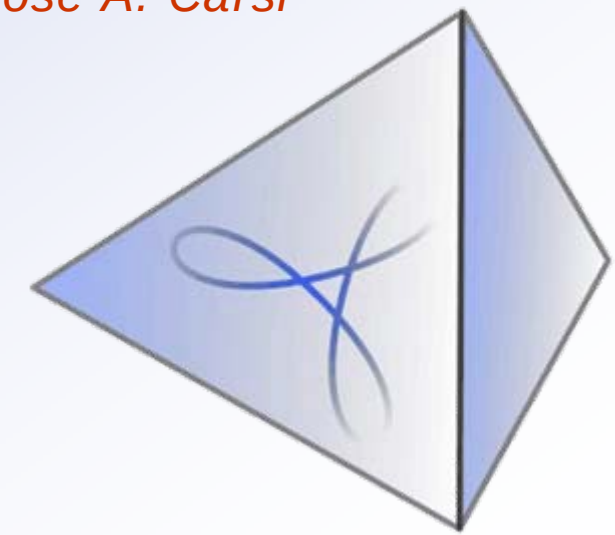
UNIVERSIDAD POLITECNICA DE VALENCIA

MODELS, ENVIRONMENTS, TRANSFORMATIONS AND APPLICATIONS

Ingeniería del Software y Sistemas de Información

# Dynamic Adaptation of Aspect-Oriented Components

***Cristóbal Costa***, *Jennifer Pérez, Jose A. Carsí*



**10th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE'07)**

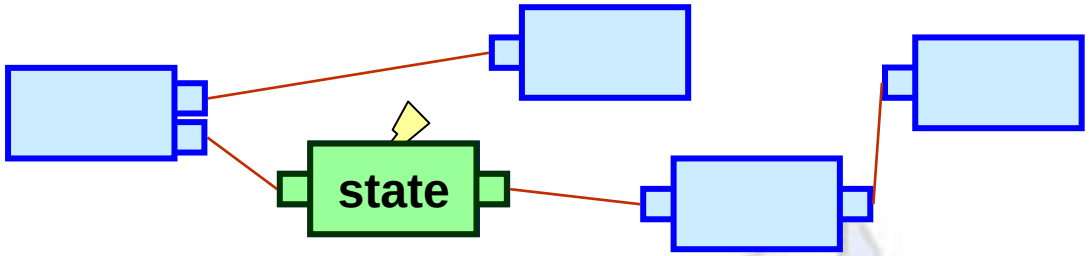
July 9-11, 2007 – Tufts University, Medford, Massachusetts, USA

# Outline

- Motivation
- Aspect-oriented components
- Case Study
- Evolving component types
- Evolving component instances
- Conclusions and further work

# Motivation

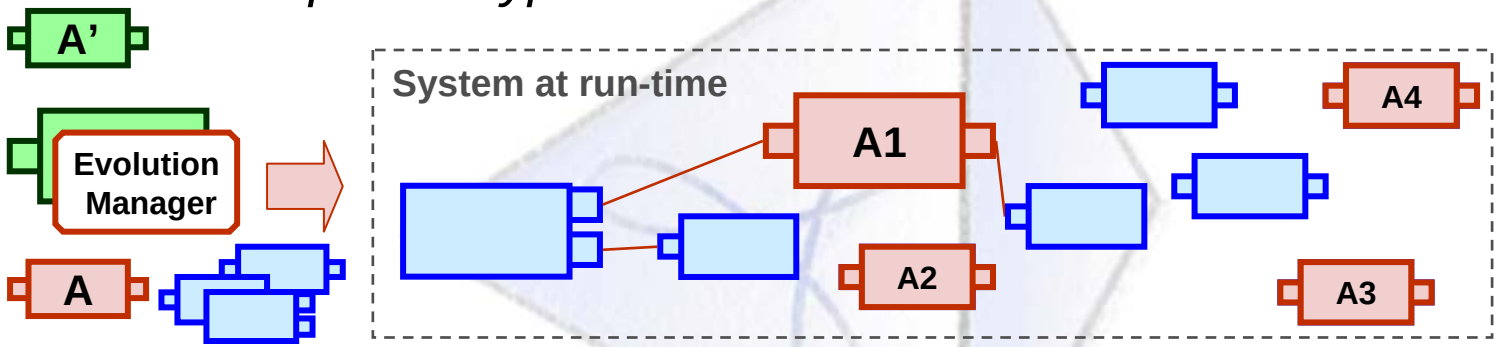
- Complex software systems undergo changes




- Component replacement has disadvantages
  - It is not appropriate when minor changes are done
  - Component state is lost
  - The overall system is affected by the replacement process
  - To restart a Component increases the performance cost
  - Some systems cannot stop their activity
- Can we design components capable from being adapted at run-time?

