Towards the Definition of a Pattern Sequence for RT Applications using a MDE Approach

Universidad Politécnica de Cartagena

Juan Ángel Pastor, **Diego Alonso**,
Pedro Sánchez, Bárbara Álvarez

DSIE
DIVISIÓN DE SISTEMAS E INGENIERÍA ELECTRÓNICA

Table of Contents

1.- Context of the Problem

- Component-Based (CB) applications with Real-Time requirements in Robotics → *frameworks*
- Main drawback: despite being CB in their conception, designers must develop, integrate and connect components using Object-Oriented (OO) technology
  - CB designs require more/different abstractions and tool support than OO technology can offer
  - Normally framework design ignores real-time issues

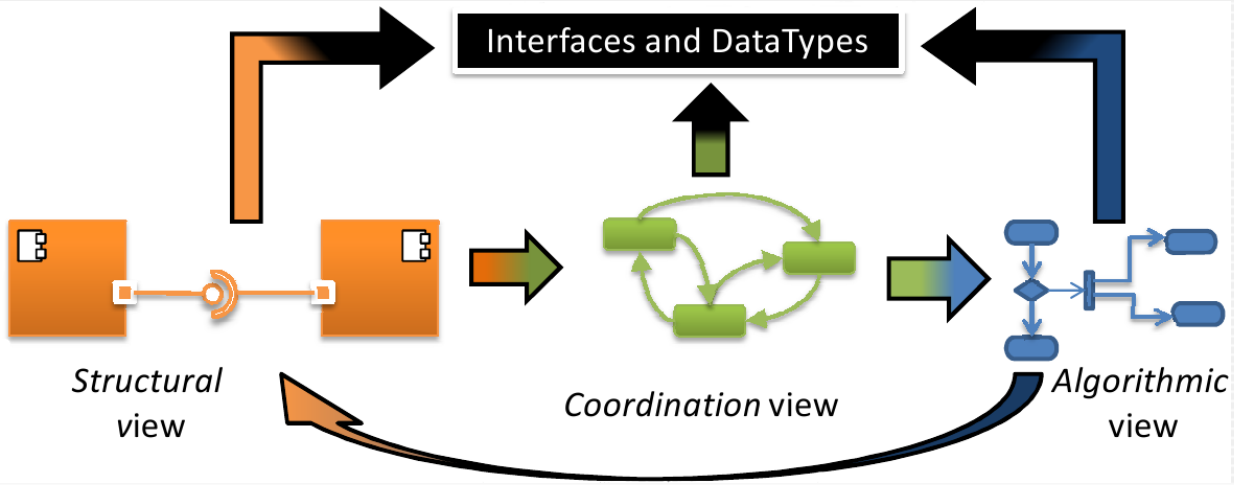
2.- Our solution, From 20.000 feet

- In our opinion, it is needed a new approach for CB software development that:
  1. Considers components as architectural units
  2. Enables components to be truly reusable among frameworks, by separating their design from the implementation details
  3. Considers domain-specific requirements (time in this case)
- Model-Driven Software Engineering can help:
  1. Providing formal languages for modeling CB applications, checking their correctness, performing V&V actions, etc.
  2. Providing model transformations for automatically generating code from input models

2.- Modeling CB applications

- In general, any language (UML, SysML, AADL, etc.) can be used as long as it provides both structural and behavioural modelling

