


HRT-UML/RCM: **an overview of the inroad to correctness by construction**

Tullio Vardanega

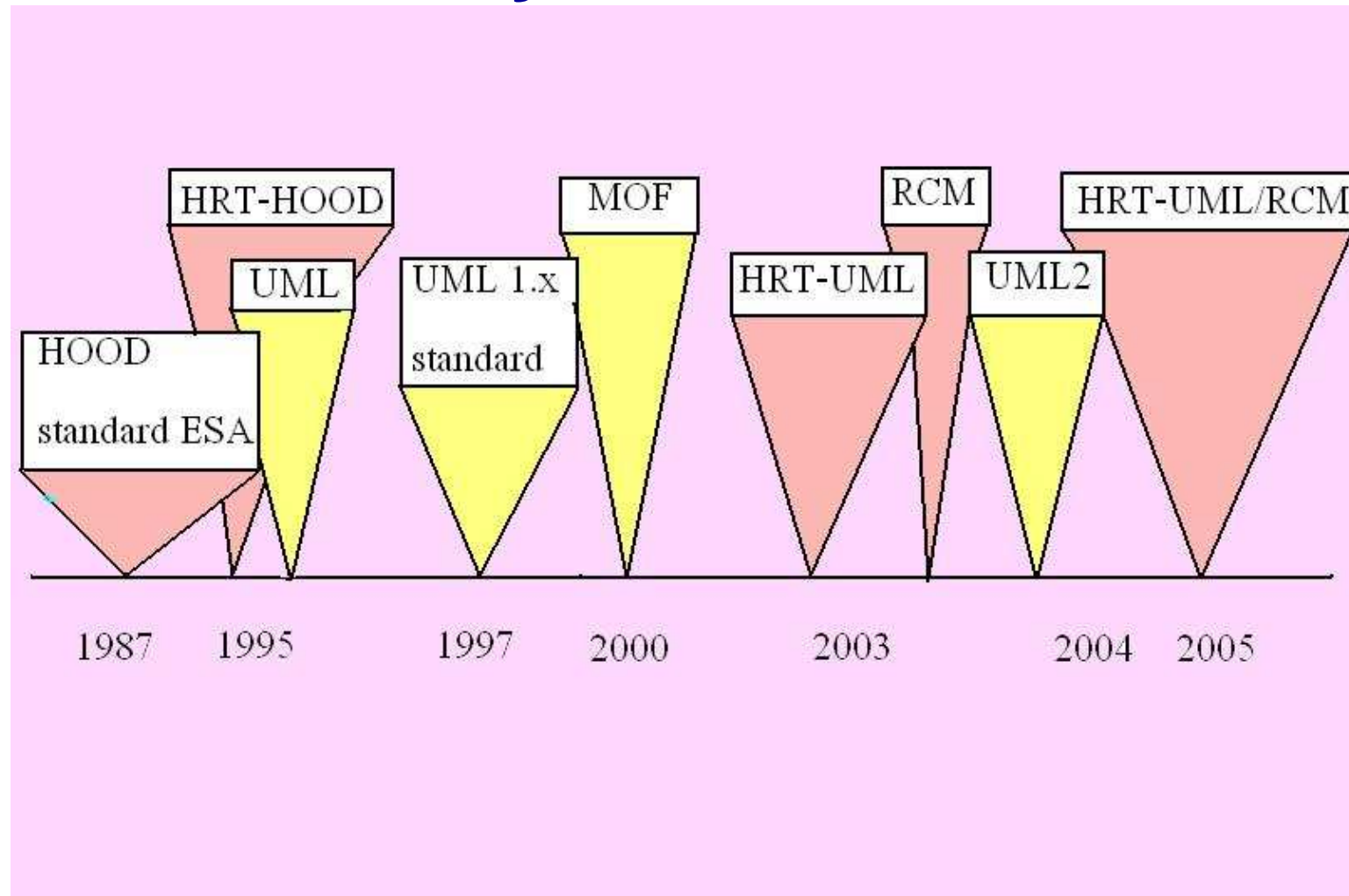
(with Daniela Cancila and Matteo Bordin)

University of Padua

(Progress through) Contents

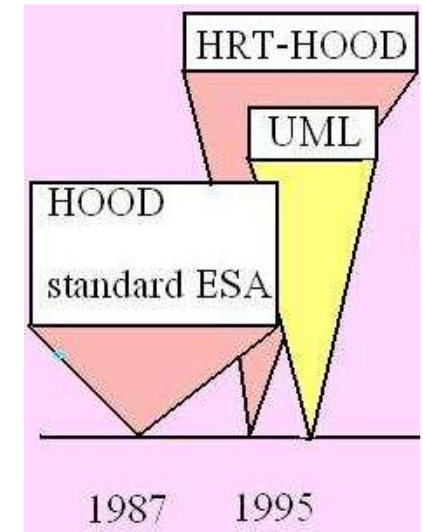
1. **A bit of history** 
2. The HRT-UML/RCM meta-model in UML2
3. The HRT-UML/RCM Profile
4. *Model-based automatic meta-model generation*
5. Model-based automatic code generation

A bit of history



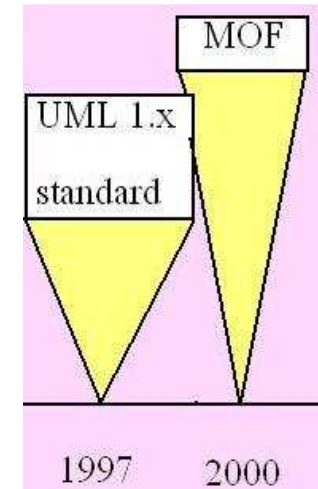
HRT-HOOD (1995)

- Based on singleton objects as in HOOD 3.1
 - Object model only
 - Single-instance objects only
 - The active object possesses an own thread of control
- Breaks the active object down to “cyclic” and “sporadic”
- All interactions have to be asynchronous and take place via protected objects
 - As allowed by Ada 95
- Mutual exclusion and bounded priority inversion warranted by Priority Ceiling Protocol
- Protected, sporadic and cyclic objects decorated with HRT attributes
 - WCET, priority, period, deadline, priority, criticality, ... (as appropriate)

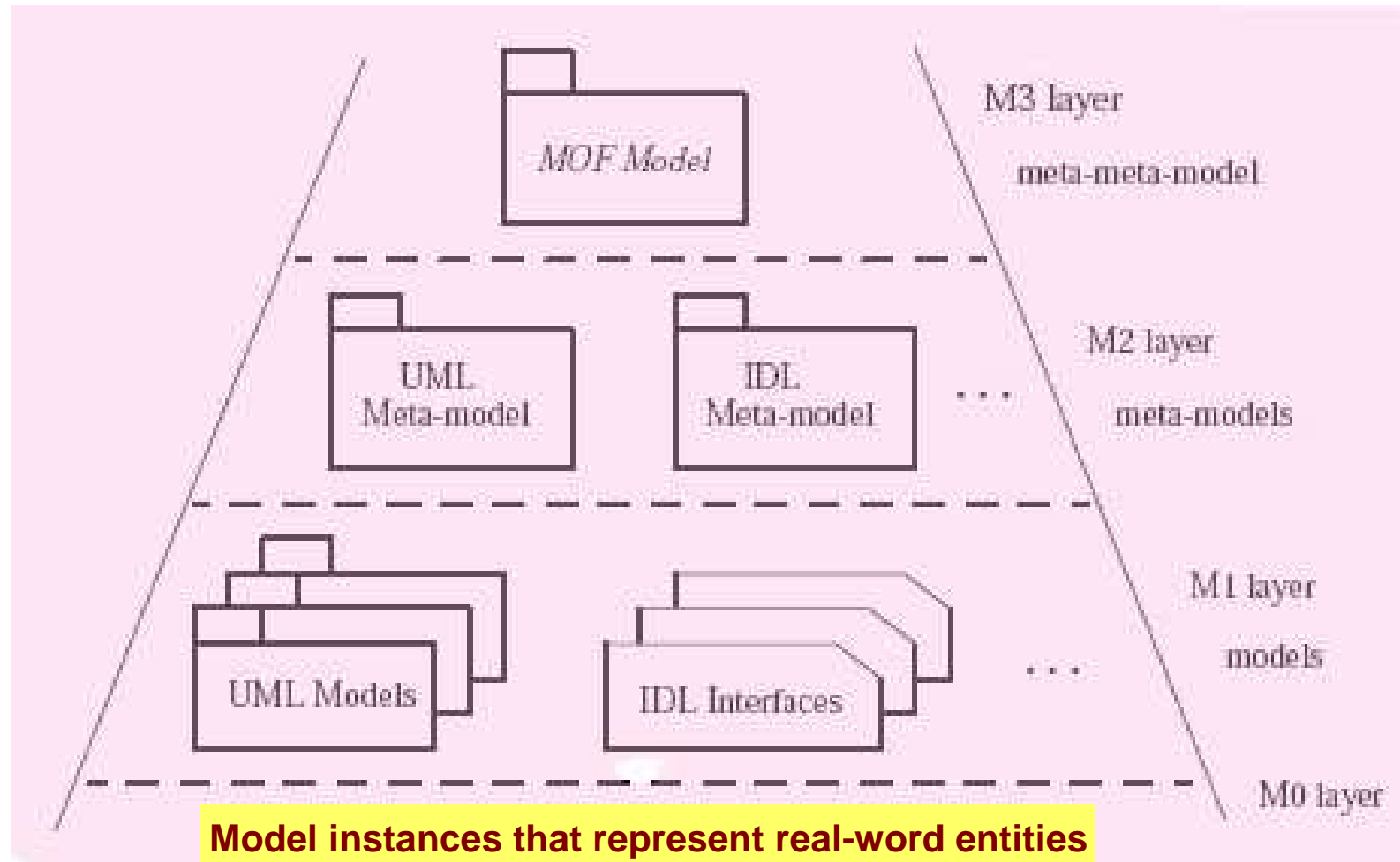


UML 1 and MOF (1994-2000)

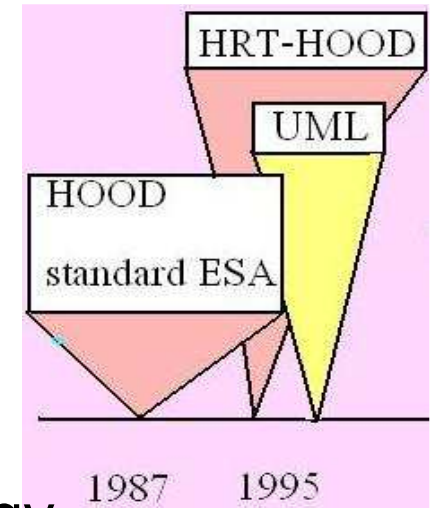
- **1994 – 1995: First appearance of UML**
 - General-purpose, “true” OO modeling language
 - Far greater expressive power than with HOOD*
 - Explicit manipulation of classes and interfaces
- **1997: OMG accepts UML as a standard**
 - Major success
 - De-facto standard
 - Yet with shallow semantic discipline
 - Need for specialized profiles for (H)RT application domains
 - Insufficient support for embedded systems in control, automotive, aerospace domains
- **2000: Introduction of the Meta-Object Facility**
 - <MOF, UML 1.x> permit the definition of multiple meta-models that all (may) share the same semantic roots
 - Interoperability across the entire development cycle



MOF hierarchical modeling (2000)



HRT-HOOD and UML 1.x



- **HRT-HOOD**

- Detached from the standard UML technology
 - Sadly not a viable industrial option anymore
 - Yet profound semantic roots and engineering discipline
- No underlying meta-model
- No (obvious) support for multiple instances
- Objects do not exist if their required interface is not fully satisfied (!?)

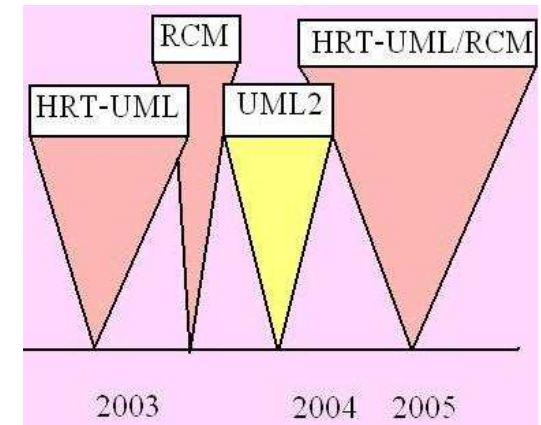
- **UML 1.x**

- Very poor support for HRT systems
 - Non-functional requirements modelled as “notes” (!?)

