# Model Transformations and Code Generation


## Ecole IN2P3 Temps Réel

Ansgar.Radermacher@cea.fr

# École d'été, 26.11

- **08h30 – 10h00: Cours S1 – Component models CCM and FCM (connectors)**
  - CCM – CORBA component model
  - FCM – un modèle par composant flexible avec ports générique, connecteurs et containers.
  - Connecteurs : schéma d'interaction et leur implémentation

- **10h30 – 12h00: Cours S2 – Déploiement**
  - Déploiement – instanciation des composants: choix des implémentations, affection valeurs aux attributs, allocation sur nœuds
  - Utilisation du FCM pour supporter l'exécution modèles MARTE
    ⇨ Mapping du MARTE GCM vers FCM et la chaine d'outil eC3M
  - Rôle des bibliothèques modèles

# Outline

- **FCM: Flex-eWare (Flexible) component model**
  - Meta-model, main principles
  - Derived UML profile
- **FCM profile usage (demo)**
  - Ports
  - Connectors defined in model libraries
- **Link with MARTE**
  - Automatic MARTE/FCM synchronization (work in progress)

# A flexible component model

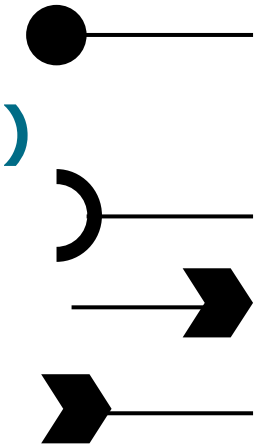
- **Different existing standards:**
  - UML,
  - MARTE GCM
- **With execution support**
  - CORBA Component Model (CCM v4, OMG formal/2006-04-01)
  - Fractal (multiple implementations)
  - Several academic approaches
    - SOFA2, RUNES, TinyOS
  - Outside embedded
    - Service oriented architecture (SOA), OSGi (used by Eclipse, Spring)
    - Web-services

# CCM Excursus – CORBA Component model

- OMG Standard, lightweight profile exists
- Based on component / container pattern (separation of concerns)
- Explicit declaration of used services (through ports)

- But … "mostly dead" (big specification, not many vendors, afraid of CORBA)
- But … only pre-defined container with fixed services
- But …only supports small set of *interaction patterns*
  … with specific and *fixed* implementations
  - Synchronous method calls (via CORBA)
  - Event based communications (only push/push style)
  - Streaming (recently added)

