Specification and Verification for Grid Component-based Applications

E. Madelaine

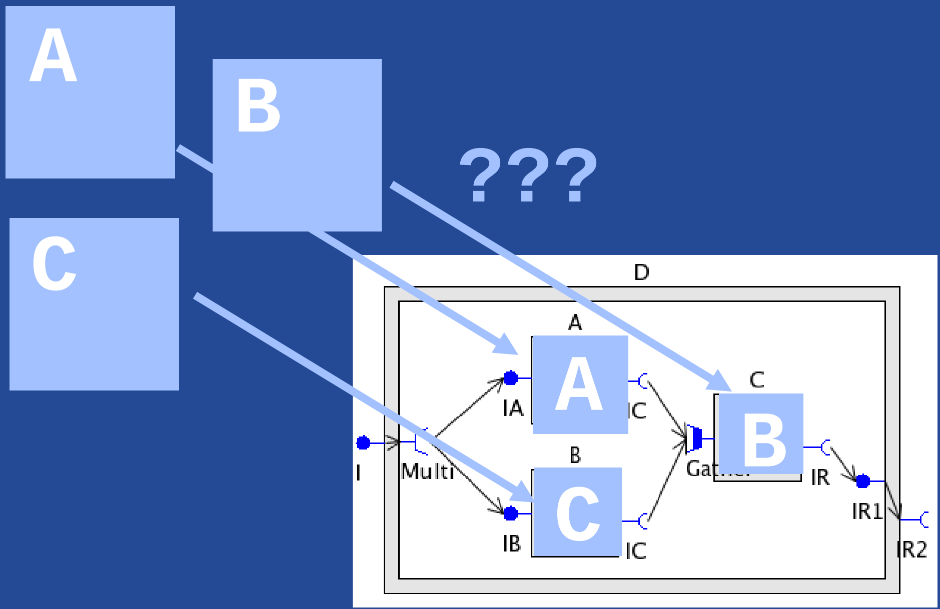
GridComp project

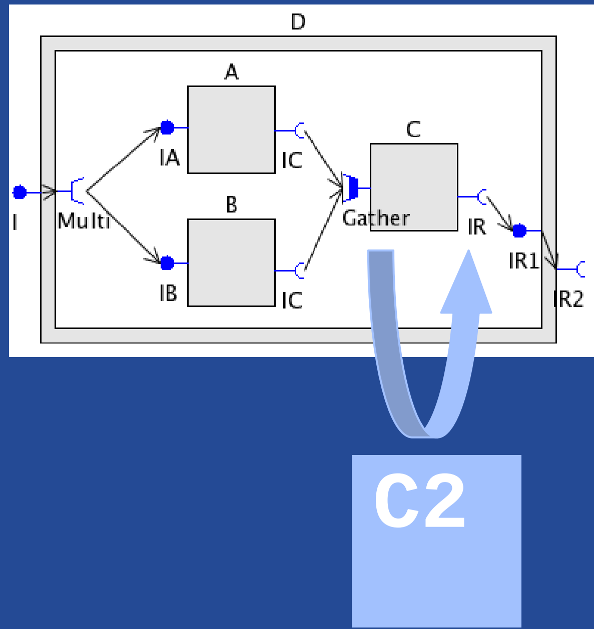
Oasis team

INRIA -- CNRS - I3S -- Univ. of Nice Sophia-Antipolis

FMCO '08 Sophia-Antipolis – oct. 21-23, 2008

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

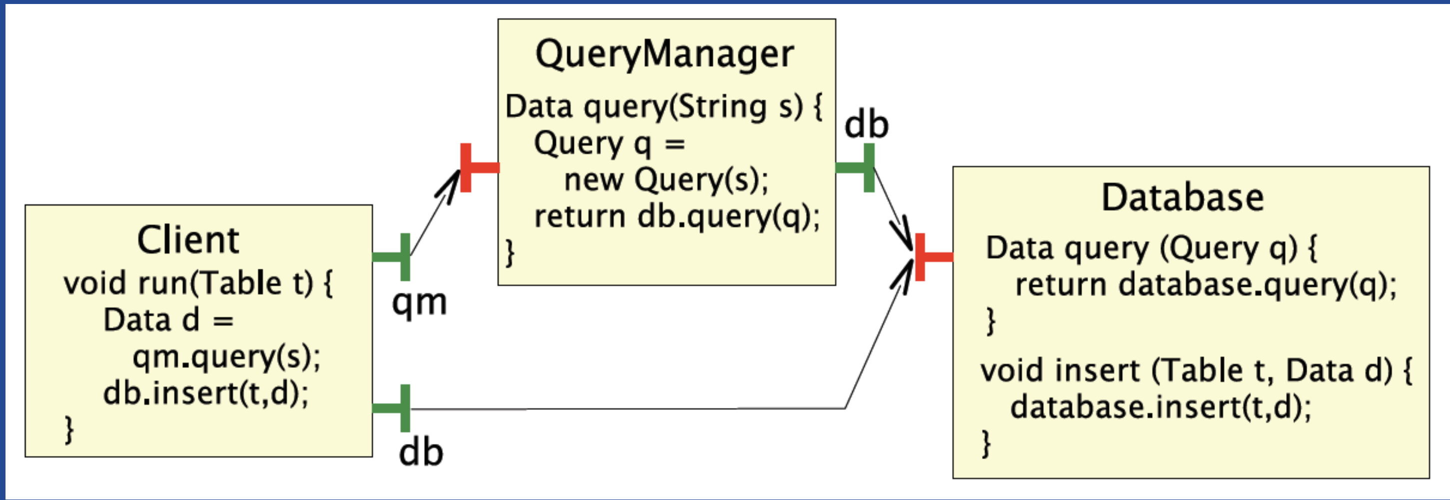




Safe COTS-based development
=>
Behaviour Specifications

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.

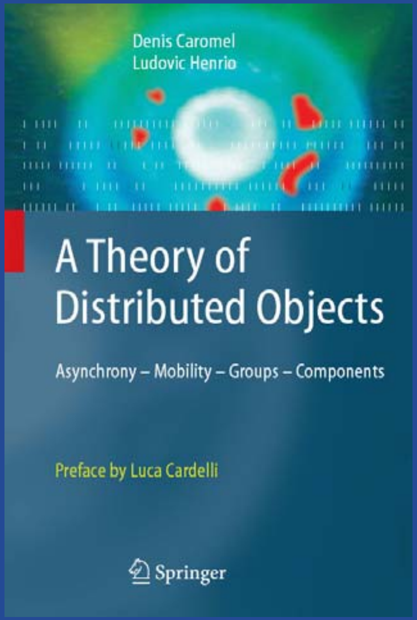
Transparent Futures do not solve all inter-component deadlocks

Agenda

- **Context**
  - **Active Objects, Components and Grids**
- **Safe Distributed Components**
  - **Definitions**
  - **Model generation *(= operational semantics)***
  - **Checking Properties**
- **Specification and Verification Tools, Case Study**
- **Conclusion & Perspectives**

Asynchronous and Deterministic Objects

[Denis Caromel – Ludovic Henrio]



ASP (Asynchronous Sequential Processes) =

- Distributed Active Objects
- Asynchronous method calls
- Futures and Wait-by-necessity

