Verification of Distributed Hierarchical Components

T. Barros, L. Henrio, E. Madelaine

OASIS Team,

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

(FACS'05), Fractal workshop, Grenoble 29 nov 2005

Université Nice SOPHIA ANTIPOLIS

CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

INRIA

Agenda

- Context
  - Components and safe composition
  - Fractive
- Building Behaviour models
  - Formal Model
  - Configuration and Introspection
  - Asynchrony
- Checking Properties
- Conclusion & Perspectives

Software Components

Definition :

Software modules, composable, 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. **Separate administration**
   Architecture Description Language (ADL), administration 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 :

- Build complex systems from off-the-shelf components
- Check behavioural **compatibility** between sub-components
- Check correctness of component **deployment** and **behaviour**
- Check correctness of the **transformation** inside a running application.

Fractive's components

FRACTAL : Component model specification

+ ProActive : Java library for distributed applications

= Fractive

Features:

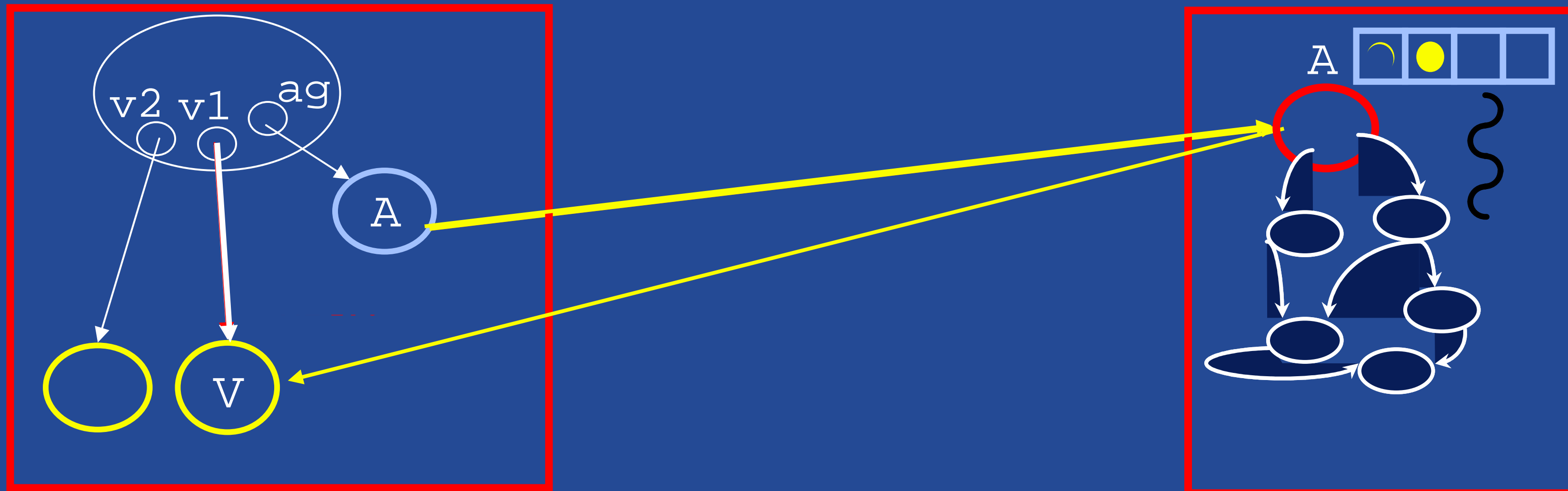
- Hierarchical Component Model
- ADL description (Fractal's XML Schema/DTD)
- Separation of functionality / management
- Distributed components (from distributed objects)
- Asynchronous computation (non-blocking method calls)
- Strong Formal Semantics (ASP) => properties and guarantees