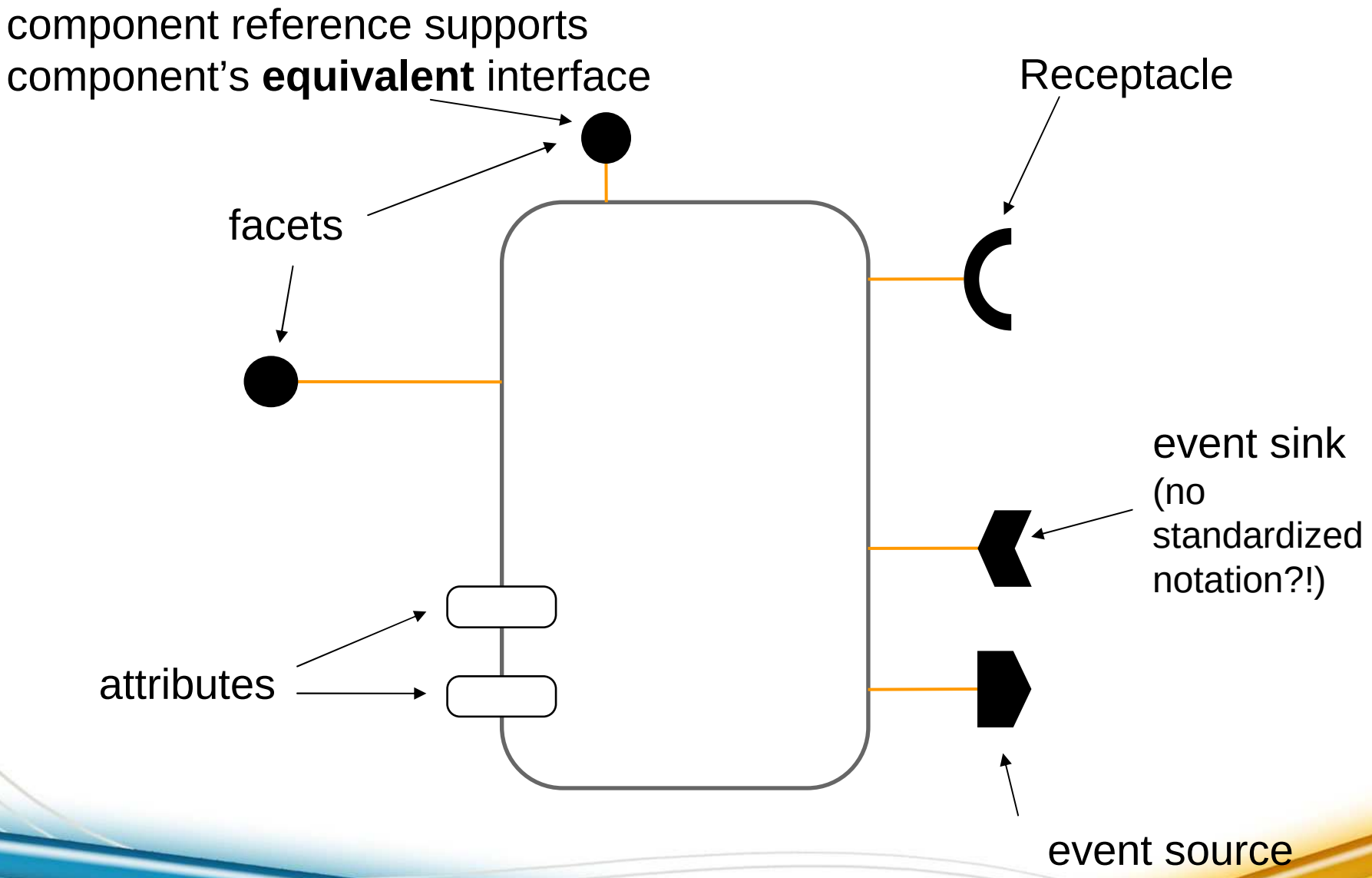
# CCM Excursus – CCM Ports



- Facets are provided interfaces for clients (interfaces are defined in IDL in Java like syntax)
- Receptacles denote connection points
- Event Sources
- Event Sinks
- Streaming (not treated in this presentation)
- Attributes for configuration purposes

⇨No complex ports as in UML2 = not possible to group related ports e.g. receptacle and facet for an associated callback in a single port

# CCM Excursus – Ports (cont'd)



# CCM Excursus – Facets/Equivalent interface

- **Facets = entry points for invocations ("server interfaces")**
- **Facets have independent object references**

## Equivalent interface

- **Component has single distinguished equivalent interface.**
- **Used by clients for navigation:**
  - Obtain facet reference from equivalent interface (`provide_facet and provide...` methods).
  - … and vice versa (`get_component()`)
- **… and connection of receptacles**
  - (`connect and connect_...` methods).

# CCM Excursus – Receptacles

- ***Receptacles*** **denote component's need to use services provided by other components ("client interface").**
- **When a component accepts object reference, this is called a *connection*.**
- **Store a simple reference or multiple references**
- **Configuration**
  - Typically, connections are set-up during assembly
  - Dynamically managed at runtime to offer interactions with clients or other components (e.g. callback)

# CCM Excursus – CCM Event + Home

- **Connects a Producer and a Consumer**
- **Based on publish/subscribe pattern**
  - Events are mapped on "Consumer" interfaces (associate push operation with event)
  - Event publisher provide subscription operations
  - Event sinks provide reference for data delivery (consumer)
- **Event delivery always via a push/push model**

- **Homes: manage component lifecycle, in particular creation**
  - provides factory & find method
  - In addition: arbitrary user-defined methods

# CCM Excursus – Component / Container Model

CIF = Component Implementation Framework

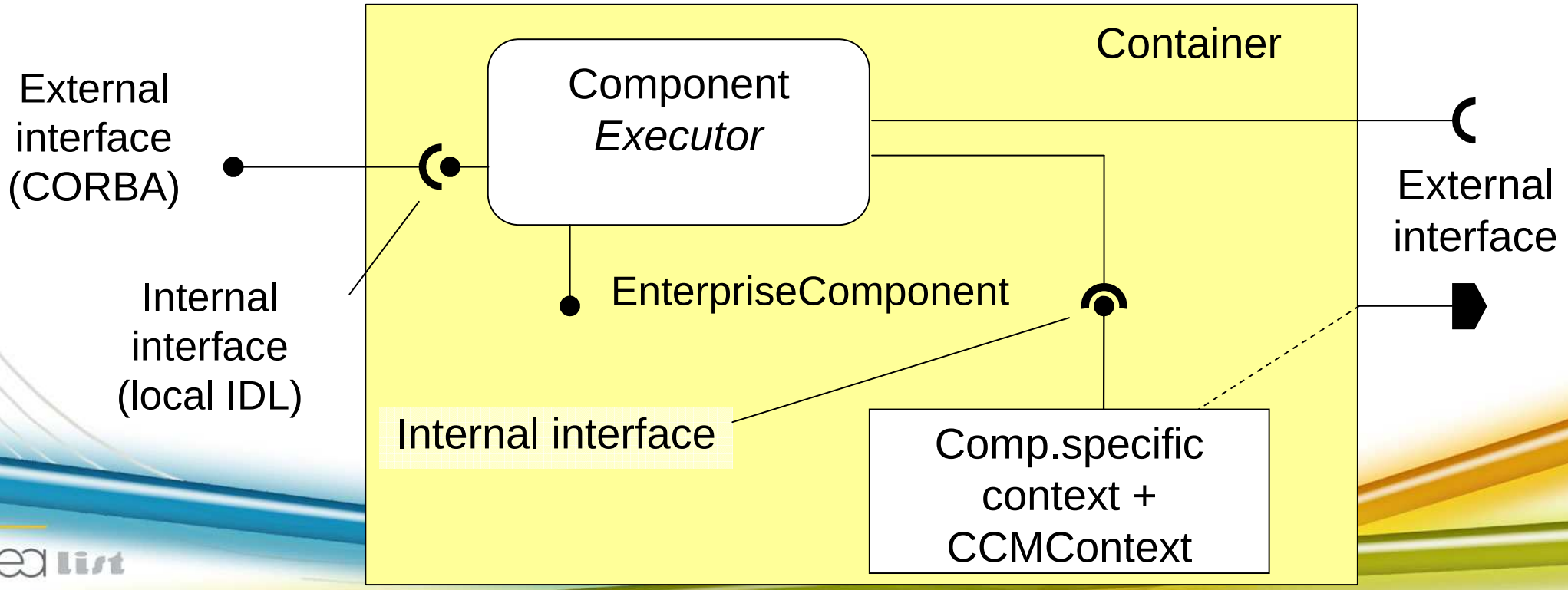
standardizes which *interfaces an executor has to implement*

and *which interfaces the executor can use*

For each provided interface provide get_<port-name>

For each require interface, use getcnx_<port-name>

**Re-used later!**



# Flex-eWare (Flexible) component model

- **FCM: Flex-eWare (Flexible) component model**
  - Meta-model inspired by UML, Fractal and CCM
  - Connector extension