# Motivation

- Several works have addressed dynamic adaptation
  - Are developed for a specific platform
  - Require a centralized evolution infrastructure
  - Only consider the adaptation of component instances *… component types*?



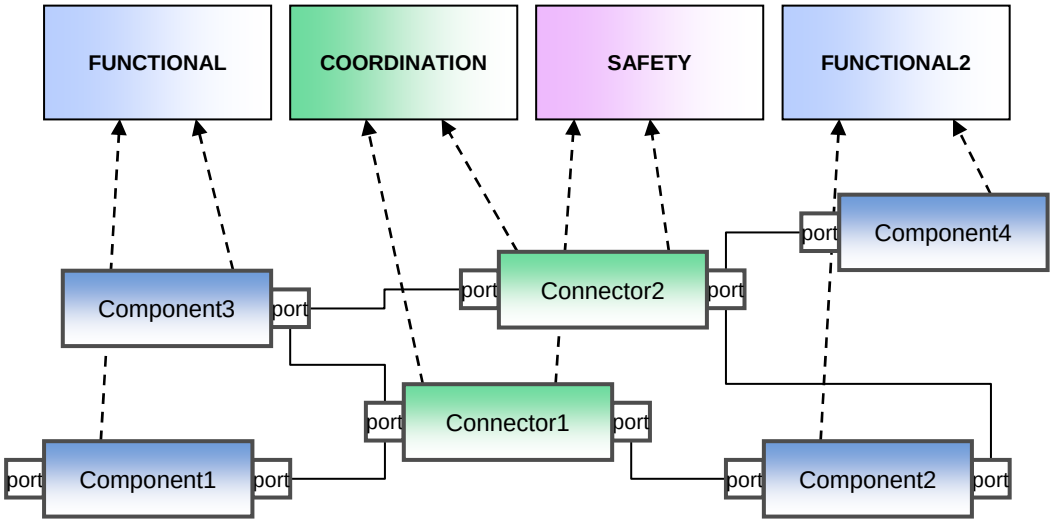

- We outline how to build a platform-independent infrastructure that allow each component type to be adapted dynamically

# Outline

- Motivation
- Aspect-oriented components
- Case Study
- Evolving component types
- Evolving component instances
- Conclusions and further work

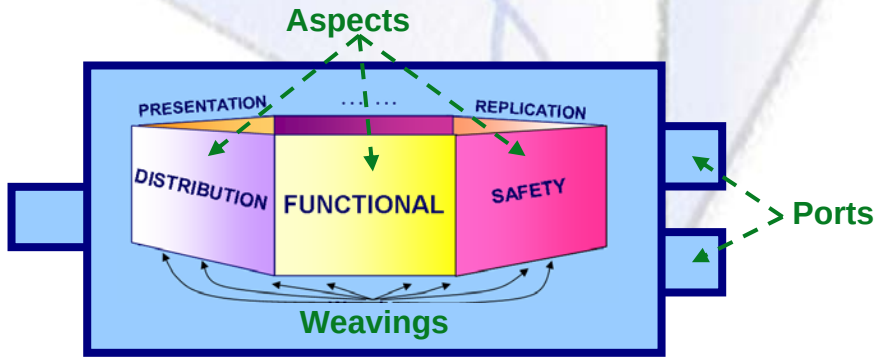
# Aspect-oriented components

- We use aspect-oriented components to benefit from its flexibility and maintenance
- Aspects modularize crosscutting-concerns:
  - One crosscutting-concern can be modularized in one or more aspects
  - Reduction of Complexity: reduction of tangled code



# Aspect-oriented components

- PRISMA
  - An approach to develop technology-independent, aspect-oriented software architectures
  - Components, aspects, and weavings are independent of each other
  - Follows the Model-Driven Development paradigm
    - PRISMA Software Architectures can be automatically compiled for a technological platform through code generation techniques

