Semantic Formalisms 2:
Software Components

- Formal Methods
  Operational Semantics:
  CCS, Bisimulations
- Software Components
  Fractal : hierarchical components
  Deployment, transformations
  Specification of components
- Application to distributed applications
  Active object and distributed components
  Behaviour models
  An analysis and verification platform

Eric Madelaine
eric.madelaine@sophia.inria.fr

INRIA Sophia-Antipolis
Oasis team

UNICE – EdStic
Mastère Réseaux et Systèmes Distribués
TC4

Program of the course: 1: Software Components

- Fractal : hierarchical components
  - Specification of Component Systems
  - Modelling with UML diagrams
- Specification and verification of behaviours
  - Generating non-functional controllers
  - Expressing and proving properties

Fractive's components

- FRACTAL : Component* model specification, implemented using
- ProActive : Java library for distributed applications

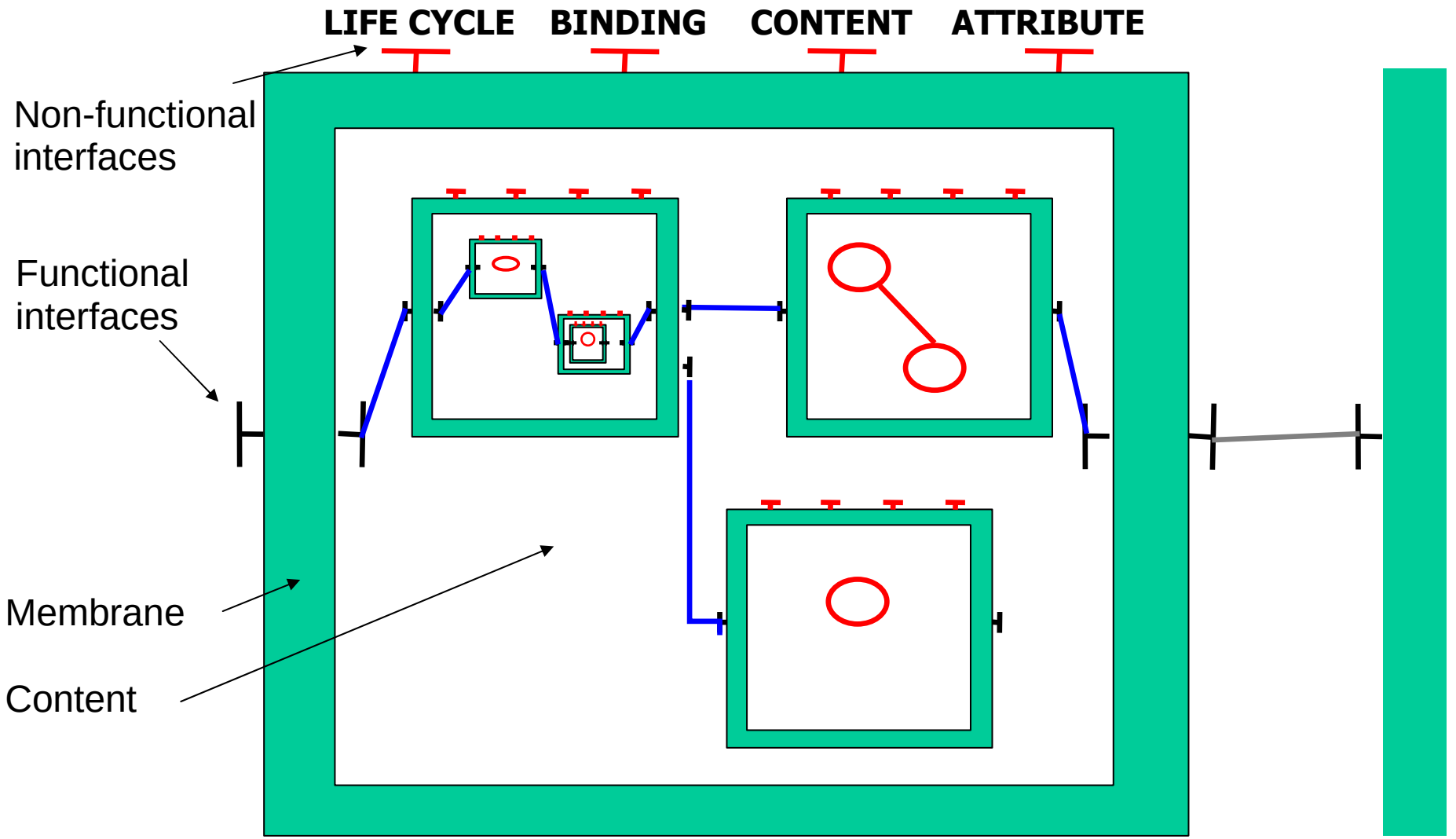
= Fractive

*** Component :**

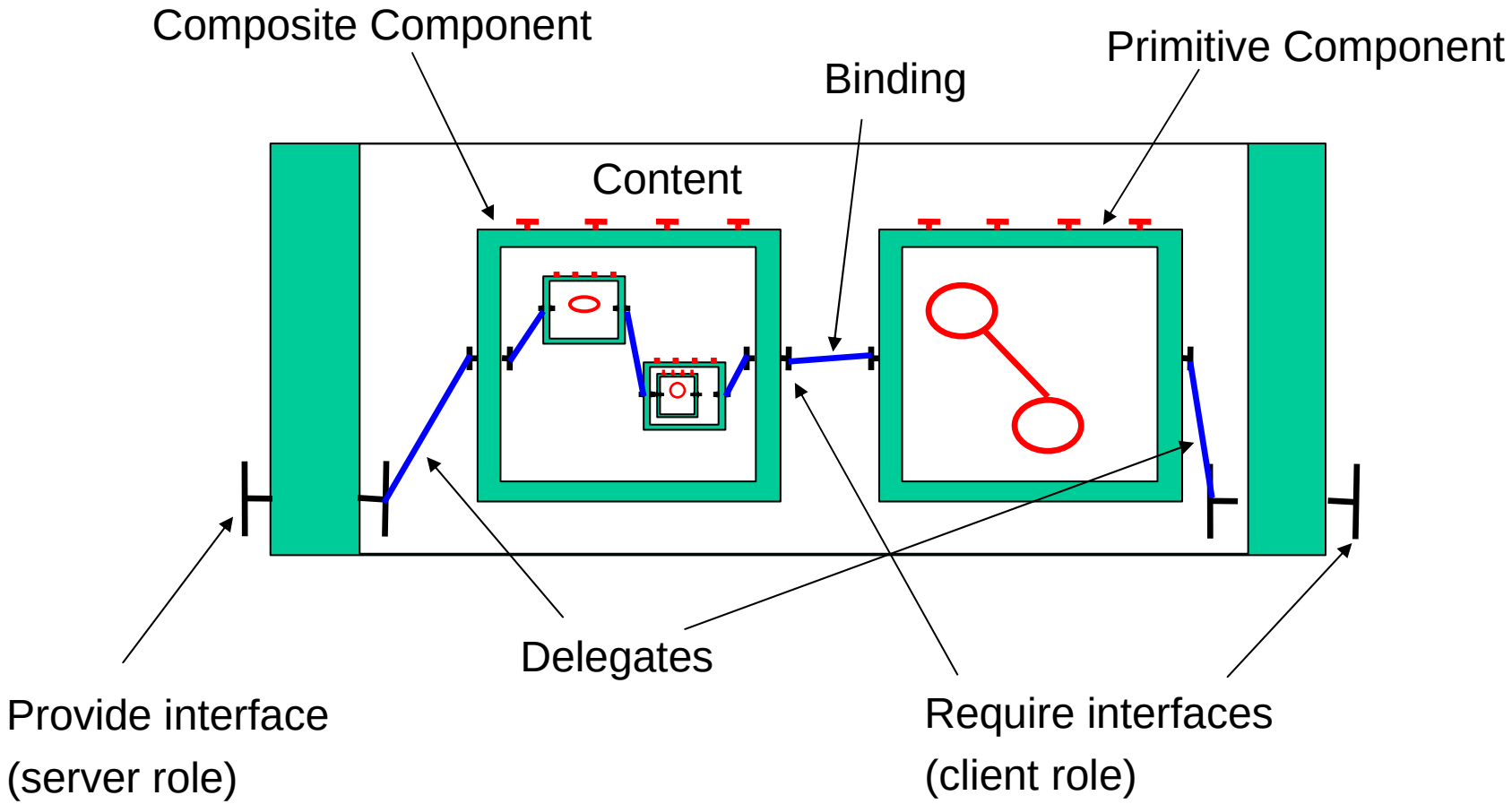
self-contained entity, with well-defined interfaces, reusable, composable (hierarchically)

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

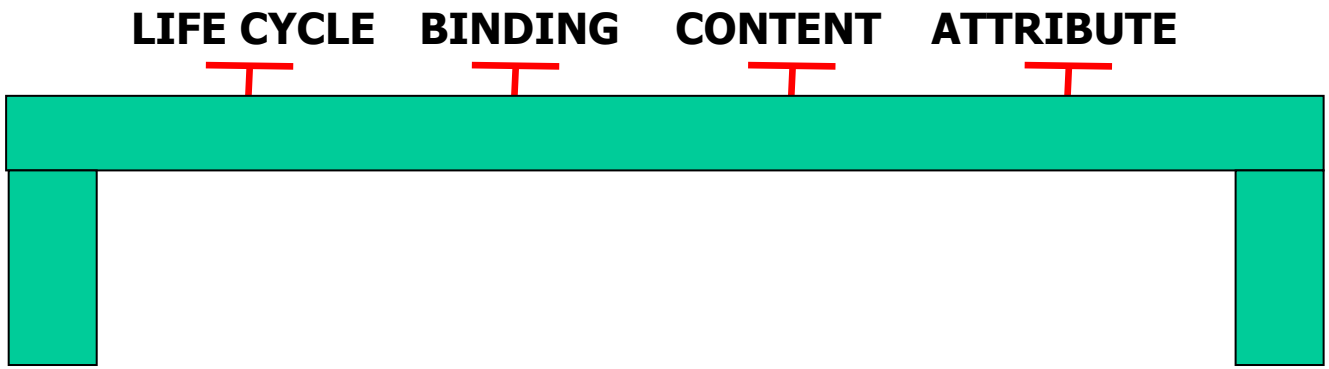
Fractal's Components



Fractal's Components : Architecture



Fractal's Components
Non-functional interfaces



Life-cycle : start / stop the component

Binding : bind / unbind a connection between interfaces

Content : add / remove sub-components

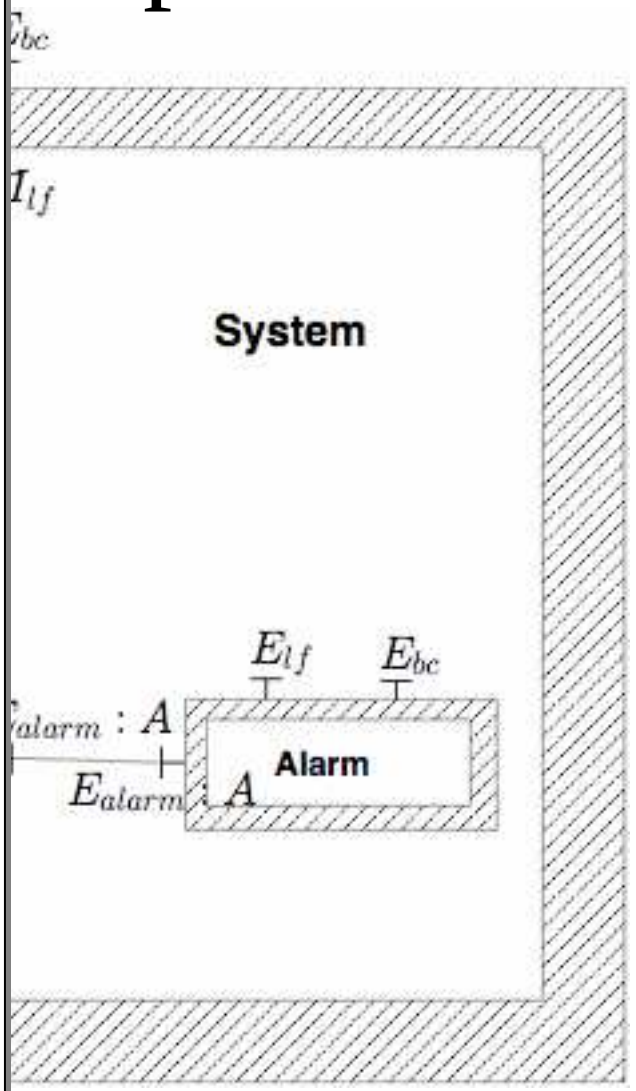
Attribute : get set the value of attribute values