V³CMM



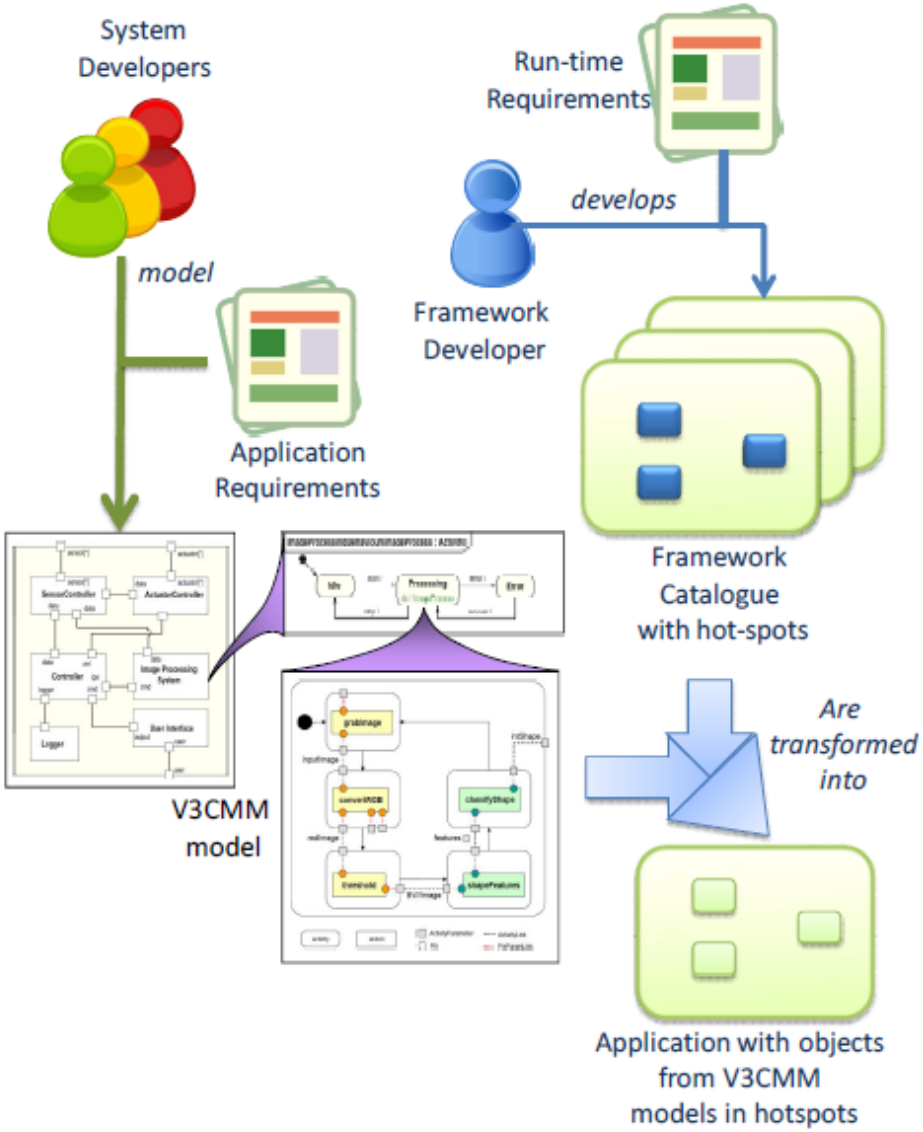
- Simplicity and economy of concepts: just 3 views
- Both view and component reuse
- Controlled semantics
- Open for extension

3.- A Close Look at the Proposed Problem

- The behavioural views (state-charts and activity diagrams) abstract designers away from run-time issues (e.g. number of tasks, concurrency model, etc.)

- These details must be realised in executable code in a way that:
  1. Reflects the behaviour of the original CB model
  2. Is organised in a set of tasks compliant with the application-specific timing requirements