- **Basic principles**
  - UML like: components with
    - Ports
    - Hierarchical components: inner parts
    - Connectors between parts

# FCM – Extensions, differences with UML

- **Ports are different compared to UML (next slide)**
- **Connectors have types and implementations**
- **Flexible containers (similar to QoS4CCM)**
- **Deployment in a D&C like manner**
  - Platform description (more elaboration required)
  - Deployment of instances on a node

# FCM Ports

- **Ports are characterized by a type *and* a kind**
- **Use of an interface does *not* require an auxiliary class definition (as in UML)**
- **Port kind has informal semantics**
- **Kind-specific mapping rules towards provided and required interfaces**
- **Examples:**
  - Port kind “UseInterface”, type “MyInterface”
  - Port kind “FeatureBasedCS”, type “ClientServerSpecification”po
  - ⇨ derived provided and required interfaces
- **Important: port kinds are defined in a model library and cn thus be extended**

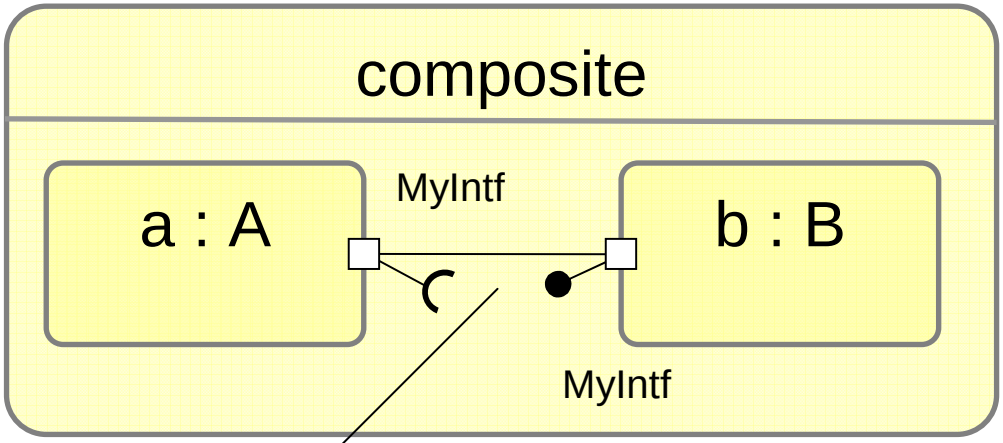
# FCM Connectors

- **Connector support allows to specify**
  - Interaction pattern during component development time
  - Interaction implementation uring deployment time

- **Basic principles: Connectors are …**
  - … like components: can be configured, have implementations (Assembly implementations in case of distribution)
  - … almost: ports don't have fixed interface types, connectors need to be instantiated (generated) from a template like definition