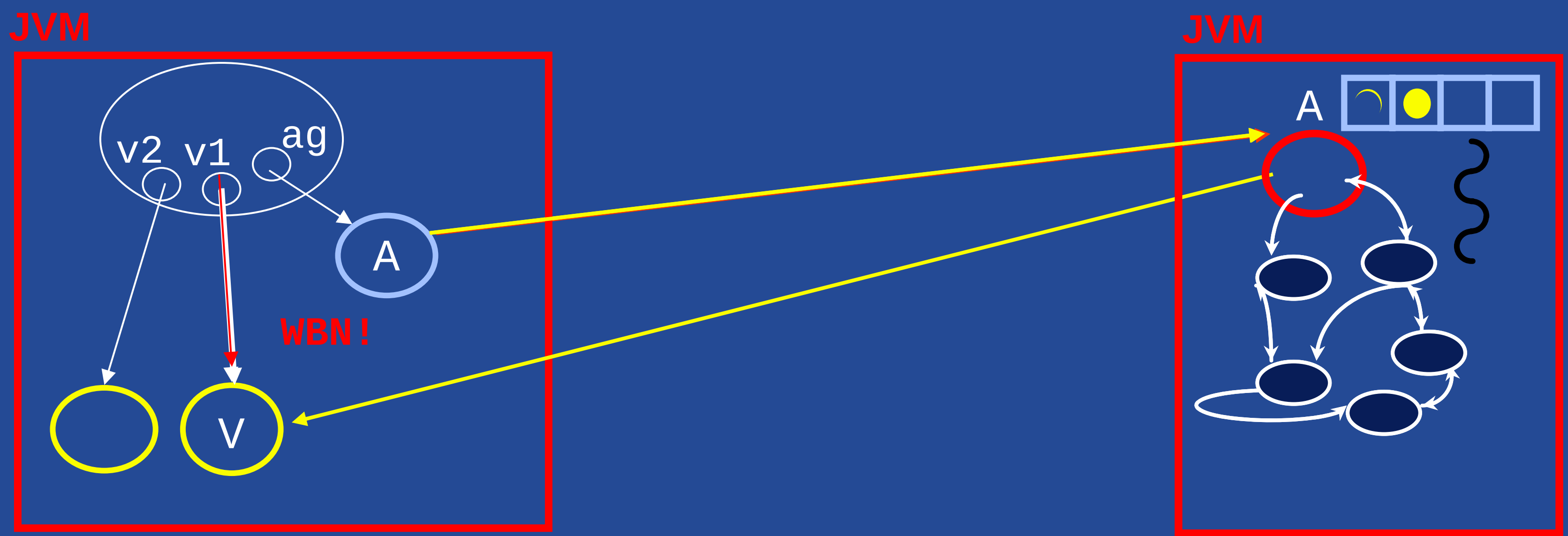
→ Determinism/Confluence properties

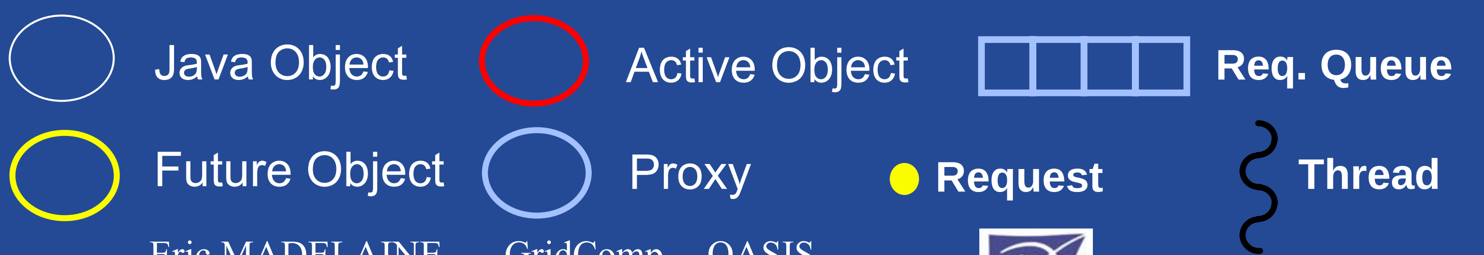
→ Programming abstractions

→ Formal Basis for Verification

ProActive : Active objects

```
A ag = newActive ("A", […], VirtualNode)
V v1 = ag.foo (param);
V v2 = ag.bar (param);
...
v1.bar(); //Wait-By-Necessity
```

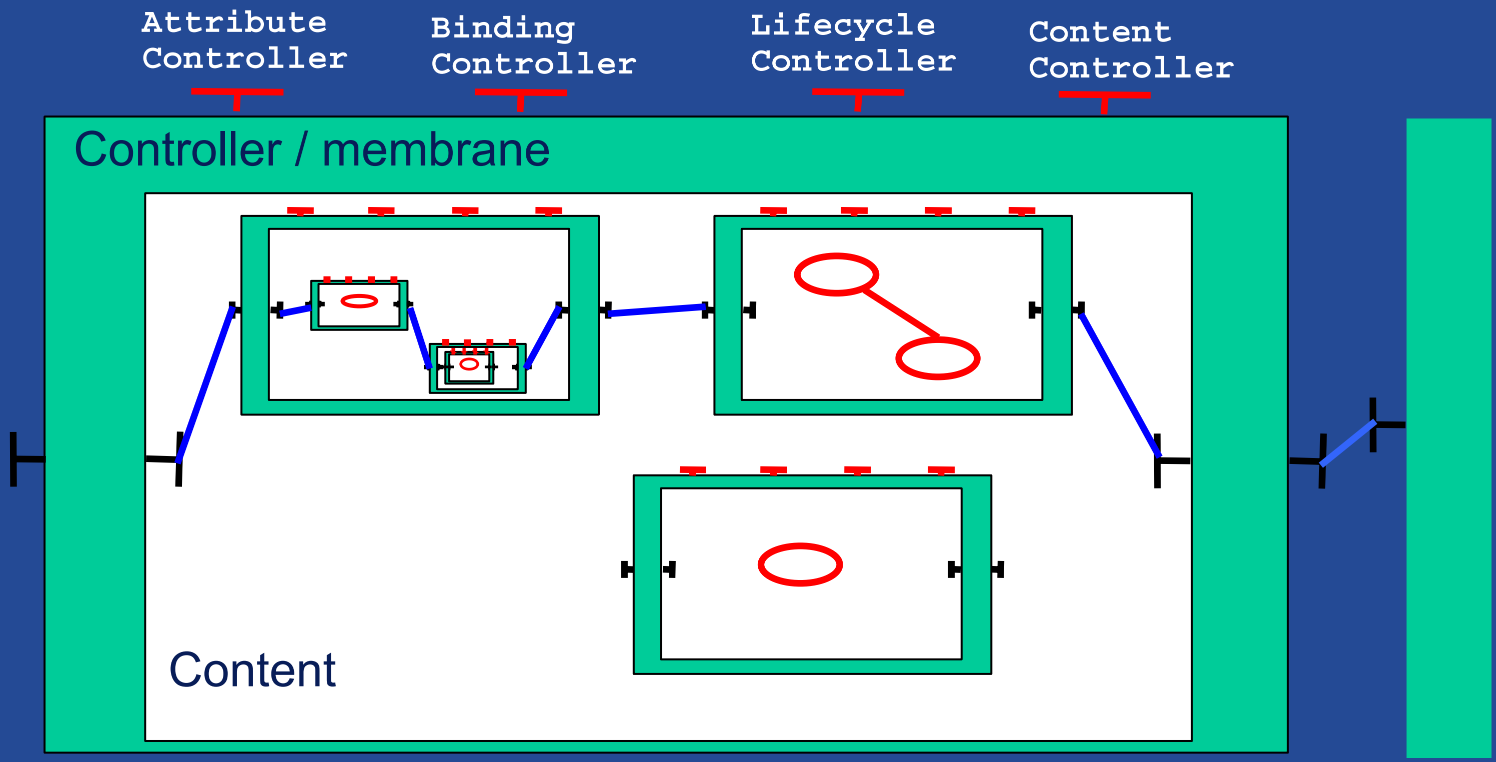




Wait-By-Necessity is a Dataflow Synchronization

Fractal hierarchical model :

- Provided/Required Interfaces
- Hierarchy
- Separation of concern: functional / non-functional
- ADL
- Extensible



composites encapsulate primitives, which encapsulates code

GCM

[Caromel, FMCO'07]

Scopes and Objectives:

Grid Component Model

Extension of Fractal for programming Grids

Innovations:

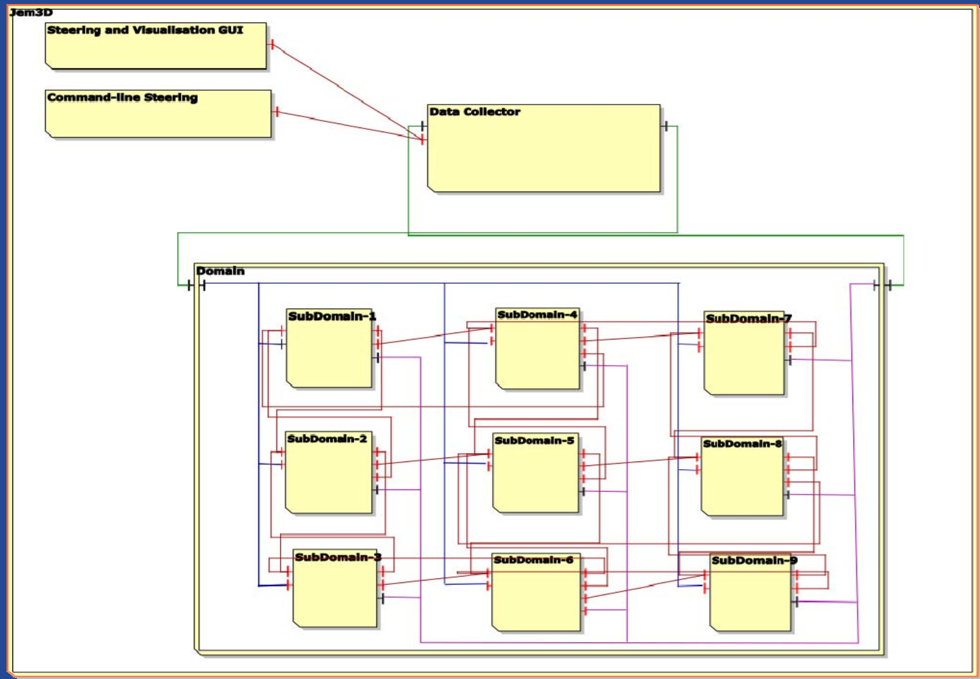
Abstract Deployment

Multicast and GatherCast

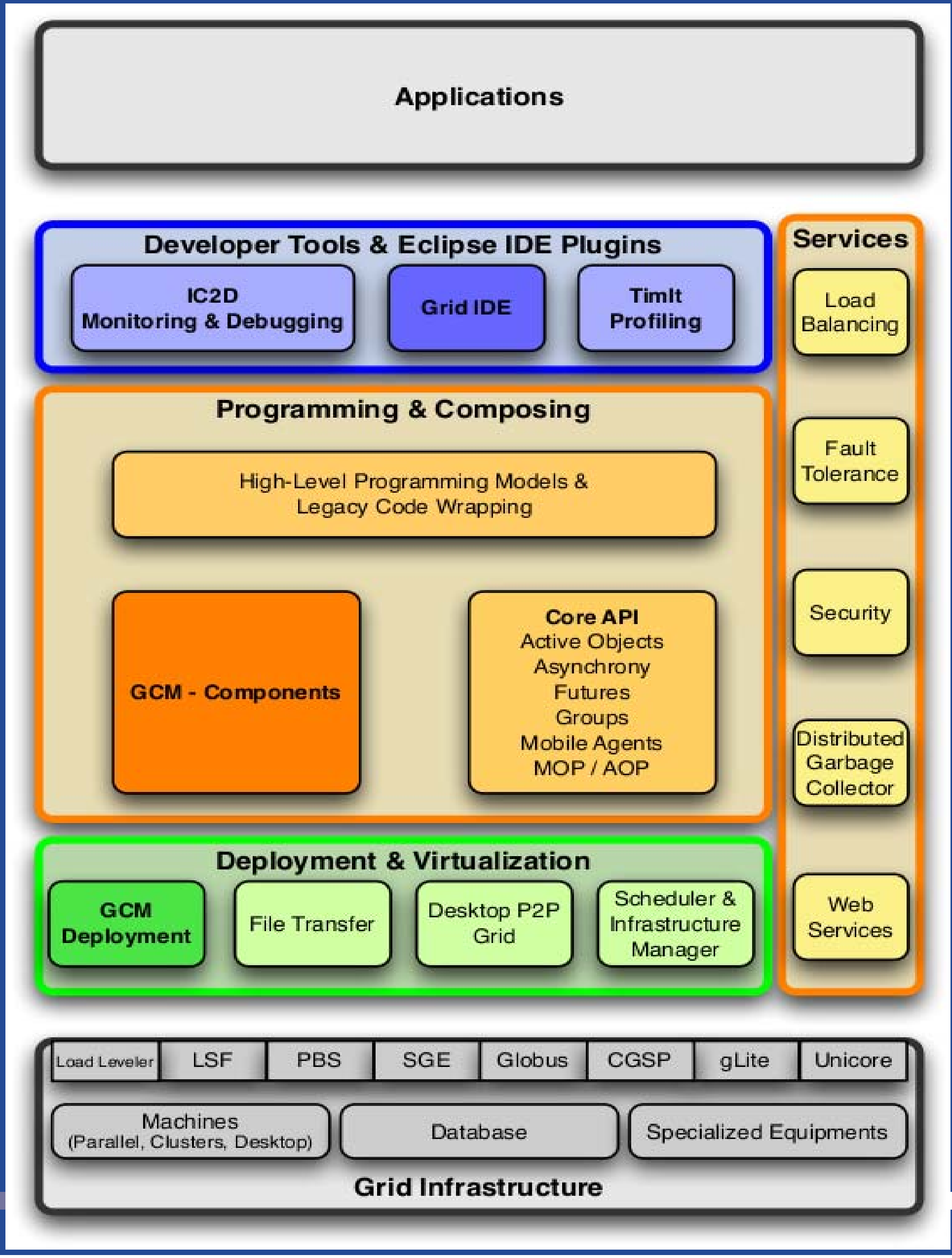
Controller (NF) Components

Standardization

By the ETSI TC-GRID



ProActive Parallel Suite



Spin-off company 2007 :

ActivEon
SCALE BEYOND LIMITS

Agenda

- Context
  - Active Objects, Components and Grids
- **Safe Distributed Components**
  - **Definitions**
  - Model generation
  - Properties
- Specification and Verification Tools, Case Study
- Conclusion & Perspectives

Software Components

My Definition :

Software modules, composable, reconfigurable, with well-defined interfaces, and well-defined black box behaviour

Our interests :

1. **Encapsulation**
   Black boxes, offered and required services
2. **Composition**
   Design of complex systems, hierarchical organization into sub-systems
3. **Separation of concerns**
   Architecture Description Language (ADL), management components
4. **Distribution (e.g. Computational Grid)**
   Interaction at interfaces through asynchronous method calls

Behaviour specification and Safe composition

Aim :

Build reliable components from the composition of smaller pieces, using their formal specification.

Component paradigm : only observe activity at interfaces.

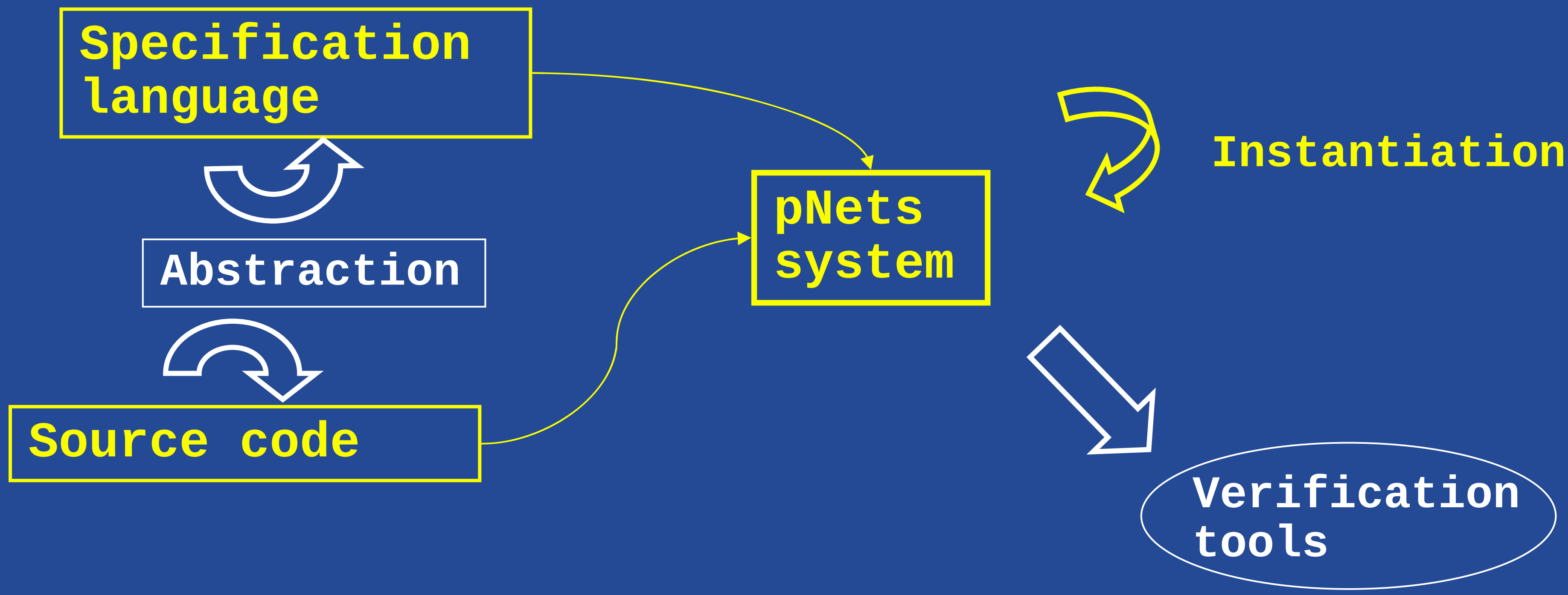
Behavioural properties:

Deadlock freeness, progress/termination, safety and liveness.

Applications :

- Check behavioural **compatibility** between sub-components
- Check correctness of component **deployment**
- Check correctness of the **transformation** inside a running application.