3.- A Close Look at the Proposed Solution

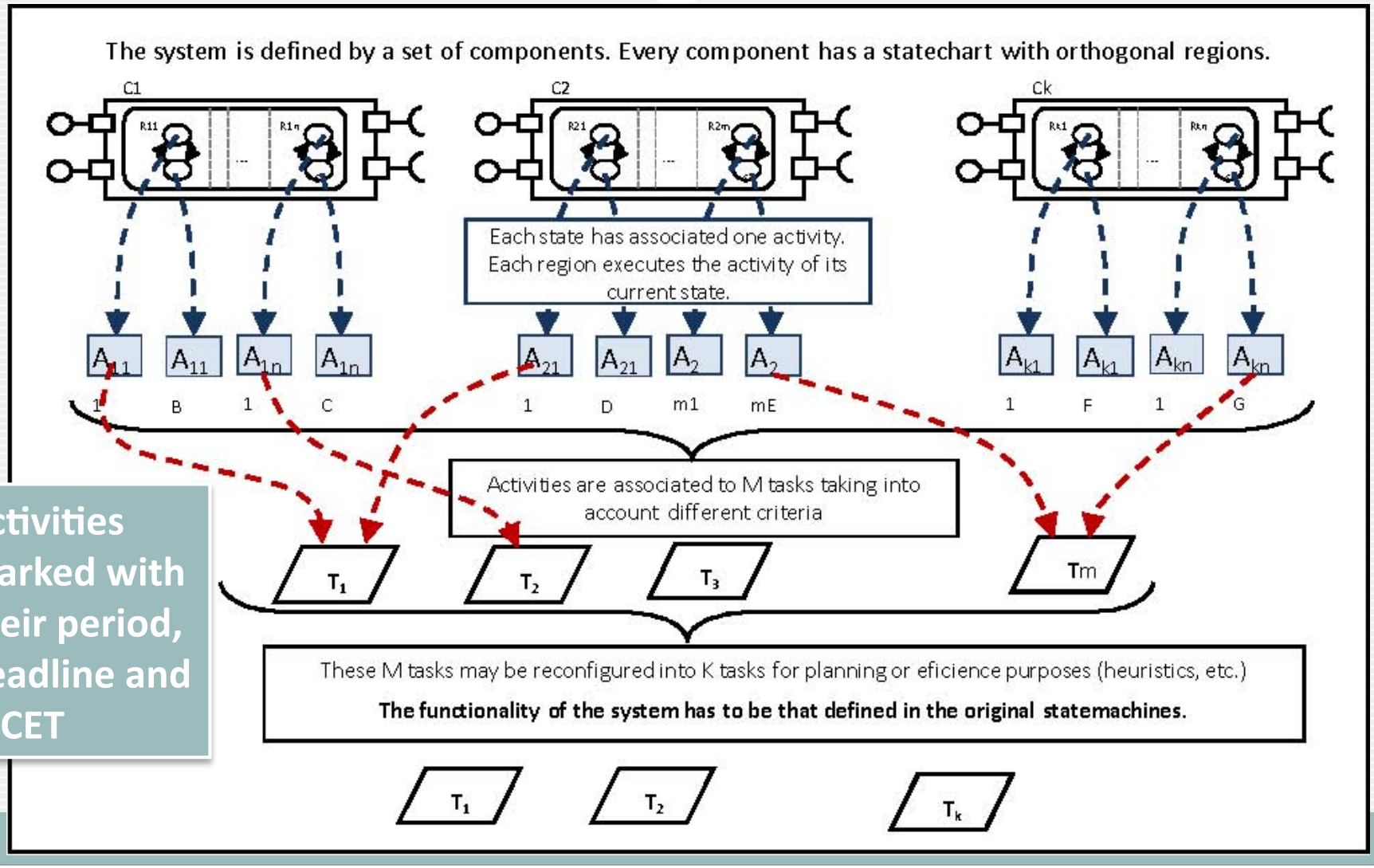


- Code structured as follows:
  - **CS1**: provides a run-time support compliant with the requirements
  - **CS2**: provides an interpretation of CB concepts
  - **CS3**: provides application code

3.- Code Sets of the Proposed Solution

- These three code sets are arranged in a way that:
  - CS1 and CS2 constitute a framework where CS3 must be integrated in order to obtain the final application
  - CS2 provides the framework '*hot-spots*' and minimises the coupling between CS3 and CS1
- As long as CS2 remains the same,
  - CS1 can be reused with different CS3
  - A suitable CS1 can be selected for the same CS3, depending on the application domain requirements
- CS1 and CS2 have been designed and implemented manually, while CS3 is meant to be automatically derived from input CB models

3.- A Close Look at the Proposed Solution



Activities marked with their period, deadline and WCET

3.- Some Requirements …

- The solution must not force a 1-to-1 relationship between components and execution tasks → flexible schemes for allocating activities to tasks, since activity allocation can be driven by
  - Real-Time requirements
  - Scheduling algorithms
  - Allocation heuristics
  - Platform constraints
  - etc.
- … which can greatly vary from application to application