# Connector Reification

- **Model transformation:**
  **eplace UML connector with a part**

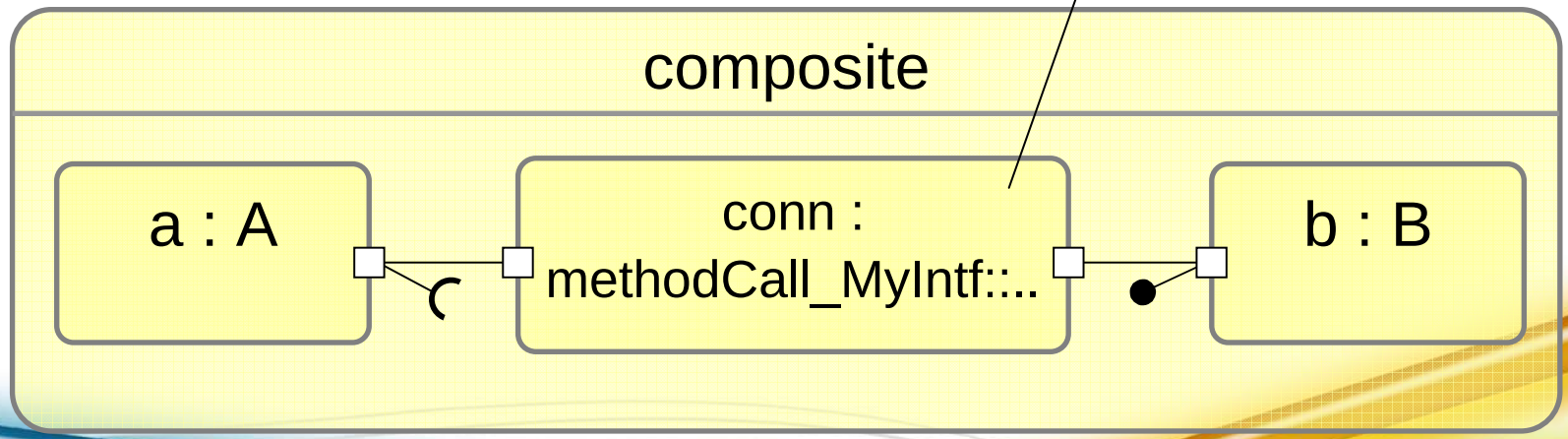



Connector type & implementation need to be adapted to MyIntf

⇨ Calculate binding based on port type

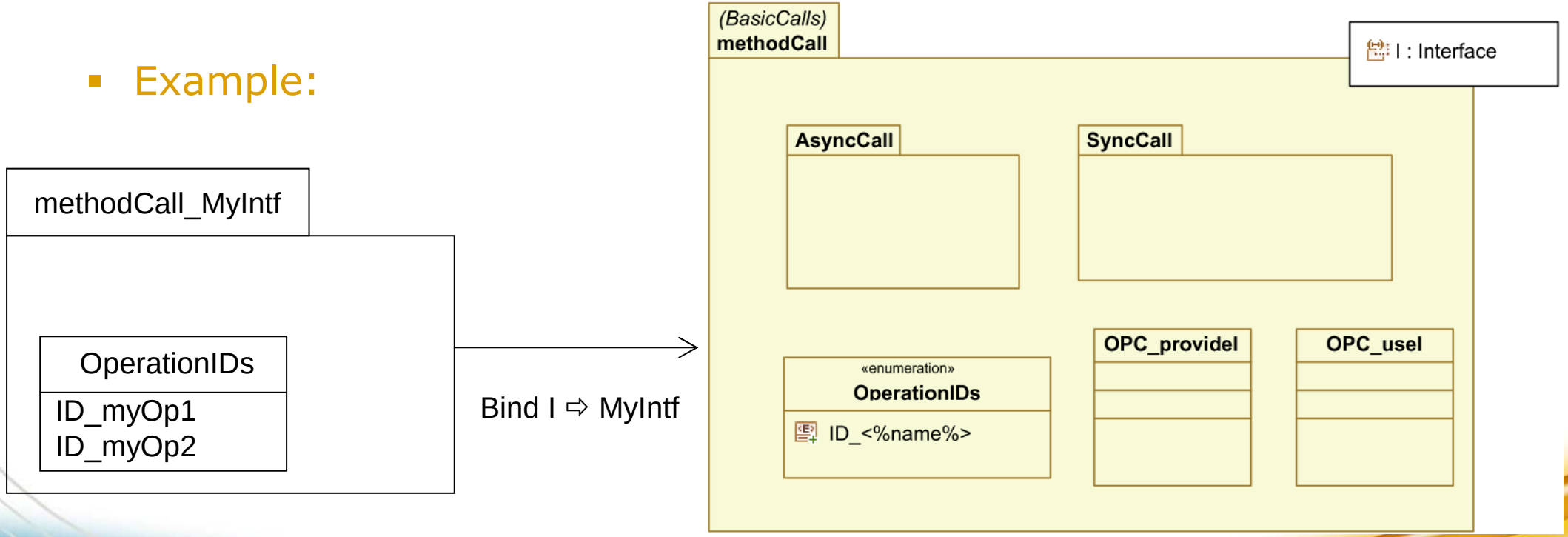
⇨ Instantiate package template (next slide)

Declarative information about connector type



# Connector Adaptation

- **Use UML package templates**
  - Own a signature with a template parameter (in most cases an interface, here by convention called I)
  - Template parameters are bound (template binding) when the template is instantiated with a concrete type (interface)
  - Example:



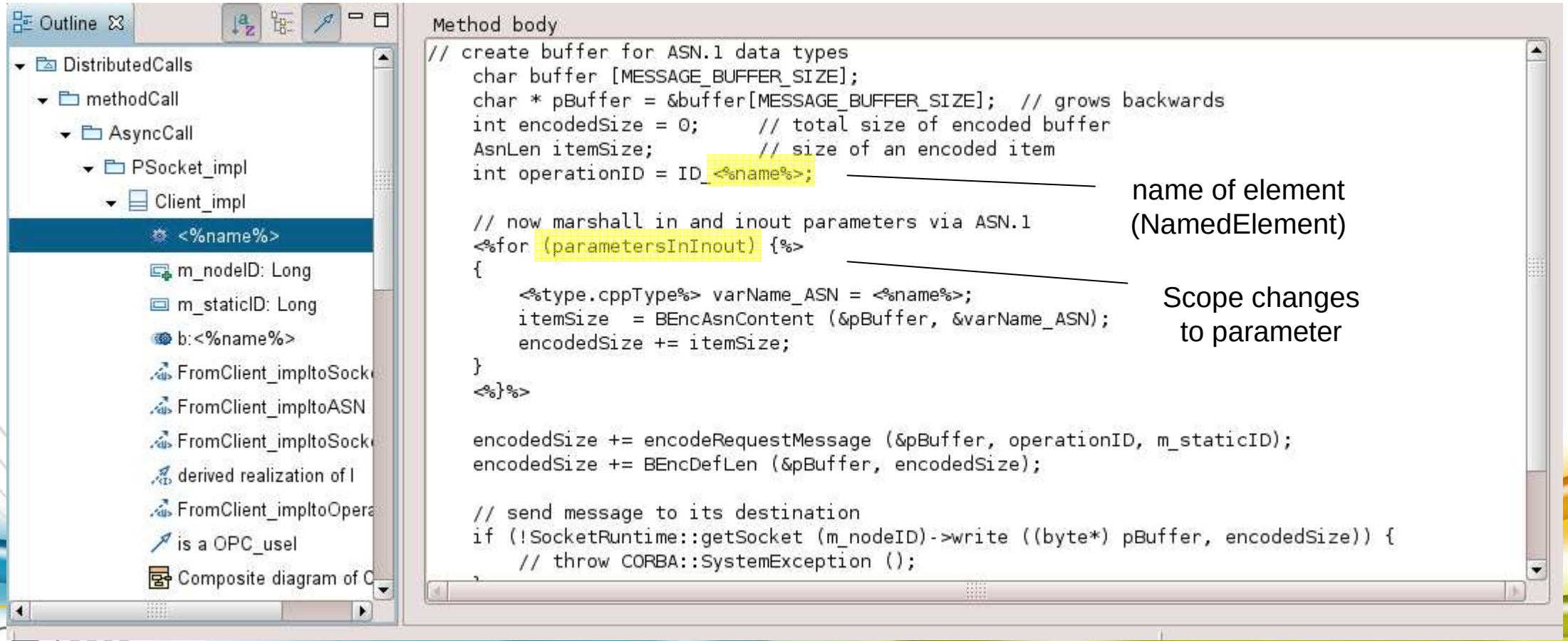
# Connector Adaptation (contd.)