HRT-UML (2003)

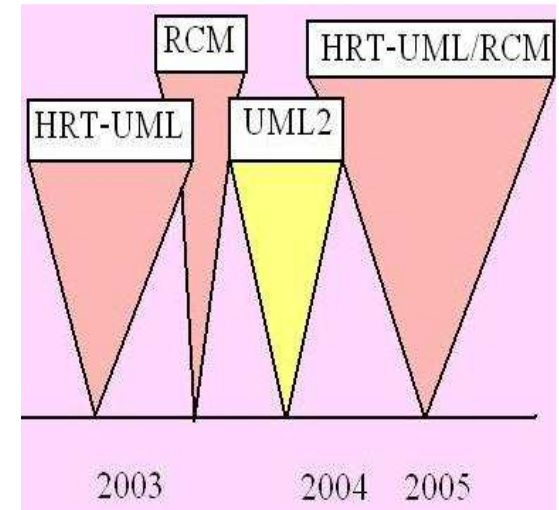
(HRT-HOOD embedded in UML 1.4)

- **Placeholder object**
 - Objects “exist” even if their required interface is not satisfied (yet)
- **Provided and required interface**
 - The beginning of a true meta-model ☺
- **Object model in the foreground and class model in the background**
 - Cleaner support for multiple instances
 - Initial (though loose) support for manipulation of the class model
- **Hierarchical design with delegation & decomposition**
- **Tool support for early static analysis**
 - Including gateway to PROMELA/Spin



Ravenscar Computational Model (2003)

- **RCM** is a direct emanation from the Ravenscar Profile
 - Statically analyzable subset of the Ada 95 tasking model
 - Now ratified as a supported formal language profile in Ada 2005
- **Key assets**
 - Destructuralization of terminal entities
 - For direct reliance on (UML) meta-model
 - Classes as compositions of meta-model elements






UML 1.x vs. UML 2 (2003–4)

- A UML2 meta-model is a model of the UML language that is itself expressed in a subset of UML
- The meta-model defines syntax and semantics of all UML modeling languages
- The meta-model improves precision and consistency of the UML specifications

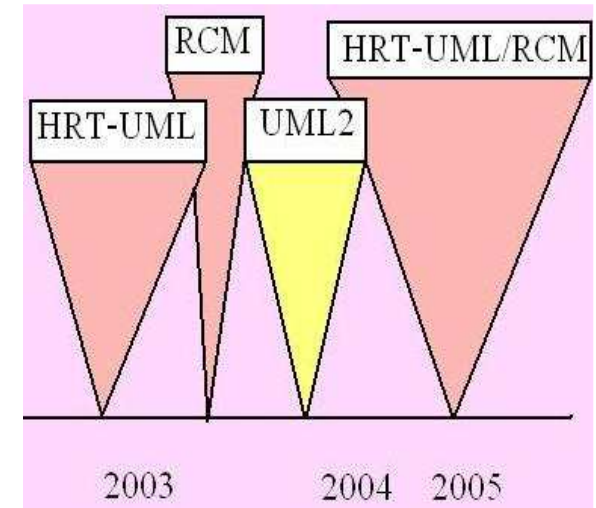
Jim Arlow, Ila Neustadt, *UML 2 and the Unified Process*, 2005, Addison-Wesley Object Technology Series

(Progress through) Contents

1. A bit of history
- 2. The HRT-UML/RCM meta-model in UML2** 
3. The HRT-UML/RCM Profile
4. *Model-based automatic meta-model generation*
5. Model-based automatic code generation

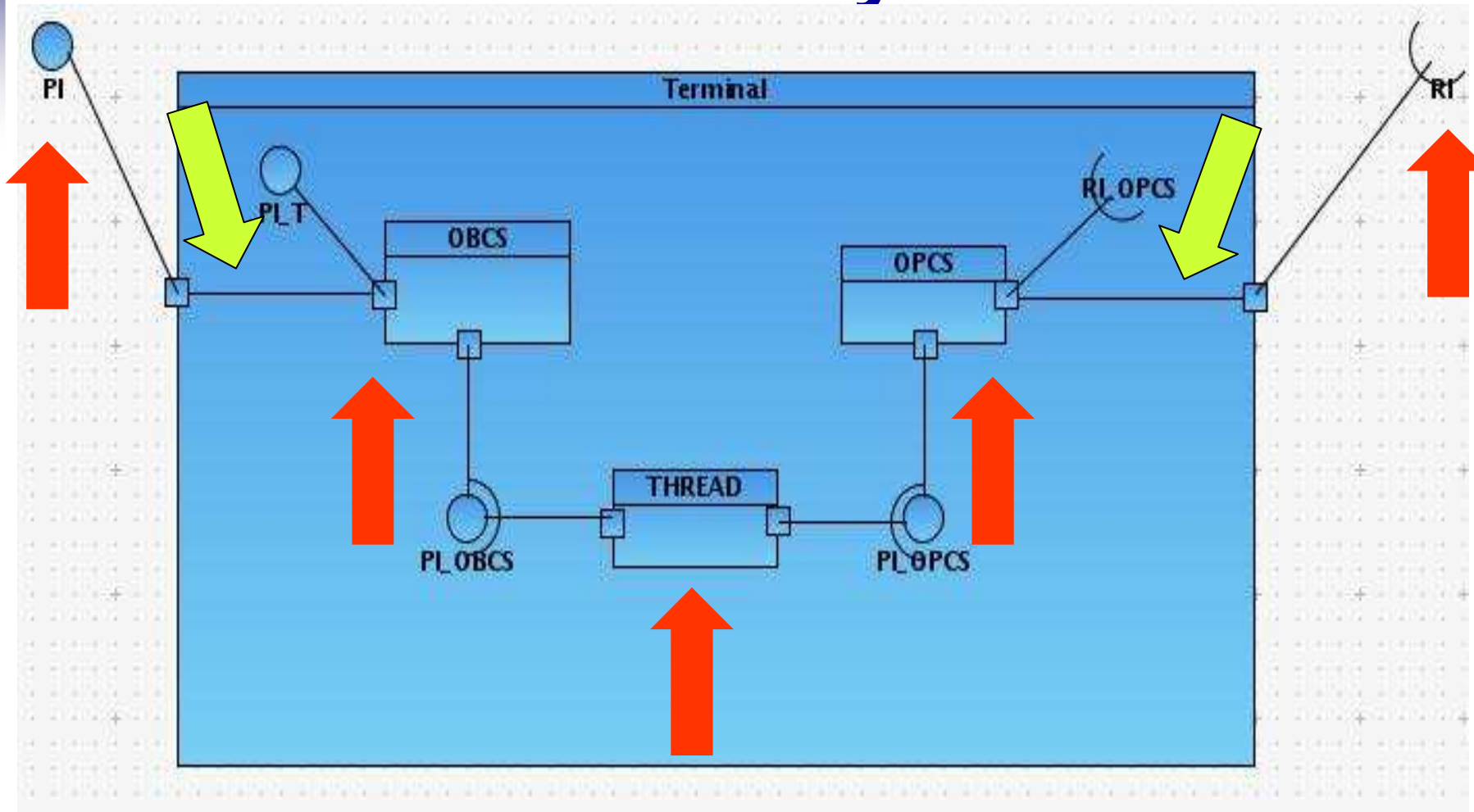
HRT-UML/RCM

- **RCM**
 - Destructuralization of terminal entities
 - Classes as compositions of meta-model elements
 - 1:1 mapping to Ada (95 and 2005!)
- **HRT-UML**
 - Placeholder object
 - Provided and required interfaces
 - Hierarchical design with delegation and decomposition
- **HRT-UML/RCM**
 - Richer meta-model and true UML2 profile
 - Model-based automatic meta-model generation
 - Model-based automatic code generation
 - Model transformation



Useful reading: “On the Dynamic Semantics and the Timing Behaviour of Ravenscar kernels”, Real-Time Systems 29(1), January 2005, 59-89

RCM Terminal Entity



Meta-model elements: PI and RI

- Every single **PI** defines *one* service that is offered by the providing entity to the environment
- Every single **RI** defines *one* service that is required by the designating entity to the environment
- The presence/absence of services in the **RI** of a model element reflects one of the three basic forms of reusable software architecture
 - **Full supply** : empty RI, all PI obligations fulfilled internally
 - **Partial supply** : nonempty RI, some PI obligations fulfilled via external support
 - **Full brokerage** : nonempty RI, all PI obligations fulfilled via external support

Meta-model elements: OBCS

- The **protocol agent** of a model terminal entity
 - Reifies execution requests (invocation of **PI** services) factorized in the form of **request descriptors**
 - The **PI** defines the synchronization semantics of the corresponding invocation request
 - ≥ 1 PAER and ≤ 1 PSER as in HRT-HOOD and HRT-UML
- It involves a **two-dimensional protocol**
 - A *horizontal* dimension
 - To warrant **mutual-exclusive** access to the internal state of the object
 - Exclusion synchronization
 - A *vertical* dimension
 - To attach a **state-based synchronization condition** to one single **PI** service depending on the internal state of the object
 - Avoidance synchronization

Meta-model elements: OPCS

- The “space” in which we specify the operational (i.e. functional) behaviour of a model entity
- Its actual design determines the **RI** of the corresponding model entity
- Every single (reified) invocation request of a **PI** service is channelled to the relevant **OPCS** in the form of a specific *request descriptor*

Meta-model elements: THREAD

- Equips model elements with an *optional* flow of control that decouples the **PI** invocation (mediated by the **OBCS**) from its servicing in the designated **OPCS**
 - The THREAD exists only in conjunction with an OBCS and a set of OPCS
 - The THREAD has no **PI** of its own but as many **RI** as the operations it must execute
- Concurrent behaviour fully conformant with the Ravenscar Profile
 - Non-terminating, with no entries, with a single suspension point
 - Implicit (time) or explicit (software or non-clock hardware interrupts)
 - The protocol agent accepts the activation event on behalf of the task and releases the task from its suspension point
 - The activation event is pushed in by an *explicit* or *implicit* call to a designated **PI**

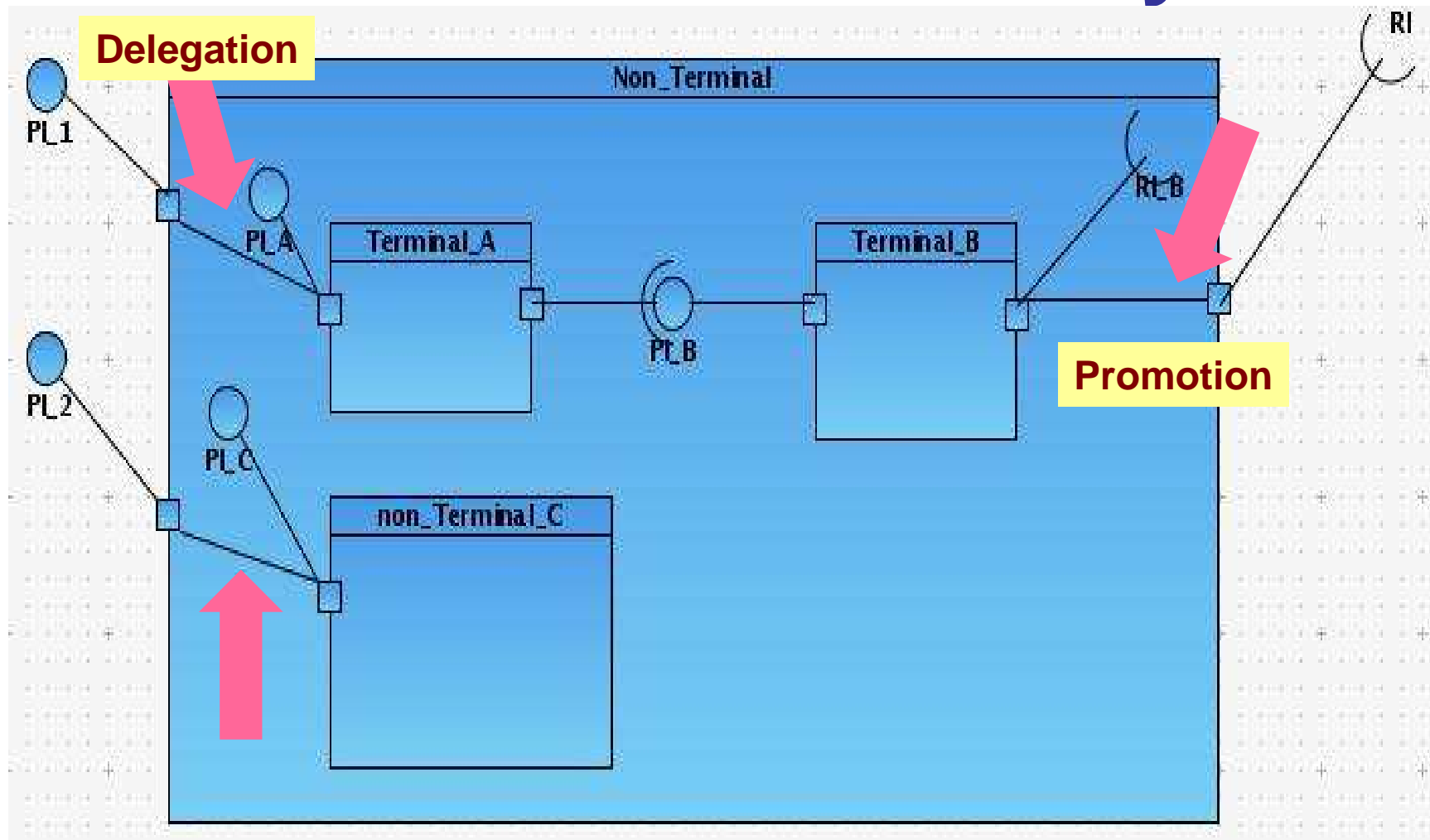
Meta-model concepts

- **<Delegation, Decomposition>**
 - Top-down architectural approach
 - An entity is *decomposed* into other entities until further decomposition is deemed no longer advantageous
 - **Delegation** establishes how parent (provided) operations resolve into child operations
- **<Promotion, Aggregation>**
 - Bottom-up architectural approach
 - Pre-existing entities are joined up in an aggregating entity
 - **Promotion** determines how the **PI** and **RI** of pre-existing entities become visible as **PI** and **RI** of the aggregating entity