Component System Specification

1. Architecture Description (ADL):
   - Primitive components, Composite components,
   - Bindings
2. Interface Description (IDL):
   - Will be a Java specification in the case of ProActive
3. Behaviour Description:
   - Any process language: LTS, CCS, value-passing CCS, Lotos…

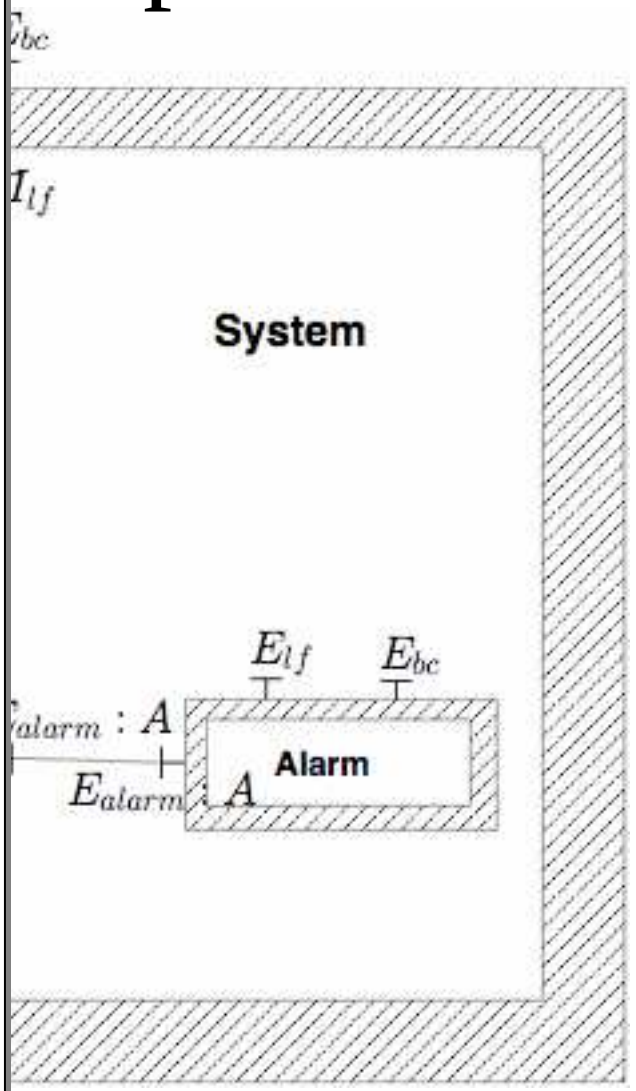
Buffer System example

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

<definition name="components.System">

 <component name="BufferSystem"
     definition="components.BufferSystem(3)">
  <interface name="alarm" role="client"
     signature="components.AlarmInterface"/>
 </component>

 <component name="Alarm">
  <interface name="alarm" role="server"
     signature="components.AlarmInterface"/>
 <content class="components.Alarm">



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



Buffer System example

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

<definition name="components.System">

 <component name="BufferSystem"
     definition="components.BufferSystem(3)">
  <interface name="alarm" role="client"
     signature="components.AlarmInterface"/>
 </component>

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



UML diagrams, modelling tools

- We use UML 2.0 diagrams as a (simple) language for describing both the architecture and the behaviour.
- Architecture : Composite structures
  - Hierarchy of boxes, ports/interfaces, interface descriptions, connexions
- Behaviour: State machines
  - States (with state variables), control structures (tests, loops), transitions, communication events.

UML modelling tool: CTTool

- Derived from TTool (*Turtle Toolkit*)
  - From ENST Sophia Labsoc (“Systems on chip”)
  - **Turtle** = Timed UML and RT-Lotos Environment

    http://labsoc.comelec.enst.fr/turtle/

- Adapted for Components (hierarchy, interfaces)

and from UML 1.5 to UML 2.0.

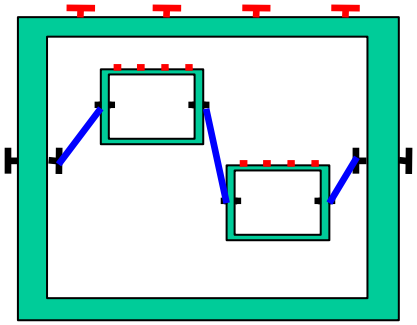
UML modelling tool: CTTool

- Introduce construction, graphically, step by step. Do it within the tool ???
- Warning: preliminary, intermediate version
- Then speak of model generation before going to proofs.
- Back to the CTTool with the CADP proofs.

Program of the course: 1: Software Components

- Fractal : hierarchical components
  - Specification of Component Systems
  - Modelling with UML diagrams
- Specification and verification of behaviours
  - Generating non-functional controllers
  - Expressing and proving properties