Fractal Components

LIFE CYCLE
BINDING
CONTENT
ATTRIBUTE

Fractive : 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

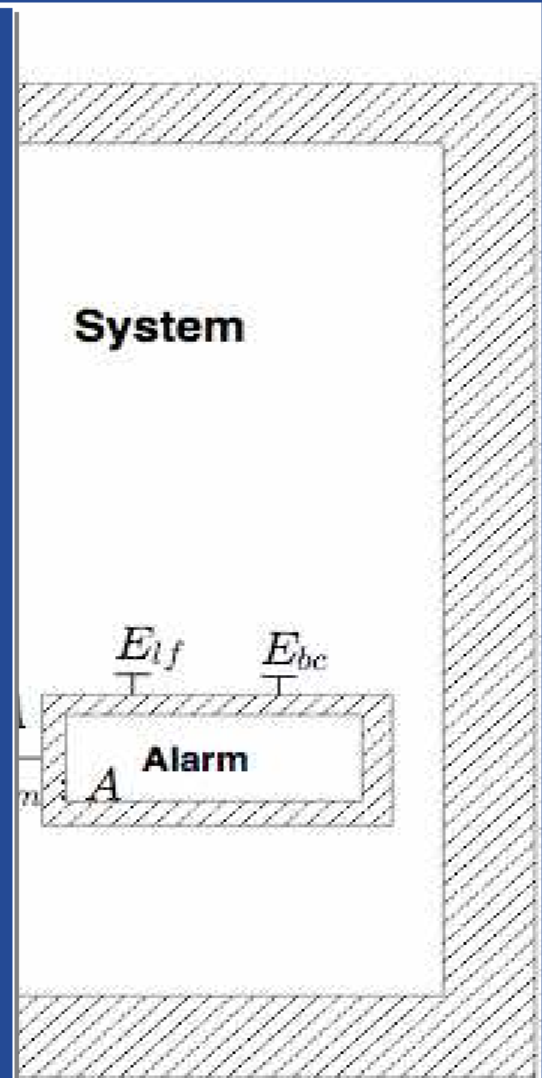
Fractive example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE .... >

<definition name="components.System">
 <component ... </component>

 <component name="Alarm">
  <interface name="alarm" role="server"
     signature="components.AlarmInterface"/>
  <content class="components.Alarm"> </content>
  <behaviour file="AlarmBehav"
     format="FC2Param"/>
</component>