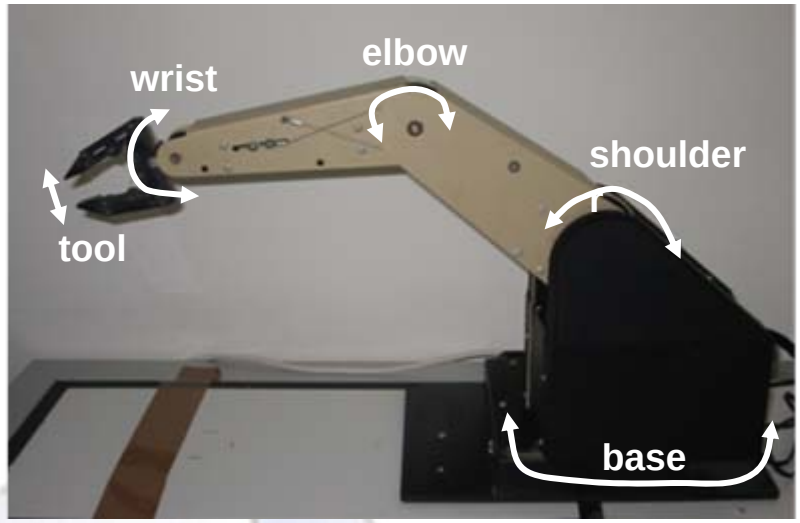


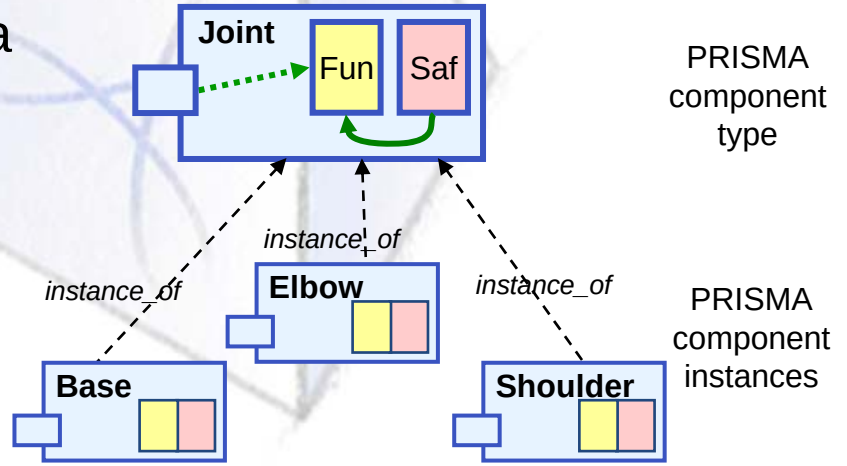
# Outline

- Motivation
- Aspect-oriented components
- Case Study
- Evolving component types
- Evolving component instances
- Conclusions and further work

# Case Study

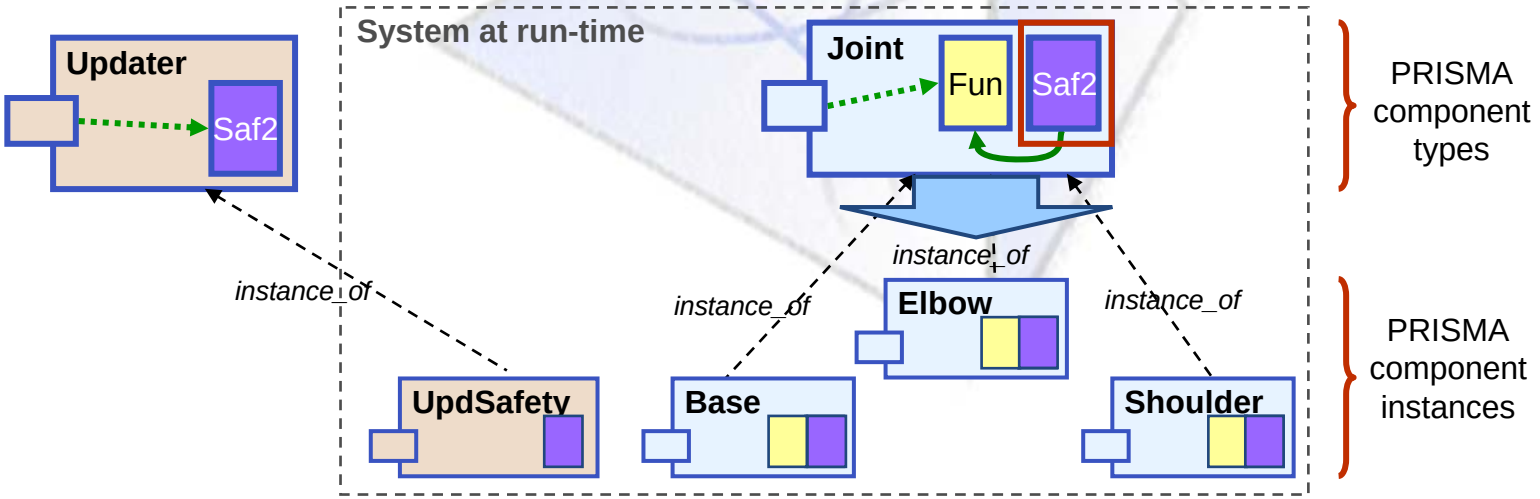
- TeachMover
  - A robotic arm whose movements are controlled by different joints
  - A joint is modelled as a PRISMA component
  - Each joint is controlled at runtime by an instance of a Joint *component type*



