

Handling Synchronization Requirements under Separation of Concerns in Model-driven Component-based development

Patricia López Martínez ¹ and Tullio Vardanega ²

¹ Computers and Real-time Group
University of Cantabria, Santander, Spain
`lopezpa@unican.es`

² Department of Mathematics
University of Padua, Padova, Italy
`tullio.vardanega@math.unipd.it`

**17th International Conference on Reliable Software Technologies
(Ada-Europe 2012)**

Stockholm, Sweden, June 11-15, 2012

Funded in part by the Spanish Government under grants grants
TIN2008-06766-C03-03 (RT-MODEL) and TIN2011-28567-C03-02 (HI-PARTES)

Table of Contents

- ❑ Our view on Separation of Concerns (SofC)
- ❑ MDE and CBSE in the development of high-integrity real-time systems
- ❑ CHES environment
- ❑ Support for synchronization requirements
 - Identification of the problem
 - Proposed solution
- ❑ Conclusions and future work

Separation of Concerns (SofC)

*“It is what I sometimes have called **"separation of concerns"**, which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of.*

*It is what I mean by **"focusing one's attention upon some aspect"**: it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant.”*

E. Dijkstra, “On the role of scientific thought”, 1974

- ❑ “Divide and conquer” strategy to **manage complexity**
- ❑ Typically applied at low-level (programming level) (aspect-oriented programming, Model-view-controller, etc.)
- ❑ More benefits if it is applied at a **higher level of abstraction**

Our view on Separation of Concerns

“The task of making a thing satisfying our needs as a single responsibility is split into two parts: stating the properties of a thing, by virtue of which it would satisfy our needs and making a thing guaranteed to have the stated properties”

E. Dijkstra, 1982

- ❑ **Specification of needs** of a system (*“stating the properties of a thing by virtue of which it would satisfy our needs”*)
 - The needs specified by the software designer not the user needs
- ❑ **Elaboration** of a **solution** guaranteed to meet the needs (*“making a thing guaranteed to have the stated properties”*)
 - Preservation guaranteed by the design and the execution environments
 - Based on adaptation and composition of consolidated well-proven solutions

SofC between the specification of needs of a system and the solution conception

- ❑ Applying an **model-driven** design process:
 - Specification of needs : Set of **properties** assigned in the **system model**
 - Solution conception => Based on **model transformations**
 - The assigned properties are used to adapt well-proven solutions that preserve the desired behaviour at run-time

- ❑ Applying a **component-based** development process:
 - System model designed as an **assembly** of **reusable** components
 - Properties representing the system needs are assigned on component instances

 - The underlying **component framework** must guarantee preservation of properties at the solution level


High-integrity real-time systems

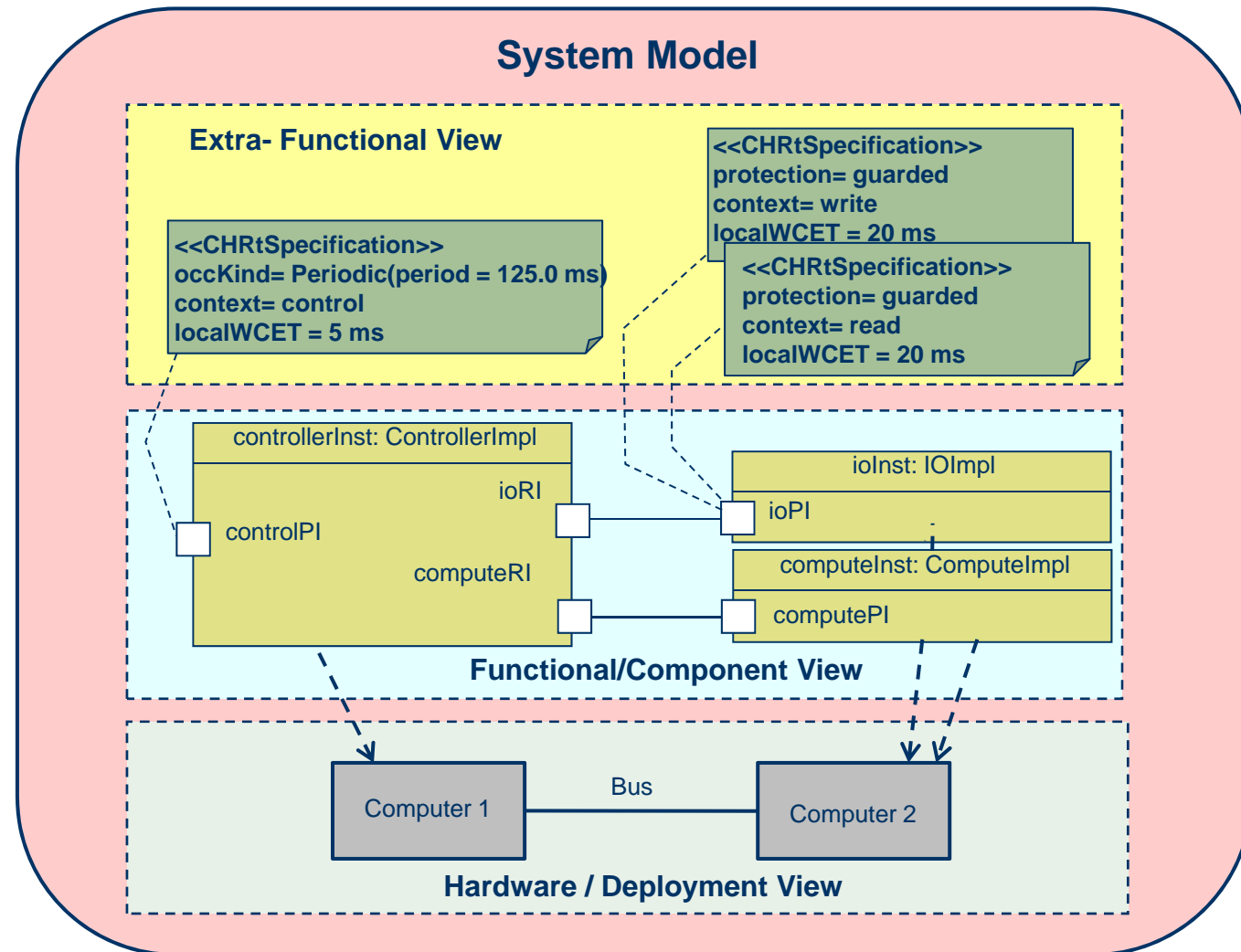
- ❑ Importance of non-functional requirements
 - Our focus is on guaranteeing the fulfillment of **end-to-end timing requirements**
- ❑ Specification of needs aimed at configuring the aspects that influence the timing behaviour of the system:
 - Concurrency, synchronization, scheduling, communication
- ❑ The application must exhibit predictable timing behaviour
 - **Schedulability analysis** can be applied to ascertain the timing requirements are met
- ❑ **MDE + CBSE + SofC** can increase the quality and the economy of the development of high-integrity real-time systems

