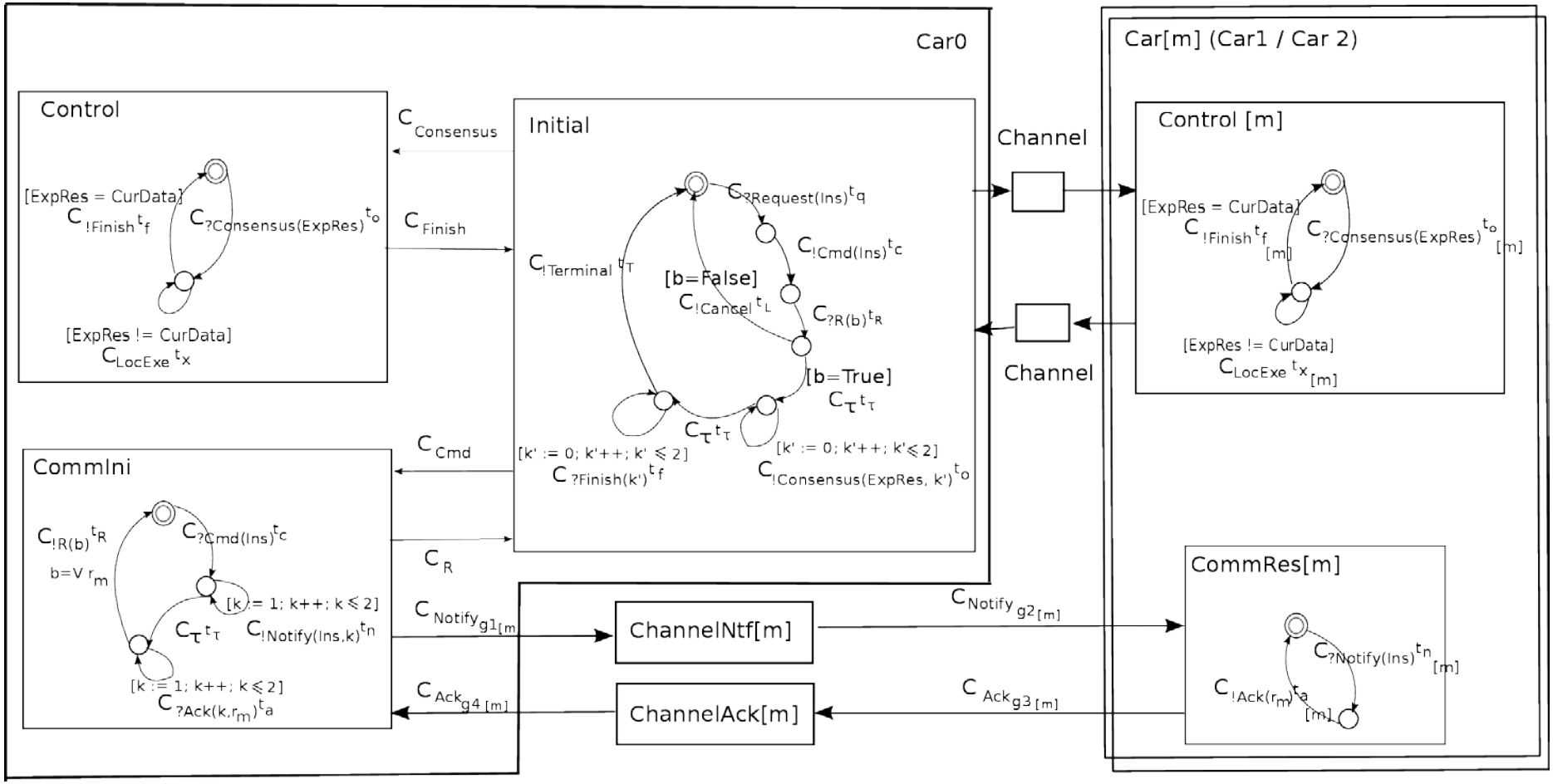
- Analyze this diagram (semantics, errors, …)



Intelligent cars scenario
(from Chen Yanwen PhD research)



System Architecture



Specific timed model transition language

1. This use-case was originally built for a timed version of the semantic formalism.
2. But in the exercises today, only consider the untimed fragment, by dropping the clocks and the time variables.

   E.g. in the CommIni component:

   $C \xrightarrow{?Cmd(Ins)} t_C$ ➔ ?Cmd(Ins)

3. GCM : RPC style here versus message oriented

   (= no return value)

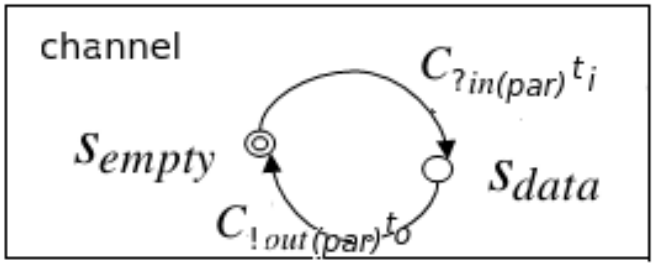
Exercice 1: Architecture

1. Build a VCE architecture diagram, for the Car0 component, with its 3 subcomponents:
   a. Only the architecture (components, interfaces, bindings) in this first step.
   b. Respect the interface names.