# Case Study

- After deployment
  - A new requirement emerges
  - Safety aspect needs to be upgraded to “Safety2”
- Dynamic Adaptation is needed
  - Performed by a new component
  - UpdSafety component instance is introduced in the running system to trigger the adaptation process

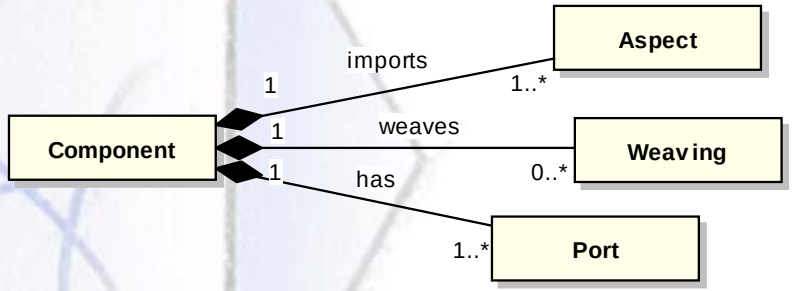


# Outline

- Motivation
- Aspect-oriented components
- Case Study
- Evolving component types
- Evolving component instances
- Conclusions and further work

# Evolving component types

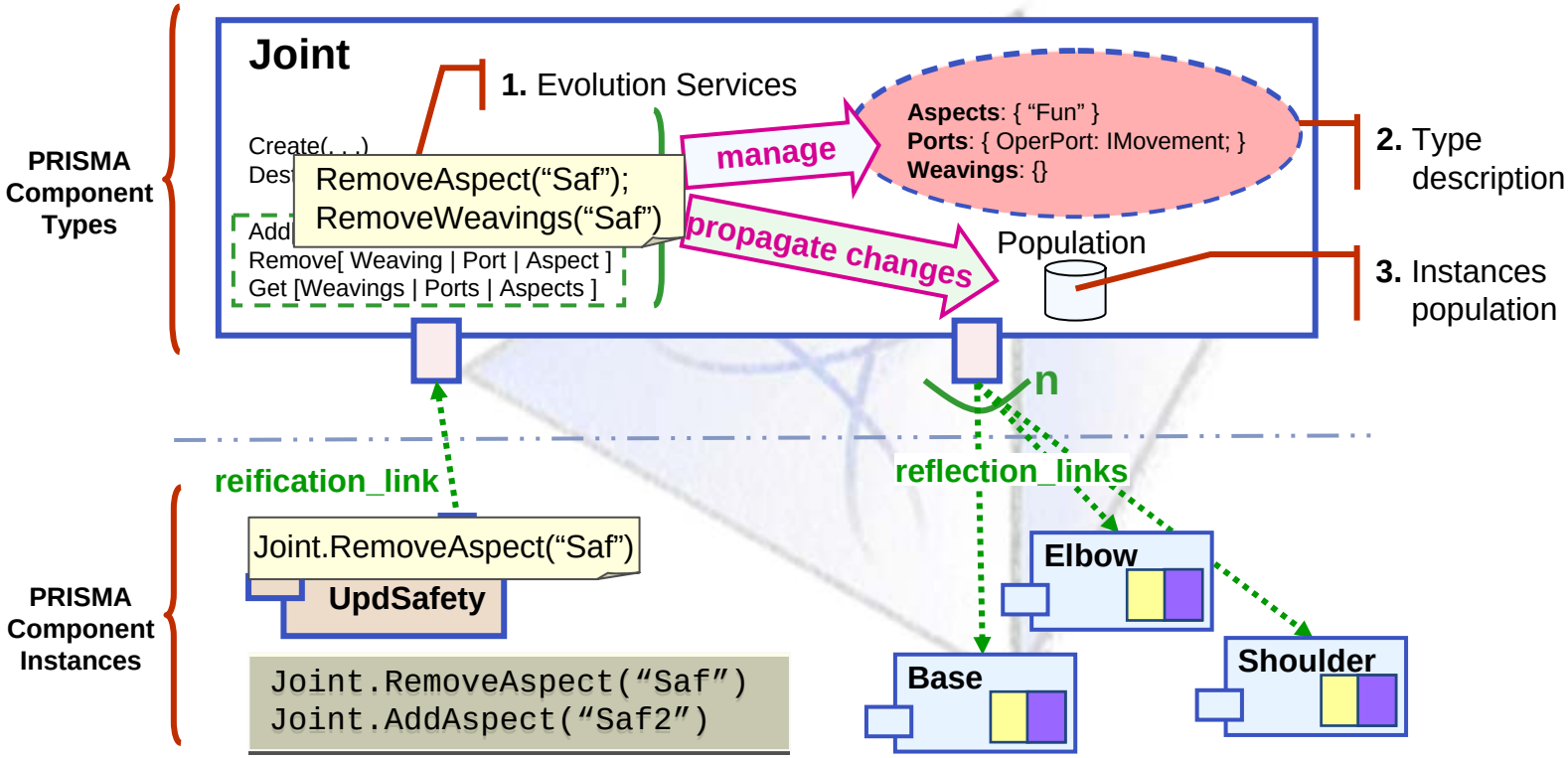
- Each component type:
  - Is a factory of component instances
  - Provides a set of evolution services to change its structure
  - Is responsible of updating its running component instances
- Evolution services modify the main parts of a component
  - To dynamically evolve a component, its structural parts have to be identified first
  - Evolution services depend on the component metamodel