- **Adapt model:**
  - Replace occurrence of formal template parameter by actual (MyIntf)
  - Replace occurrence of formal template parameter in operation names (String template based on Acceleo)
  - Adapt signature of operation to actual

- **Adapt implementation**
  - Implementation is given in form of an Acceleo template, has access to actual or an operation of actual.
  - ⇨ Implementation can perform “non trivial” operations such as parameter marshalling in the context of a generic model transformation (template controlled)

# Connector Adaptation Example – Socket client stub

- For each operation in MyIntf, create operation with same signature and implementation as given in the method body
- Access to all UML attributes of an operation (as in the UML MM) + some predefined helper functions, such as parametersInInout



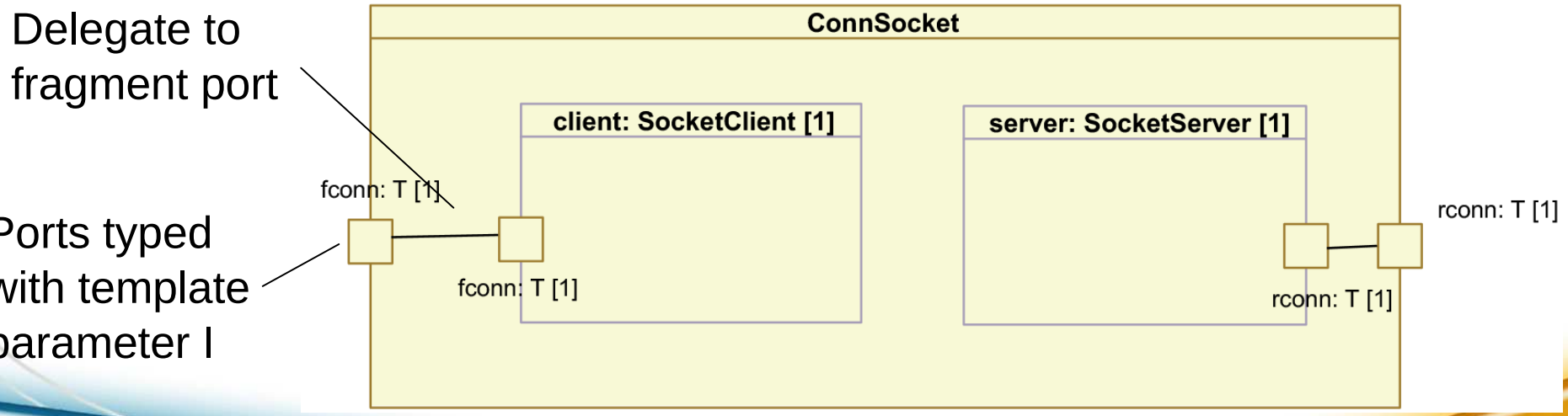
# Connector Examples

- **Basic Connectors – (domain specific) model libraries**
  - Synchronous calls via CORBA, OSEK-COM
  - Asynchronous calls via Sockets, CORBA
  - FIFO (local)
  - ACCORD (MARTE calls with real-time feature)

- **Connectors based on** composition **of basic ones**
  - FIFO – distributed via sockets, CORBA
  - Connectors supporting Fault Tolerance

