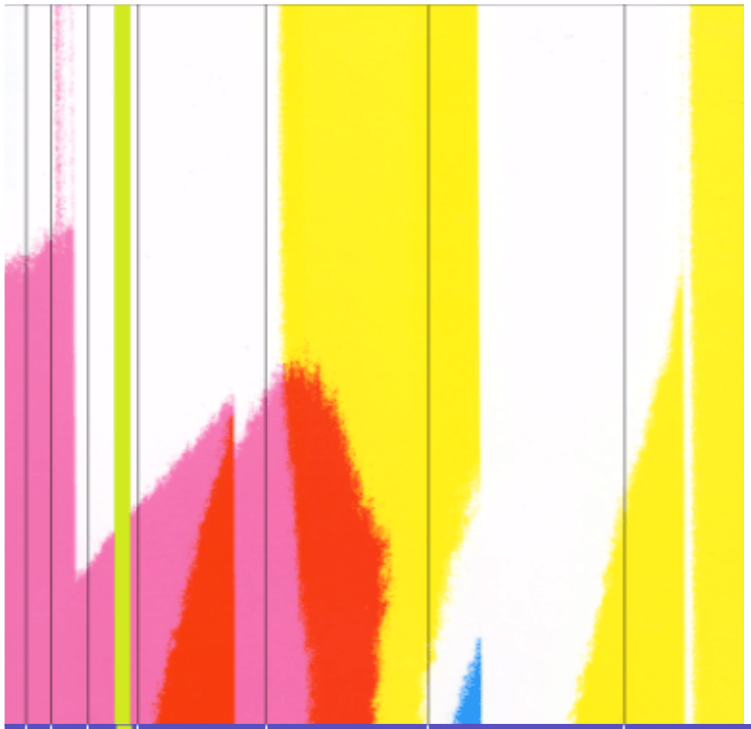


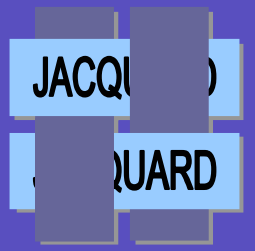
DECOR'04

1st French Conference on Software Deployment and (Re) Configuration

October 28-29th 2004
Grenoble, France.

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

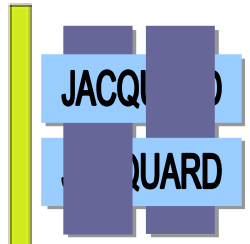


A Multi Personality Environment for Configuration & Deployment of Component-Based Applications

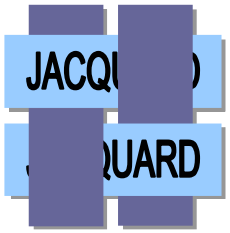
Areski Flissi, Philippe Merle

Areski.Flissi@lifl.fr, Philippe.Merle@inria.fr

Projet Jacquard INRIA Futurs
Laboratoire d'Informatique
Fondamentale de Lille (LIFL)
UMR CNRS 8022 / USTL
59655 Villeneuve d'Ascq

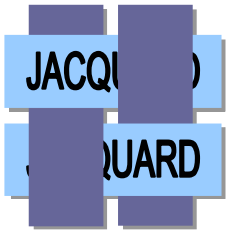


INRIA



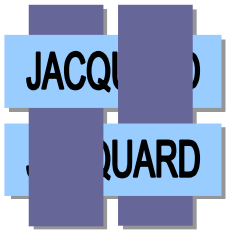
Context

- INRIA Futurs Jacquard Project:
  - Weaving of Software Components
    - *Software engineering to produce new component-oriented platforms and new methodological and technical approaches to design and exploit these applications*
- ITEA OSMOSE Project
  - WP2 Generic Deployment Frameworks
    - A general Fractal component-based orchestration framework and a set of generic tasks for deploying component-based applications independently of any component oriented platform
  - WP3 Middleware Platforms, Tools and Management
    - Task 2: CORBA/CCM Platform



Outline

1. Introduction
2. A multi-personality environment for configuration & deployment of component-based applications
3. An abstract deployment model
4. Conclusion
5. Perspectives and future work