  - Two kinds of evolution services:
    - Introspection services - provide information
      GetAspects(), GetWeavings(), GetPorts(), …
    - Type Evolution services - type modification
      AddAspect(), RemoveAspect(), AddWeaving(), …

# Evolving component types

- Component type reflective structure
  - Component types and component instances are placed in different abstraction layers

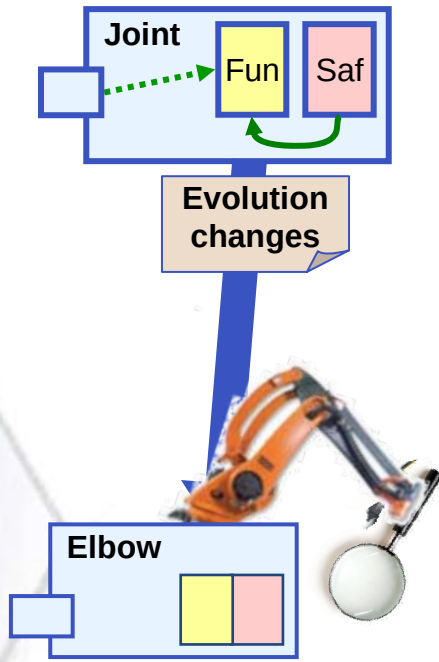


# Outline

- Motivation
- Aspect-oriented components
- Case Study
- Evolving component types
- Evolving component instances
- Conclusions and further work

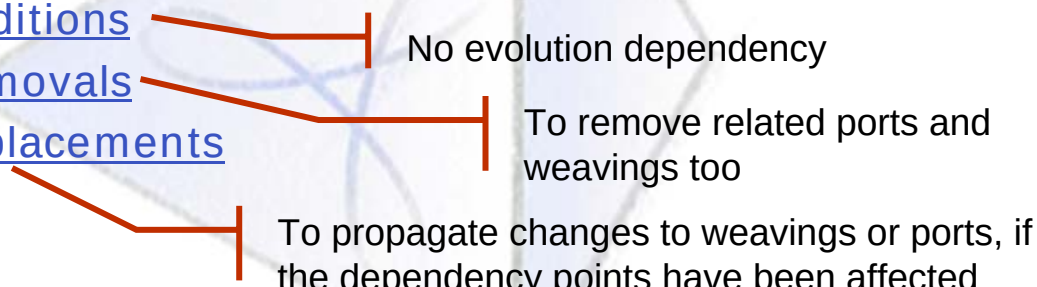
# Evolving component instances

- Component types do not directly perform evolution changes inside their instances
- Each instance is responsible of updating itself
  - It is the only one who knows itself (its state at run-time)
  - It decides **when** to execute evolution changes, ensuring that running transactions are finished in a safe way
- Runtime evolution consists of modifying only those parts affected by the change, while the others are unaware of these changes