Non-terminal entity

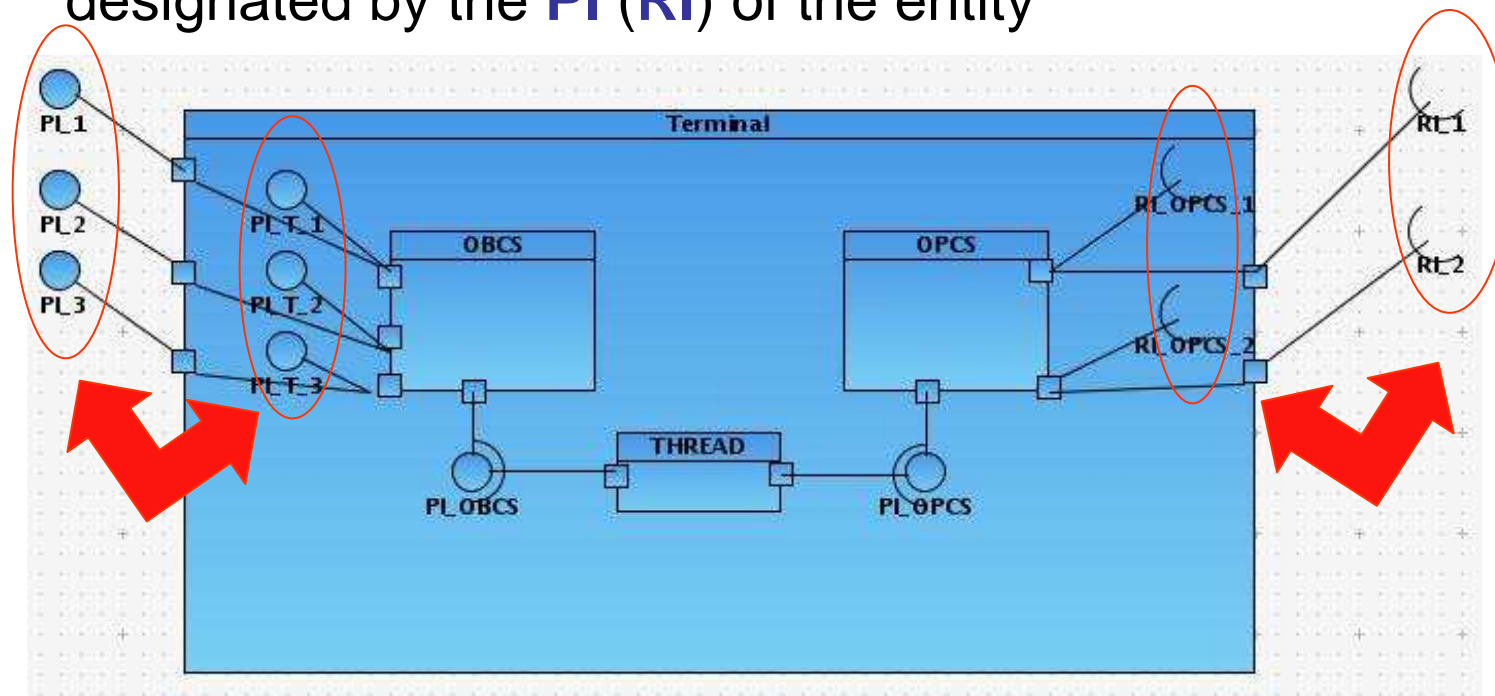
- A composite of terminal and non-terminal entities
- *Cannot* possess **THREAD**, **OBCS** and **OPCS** of its own
 - Its **PI** must be fully delegated to the (child) entities it composes
 - Its **RI** must reflect the promotion of the **RI** of the (child) entities it composes

RCM Non Terminal Entity



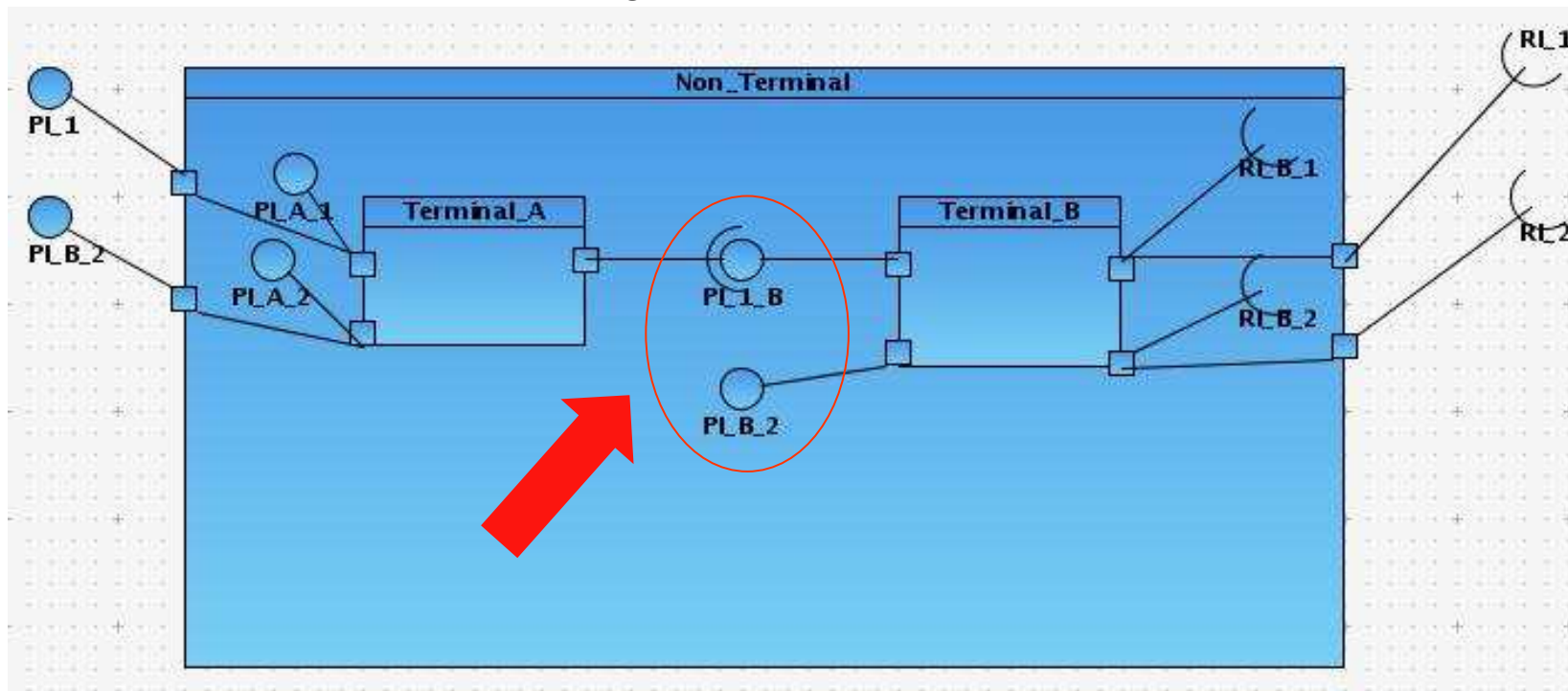
PI and RI are no monoliths – 1

- Every single **PI** (**RI**) defines *a single* method of the interface of a model entity
 - The interface of an entity is the union set of all methods designated by the **PI** (**RI**) of the entity



PI and RI are no monoliths – 2

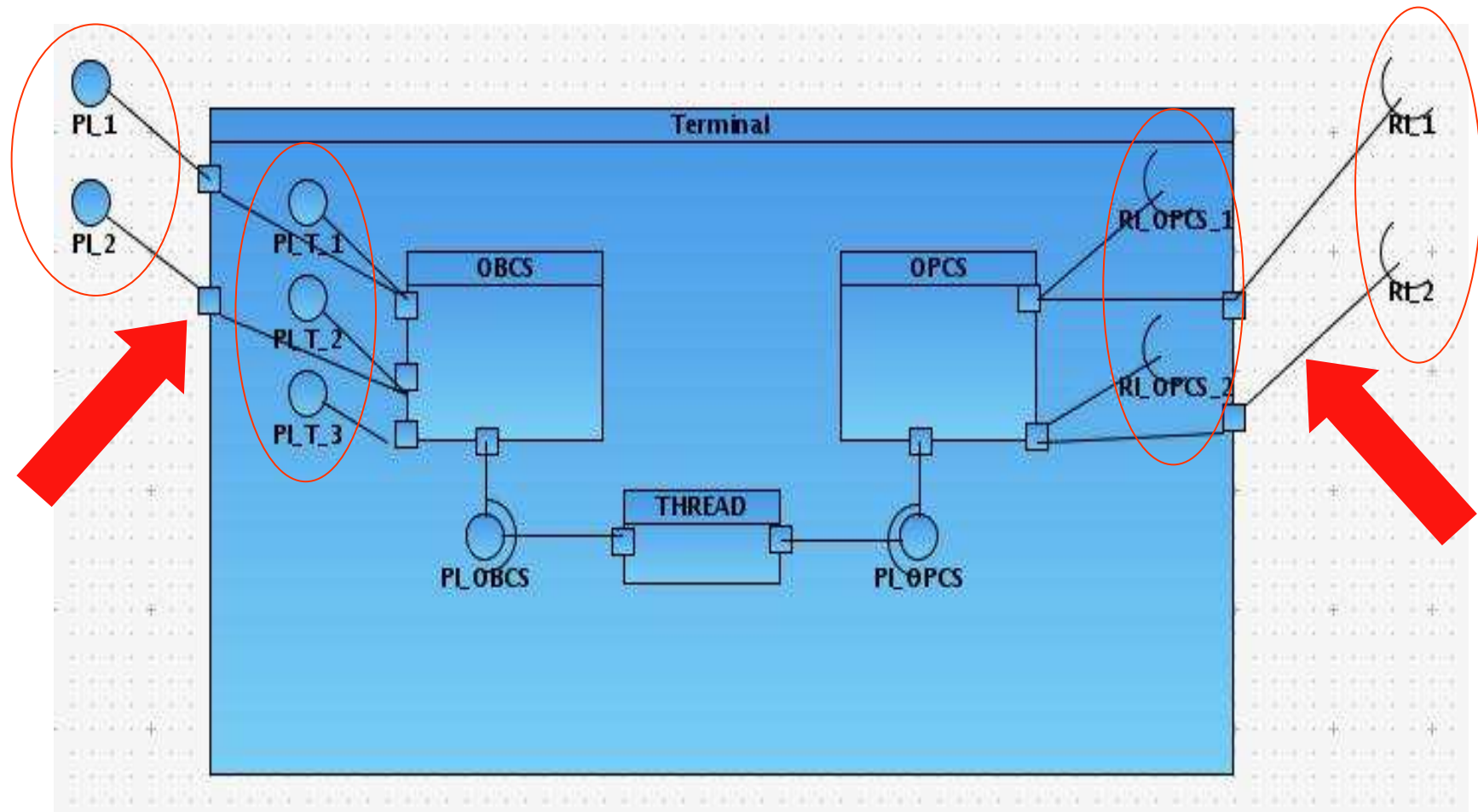
- **Drawback**
 - Interface explosion
- **Benefit**
 - Better control in the design of reusable software



PI and RI are no monoliths – 3

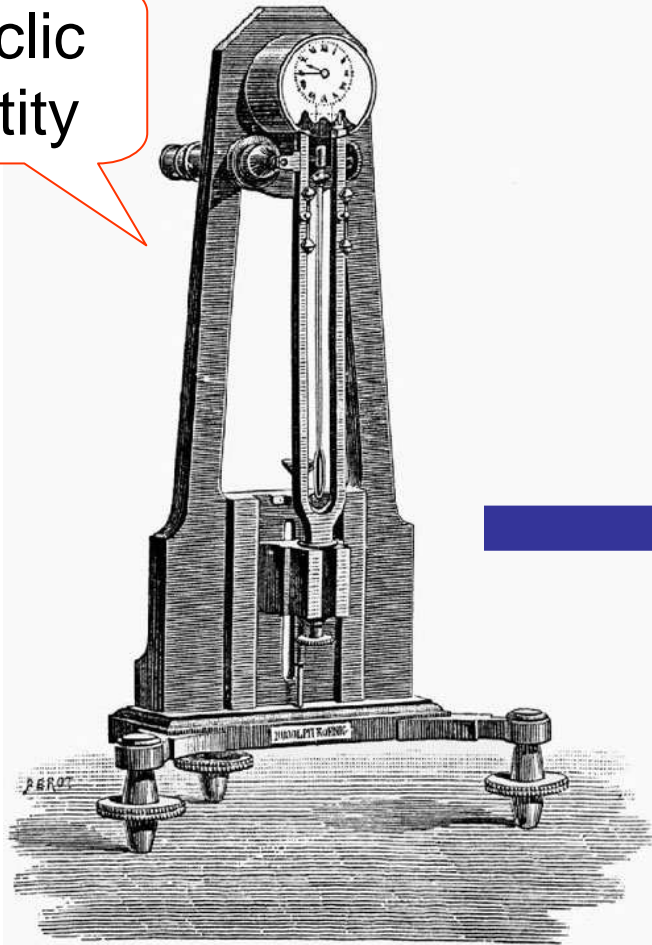
- The notion of (interface) promotion can be equally applied to **RI** and **PI**
 - The promotion of a (child) **PI** determines the **PI** of the (parent) aggregating entity
 - The **RI** of a child entity must be promoted to the **RI** of the (parent) entity
 - Which we can do thanks to the original HRT-UML notion of “placeholder”