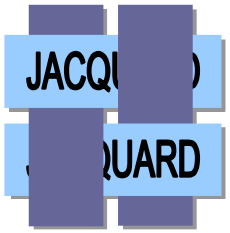


1. Introduction

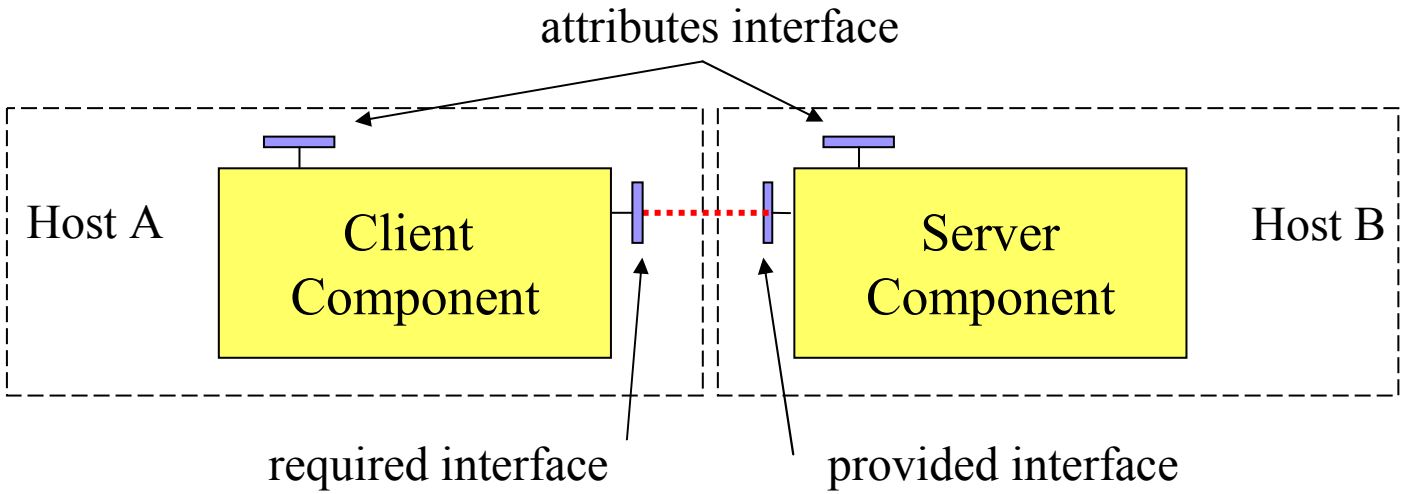
• Generalized use of the component-oriented approach to build distributed applications

• Multiplication of component models and middleware, *e.g.*
  - SUN EJB, OMG CCM, ObjectWeb Fractal, Microsoft .NET, etc.

• For each model/platform, a specific architecture description language (ADL) is used by architects to describe application software architectures, *e.g.*
  - {OMG IDL,CCM CAD/CPF/CCD} for CORBA Components Model
  - {Fractal IDL,Fractal ADL} for ObjectWeb Fractal Model

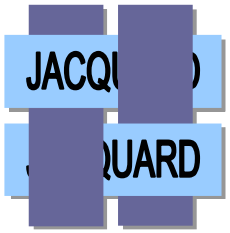


Example: Simple Client/Server Application



Configuration & Deployment characterization with two component models

	CCM	Fractal
Interface signature	OMG IDL 3.0	Fractal IDL
Components and ports type	OMG IDL 3.0	Fractal ADL
Instances configuration, deployment	XML CAD & CPF	Fractal ADL

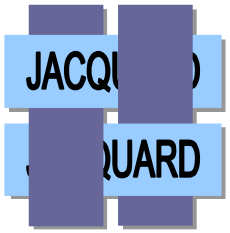


1. Introduction

- Architects are linked to technology
  - Dependency to technologies evolution
  - No reusability of component-based applications (CBA) across technologies

➔ Idea: allowing architects to capitalize their software architectures

  - Concrete configurations of CBA could be expressed via any ADL (*e.g.* in Fractal ADL) and mapped to any execution platform (*e.g.* CORBA Components platform)