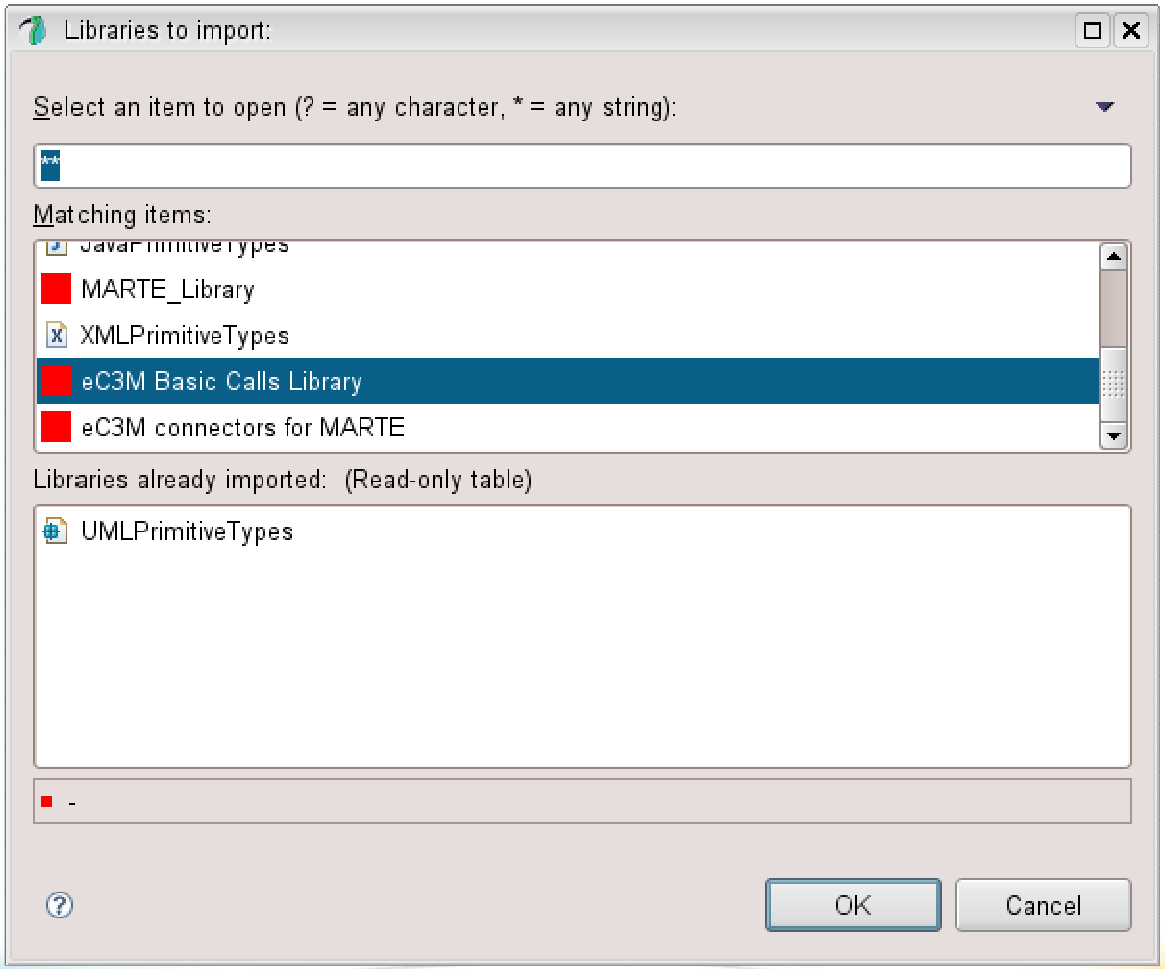
# Connectors enabling distribution

- **Connector must be local to using component**
  **⇨ connector itself needs to be distributed**
- **Implementations of distributed connectors have a composite structure (D&C assembly implementations)**
  - Internal structure captured by UML composite structure diagram
- **Example: socket connector consisting of two *fragments***



# Connector support

- **Set of predefined connector libraries, available via package import (from repository)**



# École d'été, 26.11

- **08h30 – 10h00: Cours S1 – Component models CCM and FCM (connectors)**
  - CCM – CORBA component model
  - FCM – un modèle par composant flexible avec ports générique, connecteurs et containers.
  - Connecteurs : schéma d'interaction et leur implémentation

- **10h30 – 12h00: Cours S2 – Déploiement**
  - Déploiement – instanciation des composants: choix des implémentations, affection valeurs aux attributs, allocation sur nœuds
  - Utilisation du FCM pour supporter l'exécution modèles MARTE
    ⇨ Mapping du MARTE GCM vers FCM et la chaine d'outil eC3M
  - Rôle des bibliothèques modèles

# Container

- Embedded component executors (as in CCM)
- Standard container: not an entity of its own, does not add any overhead
- Container supporting interception: manipulate port references (see next slide)
- Containers supporting extensions

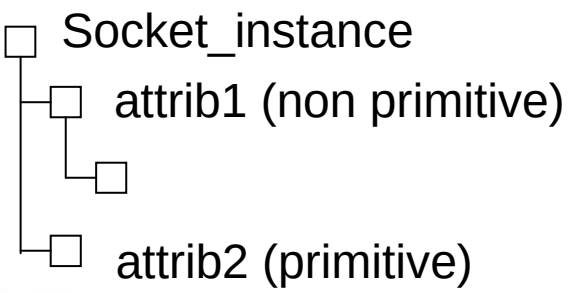
# Deployment

- **Instantiate System (a component implementation)**
  - An instance specification for the system
    - Assign values (slots) to all properties
  - Parts are typed with other components
    - Case 1: concrete implementation
    - Case 2: type/abstract implementation

    ⇨ need to find suitable implementation first, based on
      - Platform properties (supported OS, …)
      - Non functional properties (not really supported yet)

    ⇨ In particular interest for connectors

    ⇨ Slot value = instance-value, recursive instance specification (tree)