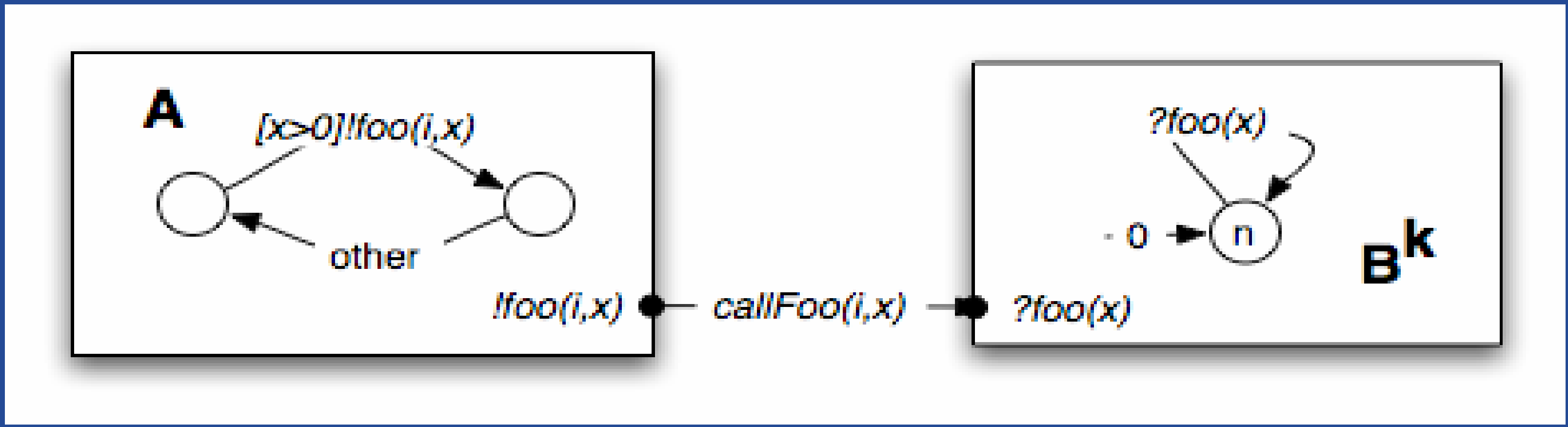
The pNET model



Constraint: domains in pNets are “simple types”.

The data domains in the source language have to be abstracted beforehand.

pNets : Hierarchical and Parameterized Models



[Arnold, Nivat 92] ***Synchronization networks***

[Lin 92] ***symbolic graphs with assignments***

[Lakas 96] ***semantics of Lotos open expressions***

- Value-passing, Dynamic architectures, etc.
- But close to code structure
- Instantiation to finite structures (through abstract interpretation)

[Forte'04: T. Barros, R. Boulifa, E. Madelaine]

[Annals of Telecomunications'08: A. Cansado, L. Henrio, E. Madelaine]

Parameterized LTSs : definition

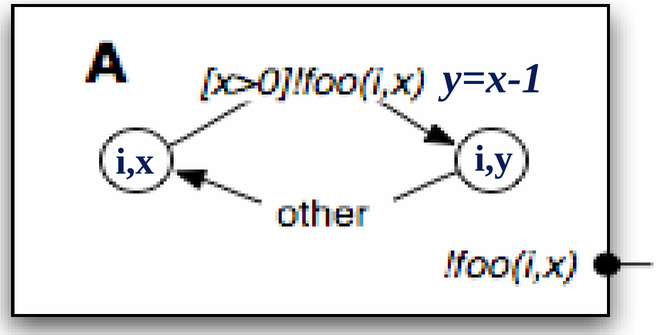
Given :

A set of parameters V (with domains in first order “simple types”)

An many-sorted term algebra $\sum_{V,}$ with a distinguished Action sort

A parameterized LTS is $<V, S, s_0, L>$ in which:

- Each state $s \in S$ has free variables $fv(s) \subseteq V$
- Labels $l \in L$ have the form $<e_B, \alpha, x_j := e_j>$
    - $e_B \in \sum_{V,Bool}$ a guard
    - $\alpha \in \sum_{V,Action}$ a parameterized action
    - $x_j := e_j$ an assignment of the state variables



Parameterized Network of Synchronised Automata (pNets) : definition

- A pNet is $<V, A_g, J, p_j, O_j, T>$ in which:
  - $A_g \subseteq \Sigma_V$ is the pNet sort, ie the set of global actions
  - J is a set of Holes, each of them with a parameter p_j and a sort O_j
  - T is the transducer (or control automaton) of the pNet, whose labels are synchronisation vectors :

    $<\alpha_g, \{a_{i,j}\}>$ that relate actions of some instances of the holes to a global action.

PhiloNET : $<$ Philo[k], Fork[k] $>$ $k \in [1{:}n]$

$A_g = \{$ Think(k), TakeL(k), … $\}$

T static (single state), with synchronisation vectors :

$<\text{Think(k)}, \text{Think}_{\text{Philo[k]}}>$

$<\text{TakeL(k)}, \text{TakeL}_{\text{Philo[k]}}, \text{Take}_{\text{Fork[k-1]}}>$

pNets and Nets : operators

- pNets are **generalized synchronisation operators** at the semantic level, in the spirit of Lotomaton.

  They address: multiway synchronisation, parameterized topologies, and dynamic topologies.

Define:

- A **System** is a tree-like structure with pNets at nodes and pLTS at leaves
- **Abstraction**: given a countable (resp, finite) domain for each parameter of a system, its **instantiation** is a countable (resp. finite) system.
- The **synchronisation product** is only defined for instantiated systems.

Abstractions and Correctness

(1) Program semantics ==> Behaviour Model (parameterized)

user-specified abstract interpretation

(2) Behaviour Model ==> Finite Model

Value Passing case : define an abstract representation from a finite partition of the value domains, on a per-formula basis

$\Rightarrow$ Preservation of safety and liveness properties [Cleaveland & Riely 93]

Families of Processes : no similar generic result (but many results for specific topologies).

Counter-example : on parameterized topologies of processes, reachability properties require induction reasoning.

Practical approach :

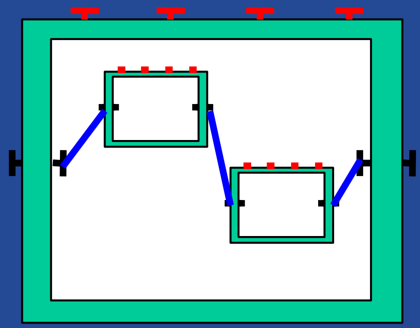
- explore small finite configurations in a “bug search” fashion,
- use “infinite systems” techniques for decidable domains when available

Agenda

- Context
  - Active Objects, Components and Grids
- **Safe Distributed Components**
  - Definitions
  - **Model generation**
  - Properties
- Specification and Verification Tools, Case Study
- Conclusion & Perspectives

Building Behavioural Models : Principles

For a given language/framework, define an operational semantics that builds pNets from the program structure.



For GCM components:

- Reason separately at each composition level

Primitive components : functional behaviour is known

- Given by the user (specification language)
- Obtained by static analysis (primitive components, e.g. ProActive active objects)
- Computed from lower level

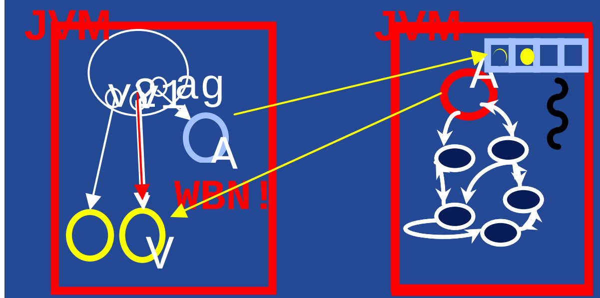
Composites : structure and non functional behaviour automatically added from the component's ADL