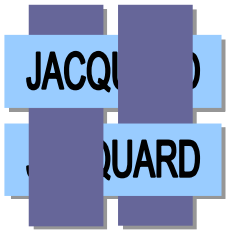


1. Introduction

• OMG Model Driven Architecture (MDA) approach could help to address this problem thanks to

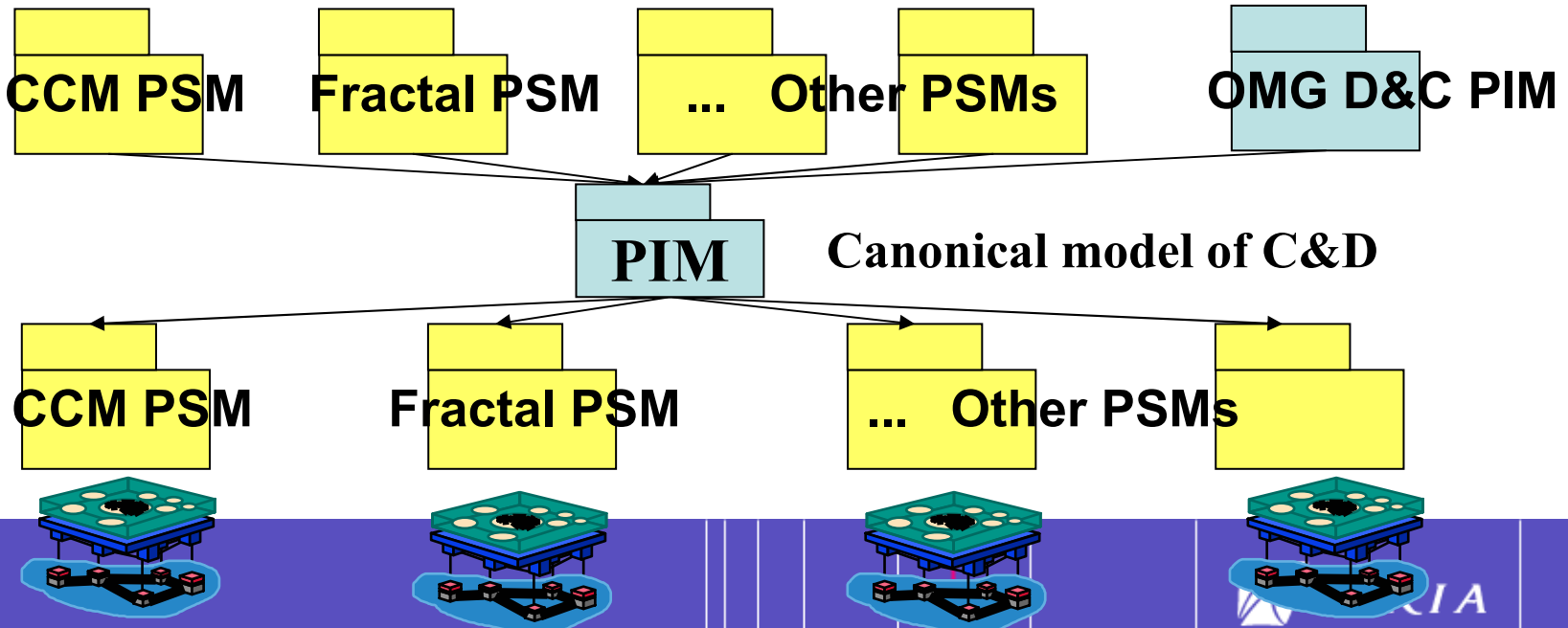
- Definition of abstract models independent of platforms (PIM)
  - OMG "*Deployment and Configuration of Distributed Component-Based Applications*" (D&C)
  - UML 2.0 component and deployment diagrams
- Mapping to platform specific models (PSM)

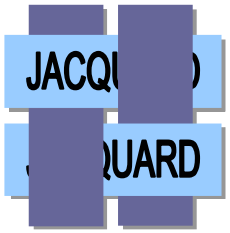
• But OMG D&C or UML 2.0 lack of generalized use and tools