Promotion of child PI and RI



OBCS vs. Protected Entity – 1

Cyclic Entity



Protected Entity with 1 PSER

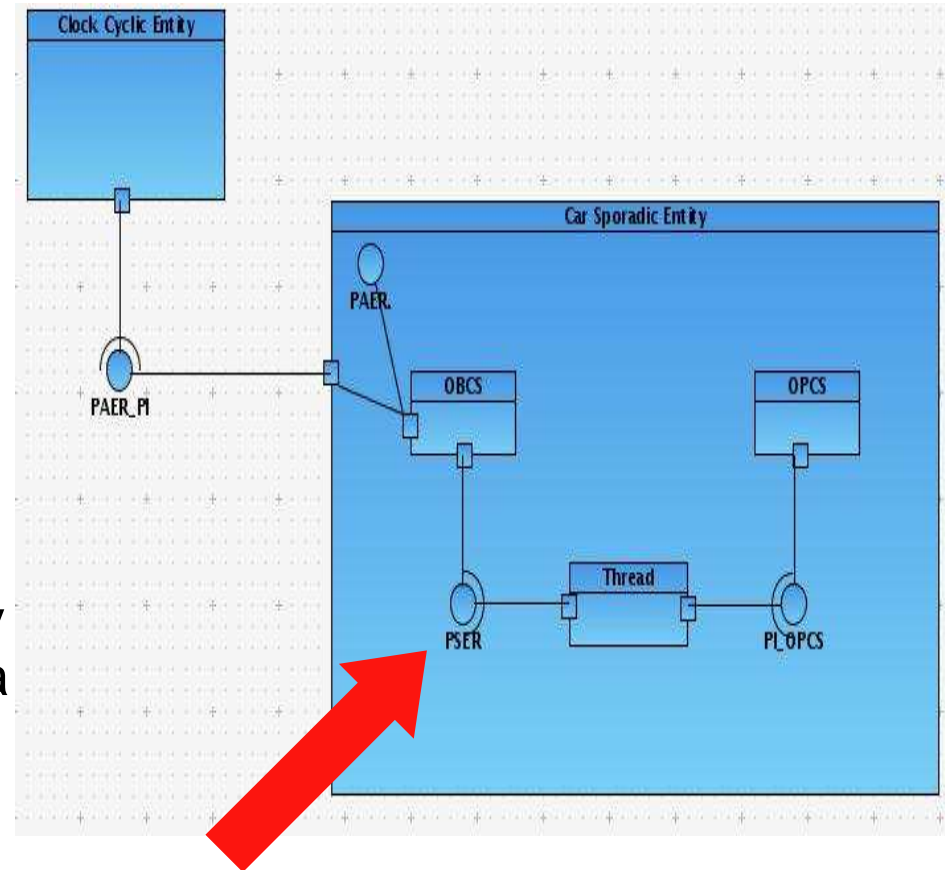


Sporadic Entity

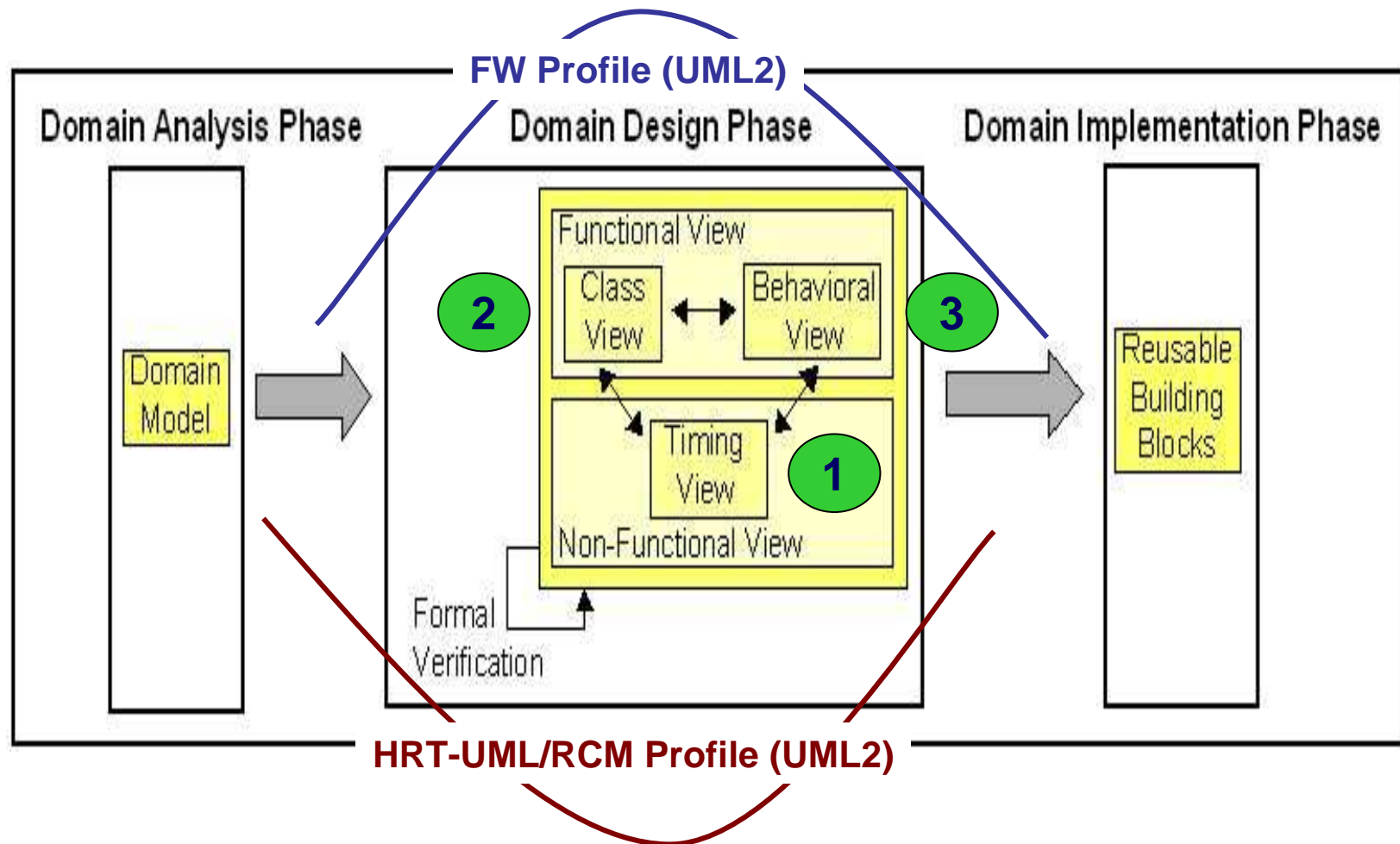
Courtesy of:
<http://www.forumpa.it/canali/vicine/strumenti/dossier.html>

OBCS vs. Protected Entity – 2

- **Proposition**
 - One Protected entity with 1 PSER is ontologically equivalent to the OBCS of a Sporadic entity
 - ≤ 1 PSER per Protected entity follows from the RCM
- **Corollary**
 - If there is a Protected entity with 1 PSER then there is a Sporadic entity that contains it and views it as its own OBCS

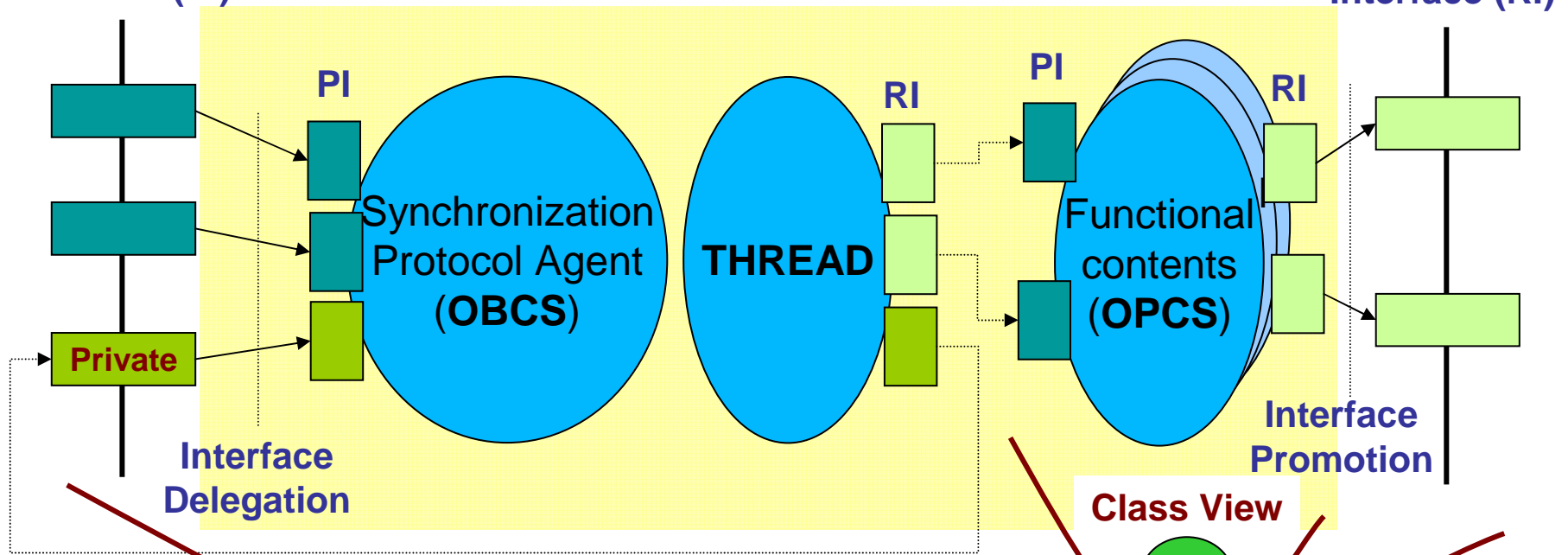


Integration with functional modeling



Integration with functional modeling

3
Component View
(behavioural)
Provided Interface (PI)



1
Timing View

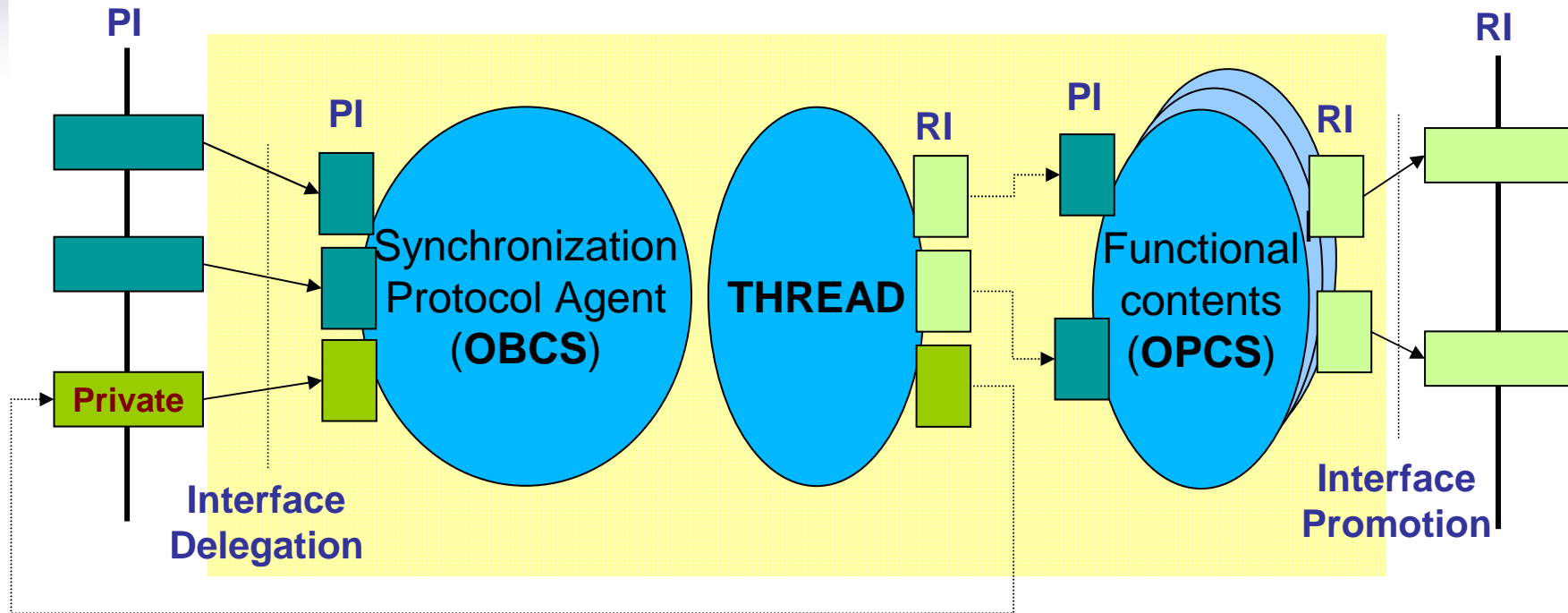
2
Class View

Preservation of properties at run time: the RCM Virtual Machine

- A run-time environment that only accepts and supports “legal” entities
 - VM-level containers = RCM terminal entities
- Services that aid run-time entities to preserve their non-functional properties
 - Execution time monitoring
 - Group budgeting
 - Enforcement of minimum inter-arrival time
 - Fault containment regions (space and time fencing)
- A compilation system that produces executable code for “legal” entities only
 - As many off-line legality checks as possible
- A concurrent computational model provably amenable to off-line feasibility analysis

**Does not that sound like the Ravenscar Profile
extended with other Ada 2005 features?**

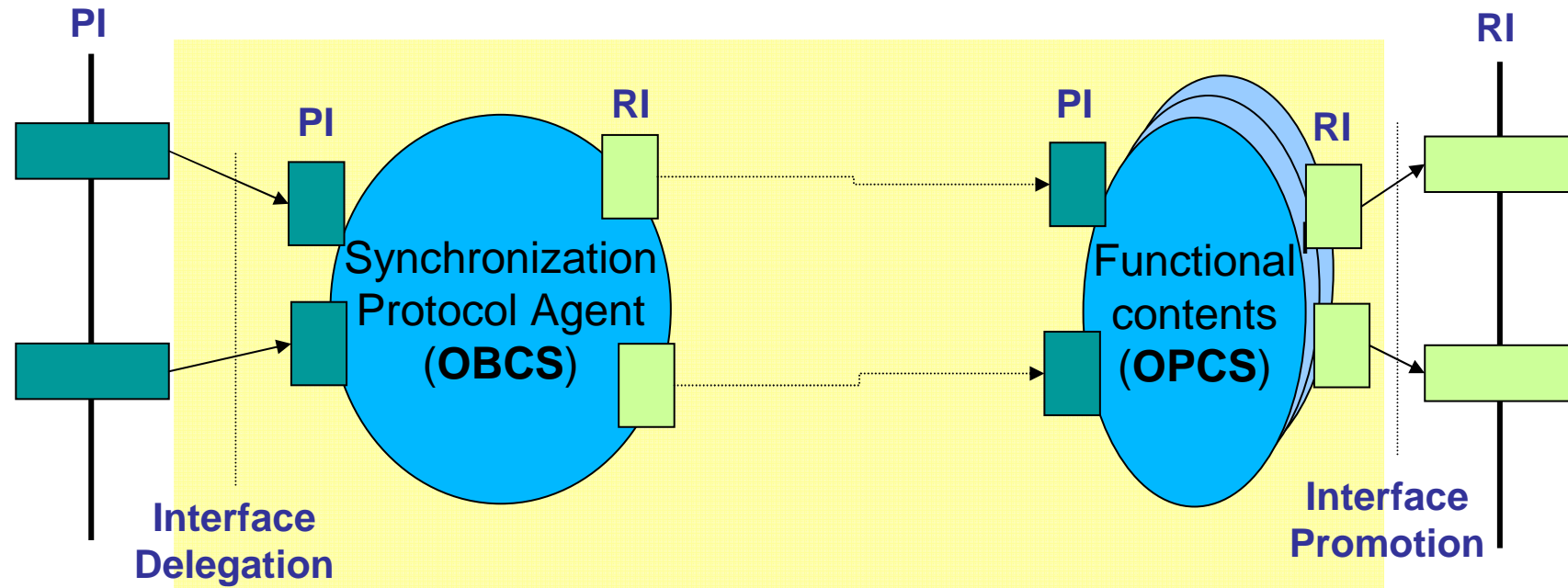
RCM entities as VM-level containers – 1



- **Threaded container**

- Needs an agent to “police” the synchronization protocol for interaction with other threads across its **PI** and **RI**