CHESS: An MDE Component-based technology for high-integrity systems

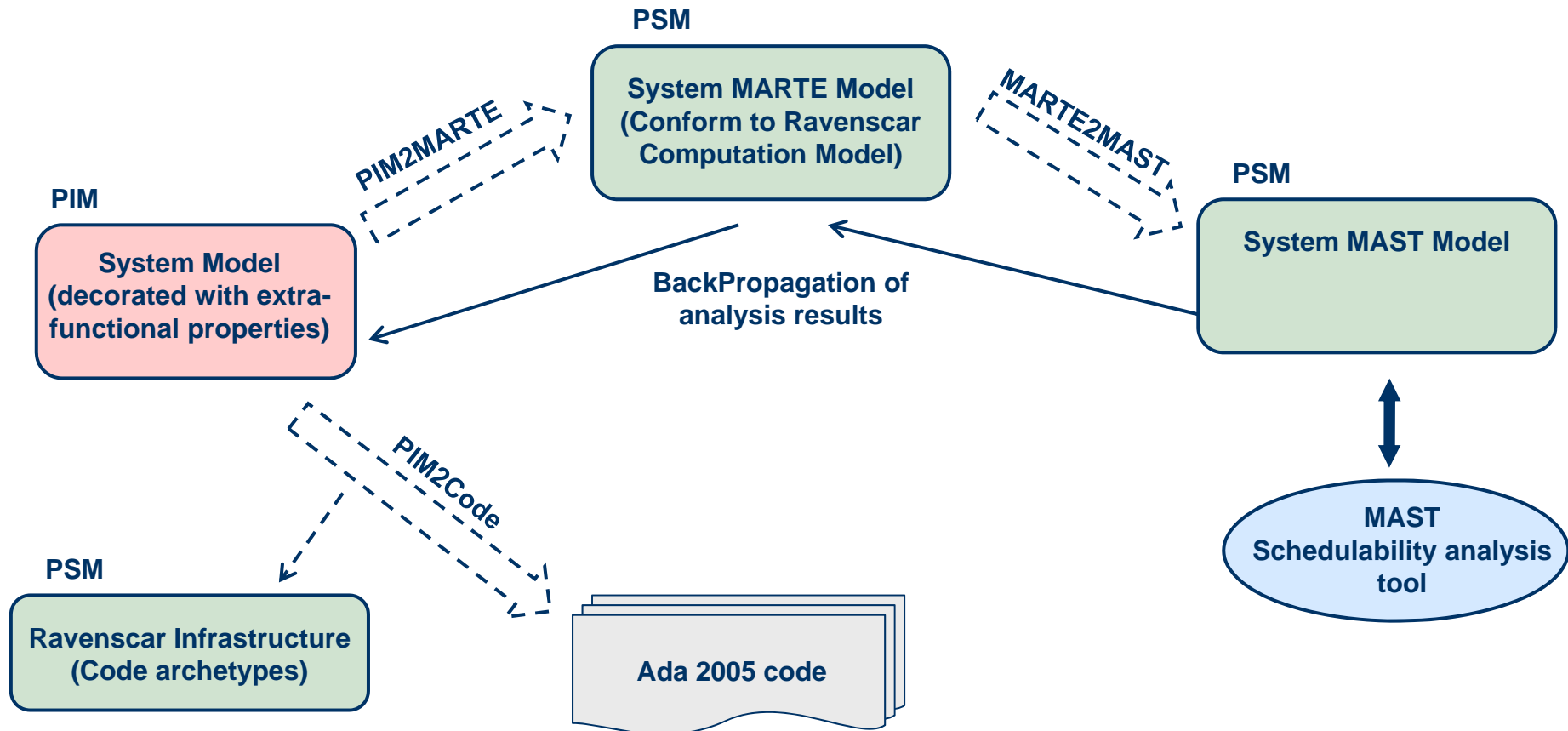
- Result of former European projects **ASSERT** and **CHESS**
- Relying on a well-defined **software** reference **architecture**, with four constituents:
 - Component model:
 - Sequential code **components**, **containers** and **connectors**.
 - Guarantees SofC at the solution level
 - Computational model
 - **Ravenscar** computational model (RCM)
 - Programming model
 - Ravenscar profile of **Ada**
 - Using code archetypes presented in “*Ada Ravenscar Code Archetypes for Component-based Development*” (Panunzio & Vardanega, AE2012)
 - Execution platform
 - Compliant with high-integrity systems