# Evolving component instances

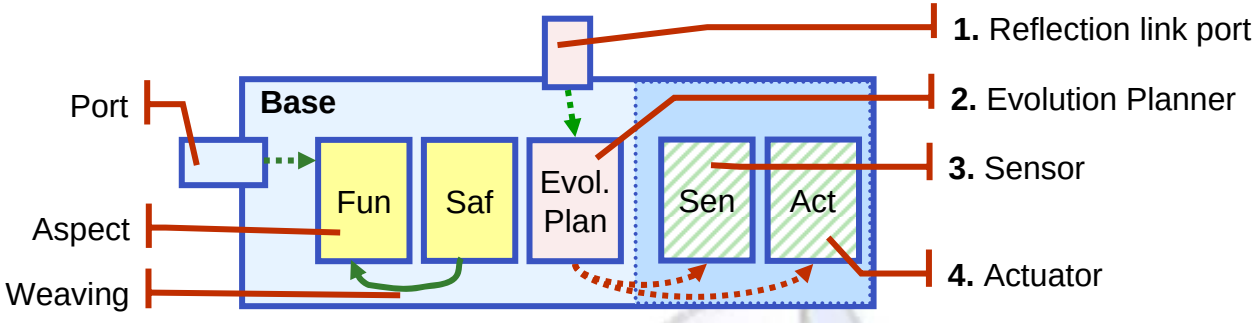
- Providing Runtime Evolution
  1. Maximize the independence among the internal parts
  2. Identify Evolution Dependencies
     - PRISMA: EvDep[ {aspect→port},{aspect→weaving}]
     - Aspect service: *MoveJoint(int coord)*
     - Weaving: *MoveJoint(int coord) after SafePos(int coord)*
     - Port: *IMovement* (which contains the MoveJoint service)
  3. Identify how to react in response to these changes
     - Aspect additions
     - Aspect removals
     - Aspect replacements




- This knowledge is introduced in the *EvolutionPlanner* aspect

# Evolving component instances

- Component instance adaptation structure



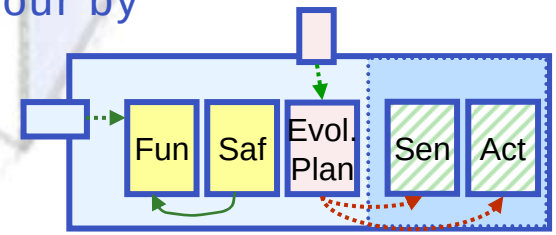

- *Reflection* link → receives the changes to be applied
- *EvolutionPlanner* → coordinates the actions to be performed by the Actuator and Sensor aspects
- *Actuator* and *Sensor* aspects
  - Perform platform-dependent evolution operations
  - Sensor → supervises what is going on (parts are ready to be changed? parts are busy?)
    - `getAspectState()`, `getWeavingState()`, ...
  - Actuator → performs changes on the running instance
    - `addAspect()`, `stopAspect()`, `startAspect()`, ...

# Outline

- Motivation
- Aspect-oriented components
- Case Study
- Evolving component types
- Evolving component instances
- Conclusions and further work

# Conclusions

- Dynamic Adaptation of Components
  - The whole system is unaware of the adaptation process
  - The evolved component does not need to be restarted
  - The state of the evolved component is not lost
- A generic infrastructure has been described
  - Adaptation of both component types and component instances
  - Autonomous point of view
    - Each component type manages its instances
    - Each component instance decides when to adapt itself
  - Use of aspects to benefit from:
    - To change the component behaviour by adding/removing aspects
    - Better maintenance of the evolution code



# Further work

- Should a component publish its internal structure to other software entities?
  - Use of component authentication techniques
- How can we avoid undesirable changes on the structure?
  - Constraints for component type evolution
    - What kind of evolution services can be executed?
- Is the use of finer evolution services convenient?
  - Dynamic evolution of services and attributes
  - High performance costs

# Questions

Thank you for your attention


UNIVERSIDAD POLITECNICA DE VALENCIA

**Cristóbal Costa**

Information Systems and Software Engineering Research Group
Department of Information Systems and Computation
Polytechnic University of Valencia
Spain

Home page: http://issi.dsic.upv.es/~ccosta
Email: ccosta@dsic.upv.es
The **PRISMA** project: http://prisma.dsic.upv.es