

---

# Specifying and Generating Safe GCM Components

---

---

INRIA – Sophia Antipolis  
OASIS Team

Antonio Cansado  
Ludovic Henrio  
Eric Madelaine

*Journée SAFA , Sophia-Antipolis, 5 déc 2007*

# Agenda

---

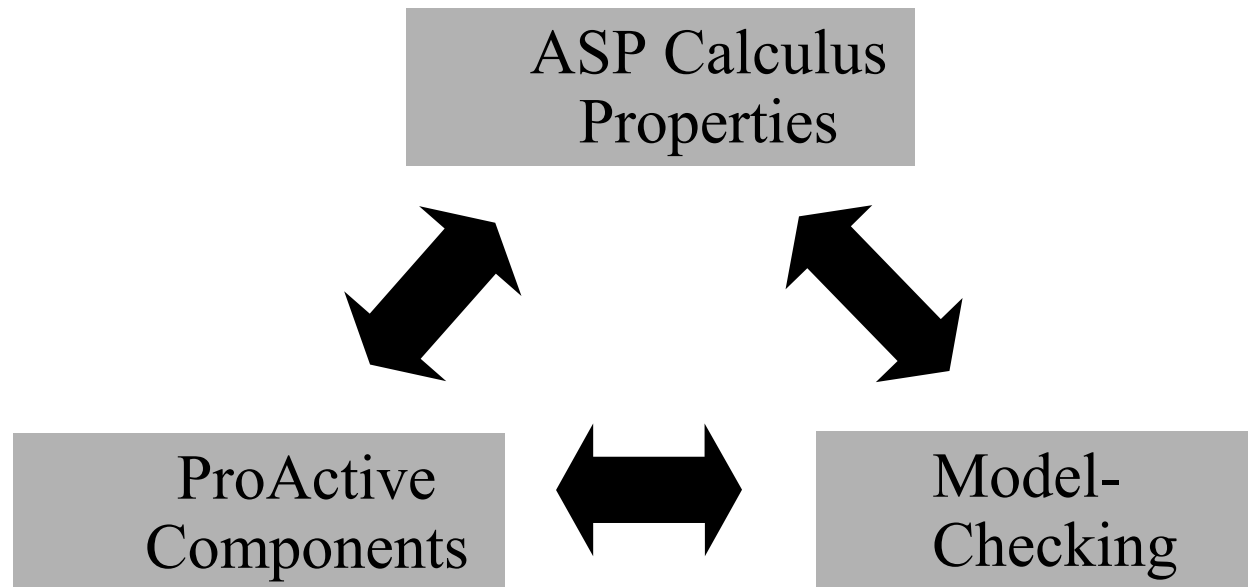
- Motivation
- VCE Prototype (Vercors Component Environment)
- Specification of GCM components
- Generation of safe components



# The OASIS Team

## A Threefold Approach

---



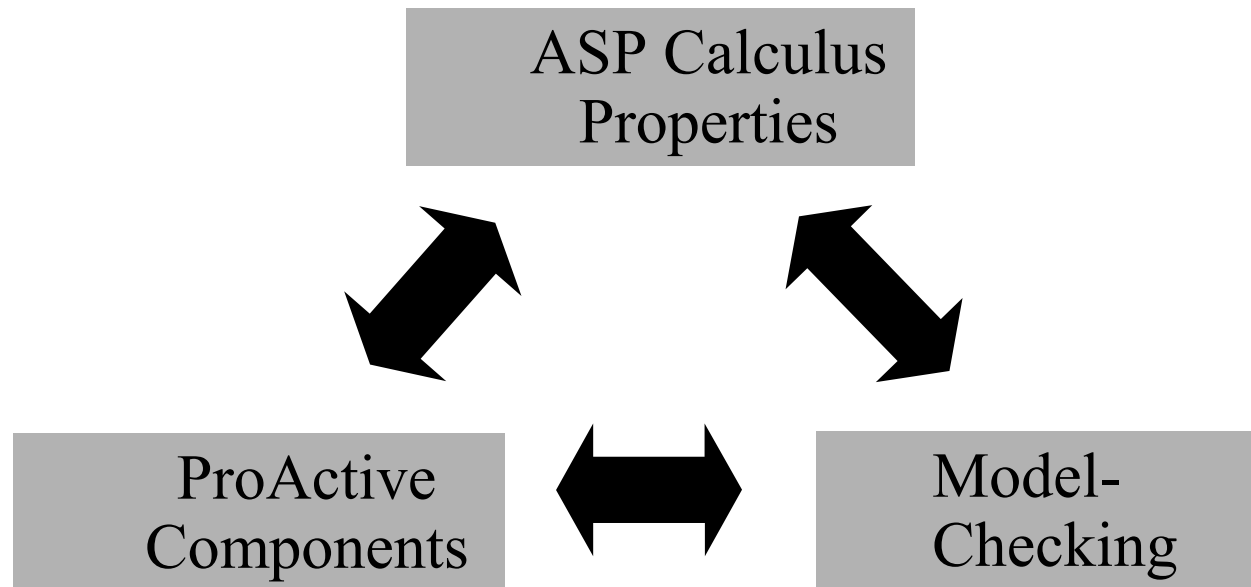
- Proposes fundamental principles, techniques and tools for the design, development, analysis, and verification of **reliable distributed systems**
- Target, application areas: Distributed, Parallel, Concurrent applications, GRID Computing