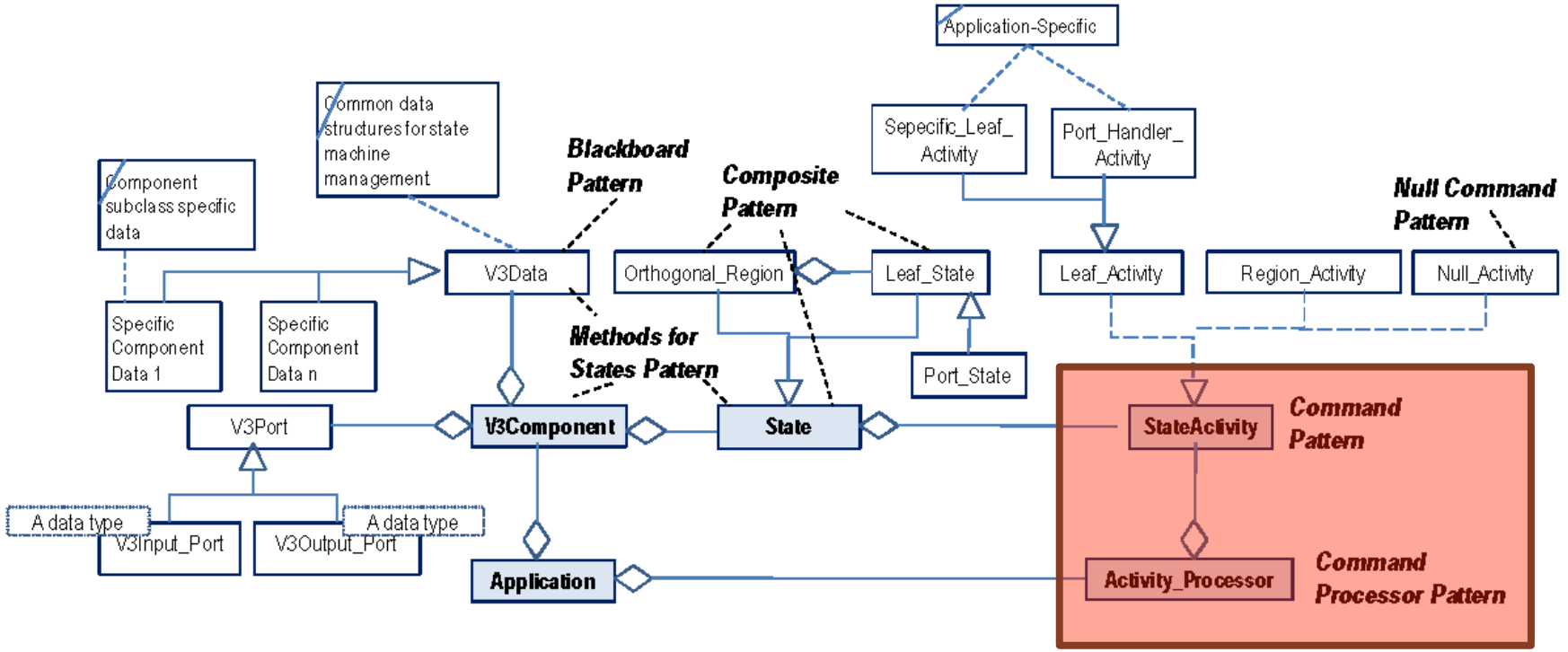
4.- Pattern Sequence: freely allocate activities to tasks

1. **COMMAND PROCESSOR** pattern: provides a task to separate service requests from their execution
2. Required by the previous pattern → **COMMAND** pattern for modelling activities
3. Derived problem: concurrent access to component's internal data → protected **BLACKBOARD** pattern