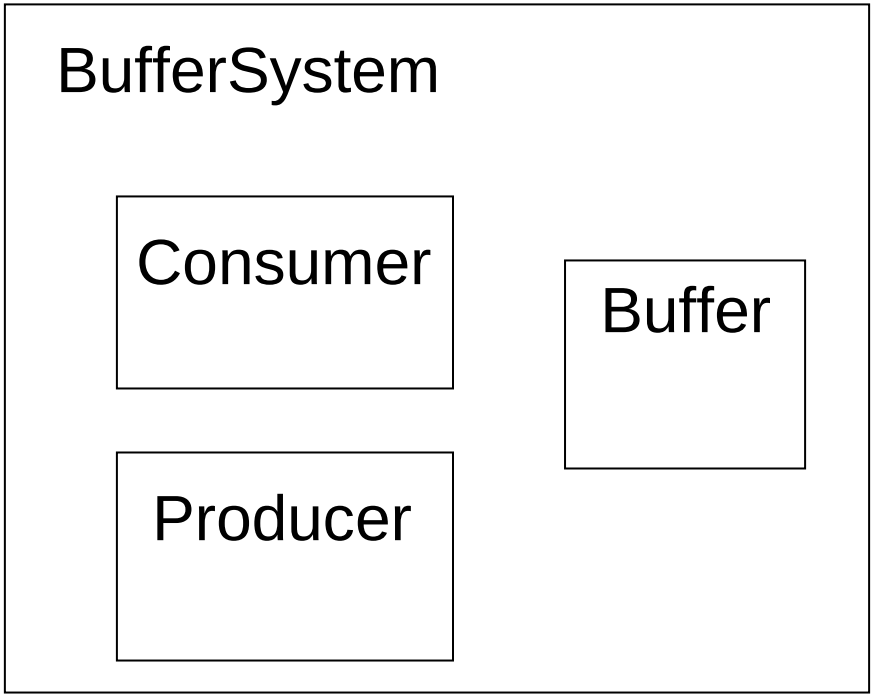
Building a Fractive Behavioural model



- Functional behaviour is known
  - Given by the user
  - Obtained by static analysis
- Non-functional (& asynchronous) behaviour is automatically added from the component's ADL
  - Automata within a synchronisation network, named controller
- Component's behaviour is the controller's synchronisation product

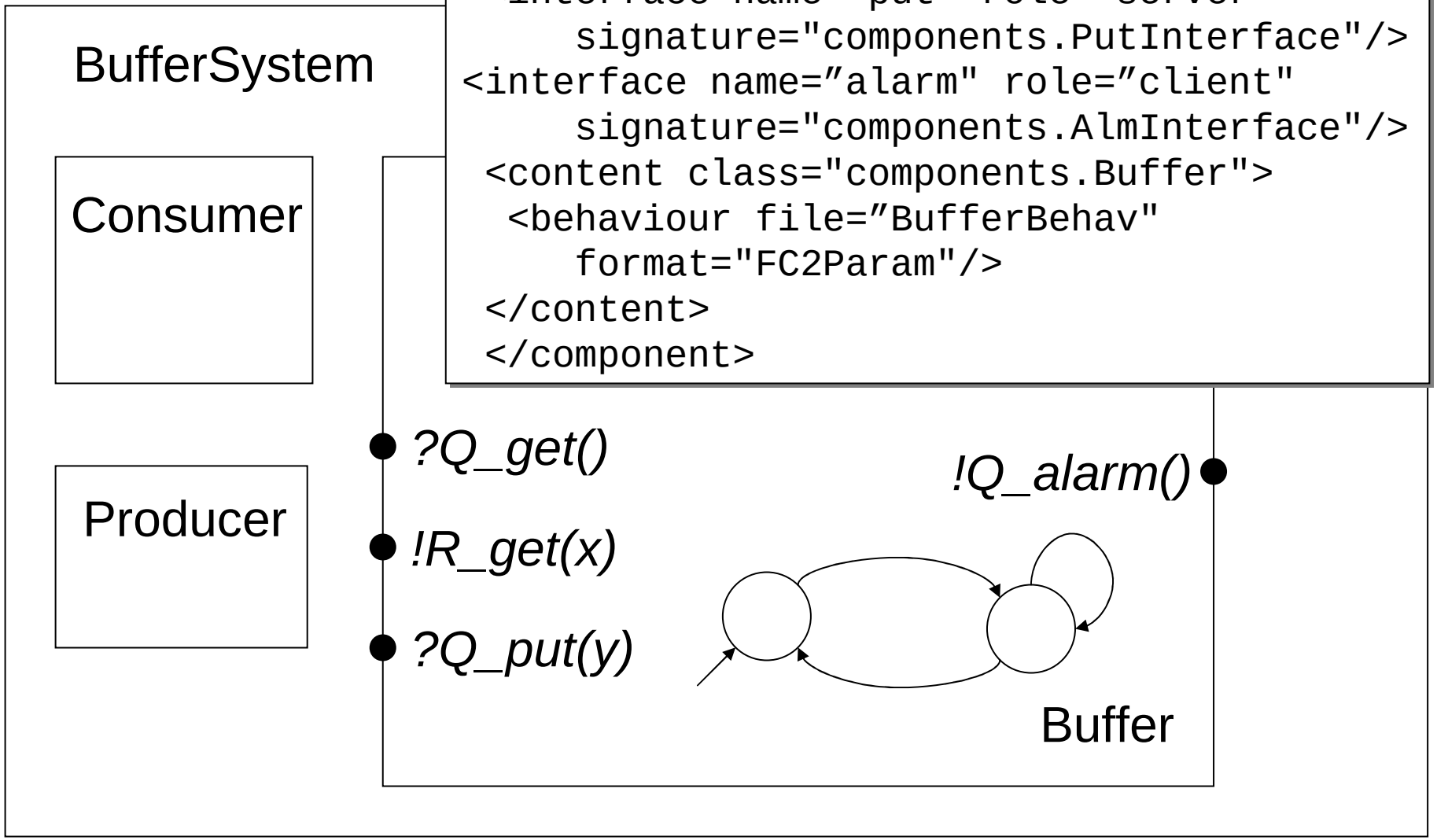
Building the Models: Topology

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE .... >
<definition name="components.BufferSystem">
<component name="Buffer"
 <component name="Buffer"
 <interface name="get" role="server"
     signature="components.GetInterface"/>
 <interface name="put" role="server"
     signature="components.PutInterface"/>
  <interface name="alarm" role="client"
     signature="components.AlmInterface"/>
 <content class="components.Alarm">
 <behaviour file="AlarmBehav"
<component name="Consumer"
 <component name="Consumer"
 <interface name="buf" role="client"
     signature="components.GetInterface"/>
 <content class="components.Consumer">
  <behaviour file="ConsBehav"
<component name="Producer"
 <component name="Producer"
 <interface name="buf" role="client"
     signature="components.PutInterface"/>
 <content class="components.Consumer">
  <behaviour file="ProdBehav"
     format="FC2Param"/>
 </content>
 </component>