# The OASIS Team

## A Threefold Approach

---



- SLOGAN /DREAM:
  - Bring the formal methods, tools, guarantees,
  - in the Development Environment => to non-expert programmers.



# Build safe component-based systems

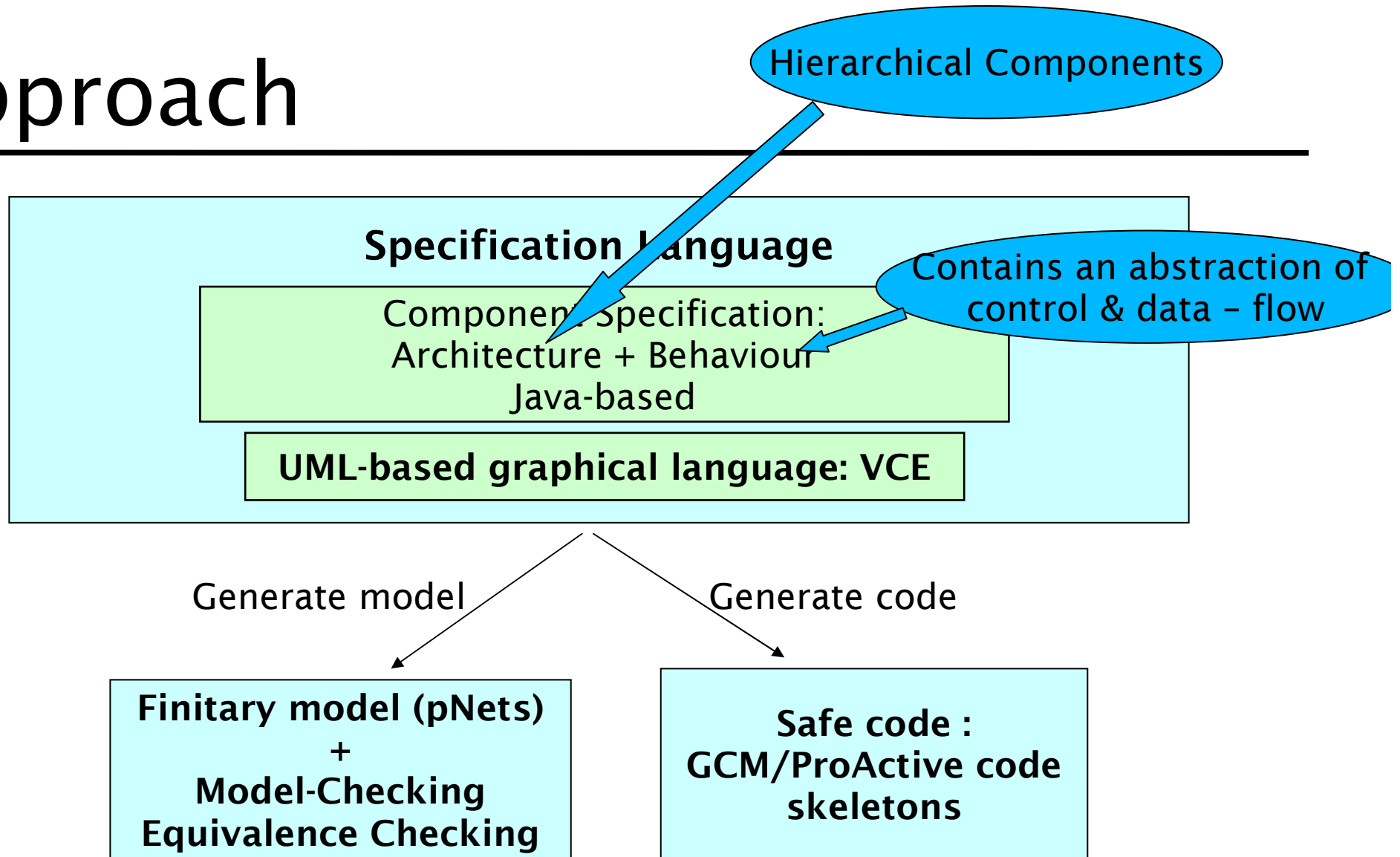
---

- Need :
  - Safe assembly of components