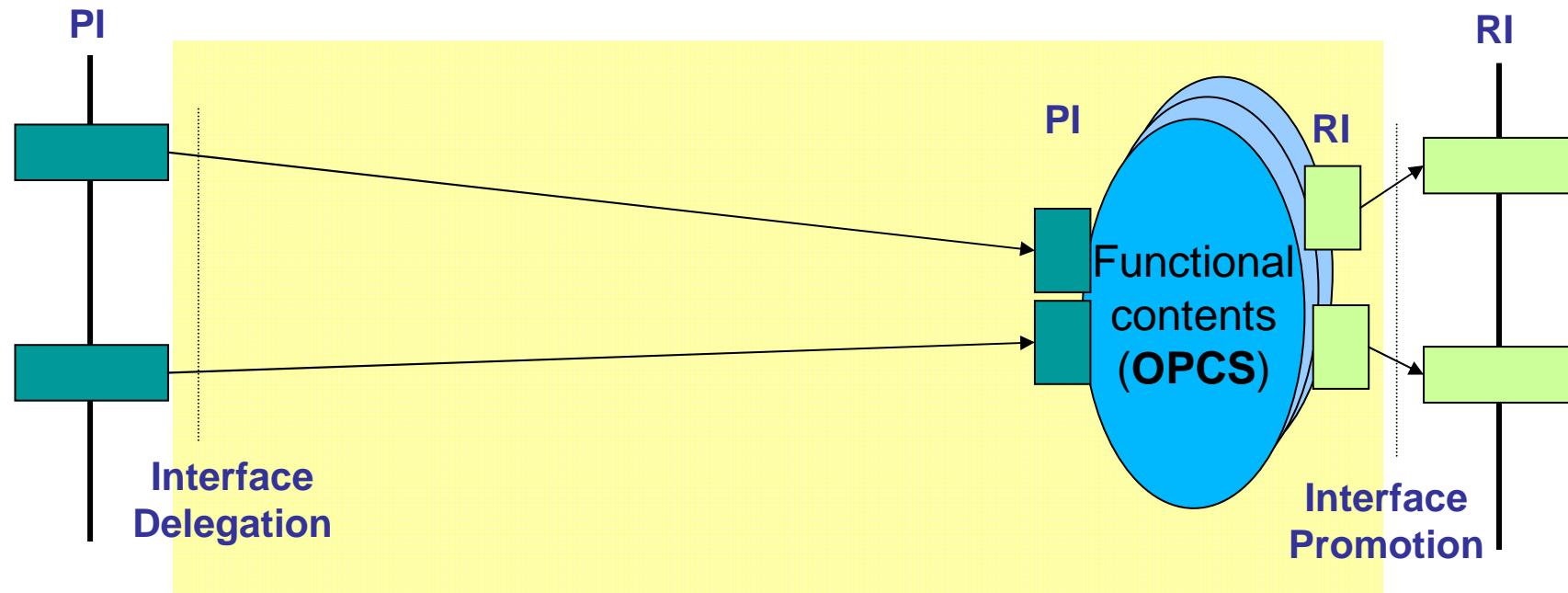
RCM entities as VM-level containers – 2



- **Protected container**

- Needs an agent to “police” the synchronization protocol that regulates the interaction among threads across its **PI** and **RI**

RCM entities as VM-level containers – 3



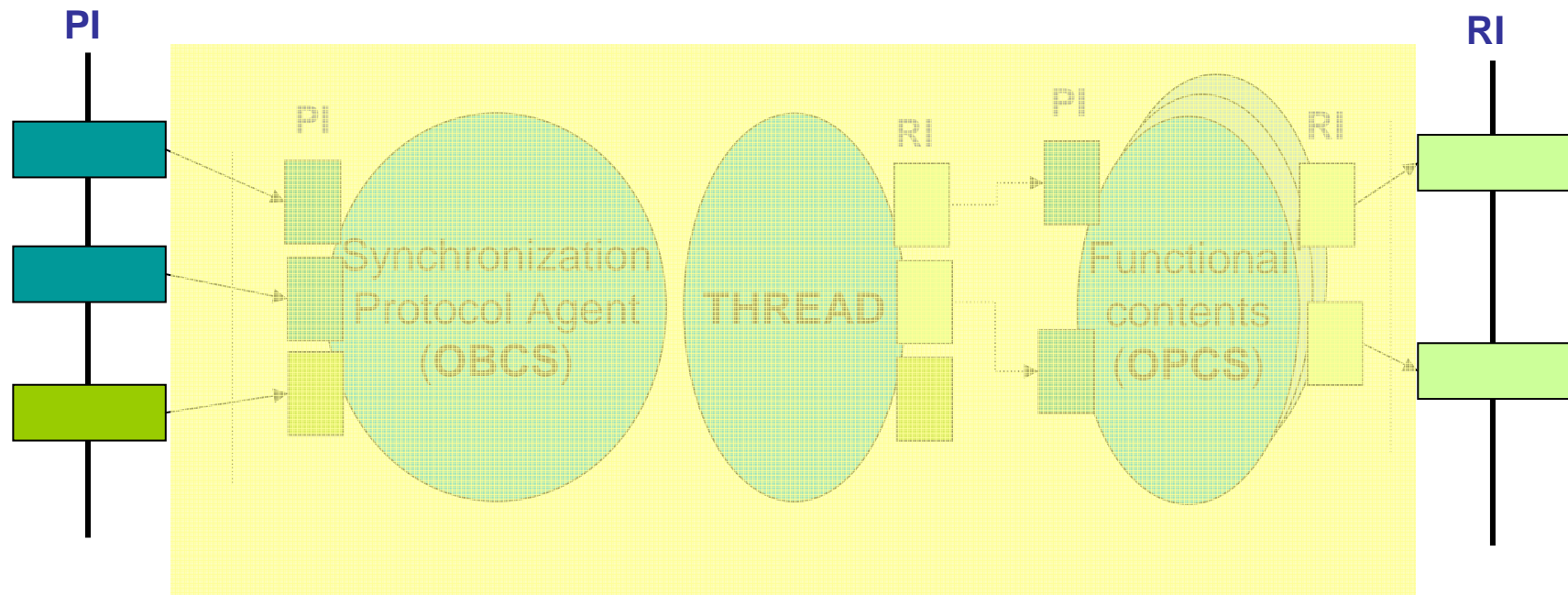
- **Passive container**

- Features a **PI** equipped with a *void* synchronization protocol
- Hence it has **no control** over access to its state by external flows of control

Component view – 1

- Defines the system architecture in terms of its interfaces
 - **PI** and **RI**, with use relation among them
- Defines the services to be provided/required by individual components in terms of
 - Signature of the operations that implement them
 - In some IDL
 - Functional activation conditions attached to non-functional attributes
 - Synchronization protocol
 - “Concurrent weight” of the PI realization
 - $W=0$ → **immediate** execution of the PI service
 - $W=1$ → **deferred** execution of the PI service
 - Distribution transparency
 - Replication transparency
- The functional activation conditions use protocol state machines (PSM) to specify the protocol to deploy on service invocation
 - The PSM regulates how the caller may effect the callee and how the invocation may effect the caller
 - In other words it specifies the **OBCS**

Component view – 2

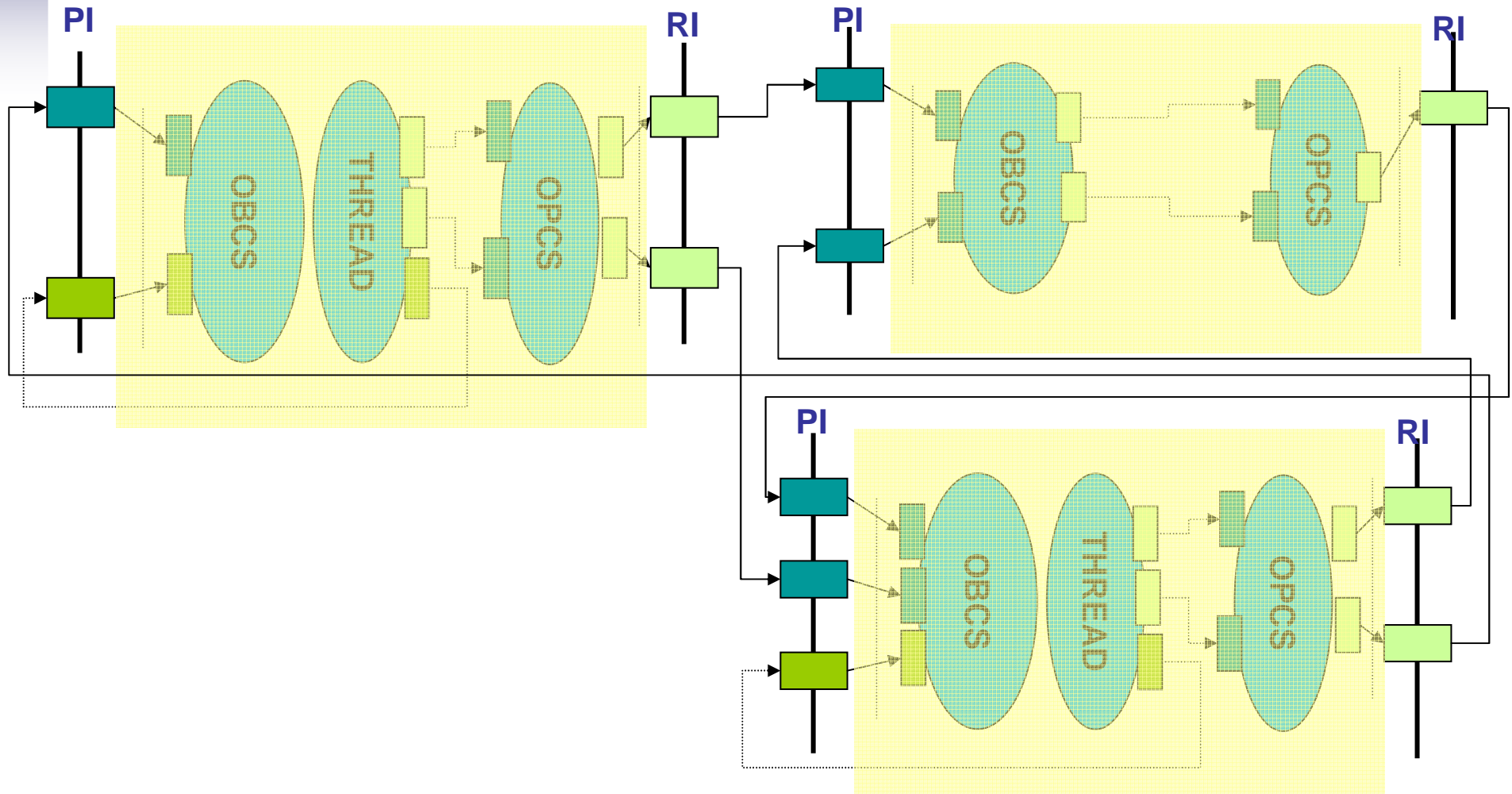


- The HRT-UML/RCM profile provides for a property-preserving mapping of PI into VM-level containers
 - A vertical transformation of model (view)

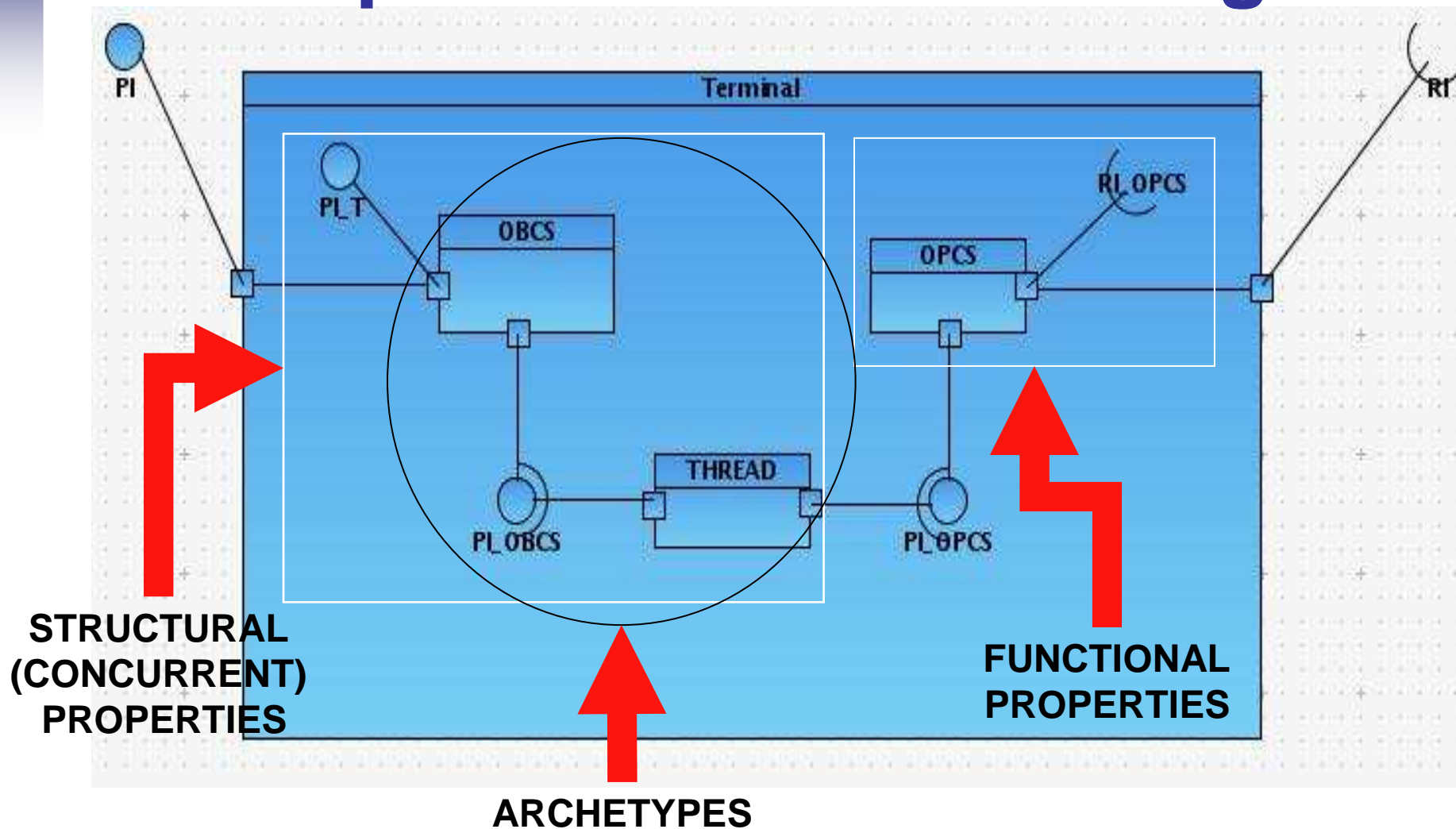
Class view

- Defines the behaviour behind the interfaces described by the component view
 - For a reusable framework it only captures the behaviour that is invariant in the domain
- Defines the adaptation points where application-specific behaviour can be inserted
 - Abstract methods
 - Virtual methods
- Behaviour is modelled through state machine constrained to fit the operational semantics expressible by **OPCS**
- Mapping to HRT-UML/RCM
 - The Class View models a set of **OPCS**
 - The provided description is at class level, not at the operation (method) level
 - Individual class methods map to individual OPCS