 <binding client="BufferSystem.alarm"
     server="Alarm.alarm"/>
</definition>
```

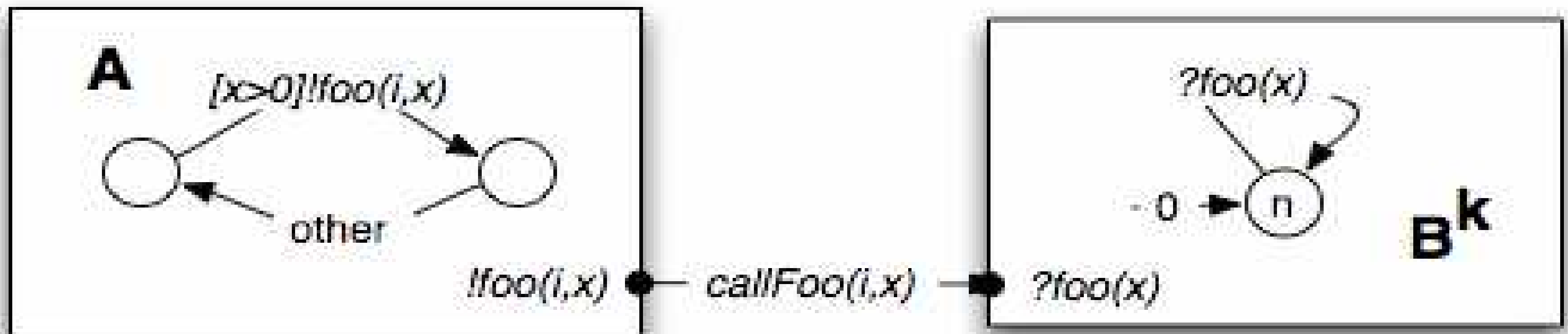


Agenda

- Context & Motivation
- Building Behaviour Models
  - Formal Model
  - Configuration and Introspection
  - Asynchrony
- Checking Properties
- Conclusion & Perspectives

Behaviour: Parameterized Networks of synchronised Transitions Systems

T. Barros, R. Boulifa, E. Madelaine: Parameterized Models for Distributed Java Objects, Forte'2004 Conference, Madrid, Sep. 2004, LNCS 3235, © Springer-Verlag



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]

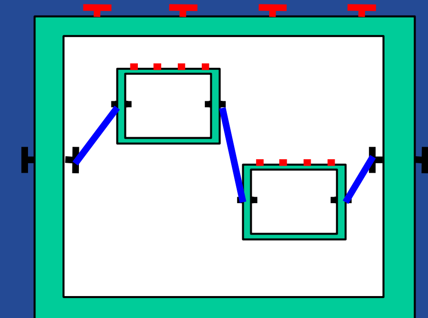
Families of Processes : no similar generic result (but many results for specific topologies).

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

Fractive Behavioural Models : Principles



Compositionality

- Reasonning at each separate composition level

Functional behaviour is known

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

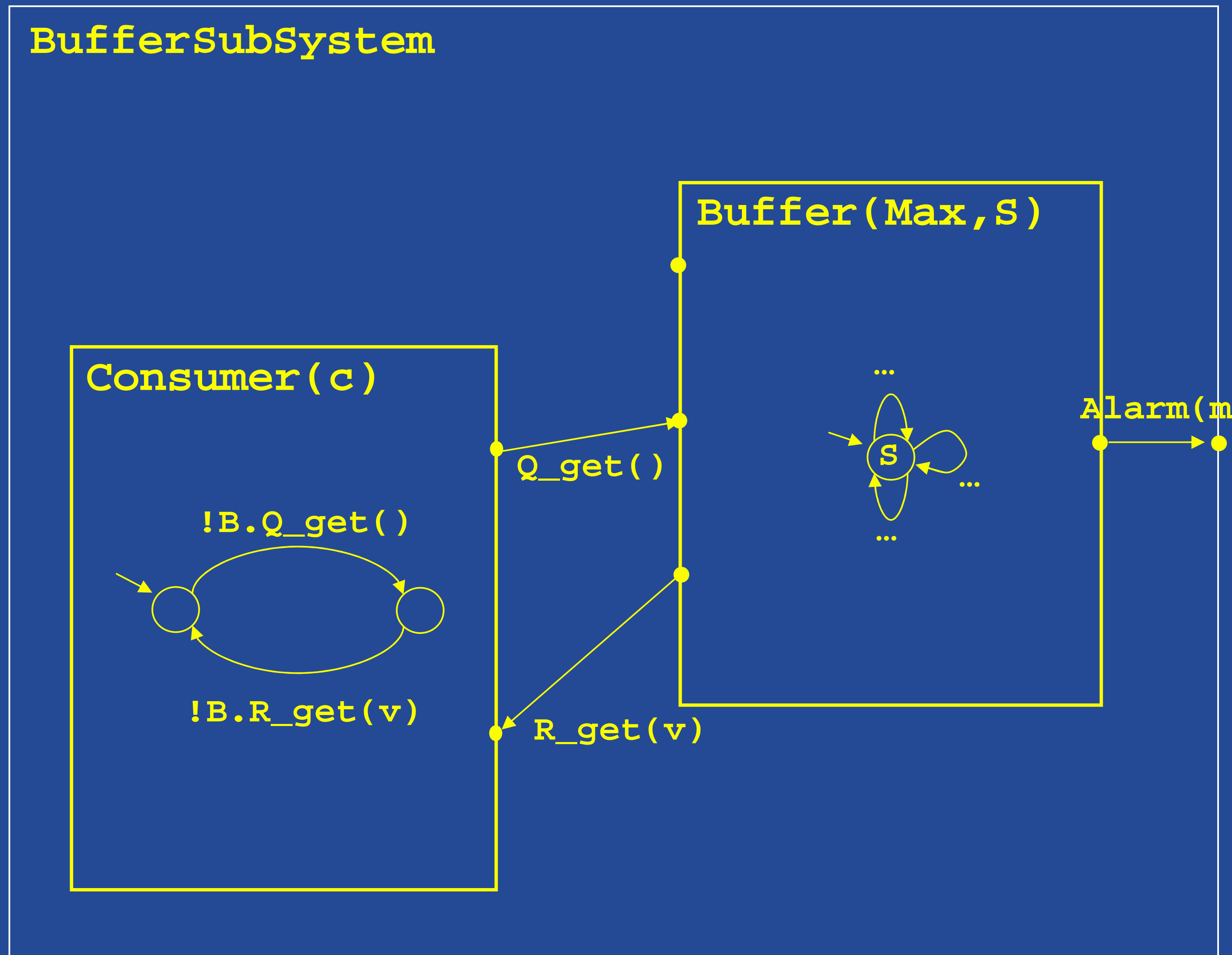
Non functional behaviour automatically added from the component's ADL

- Automata within a synchronisation network
- Incorporate controllers for management interfaces,
- and for asynchronous communication management

Build the product, Hide, Minimize….