Design of a real-time system in CHESS


 Software Integrator
 (Software designer)



Transformations chain in CHES: from PIM to code



SofC applied to handle synchronization requirements

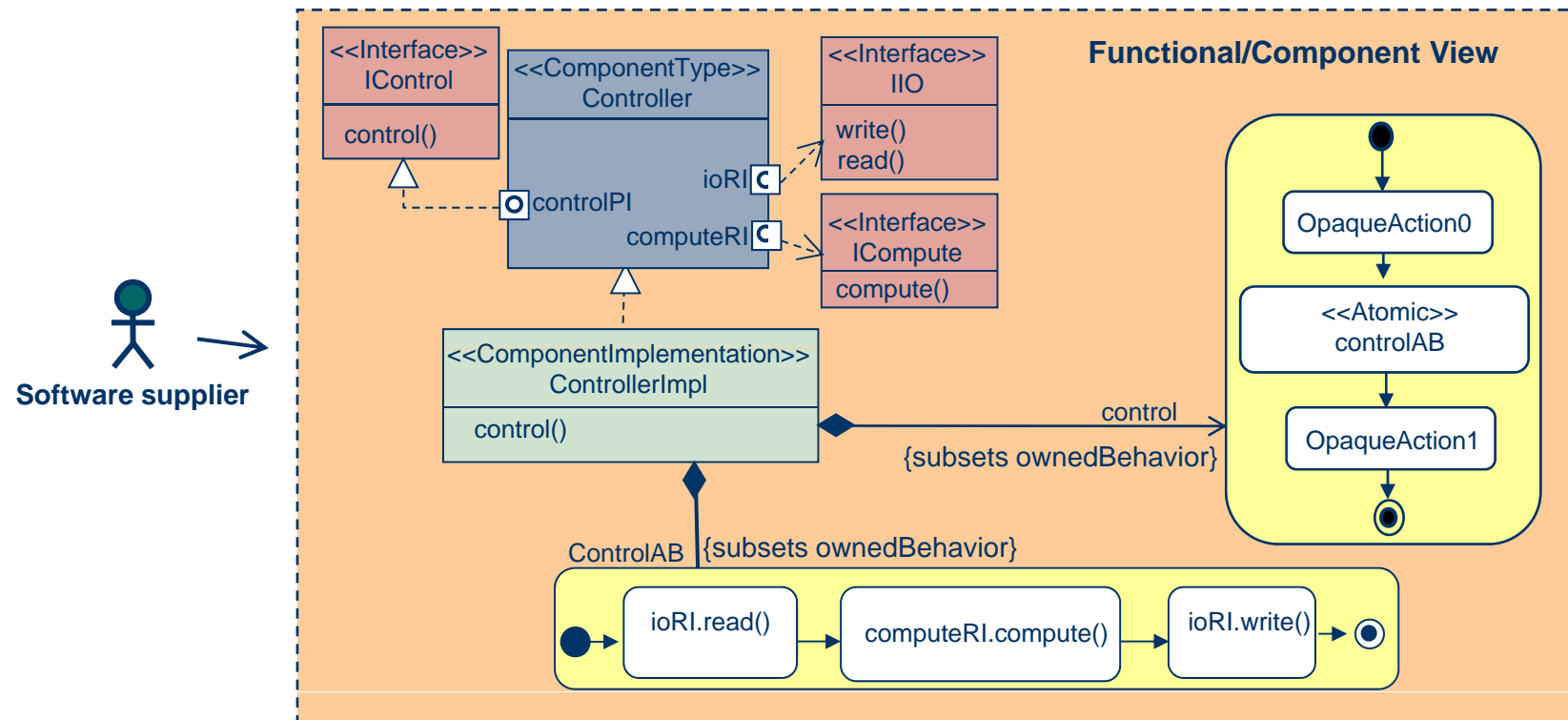
- Applying SofC to **synchronization management** requires:
 - Formulating synchronization needs => By means of “decorations” on the system model
 - Support for synchronization at the solution level => Model transformations