2. A Multi Personality Environment

- Our approach: Apply MDA through a multi personality environment capturing a canonical model of configuration and deployment (C&D PIM)
- 2 dimensions of the C&D PIM / 3 layers architecture / several scenarii (PIM to PIM to PSM, PIM to PSM, PSM to PIM to PSM)



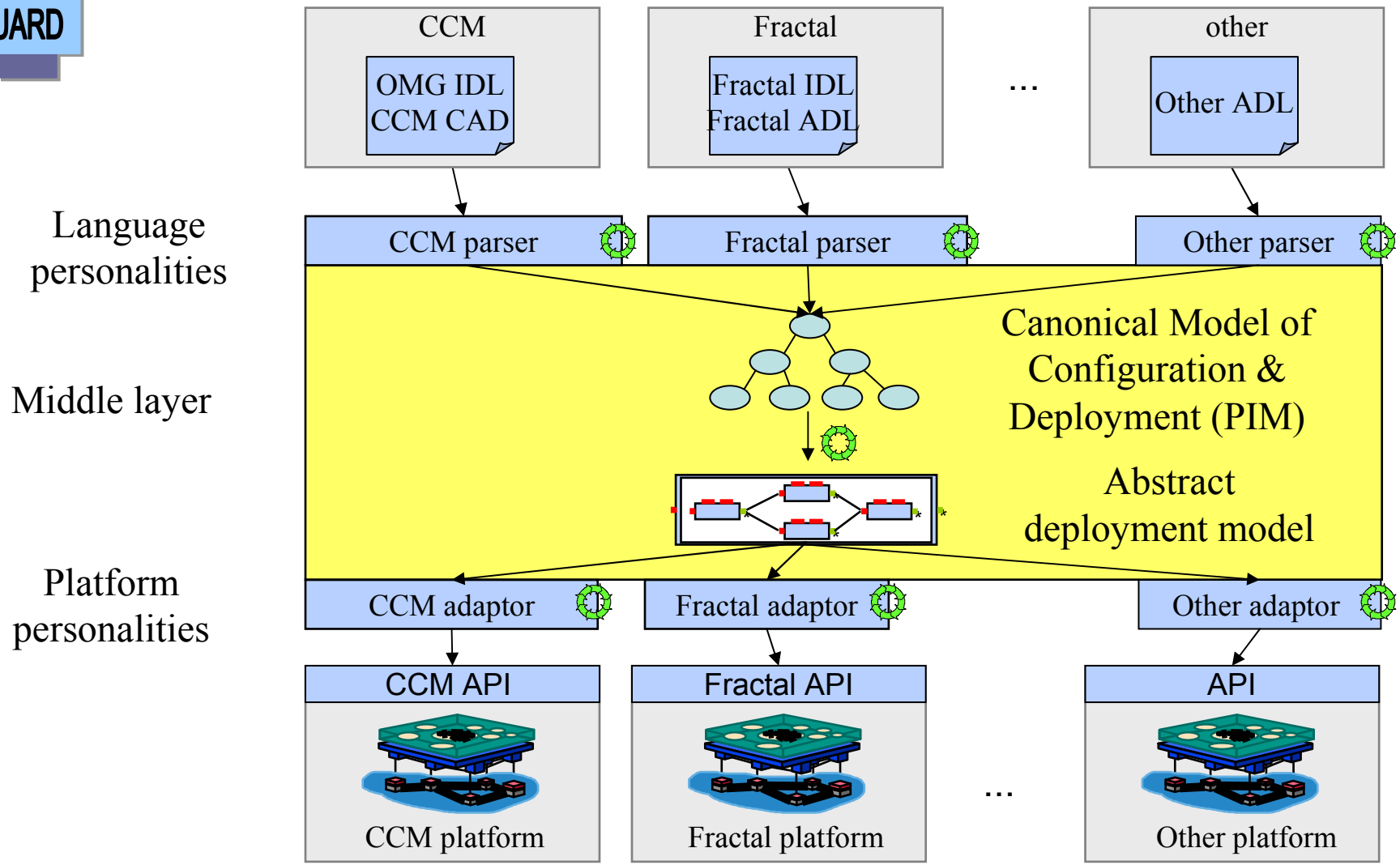


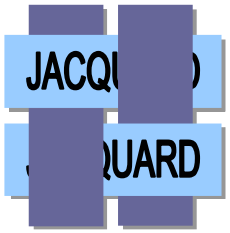
2. A Multi Personality Environment

➔ The 2 dimensions of the canonical model of C&D:

- **#1 Configuration** model means part of a PIM describing
  - Component types and ports
  - Interface signature
  - Where to install and instantiate component instances
  - Configuration properties of component instances (business/functionnal attributes)
  - Binding between component ports
  - Activation of components, etc.
- **#2 Deployment** model means part of a PIM describing how the configuration will be deployed, i.e. the deployment process

Architecture - Global View

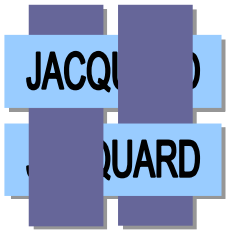




2. A Multi Personality Environment

➔ 3 layers architecture:

- A “language personalities” layer: mapping of concrete configurations to the canonical model (C&D PIM)
  - Language parsers
- A middle layer: generic C&D framework
  - Generic orchestration framework
- A “platform personalities” layer: adaptors for execution platforms
  - Specialized deployment artefacts (API, components, classes, etc.)



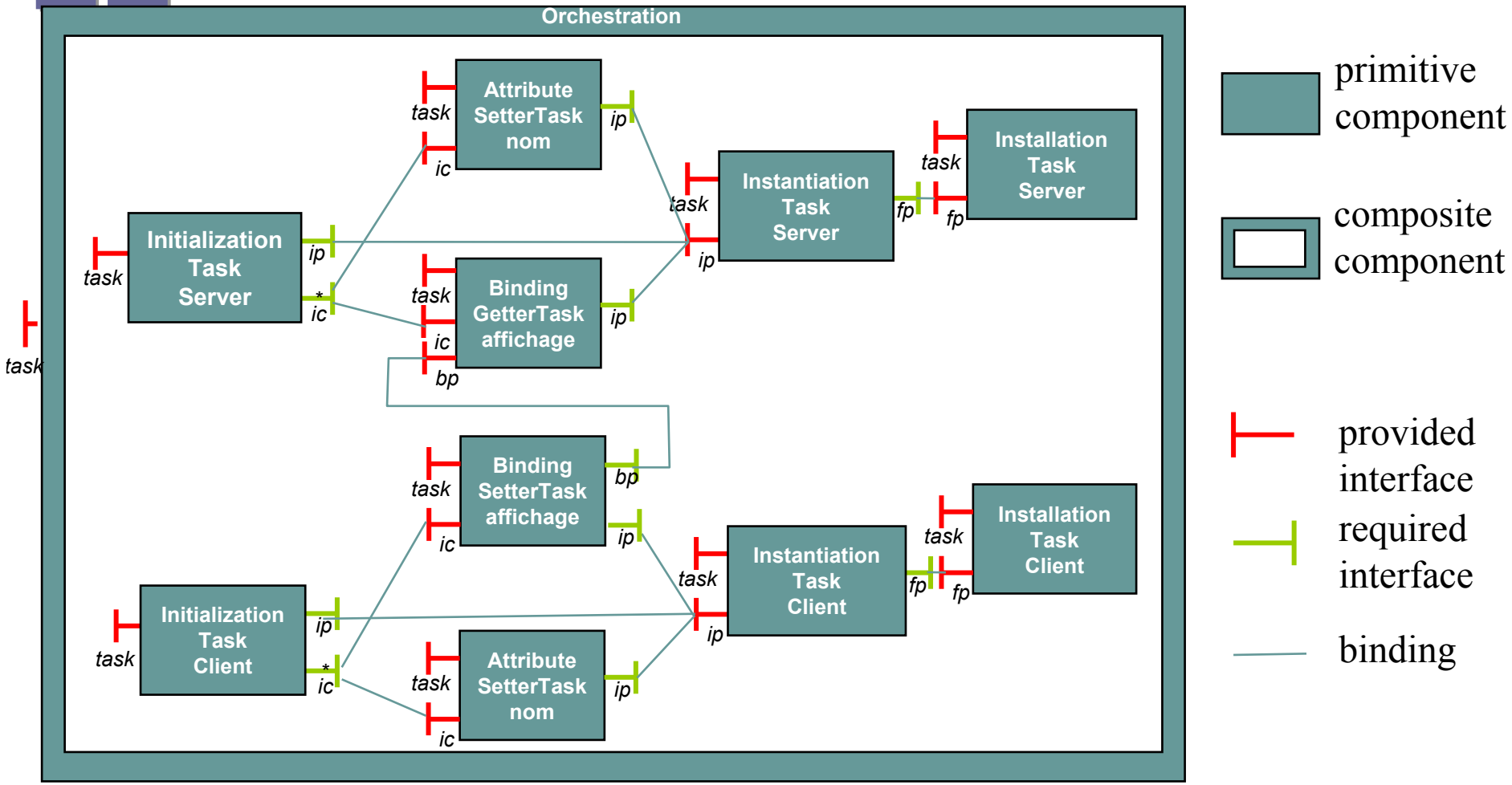
3. An Abstract Deployment Model

- Deployment (dimension #2 of our C&D PIM) is seen as a workflow process
- Each activity of the workflow process is a deployment task, *i.e.*:
  - Installation of component factories (eg binary code, executable, etc) to nodes
  - Loading into application server memory
  - Instantiation of components from factories
  - Component instance business attributes configuration
  - Binding between provided and required ports
  - Business components activation
  - etc.
- Deployment tasks are modelized with Fractal components:

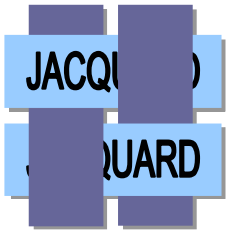
  ➔ *Fractal Deployment Components (FDC)*

- Each route of the workflow process is represented by a binding between FDC ports

Ex.: The simple C/S App. Deployment



fp : FactoryProvider *ip* : InstanceProvider *bp* : BindingProvider *ic* : InstanceConfiguration

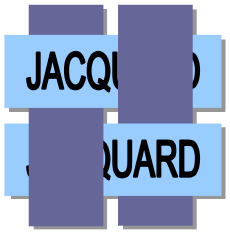


3. An Abstract Deployment Model

- 4 types of deployment interfaces required or offered by deployment components

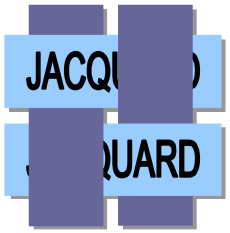
Inteface name	Provided method
FactoryProvider	*getFactory*
InstanceProvider	*getInstance*
BindingProvider	*getBinding*
InstanceConfiguration	

- 7 types of deployment components
  - *InstallationTask, InstantiationTask, AttributeSetterTask BindingGetterTask, BindingSetterTask, AddComponentTask, InitializationTask*