Building Fractive Behavioural Models

1) Assemble sub-components,



Building Fractive Behavioural Models

1) Assemble sub-components
2) add non-functional controls:
    1) Bindings

?bind(B,IA)
?unbind(B,IA)
?bind(B,IF)
?unbind(B,IF)
bound
unbound
P(p)
Q_put(x)
B
B.Alarm()
C(c)
Q_get()
R_get(v)
BSS.Alarm()
!Err(unbound,B,IA)

Building Fractive Behavioural Models

1) Assemble sub-components
2) add non-functional controls:
   1) Bindings

?bind(…)
?unbind(…)

P(p)

B

B.Alarm()

C(c)

!Err(unbound,…)

Building Fractive Behavioural Models

1) Assemble sub-components
2) add non-functional controls:
   1) Bindings
   2) **Start/Stop**

?bind(…)
?unbind(…)

!start()
!stop()

P(p)

C(c)

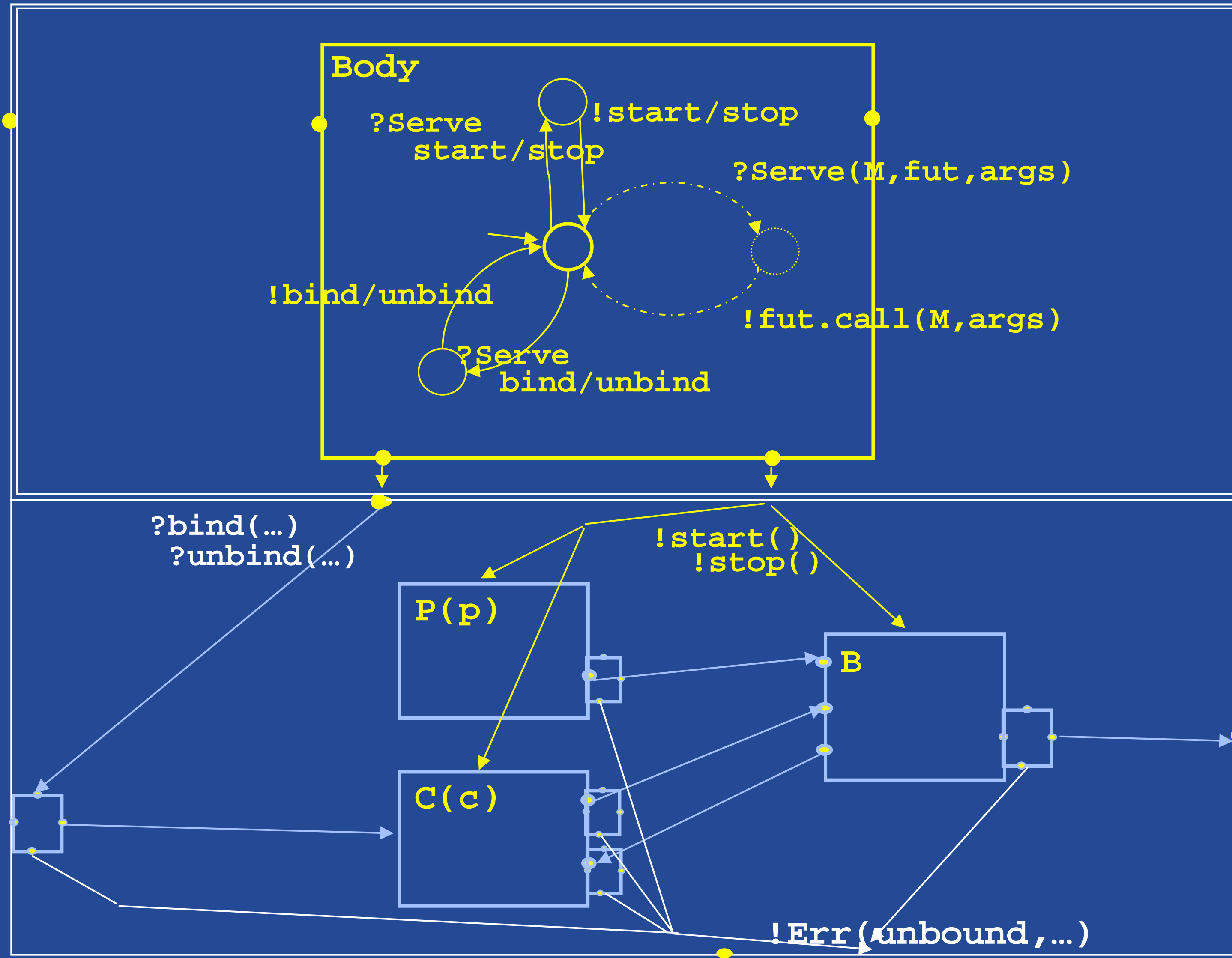
B

B.Alarm()

!Err(unbound,…)

Building Fractive Behavioural Models

1) Assemble sub-components
2) add non-functional controls:
    1) Bindings
    2) Start/Stop
3) Add Interceptor :
    1) Body



Building Fractive Behavioural Models

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

= Controller

Result : The Static Automaton

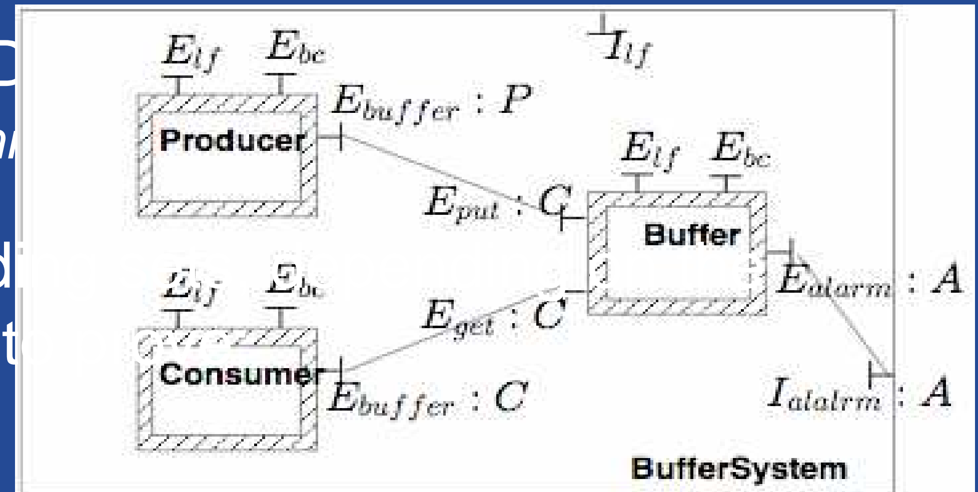
Deployment Automaton :

Static Automaton = (Controller || D

+ hiding & mini

Fine Tuning = Specify different hid
properties we want t

- deployment phase
- functional phase