- Two kinds of synchronization needs are distinguished:
 - Oriented to guarantee the **software integrator assumptions**
 - **Per-operation** => Mutually exclusive execution of single provided operations
 - Already supported in CHESS
 - Defining the “guarded” nature of an operation
 - **Per-state** => Mutually exclusive access to the internal state of the component
 - To be added
 - Prevents low-level data races

 - Oriented to guarantee the **software supplier (client-side) assumptions**
 - **Per-block-of-operations** => Mutually exclusive execution of internal sequences of operations
 - Addresses logical consistency of code
 - Avoids the presence of **high-level data races**

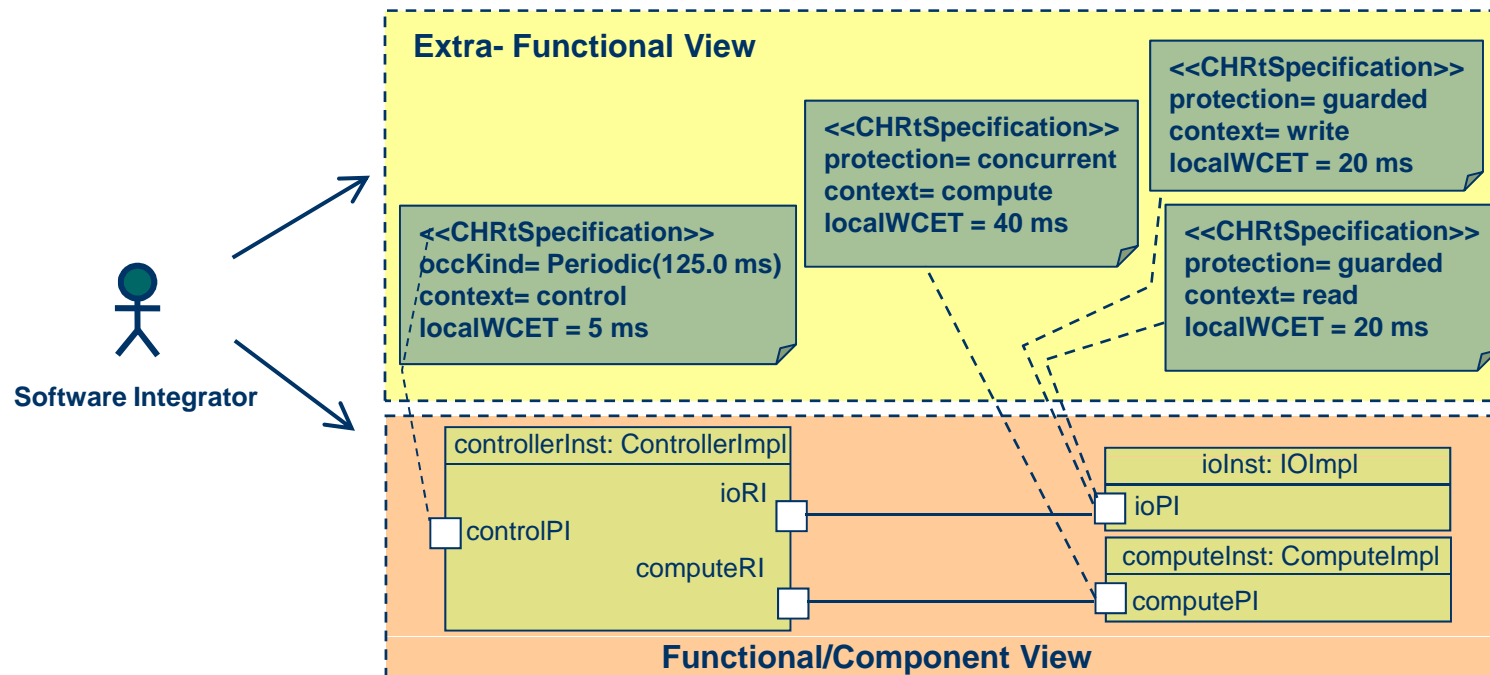
Formulating client-side assumptions

- ❑ Client-side assumptions about atomic blocks cannot be captured by only decoration of component ports
- ❑ It is responsibility of software suppliers to identify them
- ❑ It will be responsibility of the design environment guaranteeing their atomicity