Timing view



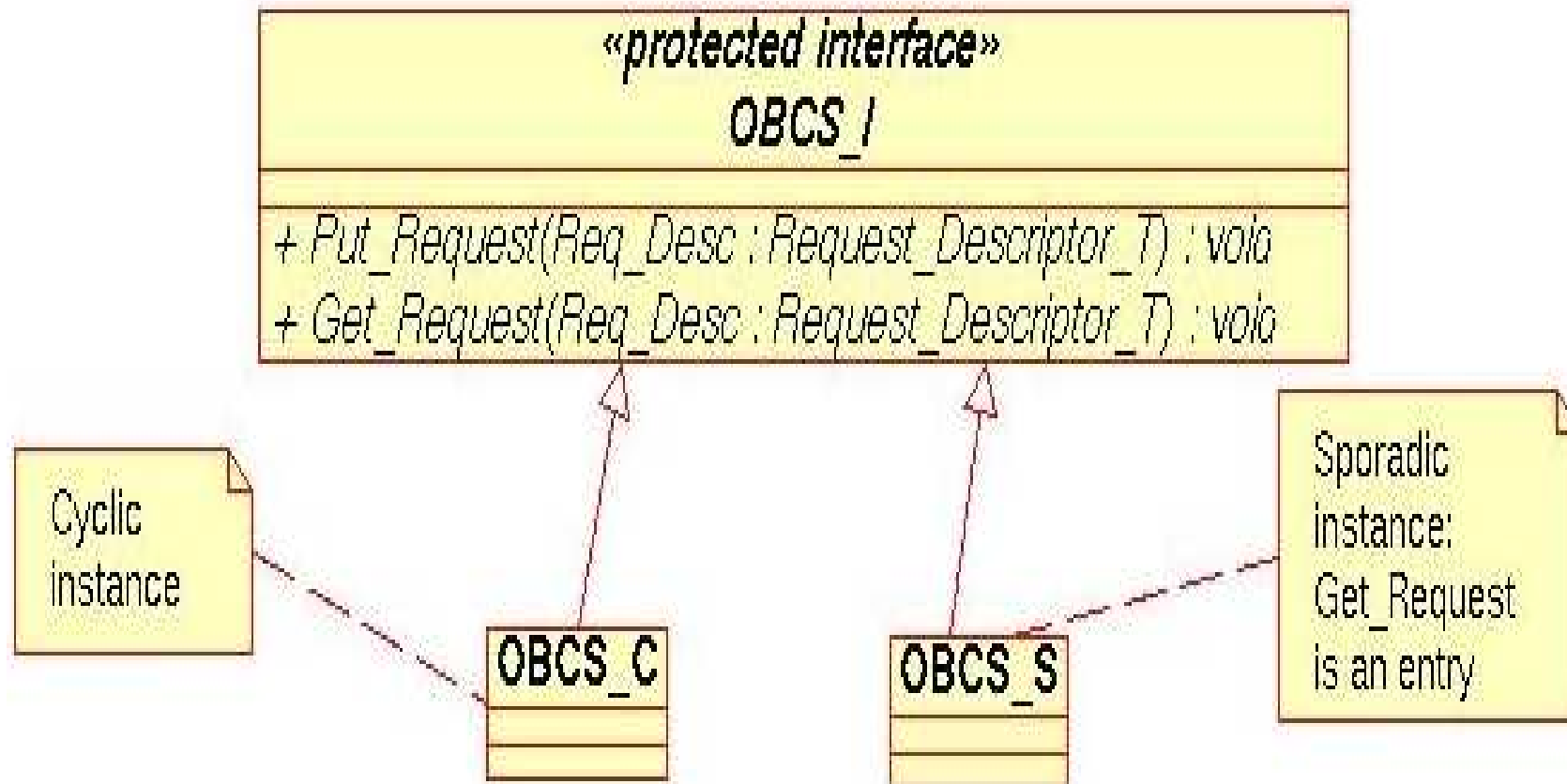
The importance of factorizing – 1



The importance of factorizing – 2

- **Individual model entities have much in common that is best factored out**
 - Factorizing the model increases modularity, isolation and cohesion
 - Factorizing the ontology of model entities facilitates proof and simplifies code generation
 - Every allowable **THREAD** and **OBCS** can be regarded as an instance of a specific (meta-model) type

Factorizing the OBCS – 1



Factorizing the OBCS – 2

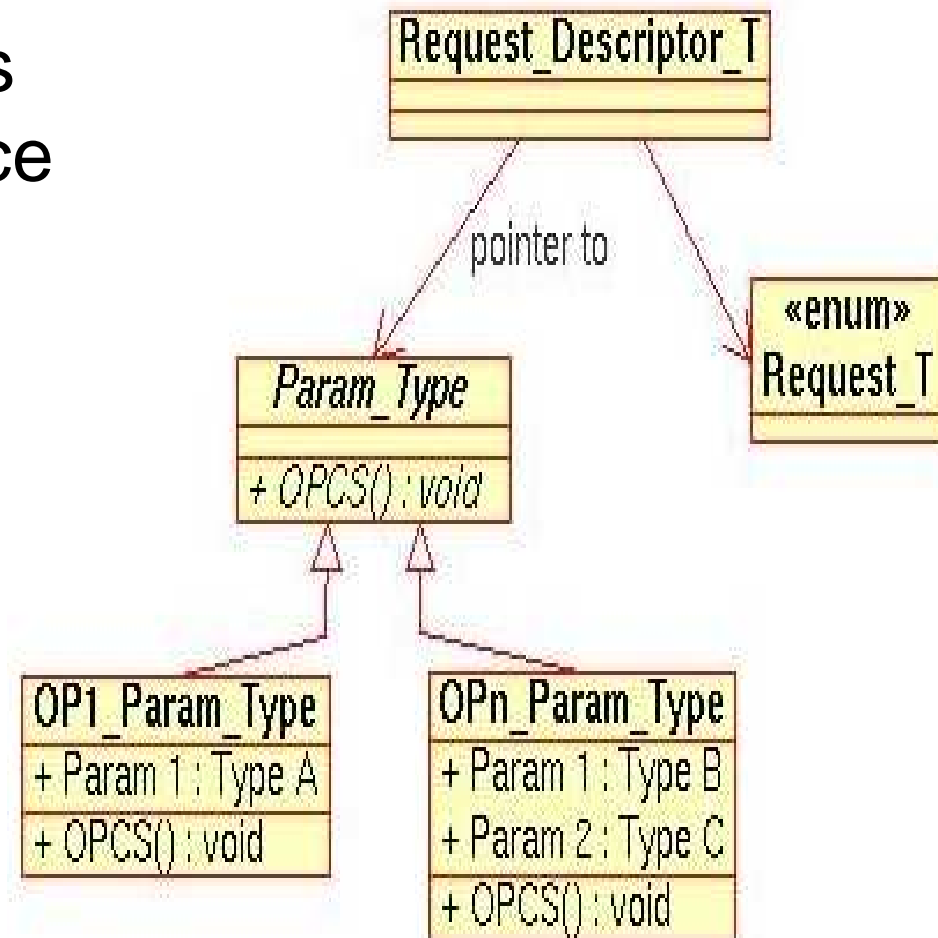
- The **OBCS** embedded in Sporadic and Cyclic entities is a distinct implementation of one and the same interface
 - Cyclic entity
 - The *Get_Request* method is a PAER
 - Sporadic entity
 - The *Get_Request* method is a PSER
- With Ada 2005 they become distinct implementations of a **protected interface**

Reification and unfolding – 1


- The **THREAD** fetches from the **OBCS** the reified *request descriptor* of the **PI** invocation to serve in accord with the applicable protocol
 - Reification occurs when the invocation request is “pushed” (in the OBCS store) by the PI caller
 - The next due request descriptor is “pulled” out by THREAD when the next activation event arrives
- The **THREAD** then unfolds the request descriptor and issues a polymorphic invocation to the service method realized in the designated **OPCS**
 - If there was no THREAD then the OBCS would have no “pull channel” and it would thus directly execute the unfolding and the OPCS invocation

Reification and unfolding – 2


- Every **OPCS** operates on a particular instance of a parameter set
- Each such parameter set is reified into a specialization of an archetypal request descriptor




(Progress through) Contents

1. A bit of history
2. The HRT-UML/RCM meta-model in UML2
- 3. The HRT-UML/RCM Profile** 
 - *Skipped in the interest of time*
4. Model-based automatic meta-model generation
5. Model-based automatic code generation

(Progress through) Contents

1. A bit of history
2. The HRT-UML/RCM meta-model in UML2
3. The HRT-UML/RCM Profile
- 4. Model-based automatic meta-model generation** 
 - *Skipped in the interest of time*
5. Model-based automatic code generation

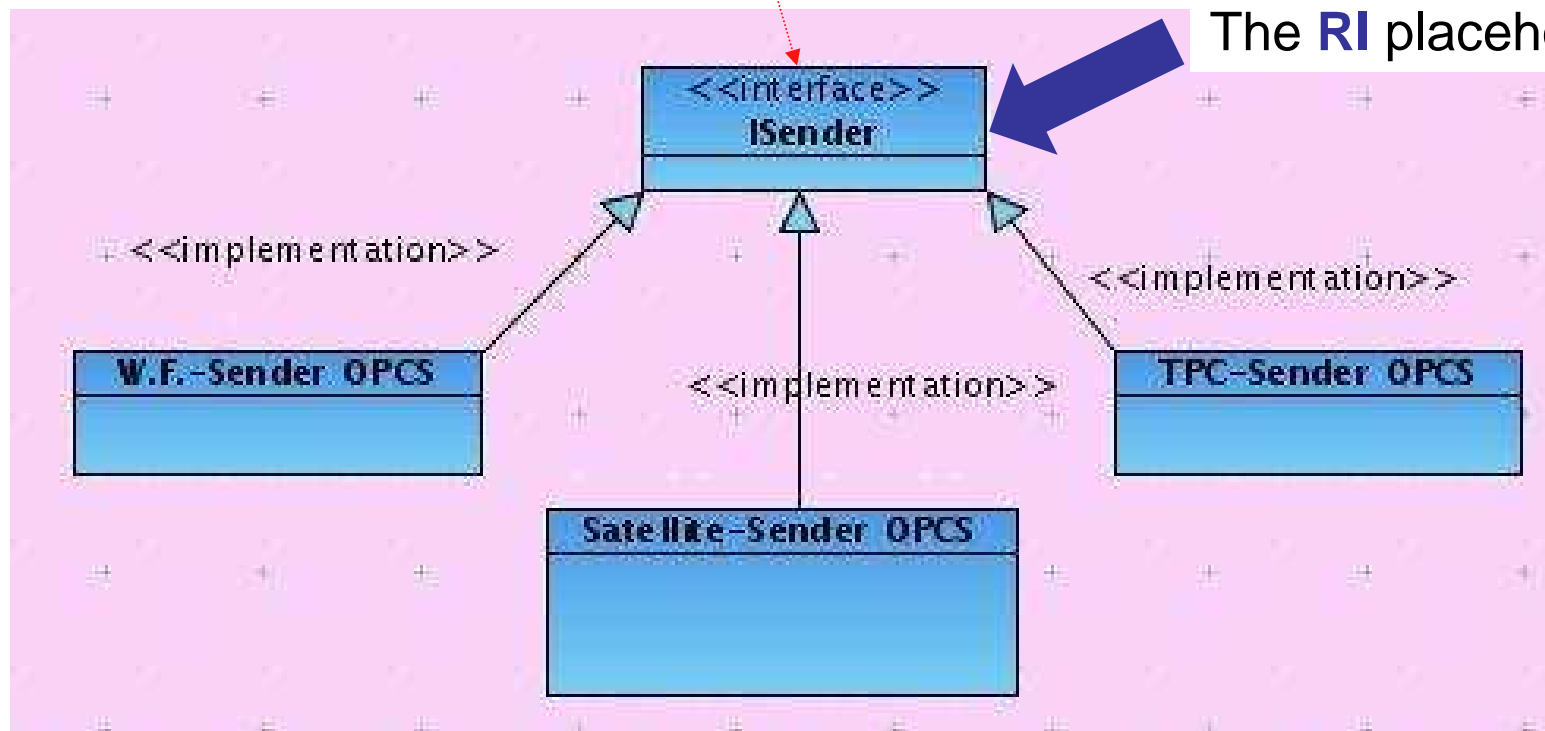
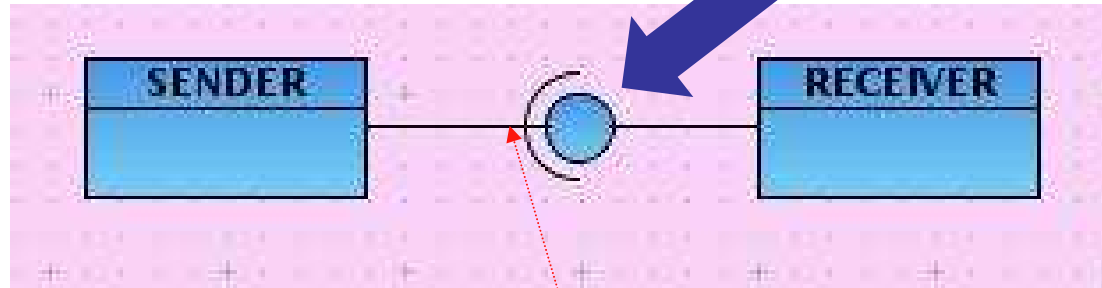
(Progress through) Contents