<binding client="Producer.buf" server="Buffer.put"/>
<binding client="Consumer.buf" server="Buffer.get"/>
<binding client="Buffer.alarm" erver="alarm"/>
</definition>
```

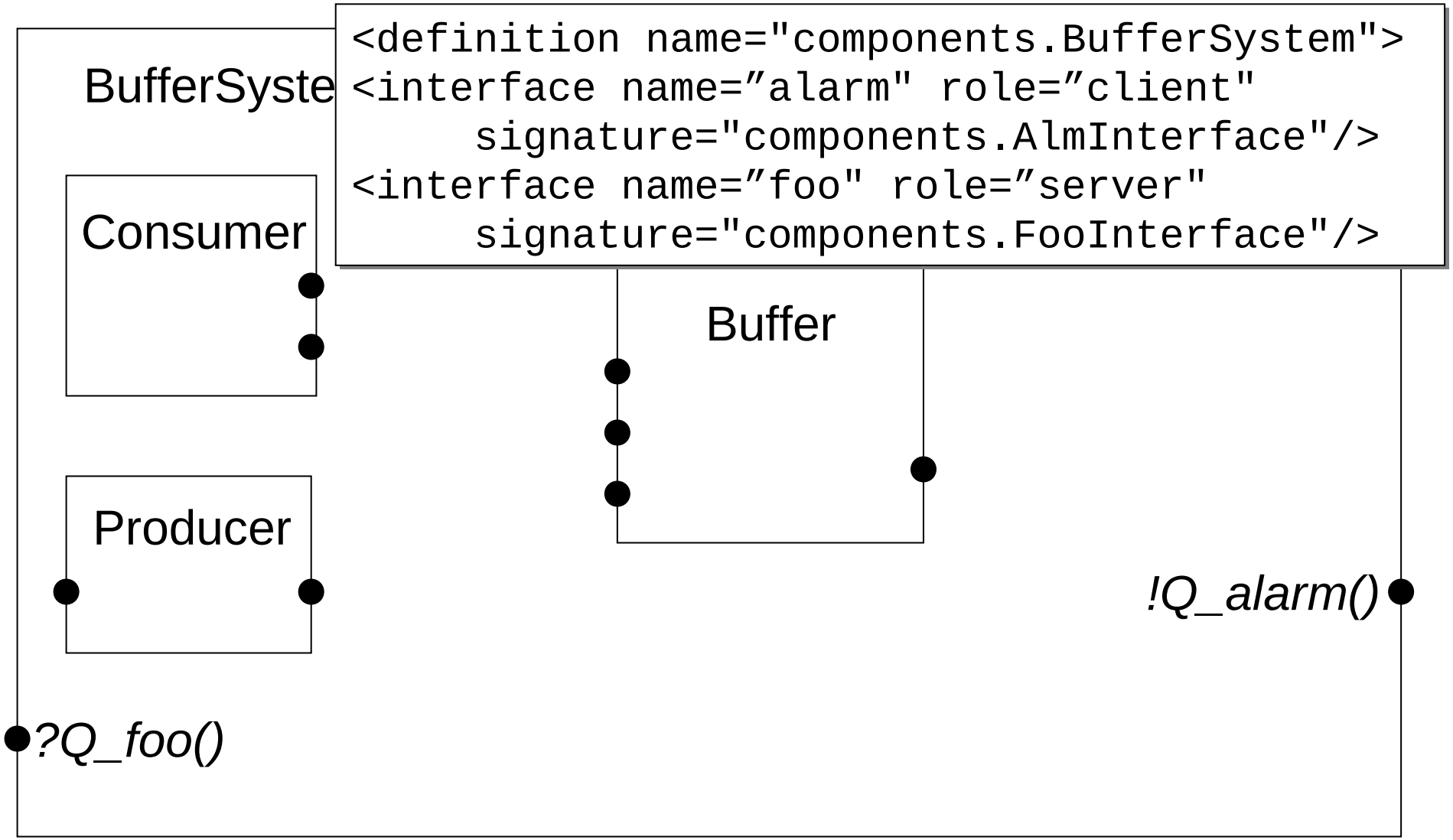


Building

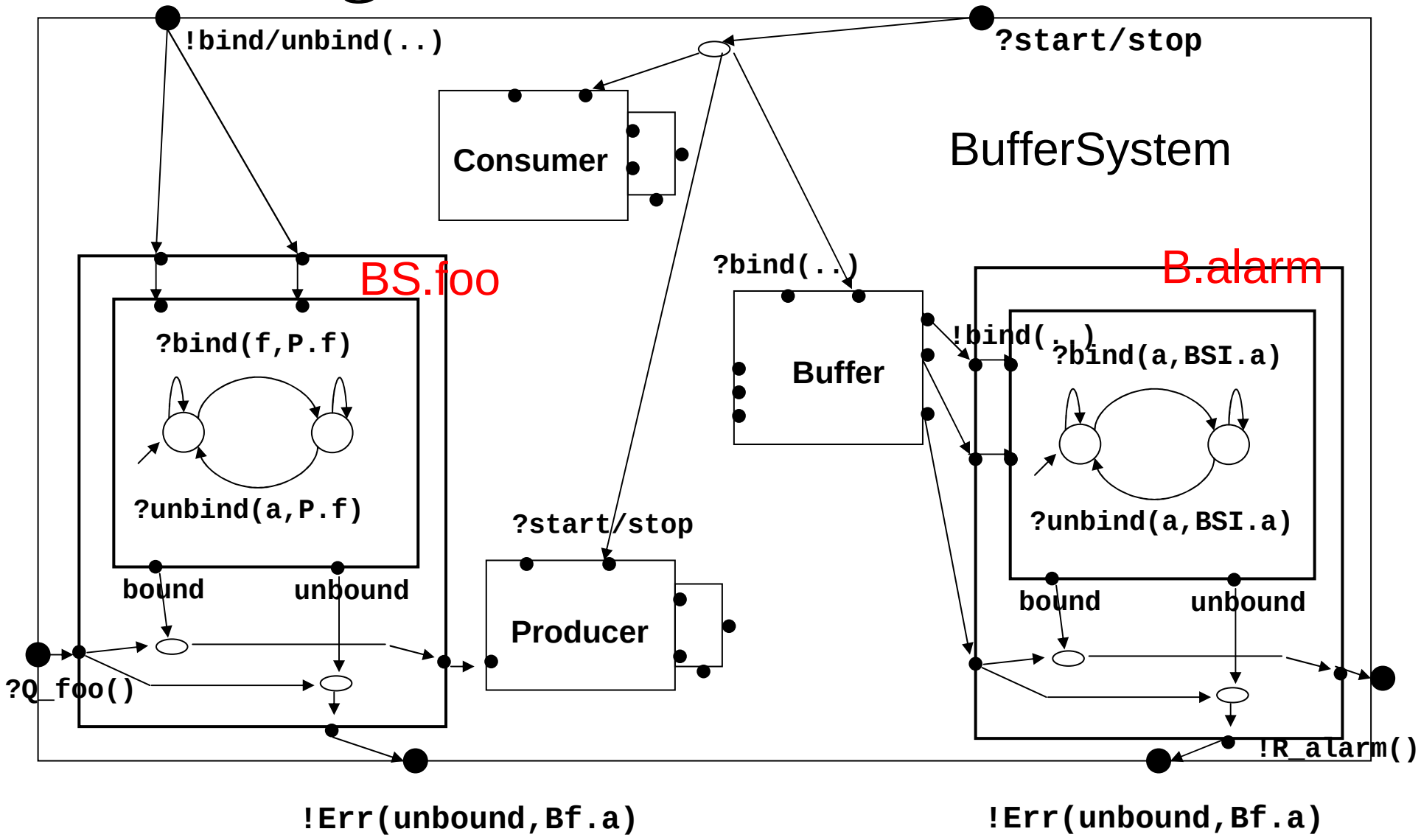
```
<component name=”Buffer"
 <interface name=”get" role=”server"
     signature="components.GetInterface"/>
 <interface name=”put" role=”server"
     signature="components.PutInterface"/>
<interface name=”alarm" role=”client"
     signature="components.AlmInterface"/>
 <content class="components.Buffer">
  <behaviour file=”BufferBehav"
     format="FC2Param"/>
 </content>
 </component>