    - **Static typing** of bound interfaces is not sufficient
    - Compatibility of **dynamic behaviour**
      - Formal specification of Components
- Specification Language :
  - Integrate **ADL** and **BDL** (architecture and behaviour)



# Approach

---



# Agenda

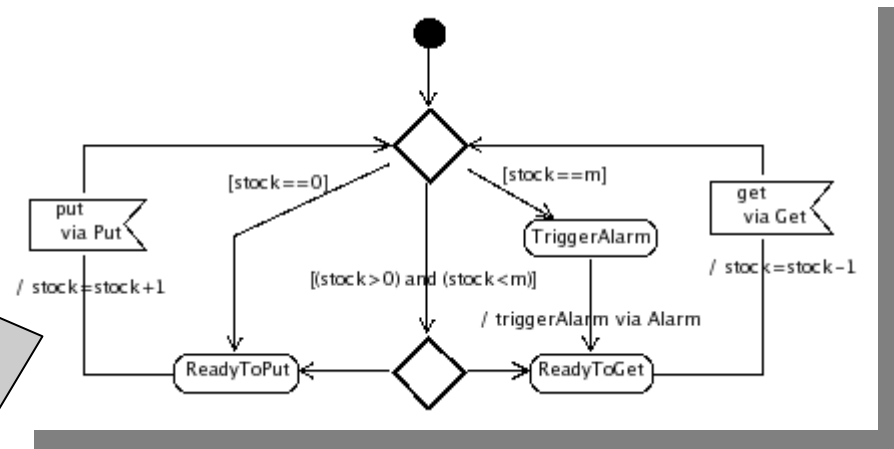
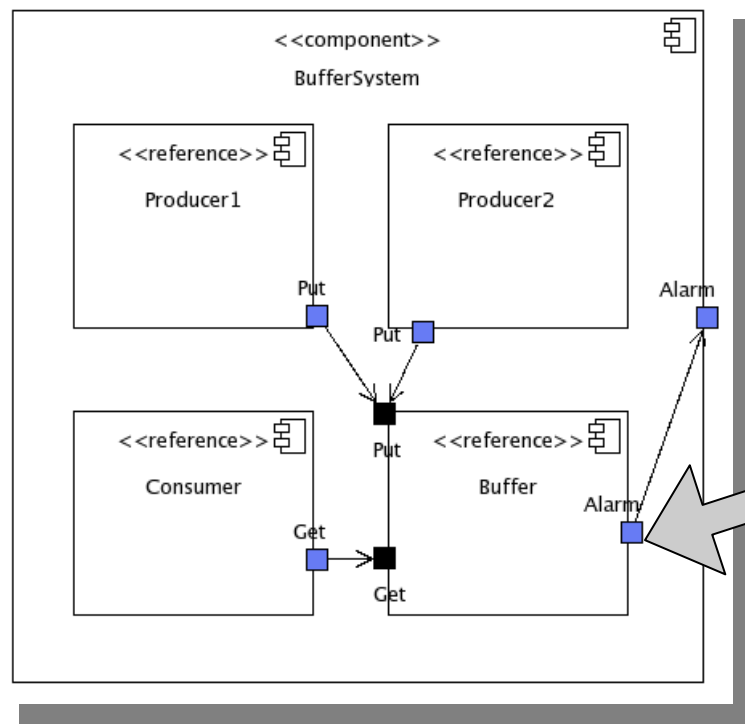
---