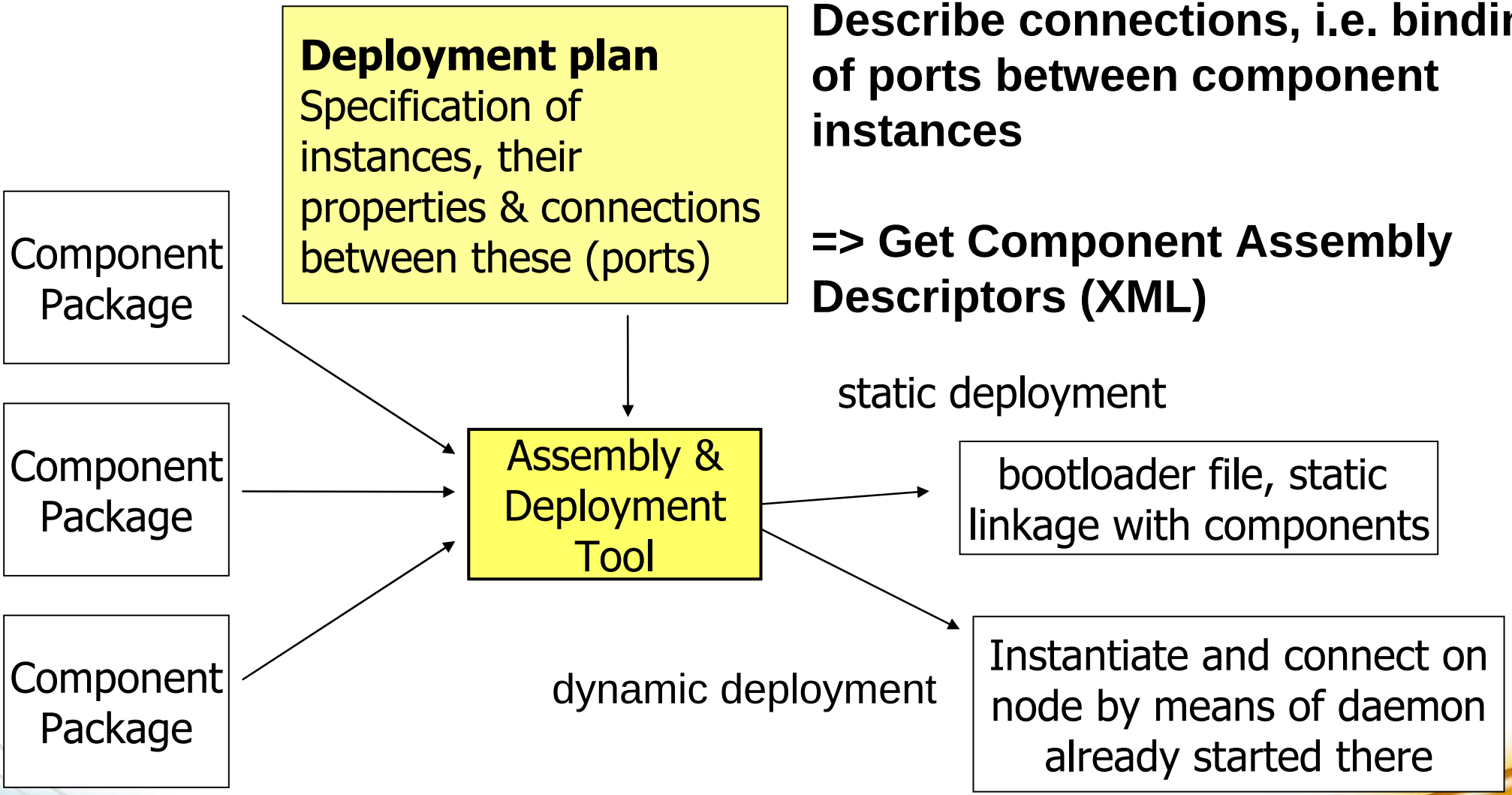
# Deployment

- **Initial creation of deployment plan (screendump)**
- **Right-click on system implementation**
  - => chose create deployment plan
  - Resulting deployment plan has fixed name, will be put into DeploymentPlans package

# Packaging and Deployment

- Define “what” (which implementation) needs to be deployed
  [called component package in  CCM]
- Define configuration, i.e. fix attribute values
- Define allocation (“where” to deploy instances)

# CCM Excursus – Deployment plan



# Deployment plan

- **Create a deployment plan (CCM, i.e. OMG D&C terminology)**

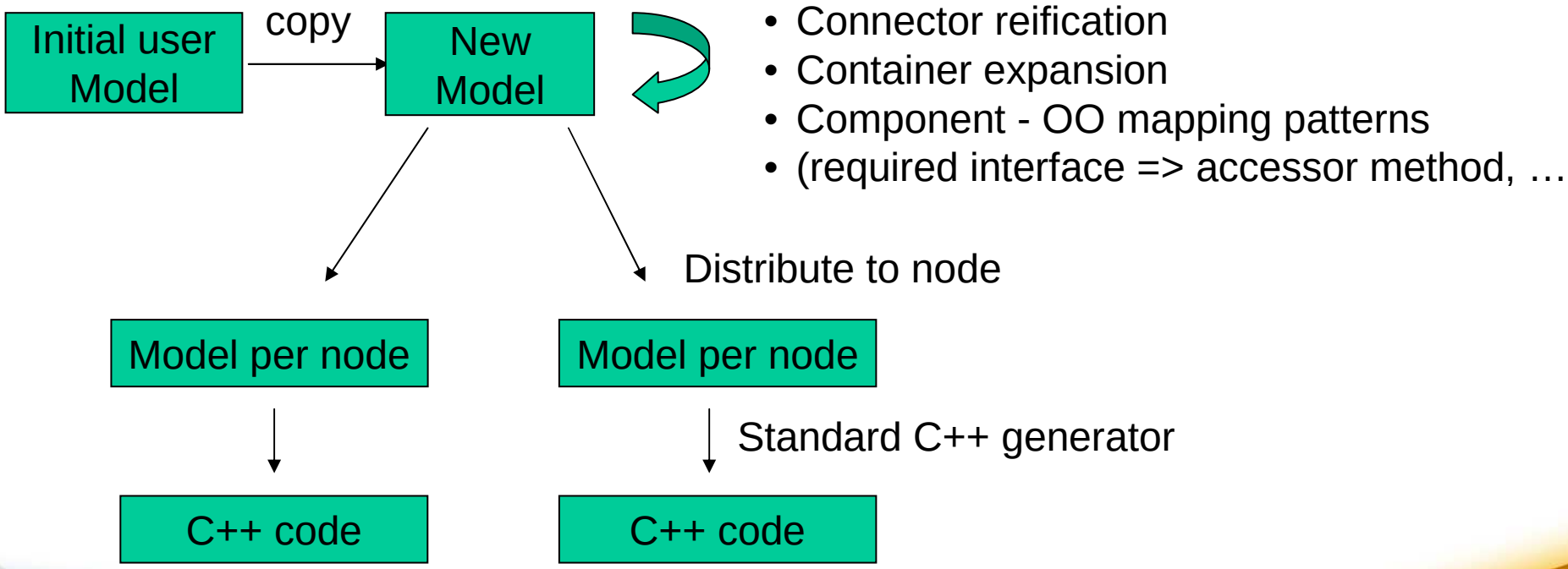
- **Plan = set of instances (UML instance specifications)**
  - Each instance references an implementation (UML class)
  - Each instance has a set of slots for configuring attributes
  - Each instance may be deployed on a node

# Allocation

- **Instances may be allocated onto nodes**
- **Create a new deployment diagram**
  - Drag&Drop nodes from platform description and instances from the deployment plan (instance specifications) into diagram.
  - Establish an Allocation relationship between these (create an abstraction between instance and node and stereotype it with the MARTE stereotype allocate, Use the profile section of the property dialog).
  - Allocation of composites:
    - based on the following rule: if a contained part is allocated on a node, the composite is implicitly allocated on that node as well.