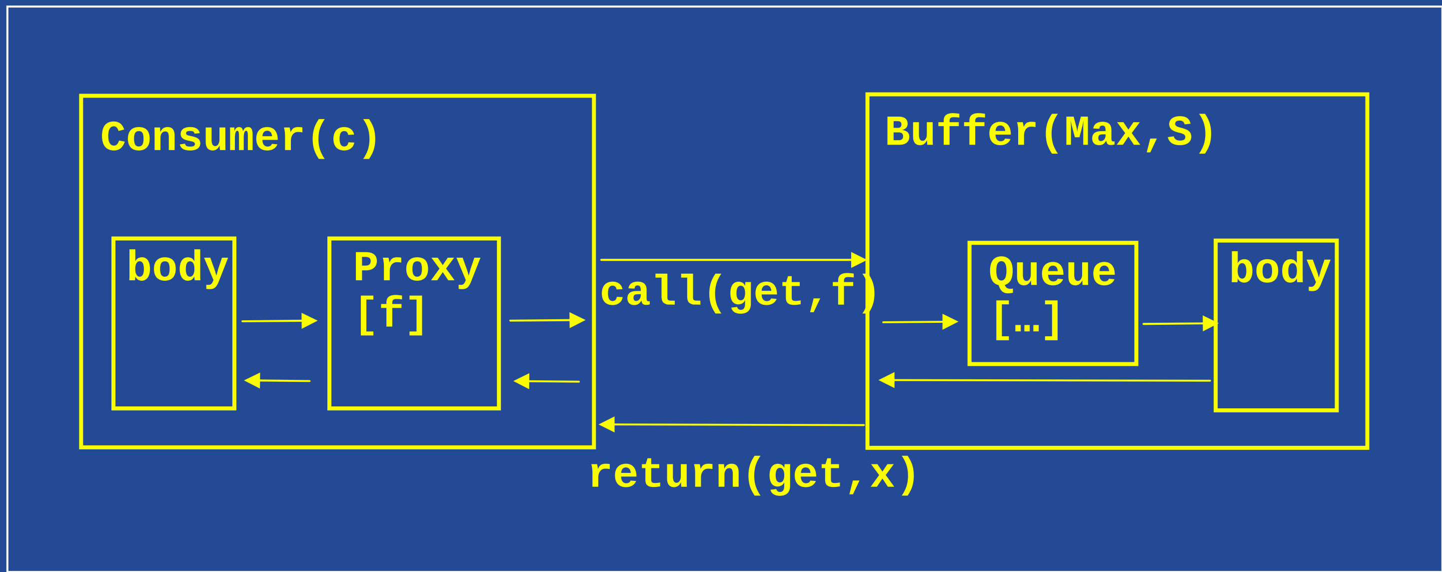
Building pNet models (ex 1)

Nets for **Active objects communication** schema :

From the set of public methods, and their signature, build :

- The (parameterized) action algebra
- The structure of the future proxies and request queue
- One synch vector per exchanged message.

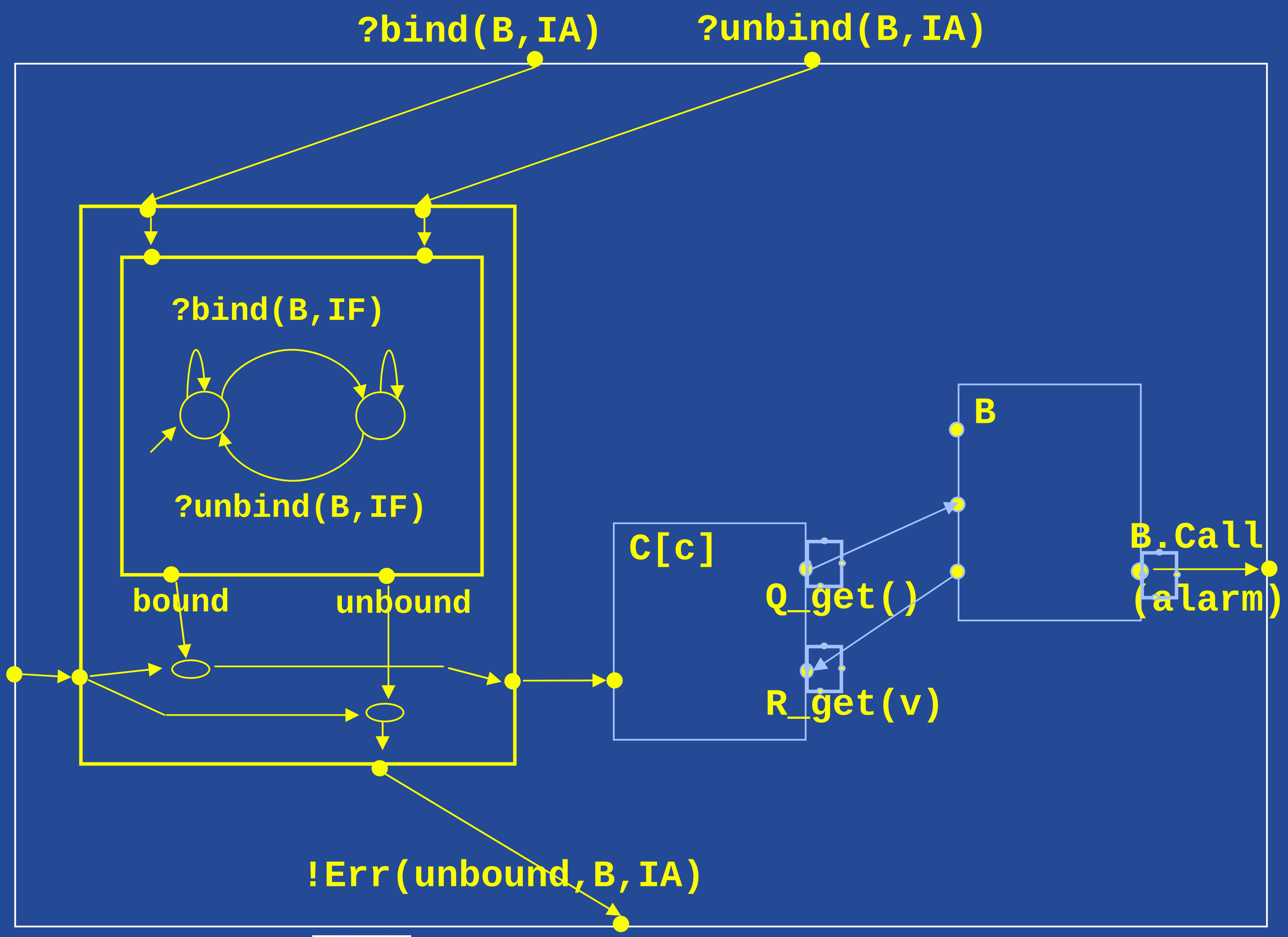




Building pNet models (ex 2)

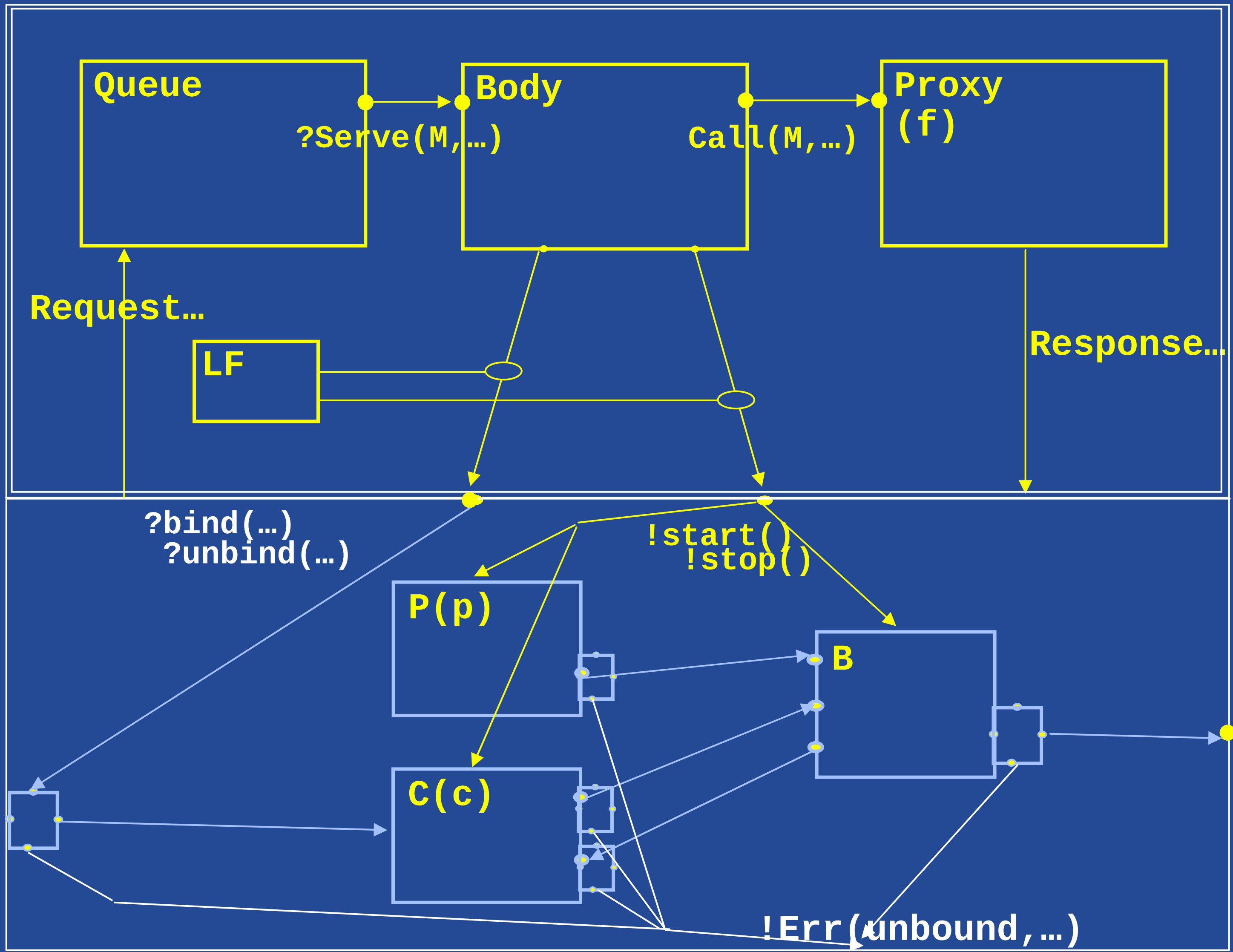
Nets and pLTS for Fractal non-functional controllers :

- Binding controllers
- Life-cycle cont.
- Content cont.



pNet Models for a GCM composite

1) Assemble sub-components
2) add non-functional controls:
   1) Bindings
   2) Start/Stop
   3) …
3) Add Interceptors :
   1) Body
   2) Queue, LF and proxies



Agenda

- Context
  - Active Objects, Components and Grids
- **Safe Distributed Components**
  - Definitions
  - Model generation
  - **Properties**
- Specification and Verification Tools, Case Study
- Conclusion & Perspectives

What do you expect to prove ?