3. An Abstract Deployment Model

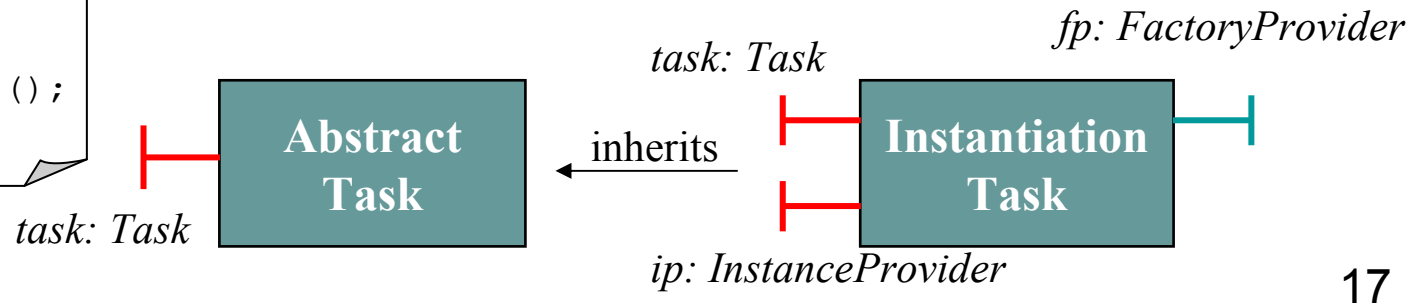
Component name	Attributes	Provided interfaces — **Task Component** — Required interfaces	
InstallationTask	*impl,target*	*FactoryProvider*	
InstantiationTask		*InstanceProvider*	*FactoryProvider*
AttributeSetterTask	*name,value*	*InstanceConfiguration*	*InstanceProvider*
BindingGetterTask	*name*	*BindingProvider, InstanceConfiguration*	*InstanceProvider*
BindingSetterTask	*name*	*InstanceConfiguration*	*InstanceProvider, BindingProvider*
AddComponentTask	*name*	*InstanceConfiguration*	*InstanceProvider*
InitializationTask			*InstanceProvider InstanceConfiguration*

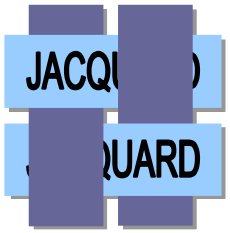


3. An Abstract Deployment Model

- Fractal Orchestration Framework: A generic deployment framework to automatically schedule tasks execution
  - http://forge.objectweb.org/projects/deployment
- An "*Orchestration*" composite contains all deployment task components
- Task components and *Orchestration* composite inherit from an abstract component that provides a "*Task*" interface offering the *execute* method

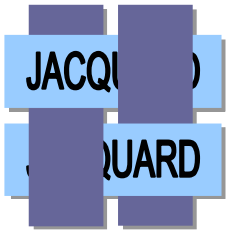
```
interface Task {
      void execute();
}
```





3. An Abstract Deployment Model

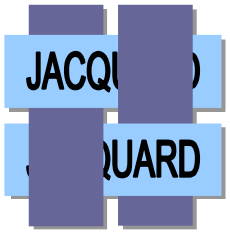
- The *execute* method of *Task* interface offered by all deployment components implements/realizes the deployment task
- When the *execute* method of *Orchestration* composite is invoked, component tasks are executed according to their dependencies
  - *e.g. InstantiationTask* cannot be executed before *InstallationTask* has been realized (*InstallationTask* offers a *FactoryProvider* interface used by *InstantiationTask*)
  - Use of bindings between task components and Fractal introspection operations to determine task dependencies



3. An Abstract Deployment Model

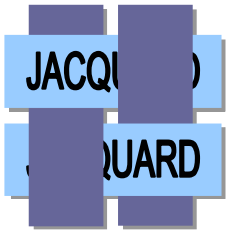
- The generic deployment components are specialized according to the targeted platform:
  - Component interface attributes
  - Returned data types of interface methods
  - *Task* interface *execute* method of deployment components

- This specialization represents a “platform personality” (*i.e.* adaptor for CCM, Fractal, etc.) of our C&D environment



Specializations for CCM and Fractal of the *execute* method of *Task* interface

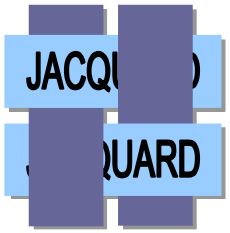
Task component	CCM	Fractal
InstallationTask	`Components::Container.install_home()`	`GenericFactory.newFcInstance()`
InstantiationTask	`Components::CCMHome.create_component()`	`Factory.newFcInstance()`
AttributeSetterTask	**Setting operations on** `Components::CCMObject`	**Setting operations on** `AttributeController`
BindingGetterTask	`Components::CCMObject.provide_facet()` or `get_consumer()`	`Component.getFcInterface()`
BindingSetterTask	`Components::CCMObject.connect()` or `subscribe()`	`BindingController.bindFc()`
AddComponentTask	`N/A`	`ContentController.addFcSubComponent()`
InitializationTask	`Components::CCMObject.configuration_complete()`	`LifeCycleController.startFc()`