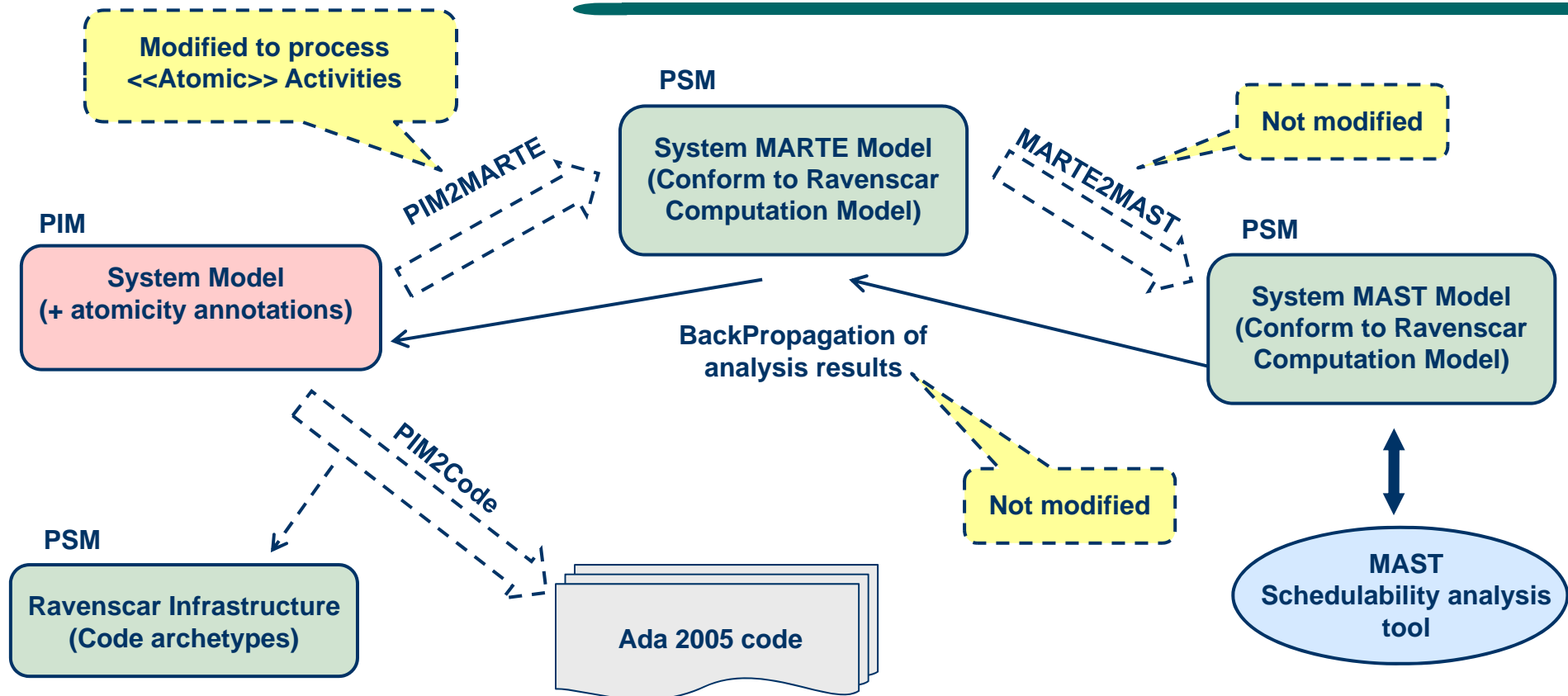
Support for the atomicity assumptions

- ❑ The software integrator designs and annotates the system model in the standard way
 - Not aware of client-side assumptions