```



Building the Models: Topology

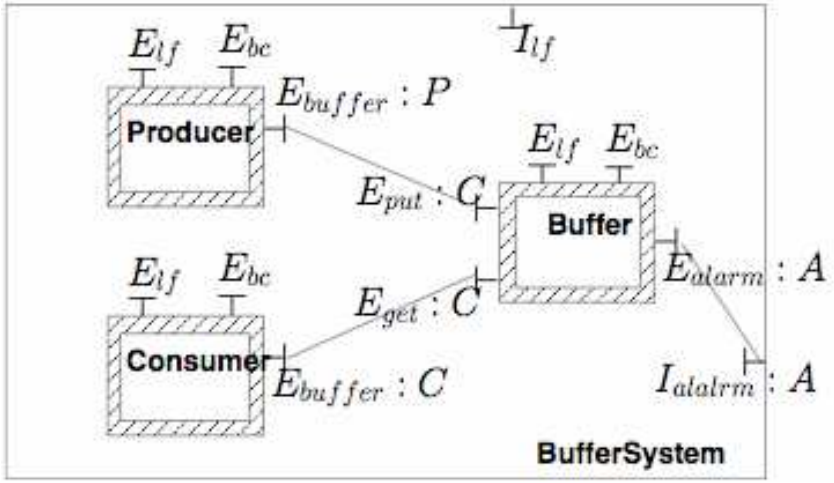


Building the Models: Non-Functional Behaviour



Building the Models: asynchronous behaviour

Static Automaton (1)



- **Content + Controllers** :

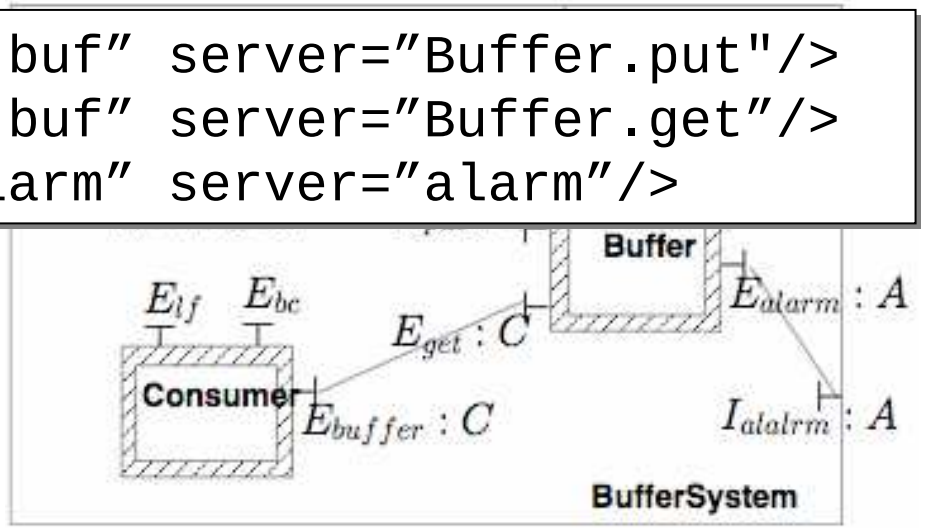
Static vision of the (initial) architecture; the bindings are not yet established, the components not started, but all controllers ready to proceed.

- **Deployment** = establish the bindings, set initial values, and start (hierarchically) all components.

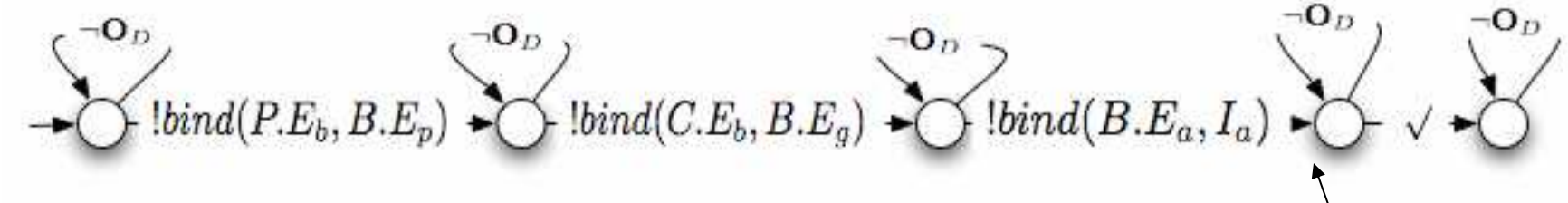
  Part of the ADL, or described in a “deployment file”

Static Automaton (2)