- topology-preserving transformations



Agenda

- Context & Motivation
- Behaviour models
- **Checking Properties**
  - **Functional and management interactions**
- Conclusion & Perspectives

Behaviour correctness
(from the user point of view)

Initial Composition

- Generic properties : successful deployment, absence of errors, deadlock freeness
- User Requirements expressed as temporal formulas

Reconfiguration preserving the network structure

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

Compositionality

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

Verification of Properties

regular μ-calculus (Mateescu'2004)

Deployment

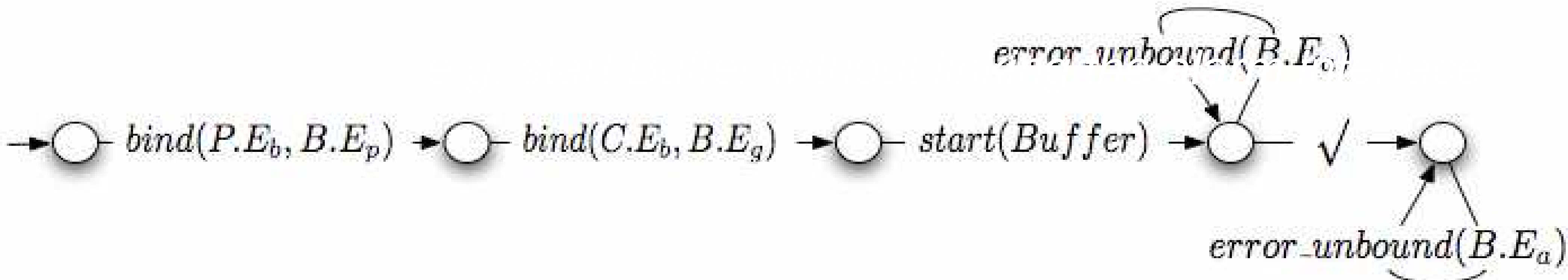
(on the Static Automaton with successful synchronisation visible)

- The deployment is always successful

$$[\ (\text{not}\ \surd\ )^*\ ]\ \langle\ \text{true}\ ^*\ .\surd\ \rangle\ \text{true}$$

- No Error during deployment

$$[\ (\text{not}\ \surd\ )\ ^*\ .\ O_E\ ]\ \text{false}$$



Verification of Properties

Functional behaviour (on the Static Automaton)

- Get from the buffer eventually gives an answer

$$[\ true^*.Req_Get\ ()\ ]\ \mu X.\ (<\ true\ >\ true \wedge [\neg Resp_Get()\ ]\ X\ )$$

Verification of Properties

Functional properties under reconfiguration (respecting the topology)