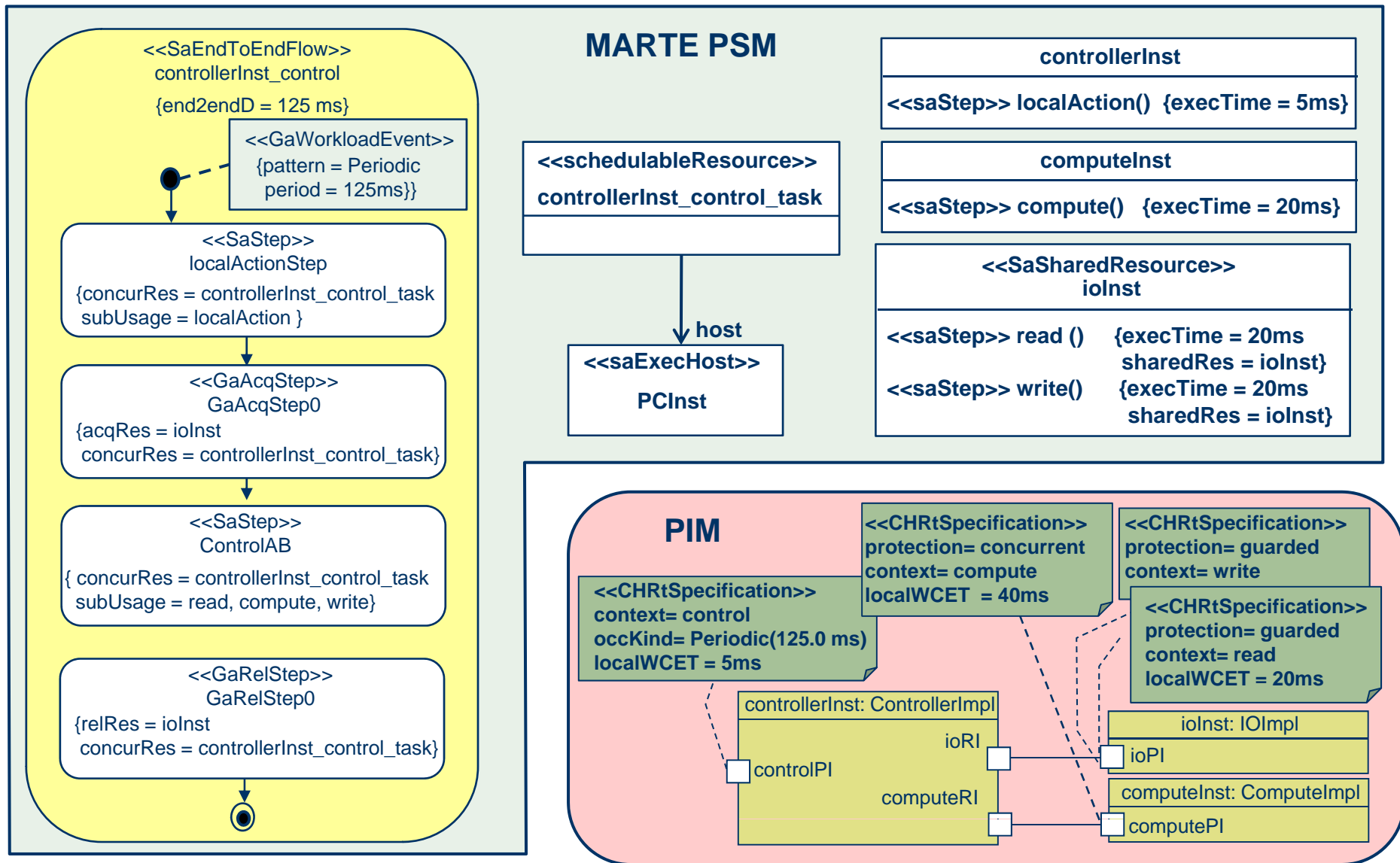


- ❑ The environment is in charge of guaranteeing both the “external” annotations and the client-side assumptions

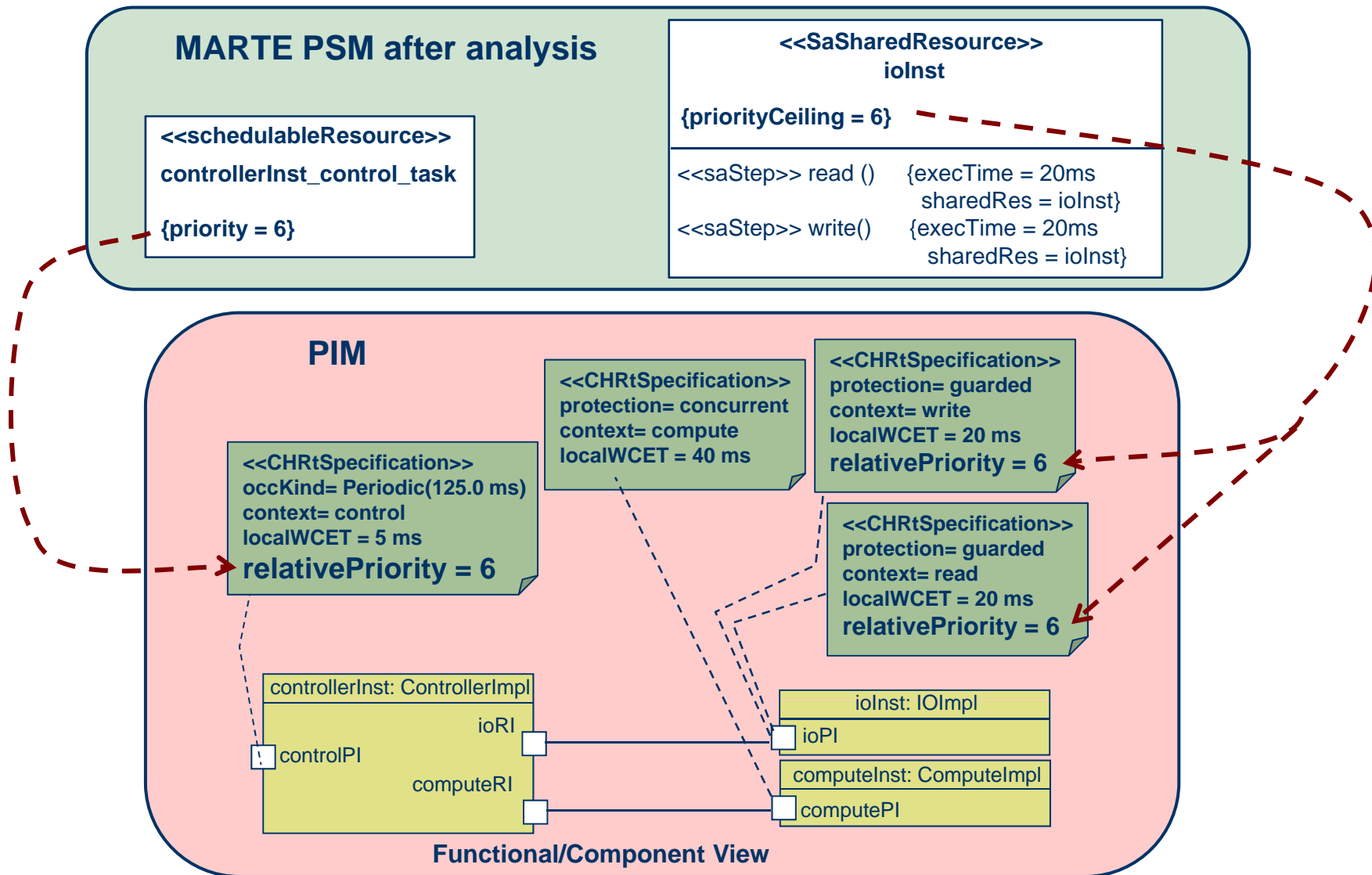
CHESS transformation chain with atomicity support