- Motivation
- VCE Prototype (Vercors Component Environment)
- Specification of GCM components
- Generation of safe components



# UML2 Prototype: Vercors Component Environment

- Component Diagram
- State Machine Diagram



Need only the **Data** influencing the **control-flow** and the **topology**  
**In practice:** use Java interfaces, + ad-hoc abstraction





# VCE Prototype

---

- **Functional behaviour** of components
  - **Specification** given as a State Machine
  - **Implementation** given as a composition of subcomponents
- Integrated into Eclipse as plugins
- Generation of **behavioural model** for Verification



# VCE: Snapshot

The screenshot displays the Eclipse IDE interface for a VCE (Value-based Component Environment) project. The main workspace shows a component diagram for 'BufferSystem' with references to 'Producer1', 'Producer2', 'Consumer', and 'Buffer'. A red circle highlights the 'Alarm' port on the 'Buffer' component. The Outline view on the right shows the composite structure, with 'Port Alarm' also circled in red. The Problems view at the bottom shows an error message: 'The port BufferSystem.Alarm doesn't have provided nor required interfaces', which is also circled in red.



# VCE: Validate and Verify

---

- Use of sound semantic model - **pNets**
  - Hierarchical, Parameterized Networks of Labelled Transition Systems
  - Simple predefined datatypes
- Generate **Behavioural Models** based on pNets
  - Functional and Non-Functional concerns
- **Model-check** the behavioural model
  - Deadlocks, Reachability, Safety, Liveness
  - Properties specified as automata
  - Functional verification + safety of reconfiguration



# Agenda

---

- Motivation
- VCE Prototype (Vercors Component Environment)
- **Specification of GCM components :**  
(Grid Component Model)
- Generation of safe components



# GCM / ProActive Components

---

- Hierarchical components
- Strong encapsulation of functional activity
- Interfaces expose all communication between components
  - Sorts given by: method calls, arguments, and return types
- State-full
  - Persistent variables (in primitive components)
  - Request queues (in primitives and composites)



# Specify the Behaviour of GCM / ProActive Components

---

- Service policy
  - How to serve requests from the **request-queue**
    - **Calls on service methods** (and local methods)
    - Depends on local component state
    - Most used: `serveOldest(filter)`
- A policy can be concurrent (for composite parallel components)



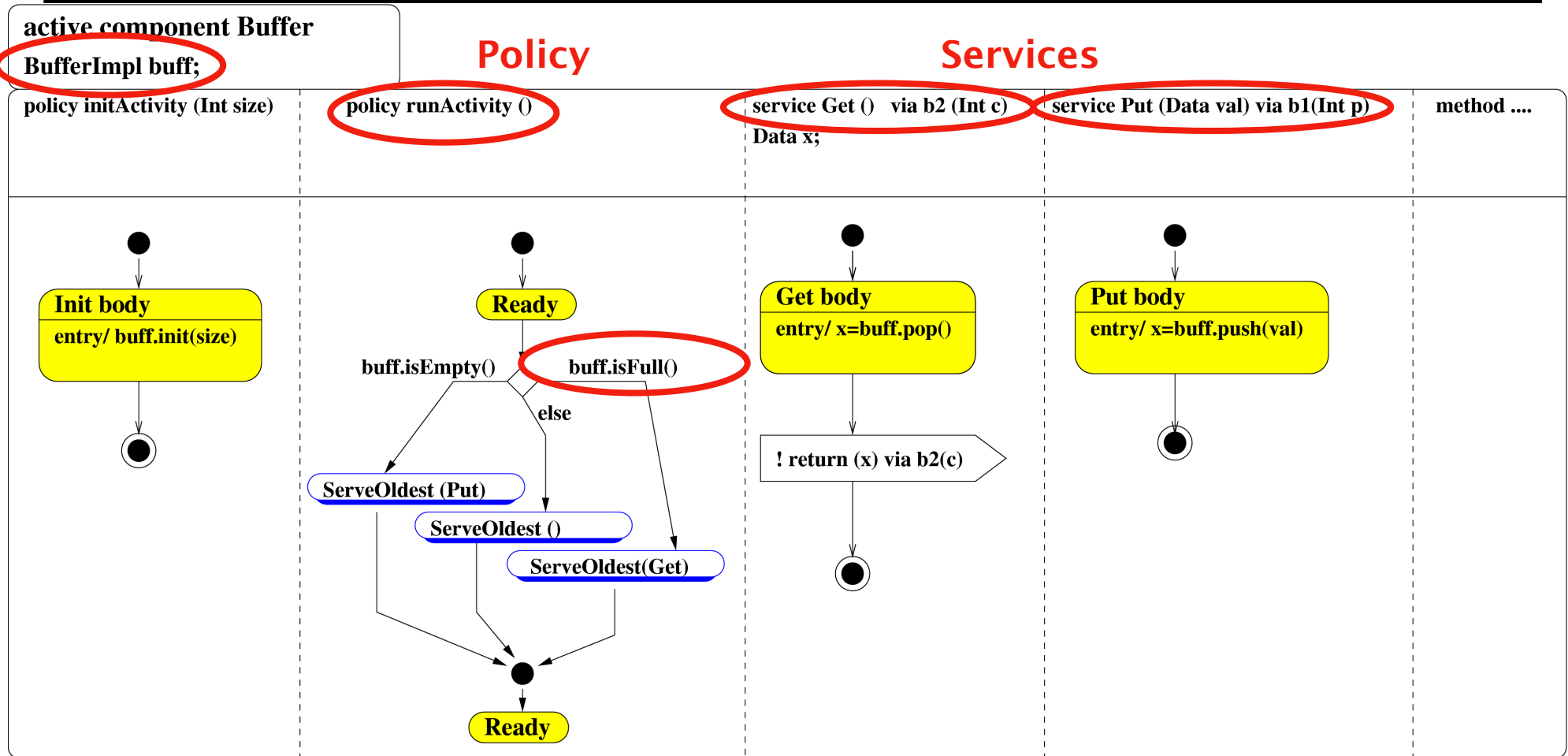
# Behaviour of GCM / ProActive Components