(the application developer point of view)

Initial Composition

- Generic properties : successful deployment, absence of standard errors (unbound interface, etc.), deadlock freeness
- User Requirements expressed as temporal formulas

Reconfiguration preserving the network structure

- Preservation of properties (aka service interaction)
- New features

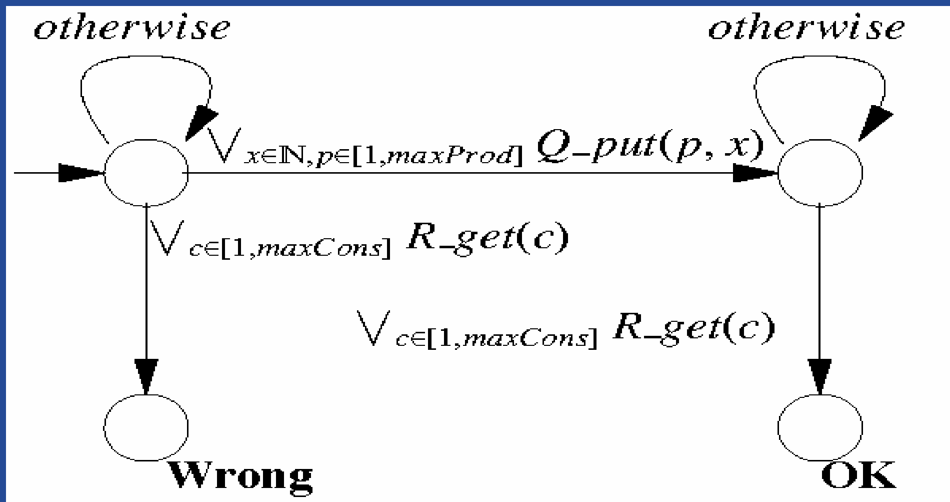
Compositionality / Substitutability

- The Component Automaton, after hiding/minimization, is the functional behaviour used at next level of composition

Verification of Properties

The question of the property definition language :

- Various temporal logics (CTL, ACTL, …)
- Regular µ-calculus (Mateescu'2004) : the assembly language of temporal logics
- Specification patterns (Dwyer'199x)
- Or parameterized symbolic automata ?



Verification of Properties

Functional properties under reconfiguration (respecting the topology)

- Future update (asynchronous result messages) independent of life-cycle or binding reconfigurations
- Build a model including life-cycle controllers,
  with the reconfiguration actions visible:

$$?unbind(C.E_b, B.E_g) \qquad ?stop(C)$$

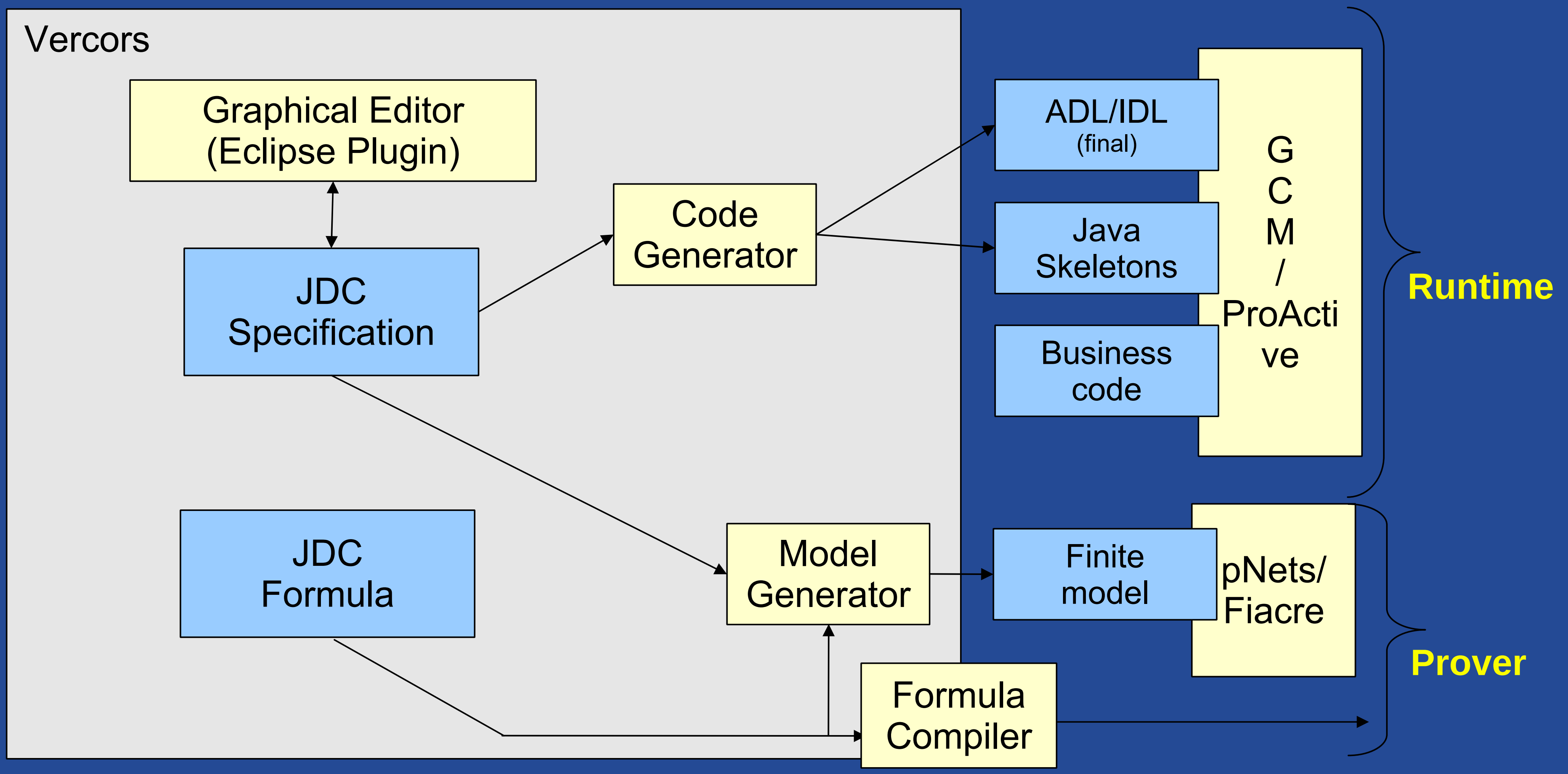
- Then we can prove:

$$[\ true^*.Req\_Get()\ ]\ \mu X.\ (\langle\ true\ \rangle\ true \wedge [\neg Resp\_Get()\ ]\ X\ )$$