2. Add a service interface accepting messages from the car driver. Name it “Driver”.
3. Check the diagram validity.

Exercice 2: Channels

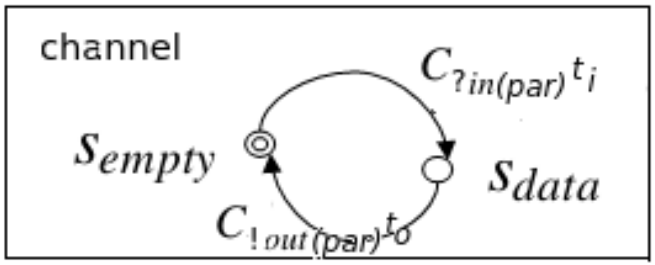
1. Channels here are primitive components with a specific behavior template:



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.

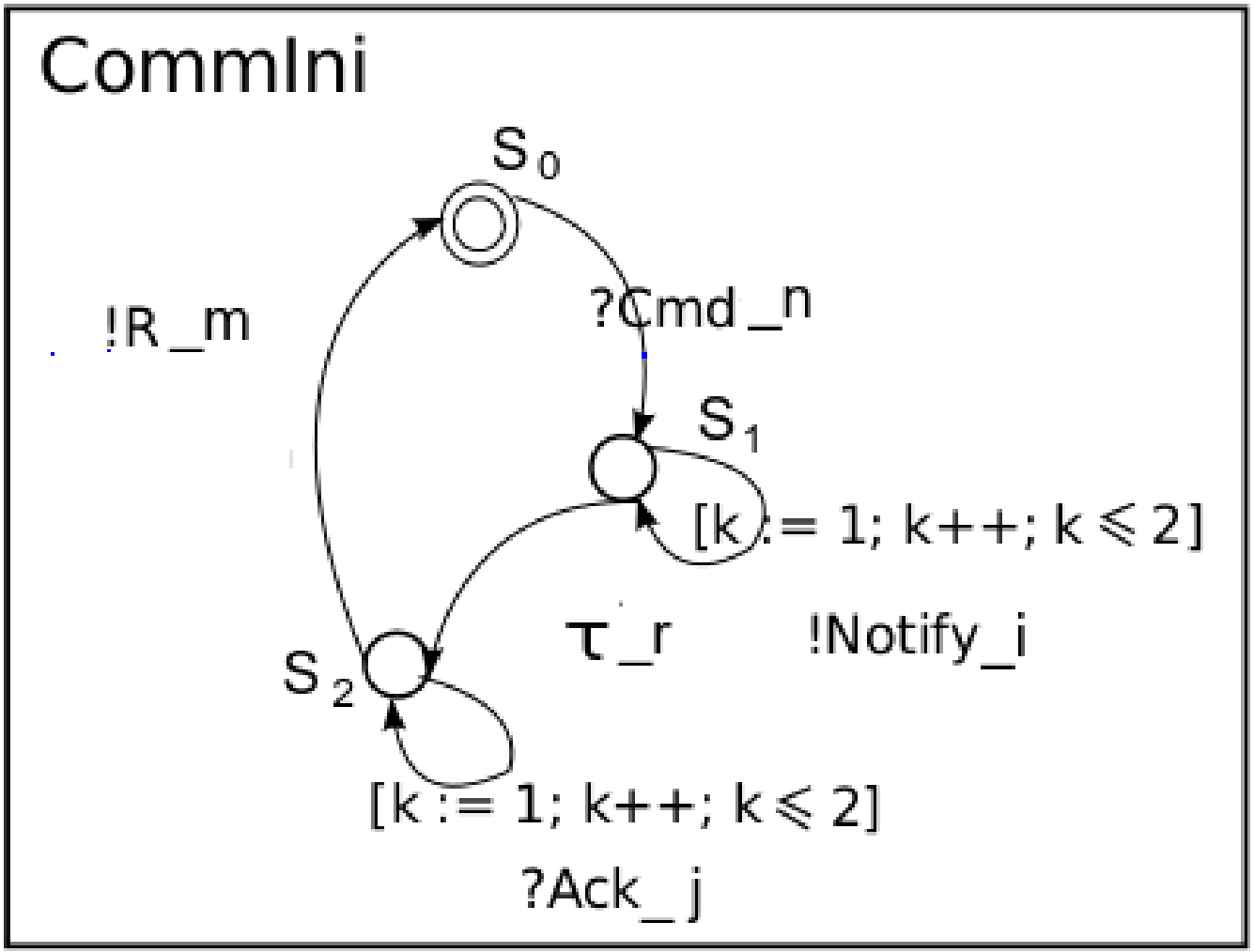
Exercice 3: Channels behavior

See Tutorial Section 2.9



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.

Exercice 4:
CommIni component



Exercice 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.