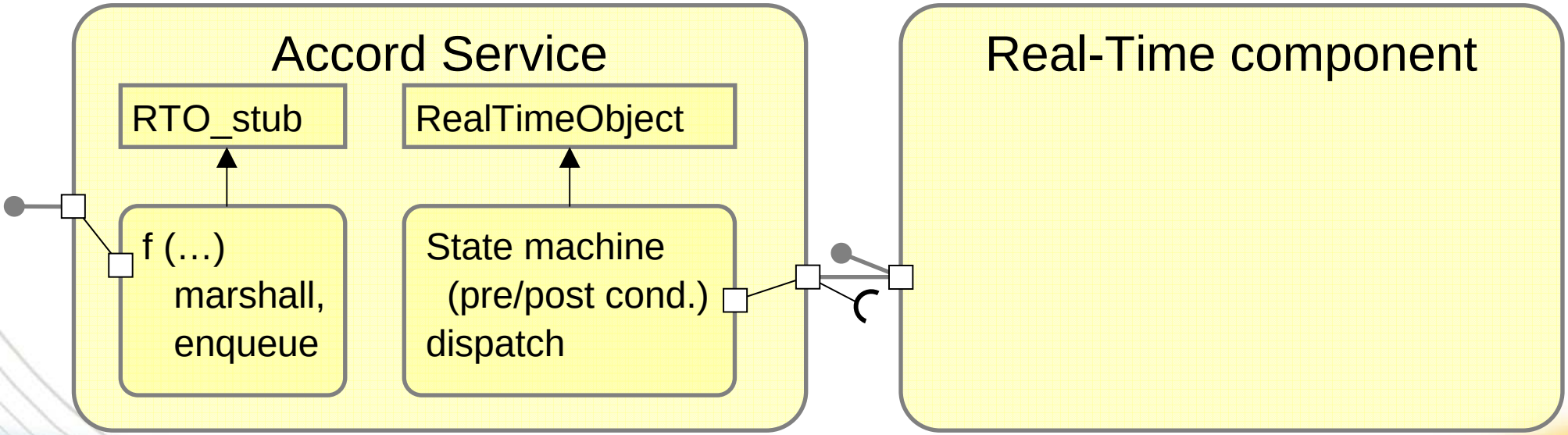
# Deployment plan instantiation

- **Instantiation of the deployment plan corresponds to a sequence of transformations**
- **Two stage transformation (Chokri's presentation)**



# Accord Integration

- **Specific connector, fragments implement (unmodified) code**
- **Container responsible for tasks that are not port specific, such as the handling of the requests in progress**



# Integration into an MDA approach

- **Component instances and interconnections specified with Papyrus UML (www.papyrusuml.org)**
  - Composite structure diagram
  - Deployment diagram
- **Profiles**
  - FCM profile
    - Deployment and configuration
    - Connectors
  - UML MARTE , for specific contexts: QoS + FT profile

# Tool Chain

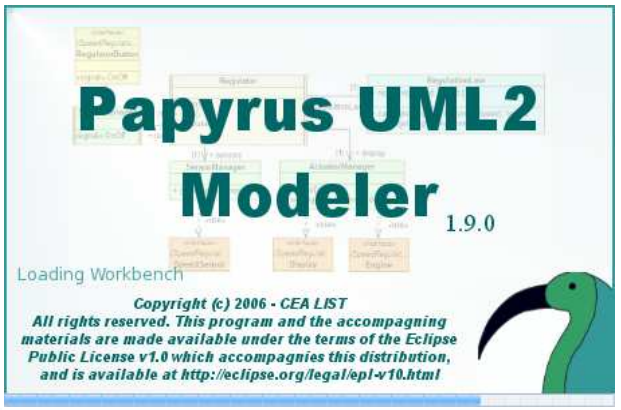
**Modeling environment**

Component descriptions

Model Library

Profile FCM

Profile MARTE





Packaging tool

IDL/IDL3 descriptions

Deployment XML (CDP, CCD)


bootcode / connector code

Binary (per node)

Transformations
model ⇨ text : Acceleo
model ⇨ model : currently Java
ATL, QVT

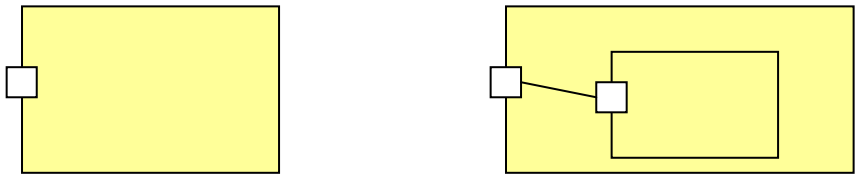
# GCM Mapping

- **MARTE GCM ports**
  - Map GCM ports to specific port kinds
  - Example:
    - ClientServer Specification => port kind of same name within model library. Port can be typed by client server specification

- **MARTE GCM PpUnits, RtUnit**
  - Map to container extension of same name,
  - currently supported RtUnit

# OO patterns + container

- **How to map ports on OO concepts? (similar to CCM CIF)**
  - For each port providing an interface, get_<portName> will return either the component reference or a reference of an inner part (delegation)



  - Depending on container type, might return a reference t o a wrapper.
  - Unlike in CCM, implementation of this operation is done by system
  - For each port requiring an interface, getcnx_<portName> will return a reference to the connected service