---

- Service methods
  - Contain an abstraction of the business code
    - Using *real* Java user classes
  - Are unaware of the request-queue
  - Are not allowed to call (local) service methods
    - the only allowed interference is through shared variables
  - Triggers events
    - calls on client interfaces



# Behaviour of GCM / ProActive components



✓ Generate skeletons for GCM components





# Agenda

---

- Motivation
- VCE Prototype (Vercors Component Environment)
- Specification of GCM components
- **Generation of safe components**



# Guidelines for Generating ProActive Code

---

- Goal
  - Same behaviour as the specification
- Generated Java/ProActive code
  - Full and final generation:
    - GCM ADL
    - Code of Fractal controllers
    - Service policy: `runActivity()`
  - Skeletons for:
    - service methods (methods defined in server interfaces)
    - With hooks to fill-in final implementation (User-defined Business code)



# Black-Box View of Components

---

- Externally, components are seen as:
  - **Sequential** component
    - These may implemented as GCM **primitives** or **composites**
    - Our choice: generate a primitive GCM component
  - **Concurrent** component
    - These must be implemented as GCM **composites** to deal with concurrency
    - Our choice: explore the architecture for finding sequential parts which will become primitives, and concurrent ones that become composites



# Generate a GCM Primitive

---

- Can be generated in an isolated manner
  - All provided and required interfaces are known
  - No need to know its environment / architecture
- **Body:** `runActivity()` + **service methods**
  - Encode a State-Machine with Java/ProActive code
    - **Non-determinism is an issue**
  - Calls on client interfaces create futures: all communication is handled by ProActive
- **Control code shouldn't be modified**
  - Ensure the same behaviour as the specification



# Generate a GCM Composite

---

- Architecture definition: ADL
  - XML with static information about the architecture of the component
- Service Policy and Sequential behaviour are treated by primitives



# Data within Components

---

- In the specification, there are user datatypes
  - Used to generate directly Java interfaces of types
- Need to specify an abstraction to be used within the state machines



# From the Architecture Definition

---

## GCM ADL

```
<component name="CashDesk">
  <interface signature="LightDisplayControlIf" role="server"
name="lightDisplayControlIf"/>
  ...
  <component name="CashDeskApplication">
    <interface signature="ApplicationEventHandlerIf" role="server"
name="applicationEventHandlerIf"/>
    ...
    <content class="CashDeskApplication"/>
    <controller desc="primitive"/>
  </component>
  ...
  <binding client="this.lightDisplayControlIf"
server="LightDisplayController.lightDisplayControlIf"/>
  <controller desc="composite"/>
</component>
```