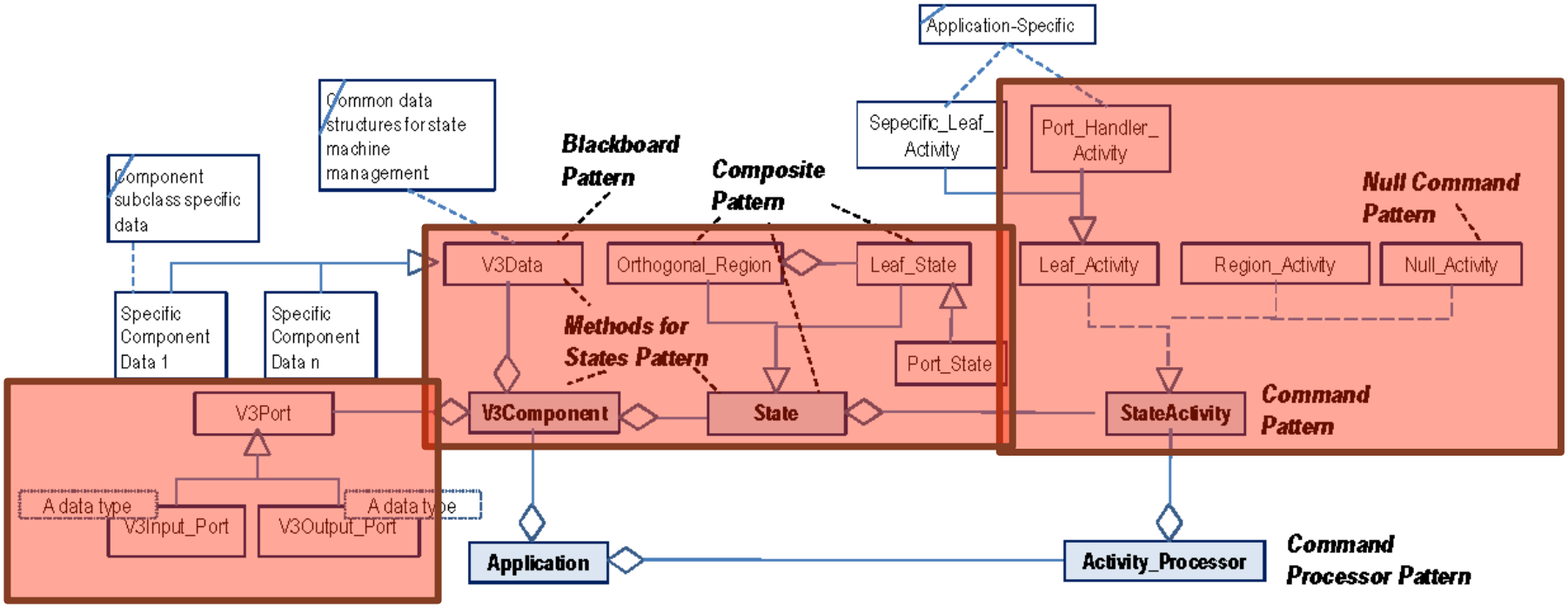
4.- Pattern Sequence: state-chart implementation

1. Structure of the state-chart → **COMPOSITE** pattern
2. Behaviour of the state-chart → **METHODS FOR STATE** pattern
   - Modification of the **STATE** pattern when there are many states sharing behaviour and data. Reduces space and overhead
3. Specific activities for considering and explicitly integrating component ports and state-chart management:
   - Region activity: manages regions (active state, transitions, etc.)
   - Port handling activity: manages component communication through their ports
   - Null activity
4. ... provides regularity and flexibility