1. A bit of history
2. The HRT-UML/RCM meta-model in UML2
3. The HRT-UML/RCM Profile
4. Model-based automatic meta-model generation
- 5. Model-based automatic code generation** 

Goals

- Model-based automatic code generation
 - Striving for 100% of automation
- Maximum separation between functional and concurrent views
 - Functional view → **OPCS**
 - Concurrent view → **OBCS** and **THREAD**
- 1:1 mapping of meta-model entities to code
 - “*Generative archetypes*”
 - More geared to proof, reuse and adaptability
- Full compliance of archetypes to the Ravenscar Profile extends to all instances
 - By definition of the guaranteed properties of the model

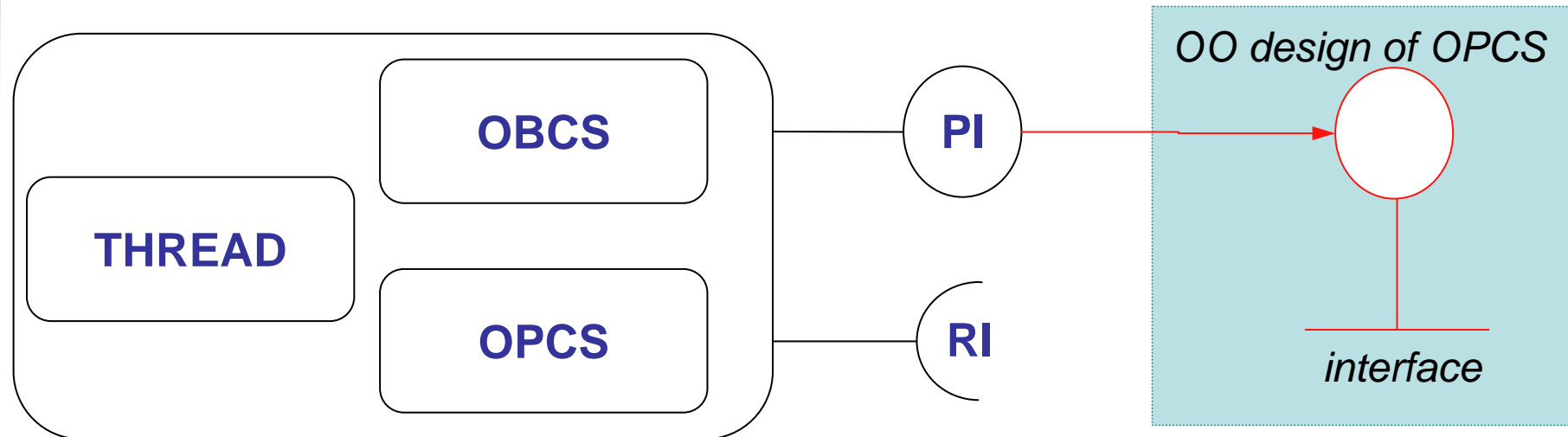
A case study



Issues

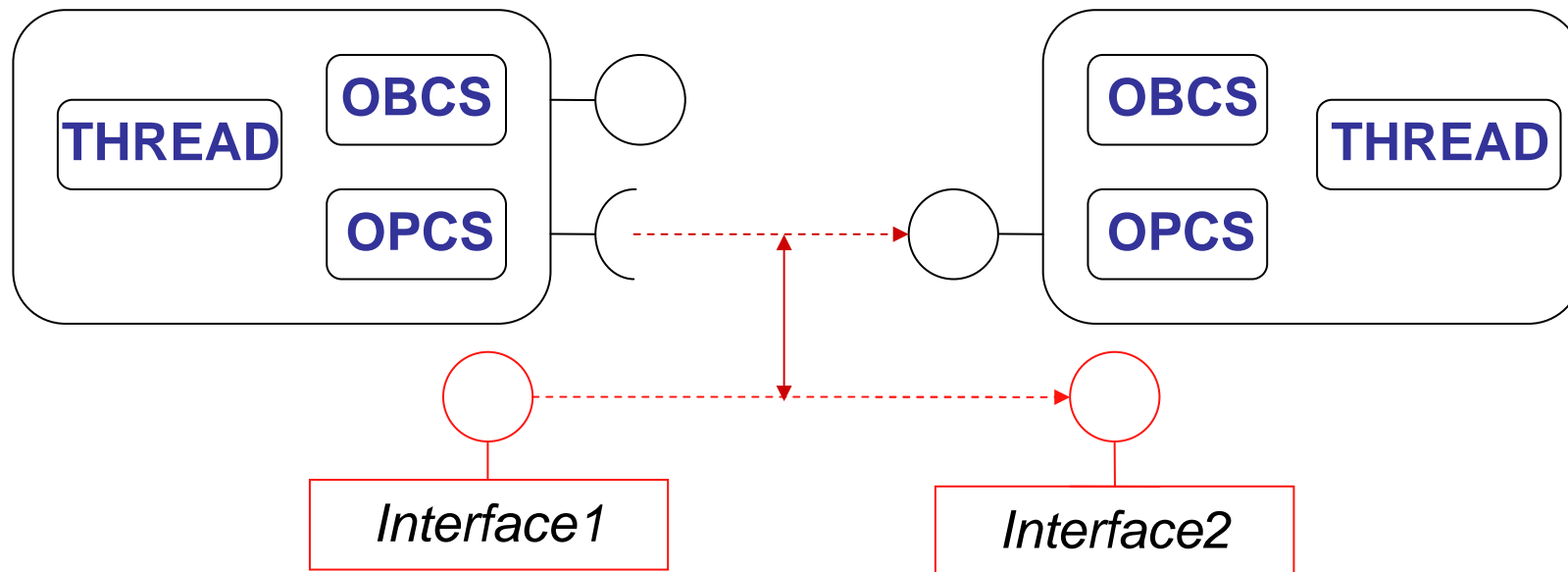
1. Distinct **PI** that implement the same interface must be able to *interchangeably* meet the corresponding **RI**
 - The client RI must be able to choose a **PI** on “value” factors (e.g. WCET, interference, etc.)
2. The **OPCS** realizes a **PI** and determines some **RI**
3. The **OPCS** must be amenable to (educated) OO design and implementation
4. The *functional state* of an entity corresponds to the “**this**” of the OO instance that realizes the **OPCS**
5. The assembly relation carries implications on the class (modeling) view

1. Interchangeability



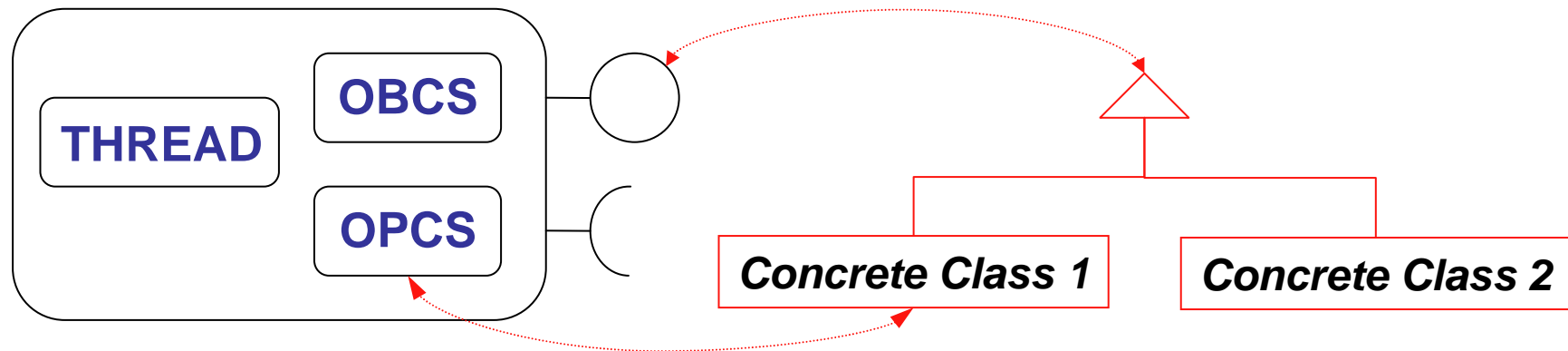
- A terminal entity should provide the *same* **PI** as the OO design of its **OPCS**
 - This identity must be warranted *across* the functional and concurrent views

2. Consistency of views



- The assembly of model entities must reflect the functional dependency between OO structures
 - If *Interface1* and *Interface2* are compatible interfaces in the component view then this must permit matching the **RI** to the **PI** in the concurrent view

3. OO design of OPCS

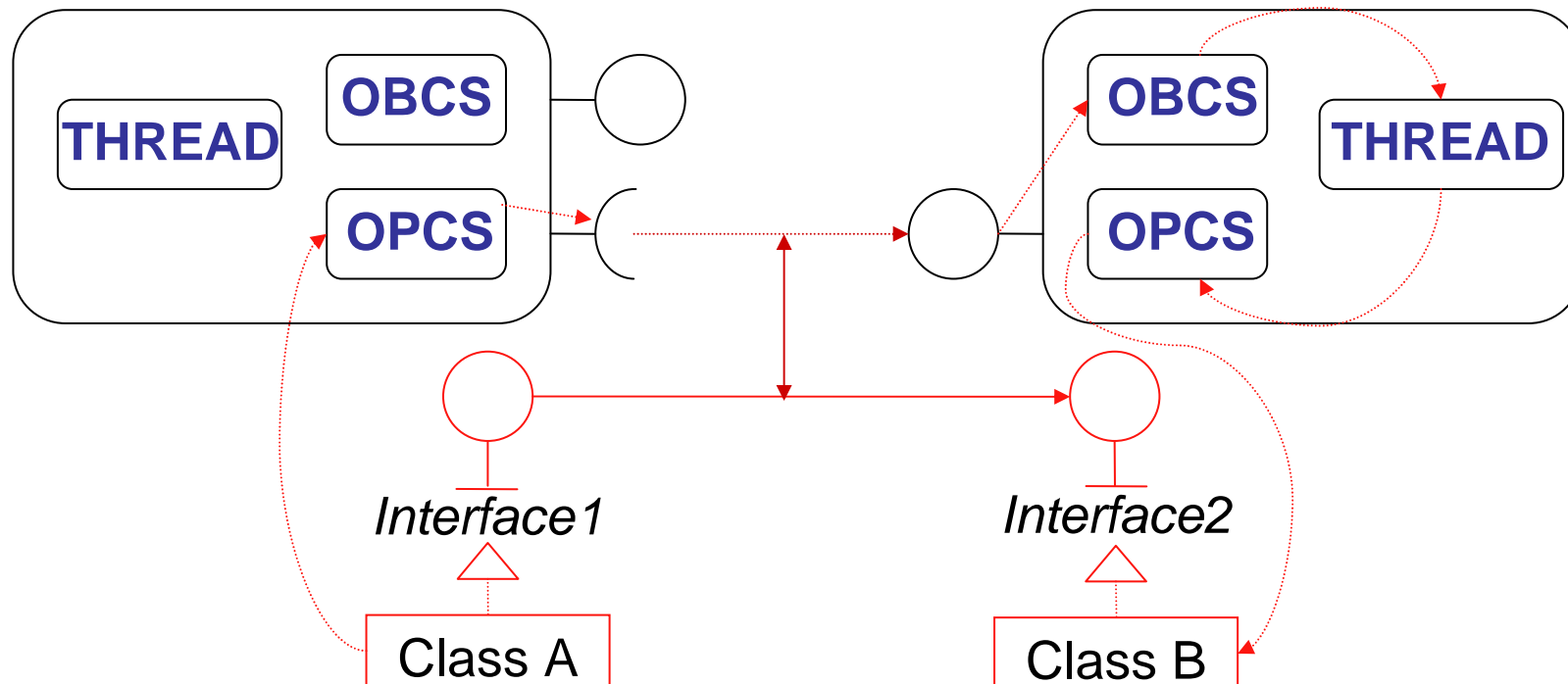


- The elementary **OPCS** maps to a class method
- The union of all elementary **OPCS** of a terminal entity is designed as the instance of a concrete class that implements the “right” **PI**