- Future update (asynchronous result messages) independent of life-cycle or binding reconfigurations
- E.g:

[true*.Req_Get()] μX. (< true > true $\wedge$ [$\neg$Resp_Get()] X)

Proved on an Extended Static Automaton allowing the following control operations:

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

Structural Transformations (ongoing work)

Scenario :

- Running application, Need to Replace/Update one sub-component
- Check the protocol compatibility before replacement

Principle :

- Use the formal Behaviour Specification
- Identify states of the application behaviour model in which the transformation is possible,
- … compute the corresponding states after transformation.
- Use the merged automaton to check properties.

Benefits :

- Identifies prerequisite and rules for safe transformation.

Agenda

- Context & Motivation
- Behaviour models
- Checking Properties
- **Related Work, Conclusion & Perspectives**

Related Work

From Process Algebras to Components

- Semantics : LTSs, congruences, refinement
- Processes, Connectors, and CSP refinement : Wright
- Hierarchical components, weak bisimulation, Buchi automata : Darwin
- Semantics of encapsulation : Kell calculus

Sofa

- Frame (spec) vs. Architecture (implementation) compliance relation based on traces
- Hierarchical construction through parallel composition, with error detection

Behavioural Contracts (e.g. Carrez et al.)

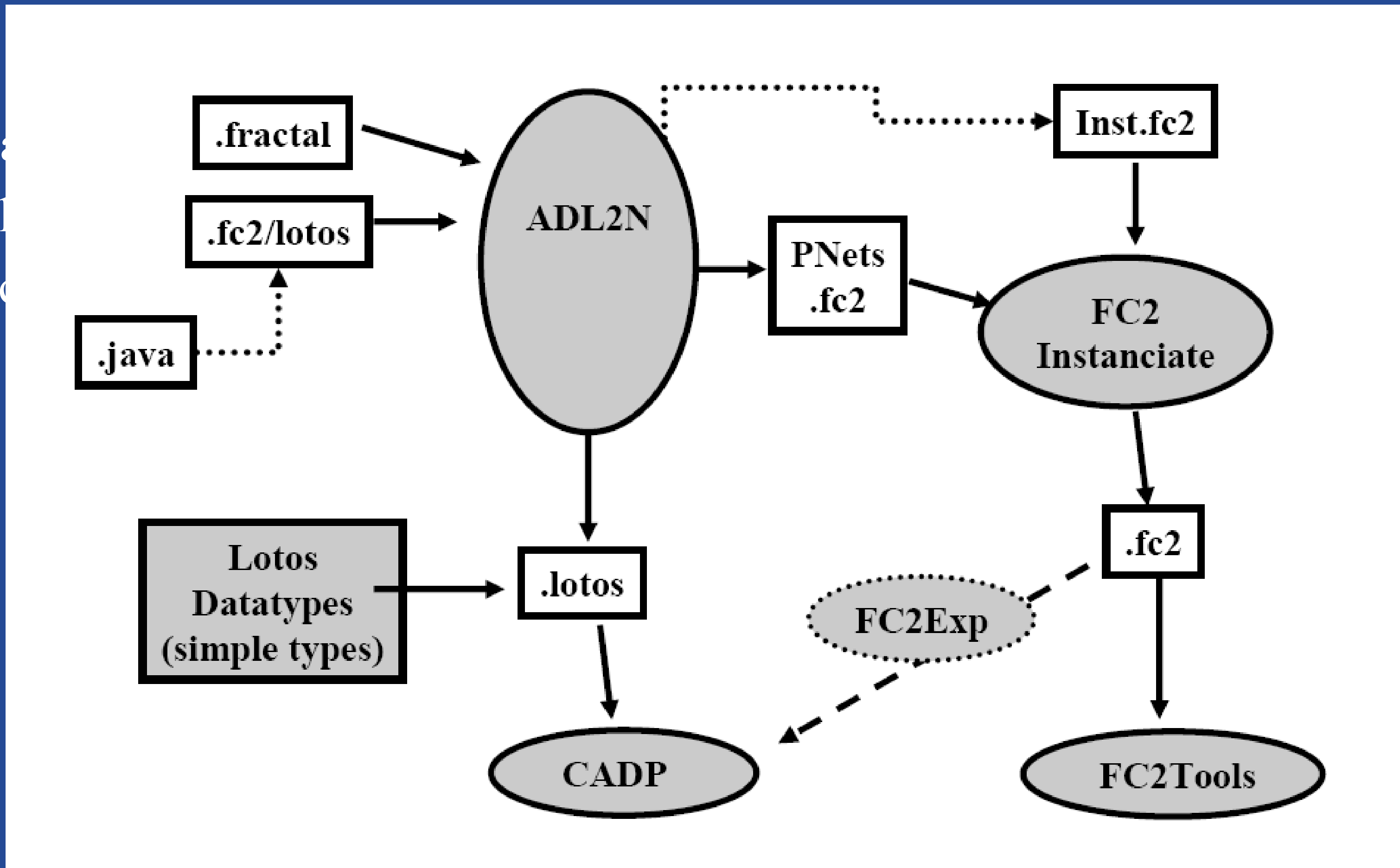
- Interface behavioural-type compatibility (decidable)
- Component contract compliance (non decidable)

No other work to our knowledge dealing with functionality + management (even in the synchronous case)

Vercors Platform

Tool set :

- Code
- Model
- Interac



Supported by FIACRE

An ACI-Security action of the French research ministry

Ongoing work

Expression of Properties :

- Pattern language specific to Grid Application

Extensions :

- Group communication

Perspectives

- New verification tools (infinite state classes)
- “Safe by construction” programming style

Conclusions

- Model for the behaviour of distributed hierarchical components
- Automatic Construction of the control parts
- Verification of properties in different phases
- Implementation of a prototype tool for model construction, using standard model-checking tools

Asynchrony is essential for large scale grid applications (hide the latency, fewer deadlocks), but brings in new difficulties (at developer level).

Papers, Use-cases and Tools at :

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