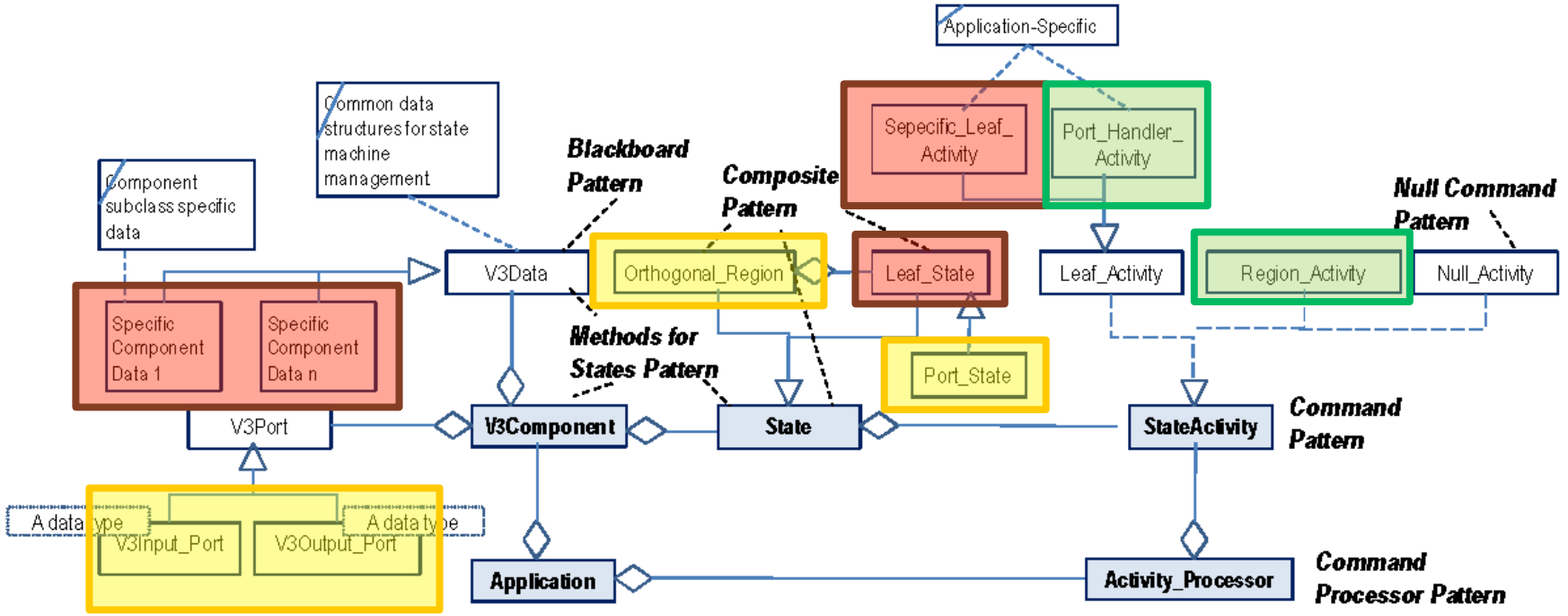
4.- Solution Implementation: CS1



4.- Solution Implementation: CS2



4.- Solution Implementation: CS3



- Other patterns not shown: **OBSERVER, PROXY, STRATEGY, TEMPLATE METHOD, COPIED VALUE**, etc.
- 18 patterns in total

4.- Implementation of Command Processor (I)

```
generic
package Common.Activity_Processor is
  procedure Set_Priority (Priority : System.Any_Priority);
  procedure Set_Period (Period: Time_Span);
  procedure Start();
  procedure Add_Activity (Act : access I_State_Activity'Class);
end Common.Activity_Processor;
```

4.- Implementation of Command Processor (II)

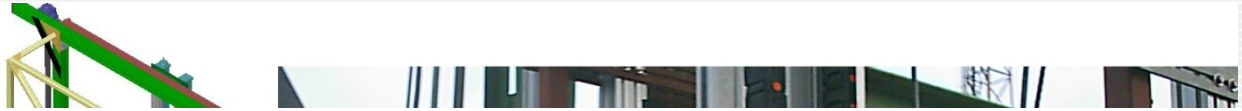
```
task body Worker is
     Next_Exec  : Time := Clock;
     Iterator       : P_Dll.Cursor;
     Element      : State_Activity_All;
begin
while Continue loop
          delay until Next_Exec;
          Next_Exec := Next_Exec + Period;
          Iterator := Activity_List.First;
          while (P_Dll.Has_Element (Iterator)) loop
               Element := P_Dll.Element (Iterator);
               Element.Execute_Tick;
               P_Dll.Next (Iterator);
          end loop;
     end loop;
end Worker;
```