```
<binding client="Producer.buf" server="Buffer.put"/>
<binding client="Consumer.buf" server="Buffer.get"/>
<binding client="Buffer.alarm" server="alarm"/>
```



Deployment automaton :



Missing "**Start**" transition here

O_D = {deployment actions}

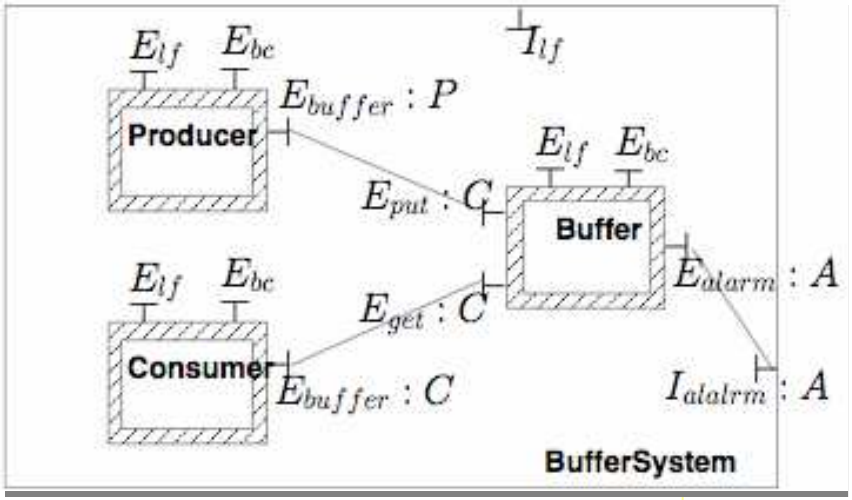
Static automaton = (Controller || Deployment)

Properties

- (1) Absence of errors during deployment (checked on the static automaton)

O_E = {error actions}

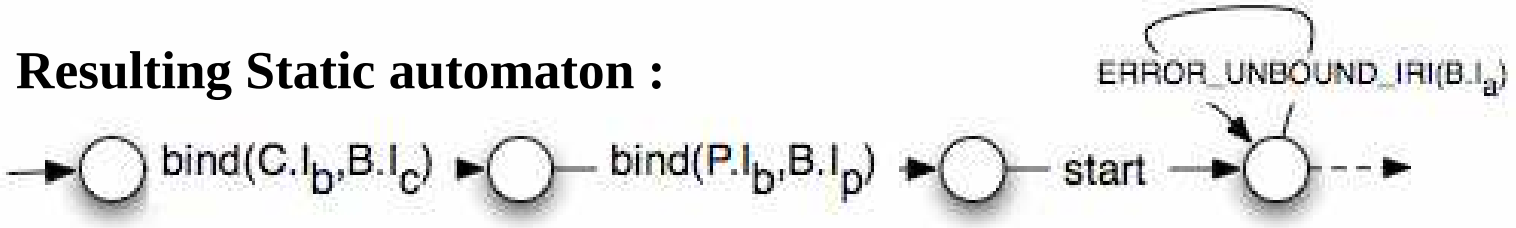
Property (ACTL) :



$$\mathbf{AG}_{true}[\mathbf{O}_E]false$$

e.g. imagine a faulty deployment specification :
-> start Buffer without linking the alarm

Resulting Static automaton :



Properties

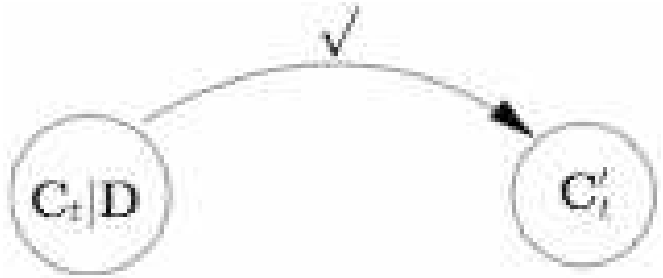
- (2) Functional behaviour (checked on the static automaton)
  - Get from the buffer eventually gives an answer

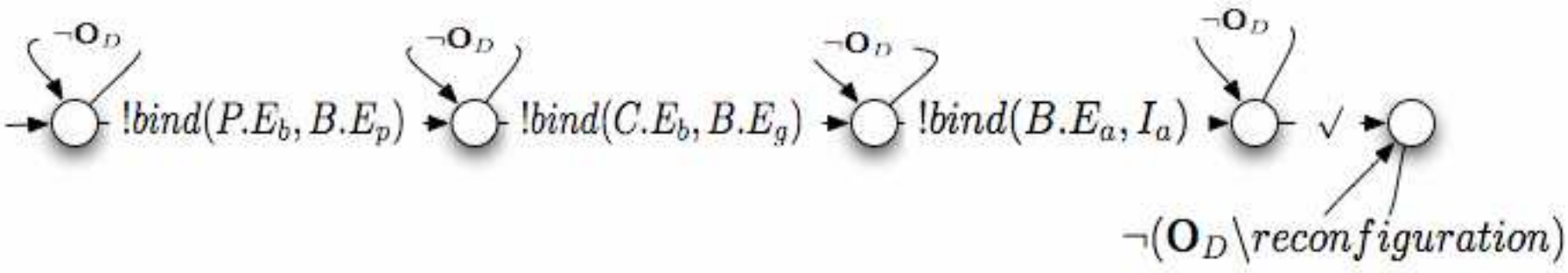
Property (regular μ-calculus) :

$[\ true^*.Q\_get()\ ]\ \mu X.\ (<\ true\ >\ true \wedge [\neg R\_get()\ ]\ X\ )$

Properties

- (3) Functional behaviour under reconfiguration
  - Selected reconfiguration actions are allowed after deployment





Properties

- (3) Functional behaviour under reconfiguration
  - Future update (once the method served) independent of life-cycle or bindings reconfigurations
  - E.g (regular μ-calculus) :

$$[\ true^*.\ Q\_get()\ ]\ \mu X.\ (< true > true \wedge [\neg\ R\_get()\ ]\ X\ )$$

  - With $C'_T \subseteq \{ ?unbind(C.E_b, B.E_g)\ , ?stop(C)\ \}$

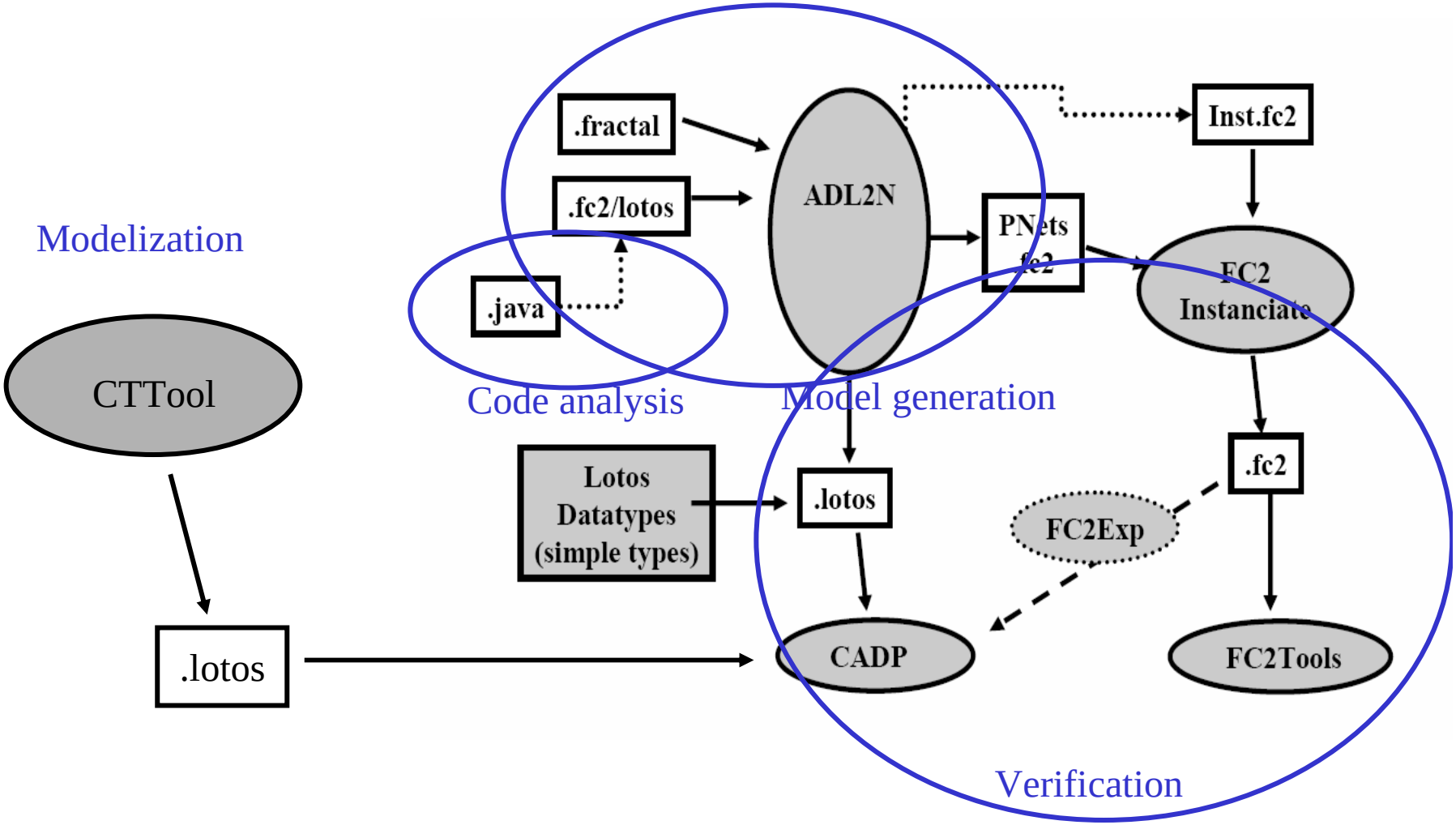
Vercors Platform

- Tool set :
  - CTTool: Architecture and behaviour specification (prototype)