4. State of terminal entity

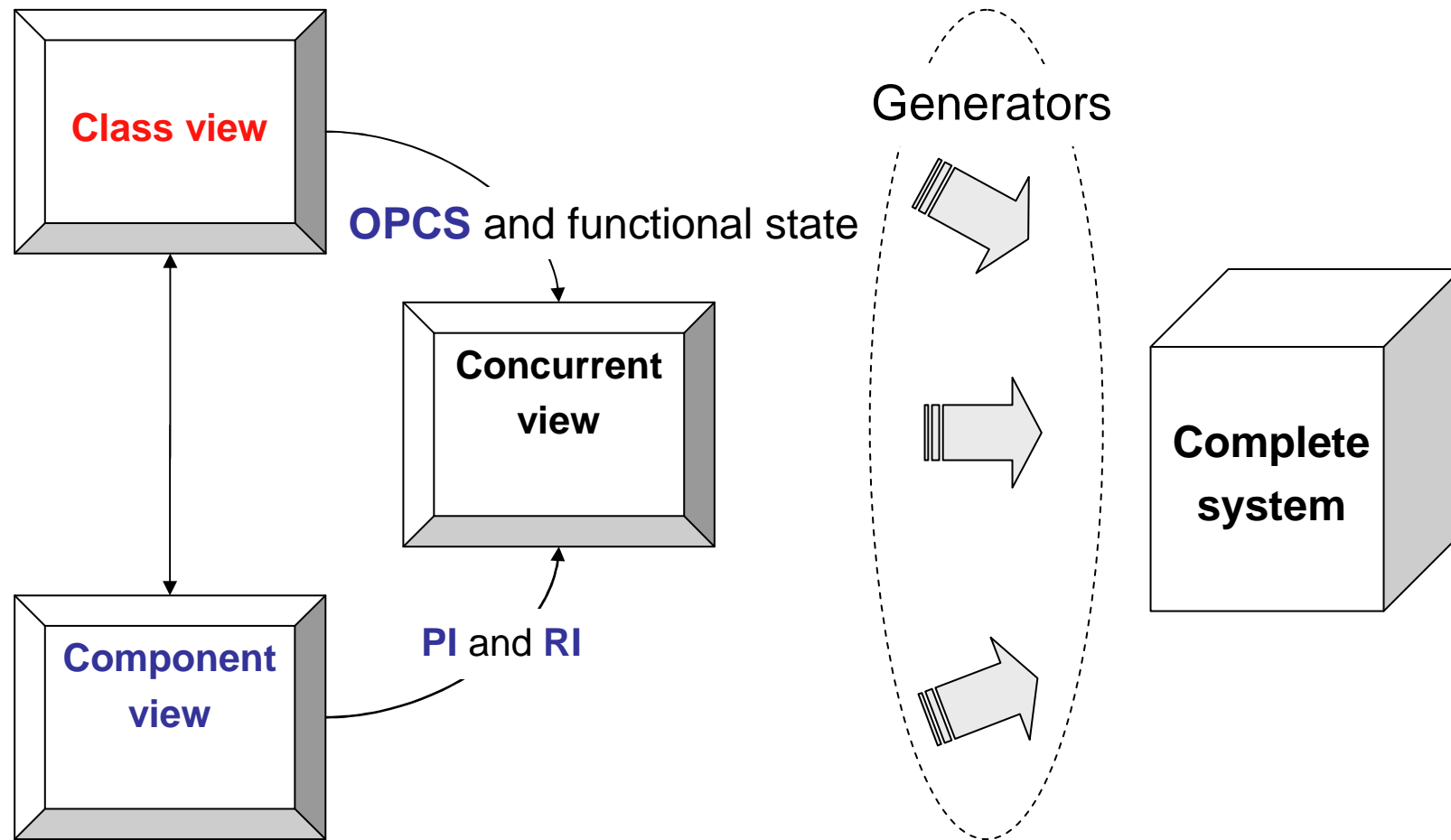
- Terminal entities have a bi-dimensional state
 - A protocol state owned by the **OBCS**
 - The **THREAD** has no state of its own
 - Relevant for the concurrent view of the model
 - A functional state determined by the members of the class that realizes the **OPCS** (\equiv “this”)
- Any part of the state can be modified from the inside following designated protocols

5. Implications of assembly

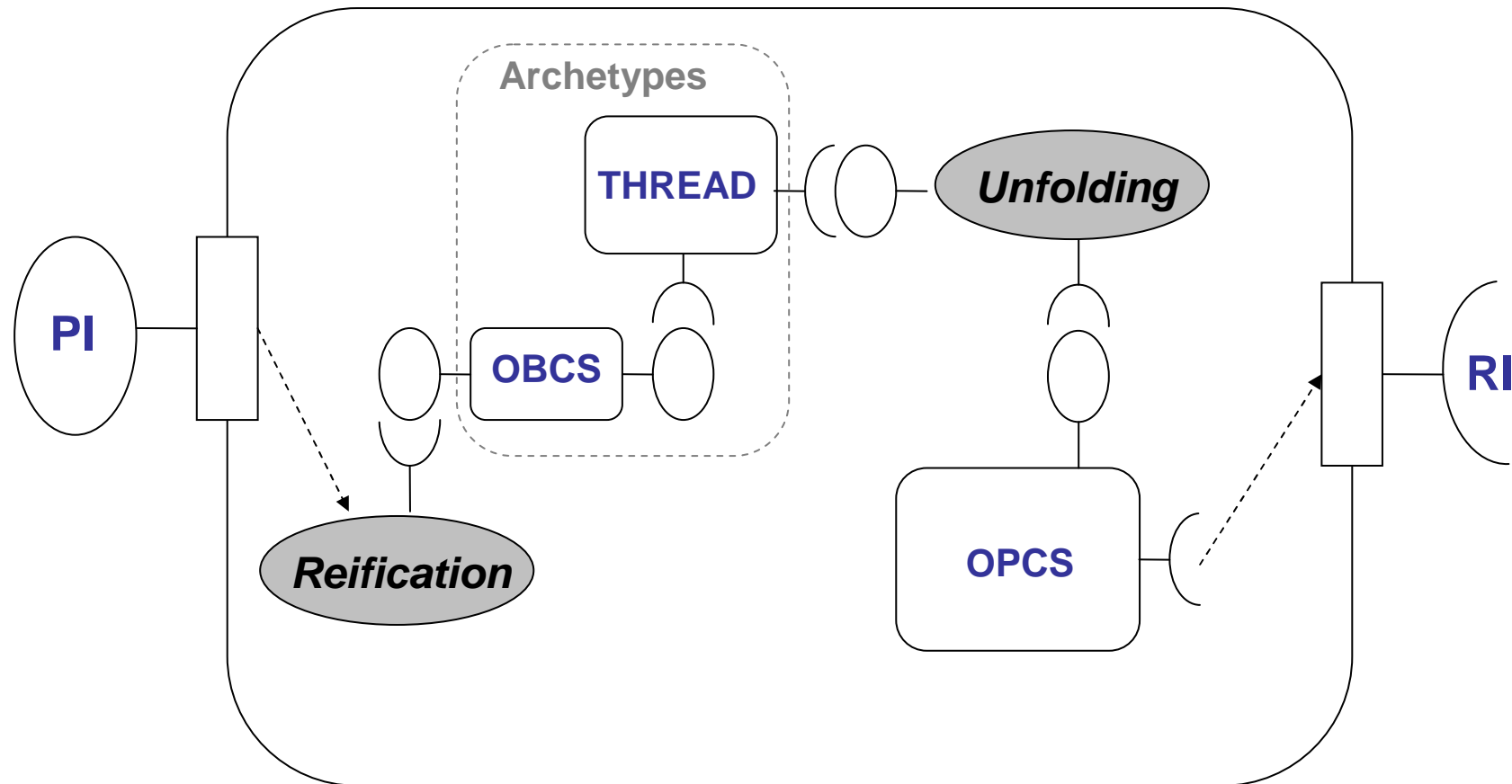


- The assembly relation implies (indirect) functional dependence between the classes that realize the corresponding **OPCS** (cf. issue 2)

Generative approach



Generative approach



A (Byzantine?) solution – 1

generic

-- type implementing the desired interface

type Implementation **is**

new <Implementations.>Interface **with private;**

package Generic_Implementation **is**

-- returns a pointer to an implementation of the

--+ desired interface

function Get_Instance

return <Implementations.>Interface_Ref;

private

-- instance wrapping OPCS and functional state

Instance : **aliased** Implementation;

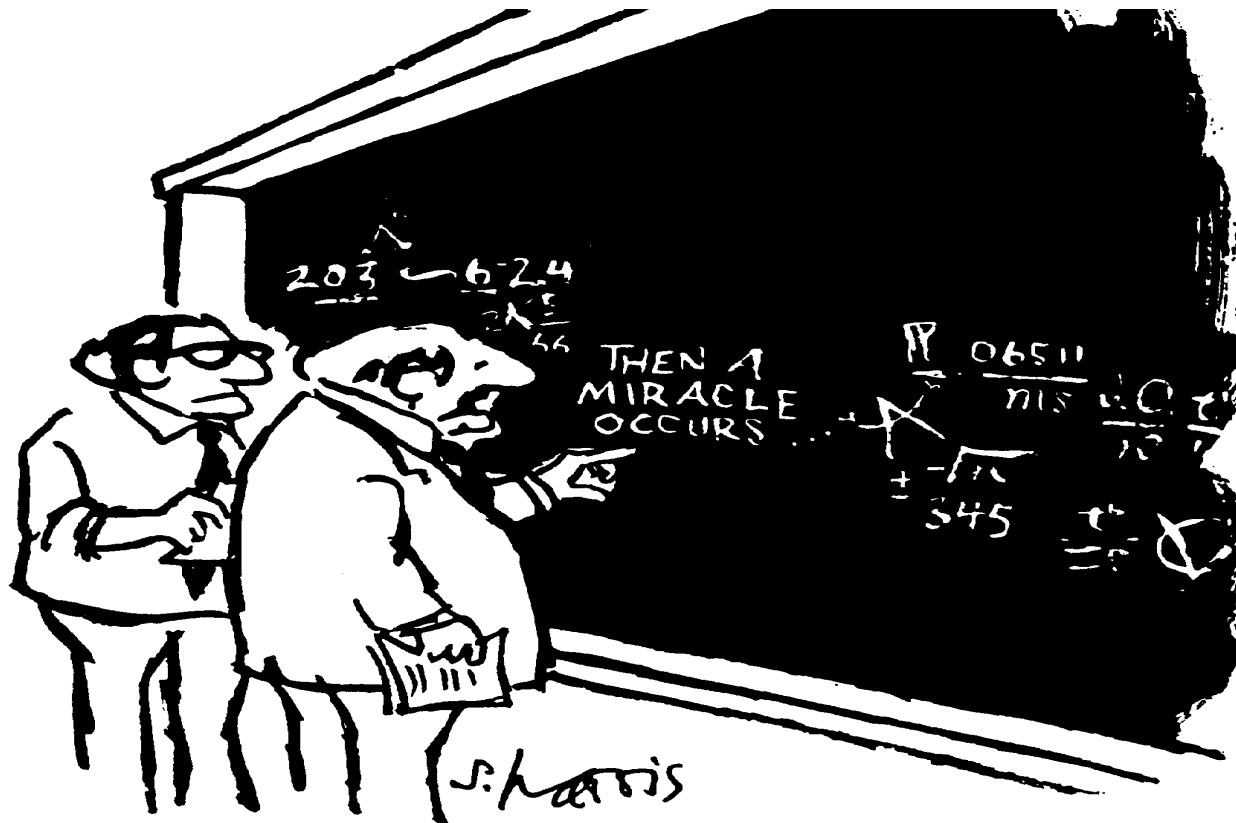
end Generic_Implementation;

A (Byzantine?) solution – 2

```
package Non_Terminal is
  procedure Method(<signature>);
private
  -- instantiate the generic to obtain multiple
  --+ instances of the same model entity
  package My_Implementation is
    new Generic_Implementation(Impl);
    -- get an instance of a model entity
    Ptr : <Implementations.>Interface_Ref :=
      My_Implementation.Get_Instance;
  end Non_Terminal;

  -- the package body implements Method by invoking it
  --+ in the designated instance at run time
  Ptr.Method(<params>);
```

Unveiling the arcane



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

© 1975 Sidney Harris — American Scientist magazine

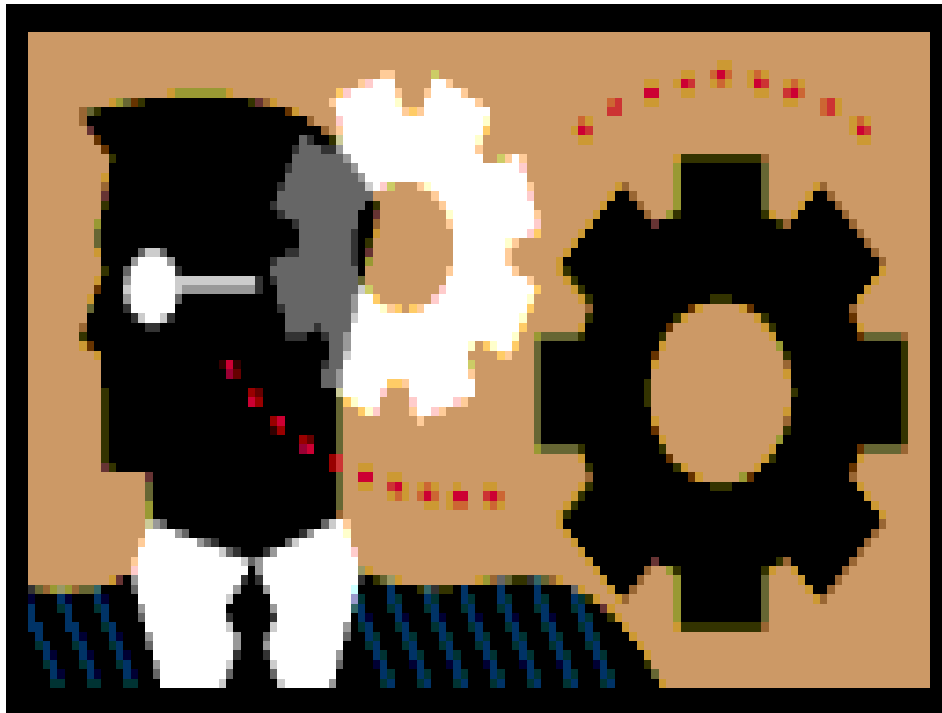
A (Byzantine?) solution – 3

- The body of the generic package defines a new type
 - Which implements the **PI**
 - Which includes functional code that reifies the invocation request and posts it to the **OBCS**
 - Hence the caller invocation maps to a *Put_Request* on the **OBCS**
 - Else (if no **THREAD**) it directly resolves into the invocation of the appropriate **OPCS**
 - Immediate vs. deferred execution
 - Function `Get_Instance` returns an instance of the hidden types as a polymorphic pointer

Byzantine wisdom

- This (admittedly convoluted) strategy wins a big bonus
 - There is a genuine OO space for modeling the functional view
 - For all **OPCS**
 - There is a rigidly controlled HRT space for modeling the concurrent view
 - For all HRT “containers” (**OBCS** and **THREAD**)
 - Rigorous rules over-impose views on one another
 - Correct-by-construction component view

Gosh, the end!



Energy left for
any questions?