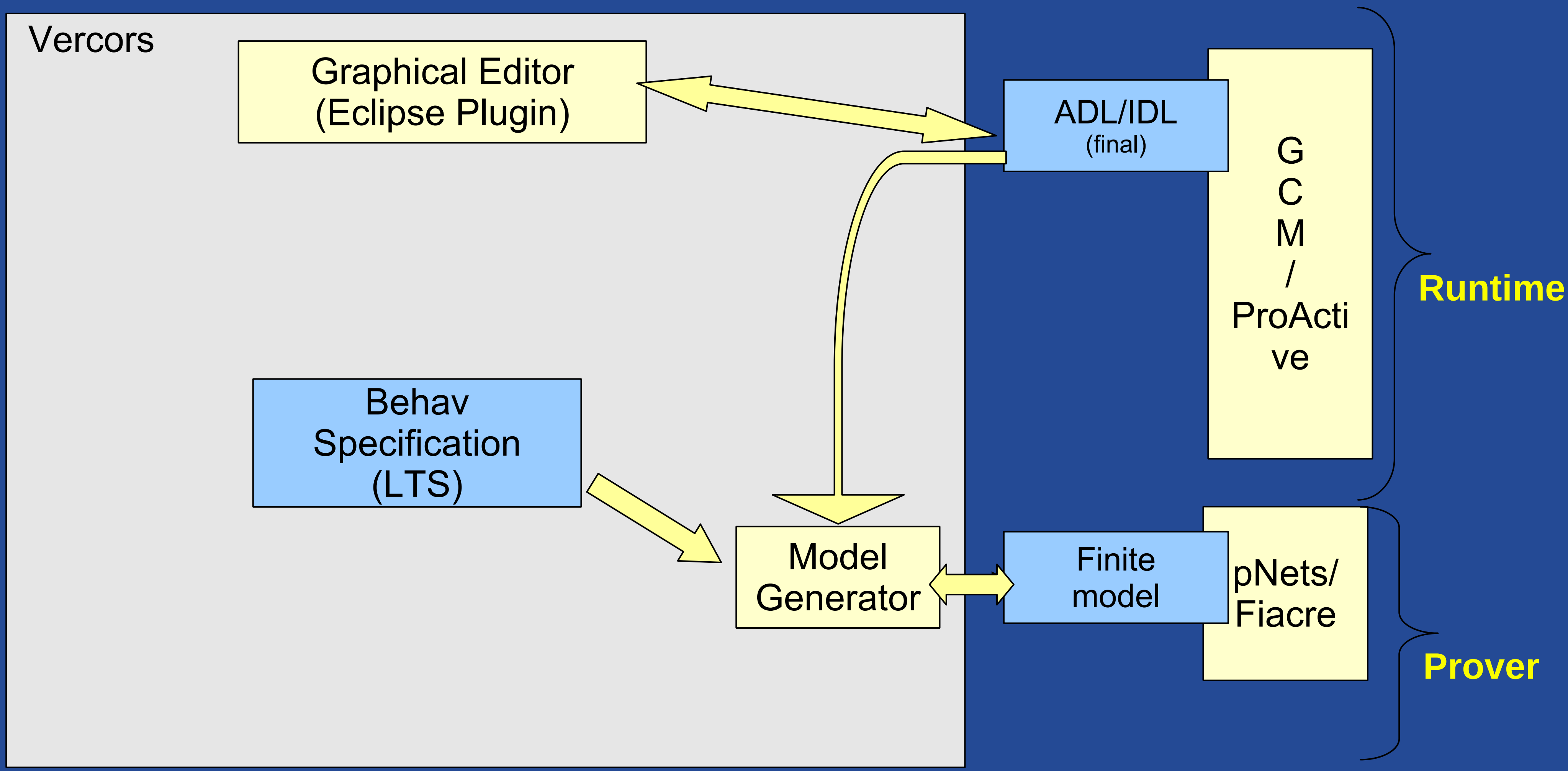
Agenda

- Context
  - Active Objects, Components and Grids
- Safe Distributed Components
  - Definitions
  - Model generation
  - Properties
- **Specification and Verification Tools, Case Study**
- Conclusion & Perspectives

The Vercors Specification and Verification Platform (middle term)



The Vercors Specification and Verification Platform (current prototypes)

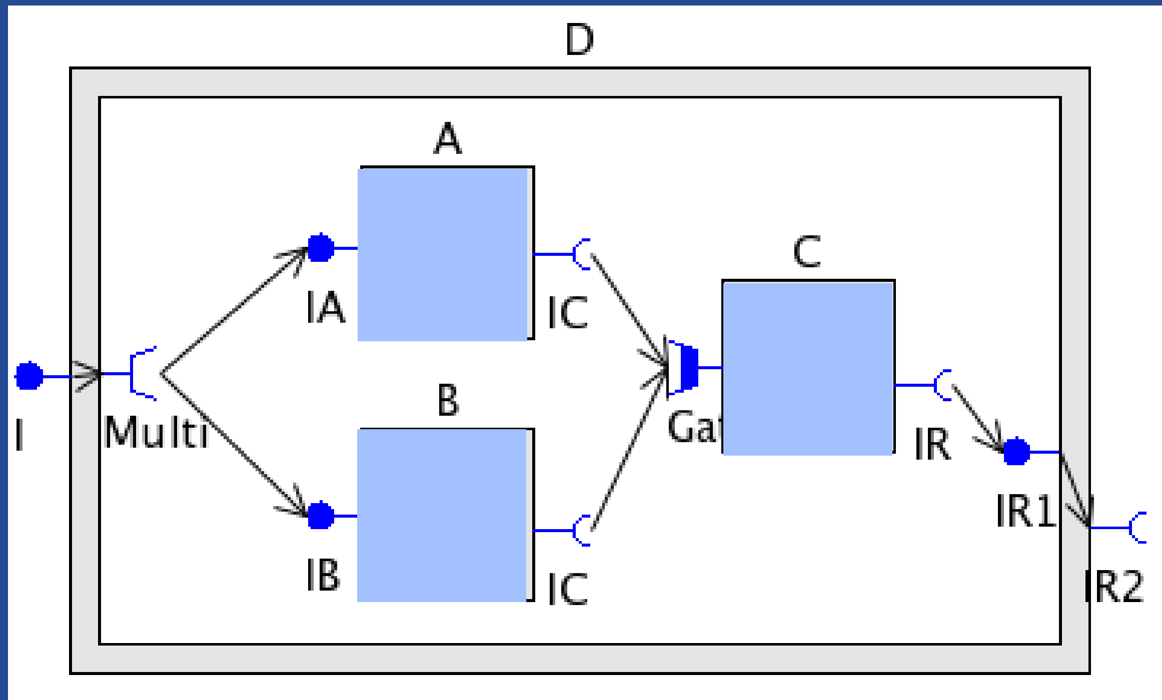


Graphical Specifications : VCE tool

GCM specific constructs:

- 1 to N and N to 1 bindings
  (multicast and gathercast interfaces)

- Open issue :
  attach a behaviour to
  these interfaces.



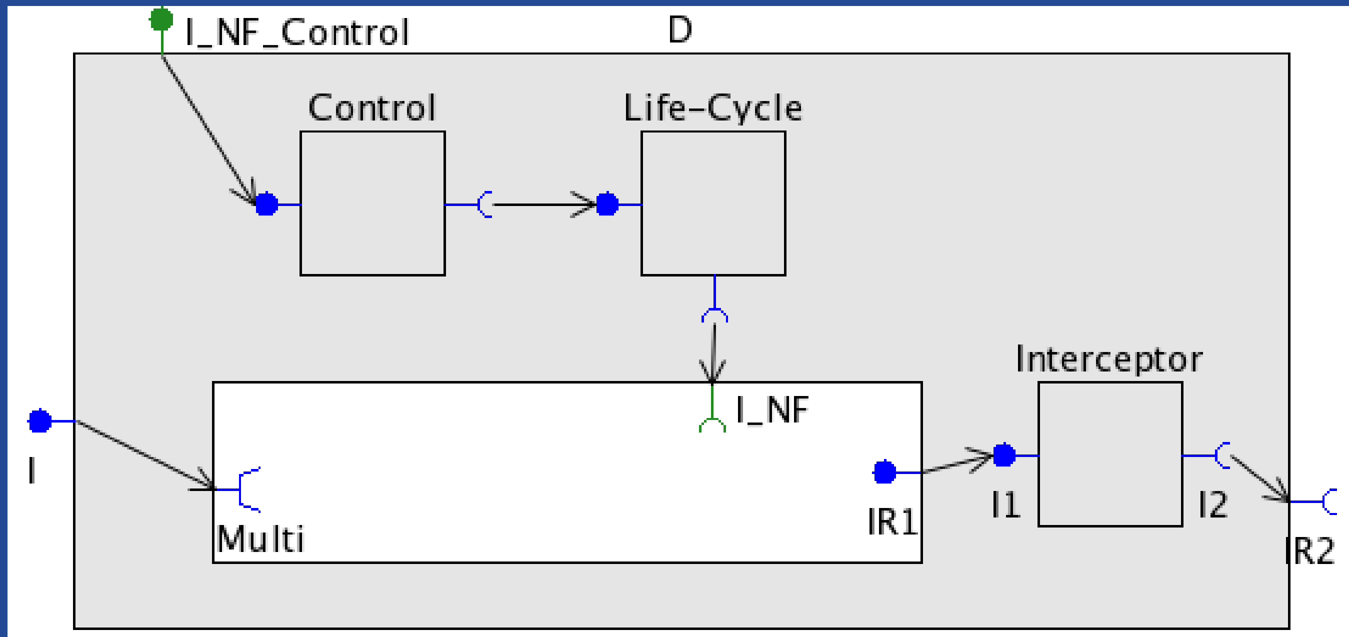
Graphical Specifications : VCE tool

GCM specific constructs:

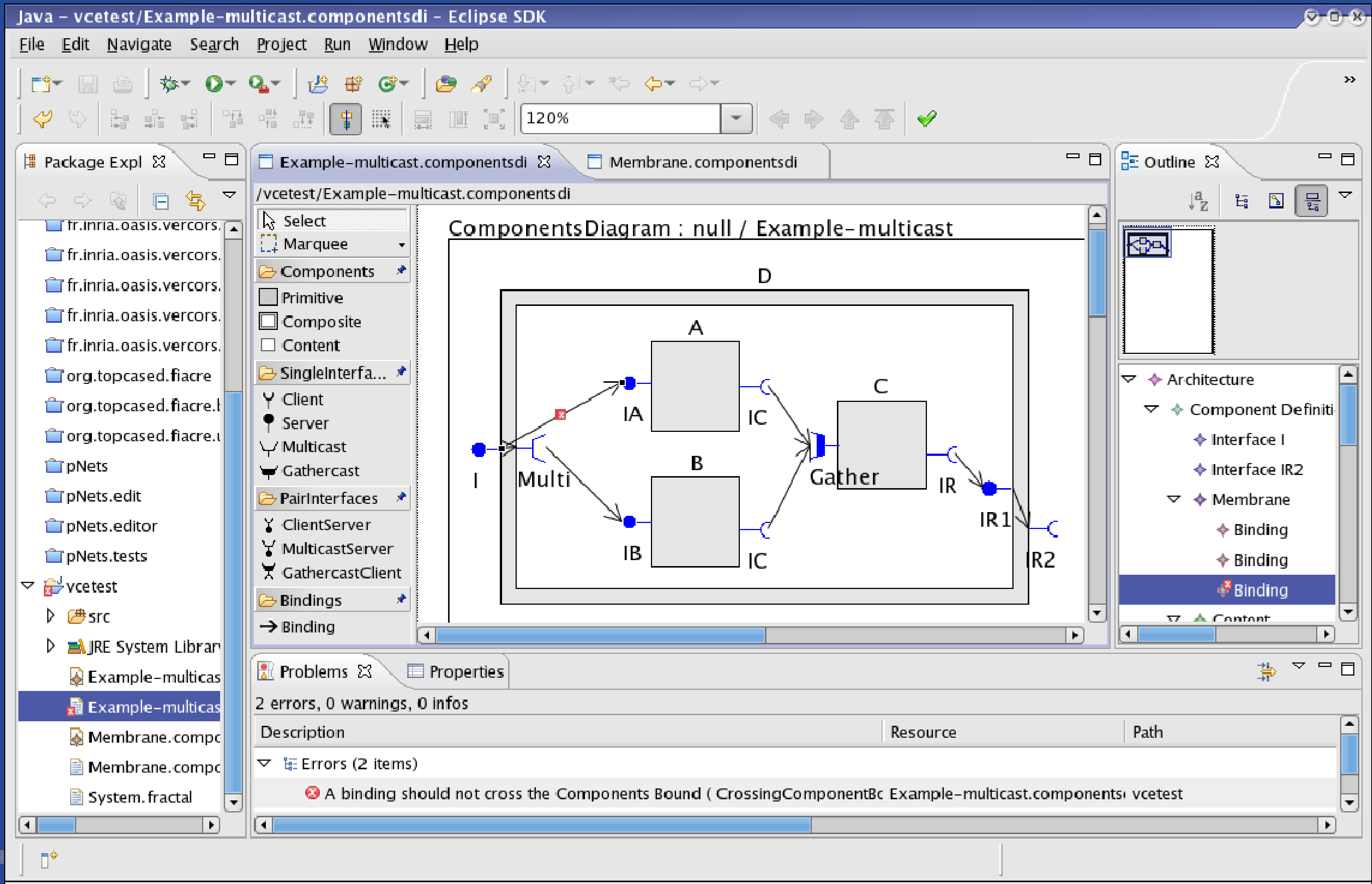
- Non-functional controller components in the membrane

    Interceptors

    Autonomic management



Graphical Specifications : VCE tool

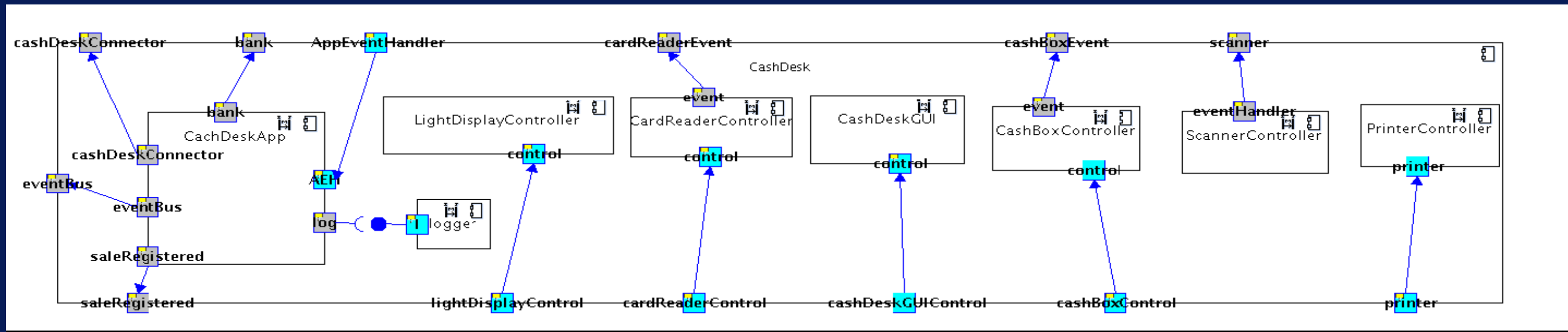


Verification Tools

CADP toolset **(INRIA Rhones-Alpes, VASY team)**

- Generic Front-end
  - (Lotos, BCG, Sync-vectors)
- Model generation: distributed on a cluster
  - Up to 100 millions of states
  - On-the-fly, Tau-reduction, Constrained…
- Verification Evaluator tool:
  - Deadlock search / Regular μ-calculus
- Bisimulation ckecking, minimizing

Case study : Point of Sale

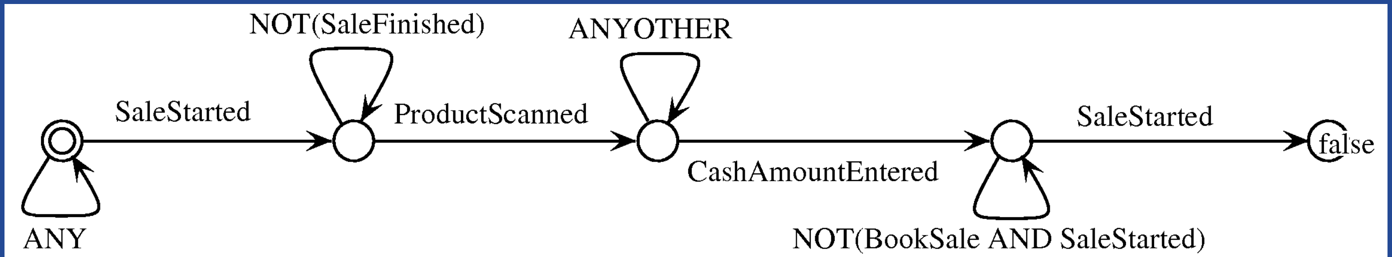


CoCoME : Common Component Modeling Example

- Hierarchical model for a Cashdesk system
- 16 components, 5 levels, 10 parameters
- Brute force state space would be 2.10^8
- optimized generation => biggest size < 100000 states
- Mastering data parameters, and broadcast communication.
- Code generation (GCM/ProActive)

Case study

- Deadlocks were found (due to synchronous versions of our encoding)
- Checking Specification Requirements:
  - Main sale process is feasible (Use Case 1)



  - Wrong behaviours

    (Booking an Empty Sale, Successful Sale with Insufficient Money)
  - Error due to incomplete specification : safety of the Express Mode (an express mode may be triggered during an ongoing sale

Ongoing work

Code Generation :

- From Architecture and Behaviour Diagrams

  … to ADL descriptions and GCM/ProActive code skeletons

Extensions :

- 1 to N and M to 1 communication
- Parameterized components in the specification language
- Tool support for abstraction specification

New verification tools :

- Specialized model-checking engines for decidable classes of problems:
  - unbound fifo channels
  - Counters + presburger

Conclusions

pNETs:

Semantic model for hierarchical, parameterized asynchronous systems
Flexible, expressive and compact.

Model generation for the behaviour of distributed hierarchical components

- Automatic Construction of the control automata and of synchronisation constructs
- Verification of properties in different phases
- Prototype platform for graphical specification, model construction, model-checking.

Papers, Use-cases and Tools at :

http://www-sop.inria.fr/oasis/Vercors