  - Code analysis (prototype)
  - ADL2N: Model generation (available)
  - Bridges with model-checking and verification tools (available)

Supported by FIACRE

An ACI-Security action of the French research ministry

Vercors Platform



Tools: Pragmatics

Avoiding state explosion

1. Distributed model generation (distributor, CADP)
2. Reduced controllers based on deployment
3. On-the-fly mixed with compositional hiding and minimisation

cation engines.

networks of

ce format.

sers.

Home Work : play with CTTool

1. **Lancement de CTTool**
   - Dans le répertoire
     http://www-sop.inria.fr/oasis/Eric.Madelaine/Teaching/RSD2006/CTTool
   - récupérez le logiciel CTTool (CTTool.jar et config.xml); installez-les chez vous dans le même répertoire.
   - récupérez la doc (CTToolReport.pdf)
   - récupérez les fichiers d'exemples: ConsumerProducer.xml et car1.xml
   - Lancez CTTool (attention java 1.5 seulement): java -jar CTTool.jar -lotos -proactive
2. **Exemple du Consumer / Producer**
   - Ouvrez le fichier ConsumerProducer.xml: (Dans l'éditeur CTTool, File->Open->...)
   - Étudiez le diagramme de composants et les diagrammes des machines d'état. Pour mieux comprendre, voir CTToolReport Chapter5.
3. **Exemple d'un Système de contrôle de boite de vitesse : Utilisation des outils CTTool / CADP**
   - **Correction semaine prochaine.**

Next course

3) Application to distributed applications

- ProActive : active object and distributed components
- Behaviour models
- Case-study
- Tools : build an analysis and verification platform

www-sop.inria.fr/oasis/Eric.Madelaine

Teaching/RSD-2006