Example of specializations: The *InitializationTask* component case

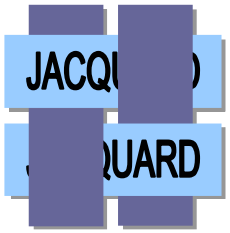
CCM	Fractal

public class **CCM_**InitializationTask
extends **AbstractInitializationTask** {
public void execute() throws Exception {
// Obtain the component to start.
CCMObject component = (**CCMObject**)
getInstanceProvider().getInstance();
// Start the component.
component.configuration_complete();
}
}
	public class **Fractal_**InitializationTask
	extends **AbstractInitializationTask** {
	public void execute() throws Exception {
	// Obtain the component to start.
	Component component = (**Component**)
	getInstanceProvider().getInstance();
	// Start the component.
	Fractal.getLifeCycleController(component).startFc();
	}
	}



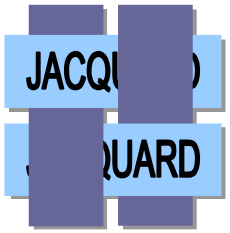
3. An Abstract Deployment Model

- Specialization for platforms easy to write
  - Already done for Fractal [FRA 02]
  - Began for CORBA Components Model in our OpenCCM DCI platform [MAR 01, OPE 02]
- The proposed abstract deployment model is valid for Fractal and CCM
  - Nevertheless, some tasks have to be more fragmented (*e.g. InstallationTask*) or added (*e.g.* to represent registration to services)
- Possibility to add new task components for supporting more component middleware platforms
  - Support of different deployment requirements



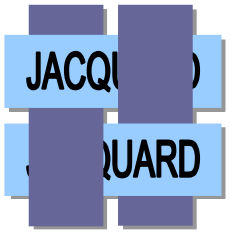
4. Conclusion

- We address the problem of the capitalization by architects of configuration & deployment of component-based applications
  - MDA-based approach through a canonical model of configuration & deployment (PIM)
  - An environment using language and platform personalities to respectively obtain this PIM and realize mapping to execution platforms
  - Contributions around the dimension #2 of PIM through an example of an abstract deployment model and personalities for two component models (Fractal and CCM)



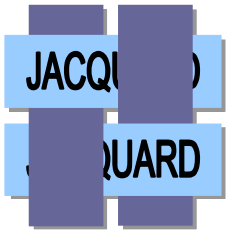
4. Conclusion

- Benefits of our approach:
  - Architects point of view:
    - Describe CBA configurations in any description languages and map them to different execution platforms
    - Migrations of existing applications through transformations from abstract configurations to concrete configurations
  - Engineering point of view:
    - A software framework (Generic Orchestration Framework) to build specific deployment machines using platform personalities
  - Scientifical point of view:
    - An abstract configuration and deployment model (which is independent of technologies) for:
      - Comparing languages expressivity and their limitations, identifying an “ideal” canonical model of software components
      - Re-configuration during execution will be possible through use of the dimension #1 part of the PIM (configuration)



5. Perspectives / Future Work

- Apply a model-driven approach to automatically generate component middleware deployment machines
  - Introducing a metamodel of workflow in order to define abstract deployment models (PIM) which are independent of underlaying technologies
  - Refinement of this PIM: specialization with functionnal/business aspects (*eg.* deploying CORBA components, Fractal components, etc.)
  - Transformation to a specific deployment model (PSM)
  - Mapping to execution platforms: code generation from PSM
    - *e.g.* Fractal deployment components code representing the platform personality of our Fractal Orchestration Framework



5. Perspectives / Future Work

- Link with OMG D&C and our PIM
  - Express transformation between D&C PIM and our abstract deployment model
- Tools to automatize MDA process to generate code from abstract configuration & deployment models:
  - Generic *Generation Framework* (GGF),
  - Platform personalities for this GGF, ie specific generators (OMG IDL, Fractal ADL, Java classes, etc.)
- Enhance the Generic Orchestration Framework , *e.g.*
  - Address failures during deployment process
  - Provide various orchestration policies, *e.g.*
    - Serialization of task execution Vs parallel executions