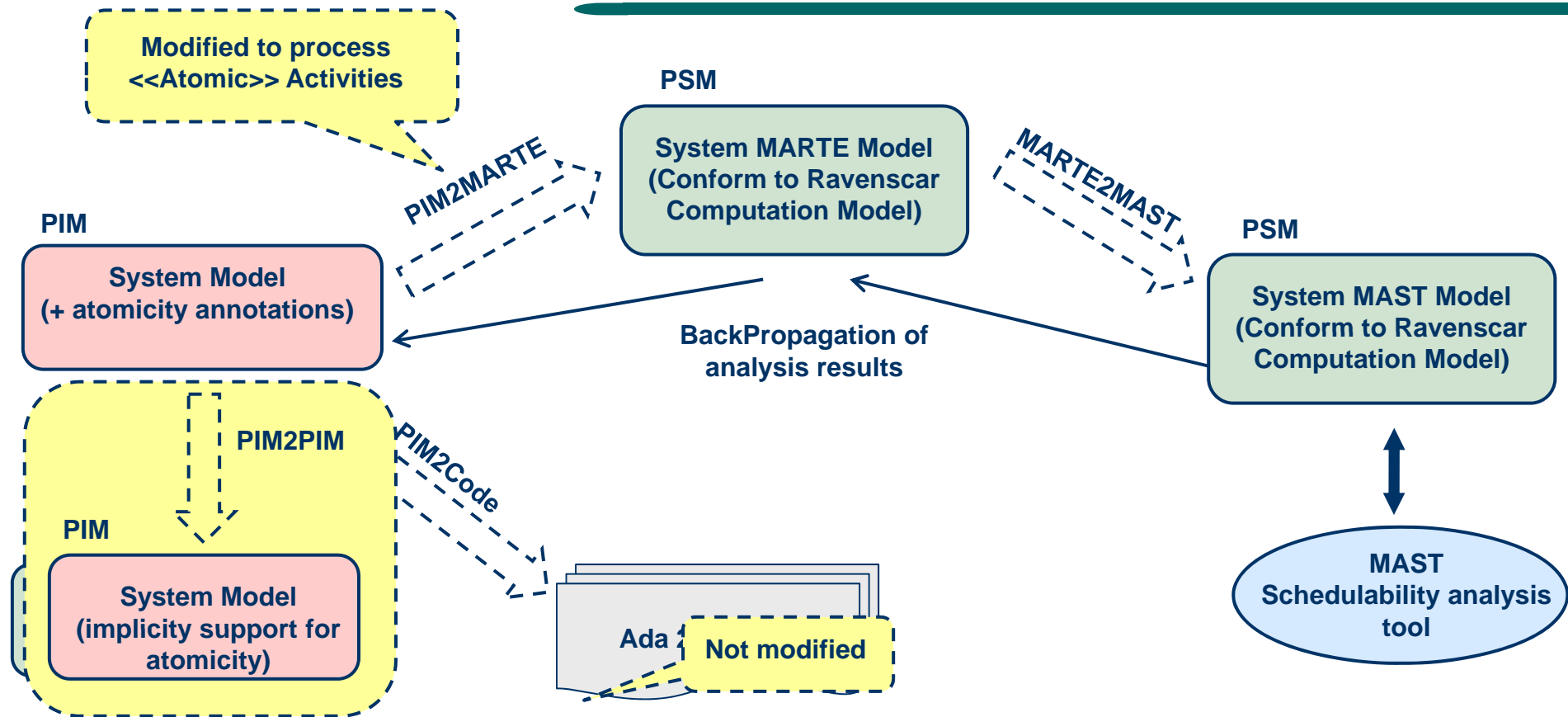
PIM2MARTE Transformation



Back Propagation of analysis results

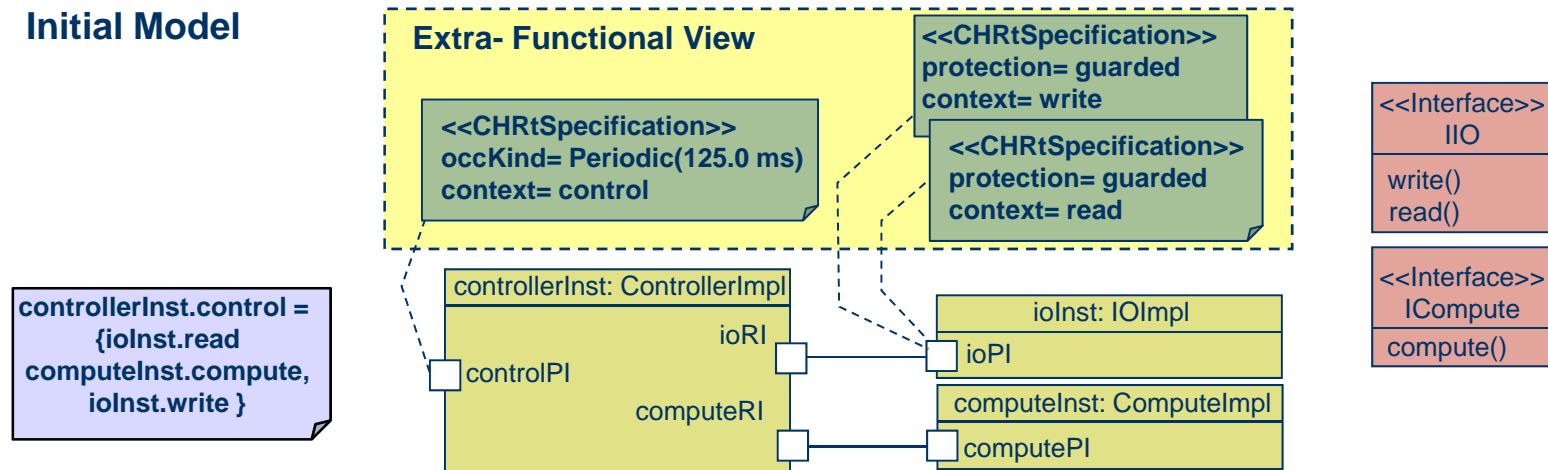


CHESS transformation chain with atomicity support

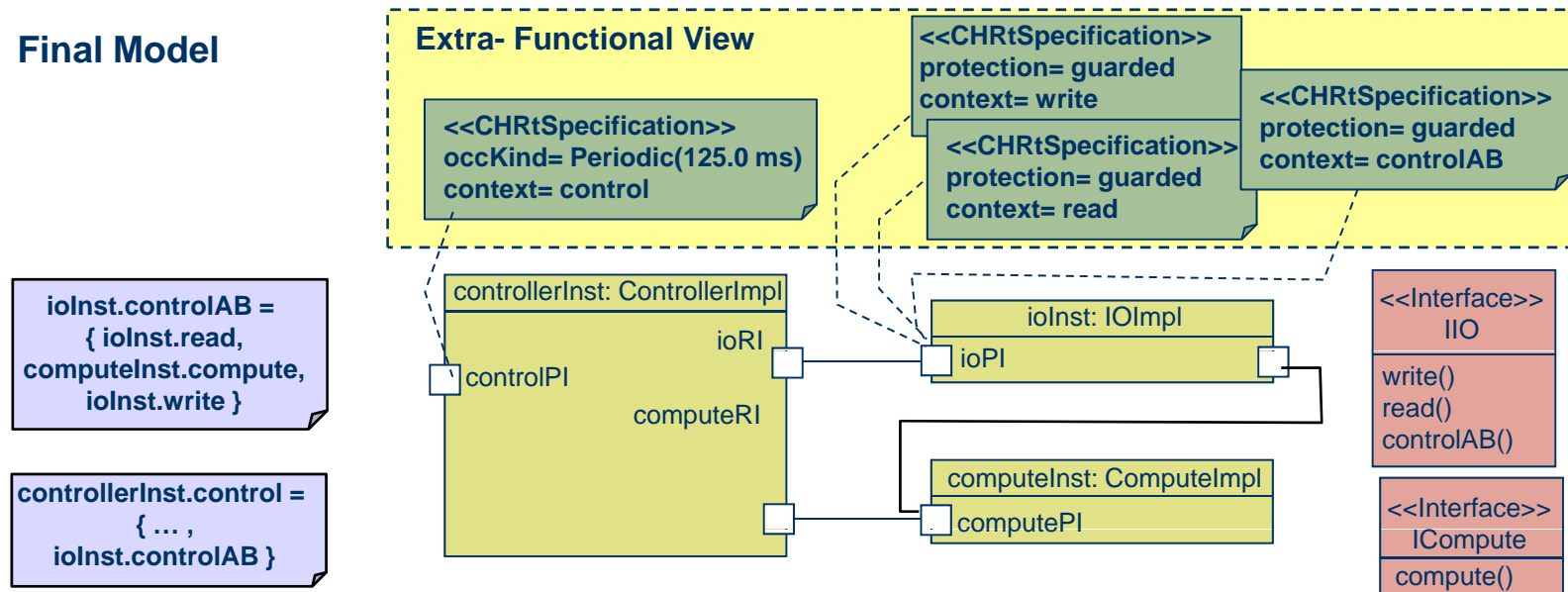


PIM2PIM: Restructuring instances architecture

Initial Model



Final Model



- ❑ Dijkstra's view on SofC to improve the component-based development process of high-integrity real-time systems
- ❑ SofC between system needs and system solution
- ❑ Approach applied to handle synchronization requirements
 - Support for client-side assumptions avoiding high-level data races
 - Specially interesting in multi-core architectures
- ❑ As future work,
 - Proposal of a less pessimistic approach
 - Better characterization of the trade-off between schedulability loss and atomicity guarantee