# From the Architecture Definition

---

## Fractal Controllers

```
public String[] listFc() {
    return new String[]{ CASHBOXEVENTIF_BINDING };
}

public Object lookupFc (String clientItfName) {
    if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
        return cashBoxEventIf;
    return null;
}

public void bindFc (String clientItfName, Object serverItf) {
    if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
        cashBoxEventIf = (CashBoxEventIf)serverItf;
}

public void unbindFc (String clientItfName) {
    if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
        cashBoxEventIf = null;
}
```





# Generate GCM / ProActive code

---

## Service Policy: runActivity()

```
public void runActivity(Body body) {
    Service service = new Service(body);
    while (body.isActive()) {
        switch(state){
            case ENABLE: // states of the automata
                if (service.hasRequestToServe()) {
                    Request request = service.getOldest();
                    if
(request.getMethodName().equals("scanCreditCard")) {
                        service.serveOldest();
                    } else if (request.getMethodName()
.equals("expressModeDisabled")) {
                        service.serveOldest();
                    }
                    else {
                        service.serveOldest();
                        state = State.DISABLE;
                    }
                }
                break;
            case DISABLE
            ... // other states
```



# Service Methods

---

- Which part of them needs to be generated from the specification ?
  - Control structure (Control-flow and data-flow)
  - Method calls performed on client interfaces
    - Creation of futures
    - Flow of futures
  - Data-usage
    - First use of a future is meaningful
      - [wait-by-necessity](#) – synchronisation points
    - Programmers may use these data within the computation



# Generate GCM / ProActive code

---

## Service Method

```
public void pinEntered(PIN pin) {
    if (creditInfo != null) {
        Transaction transId = bankIf.validateCard(creditInfo, pin);
        // call on client interface
        // arguments may be futures
        if (ProActive.getFutureValue(transId) != null) {
            Info info = bankIf.debitCard(transId, runningTotal);
            if (ProActive.getFutureValue(info) != null){
                Sale sale = new SaleImpl(
                    new PaymentModeImpl(PaymentModeImpl.CREDIT),
                    products, runningTotal);
                saleRegisteredIf.bookSale(sale);
                info.getInfo(); // wait-by-necessity
                init();
            }
            ...
        }
    }
}
```



# Correctness of the Generated Code

---

- Our modelisation preserves safety formulas
  - Including abstraction of value-passing data
- Semantics of communication is sound
  - ProActive library is based on ASP-calculus (and its properties)
  - Semantics are simulated by the behavioural model (pNets) which closely fits in ProActive's MOP
- Reliability of code is highly dependent on the set of formulas and the abstraction used in them



# Some Open Questions

---

- How to manage **hooks** for filling in the final implementation?
  - Don't limit the programmer
  - Don't interfere with the control code
- Implementation issues
  - Use of Java annotations?
- Where is the right place to put abstractions (which depend on the set of formulas to prove) ?
- Which is the best way to have human-readable code?  
*[Code based on State-machines]*



# Conclusions and Perspectives

---

- **Short-term**
  - Graphical tool for GCM Specification (**VCE**)
    - Asynchronous distributed components
  - Generation of Safe code (A. Cansado PhD)
    - Java code generated
- **Long-term**
  - Advanced GCM-specific features
    - Multicast, gathercast interfaces
    - Autonomicity : behaviour of the membranes of composites
  - Formalisation and proofs
  - Language for expressing reconfiguration of components (related to M. Rivera PhD)



# VCE : roadmap

---

- **Current (V0.5):**
  - Hierarchical components + ad-hoc State-Machines
  - Bridge to CADP through TTool
- **Next (V1):**
  - Various small enhancements, UML2.1 State-Machines
  - GUI for Data Abstraction
- **V2:**
  - Grid specific concepts (GCM/ProActive runactivity, multicast/gathercast, ...)
  - Reconfiguration specification
  - Bridge to CADP through FIACRE format and ADL2N



# References

---

- <http://www-sop.inria.fr/oasis/Vercors>
- **Behavioural Models**  
Model checking distributed components: The Vercors platform.  
T. Barros, A. Cansado, E. Madelaine, and M. Rivera  
In 3<sup>rd</sup> workshop on Formal Aspects of Component Systems  
Prague, Czech Republic, Sep 2006. ENTCS
- **Case-study: CoCoME**  
A Specification Language for Distributed Components implemented in GCM/ProActive  
A. Cansado, D. Caromel, L. Henrio, E. Madelaine, M. Rivera, and E. Salageanu  
Lecture Notes in Computer Science. Springer, (to be published 2007)
- **Diagrams for GCM components**  
Specifying Fractal and GCM Components With UML  
S. Ahumada, L. Apvrille, T. Barros, A. Cansado, E. Madelaine, and E. Salageanu.  
In XXVI Inter.Conf. of the Chilean Computer Science Society, Chile, Nov. 2007. IEEE



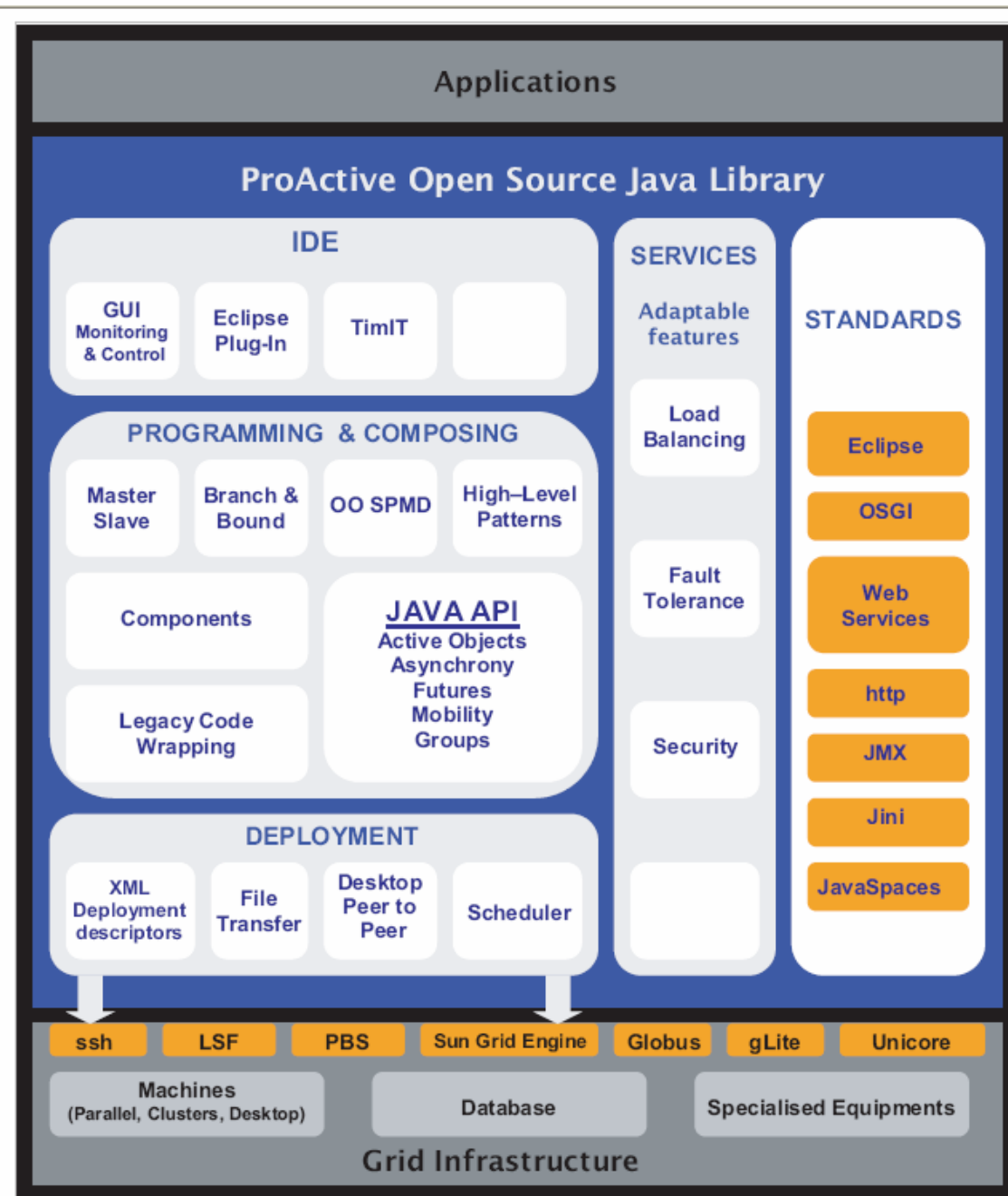


# Annexes

---

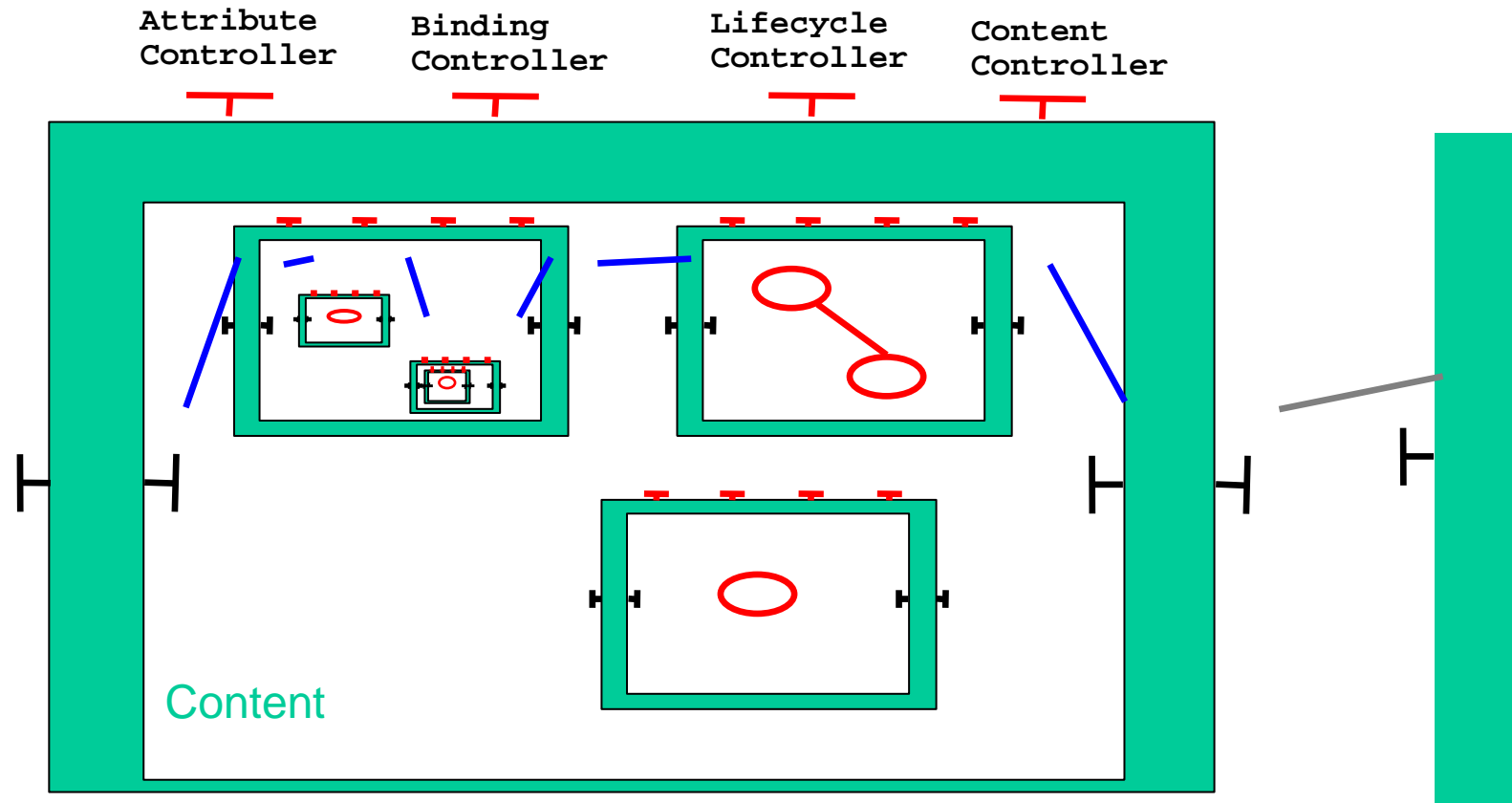


# ProActive Architecture



# Fractal hierarchical model :

composites encapsulate primitives, which encapsulate code



now the basis for the GCM

# Architectural View