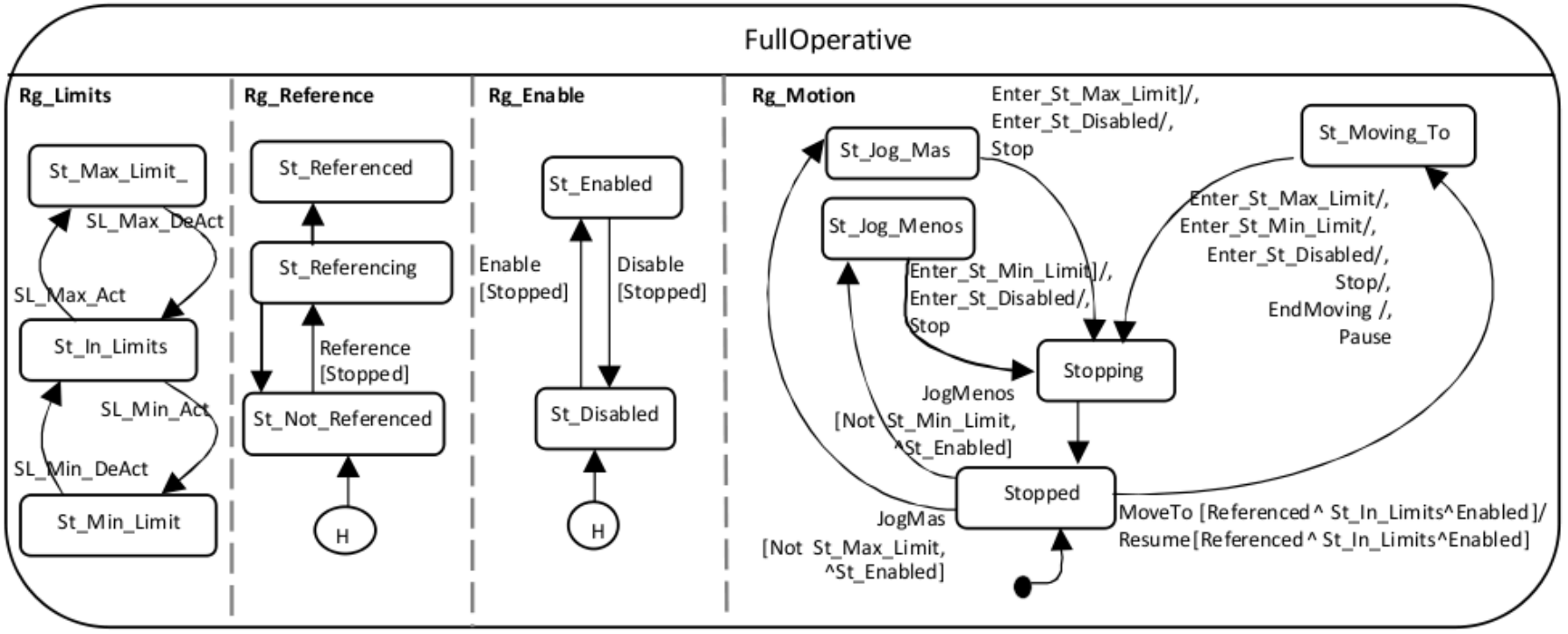
4.- Some Notes About COMMAND PROCESSORS

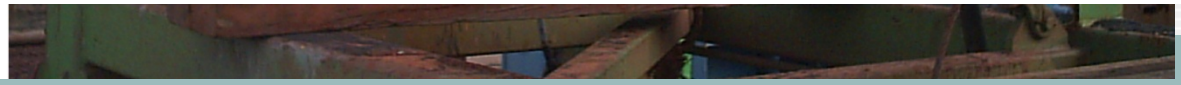
- **COMMAND PROCESSOR** is a very flexible pattern that has been constrained
- Not allowed to spawn new tasks → set of tasks is known at design time
- Activities cannot be added/removed at run-time → task load is known at design time
- Periods and priorities cannot be changed at run-time → fixed priority with pre-emption schedulers

5.- Example of Framework Usage

- Sample state-chart of a motor controller of a Cartesian robot







5.- Framework Usage: Added Activities and Regions

- After creating **CS3** …
- Region handling activity to manage each region:
  - Periodic activity with period ≤ the minimum period of the region activities
- Additional region/s with a port handling activity to manage component ports:
  - Periodic activity with period ≤ the minimum period of all the component activities
- The framework already provides sample ones (**CS2**) that can be reused or new ones can be created by developers

5.- Allocation of Activities to Tasks

- Currently, the granularity is region handling activity to a **COMMAND PROCESSOR**
- Developers are free to choose any allocation criteria, considering also the additional regions described before
  - Maximum concurrency: 1 task for each region
  - Minimum concurrency: 1 task for the whole application

5.- Schedulability Issues

- Region handling activities:
  - Execute both periodic and sporadic activities, making no distinction → heterogeneous tasks
  - Their periods are set to the lowest period of their region → worst case scenario, but only one activity is executed in each region
- Port handling regions and activities:
  - As many as needed, depending on the timing characteristics of the activities triggered by the command → provides a finer control
  - But again, their periods are set to the lowest period of their region → worst case scenario

6.- Conclusions

- The framework provides a solution for distributing component activities across tasks, is fully operative and can be used “as-is”
- It provides an OO interpretation of CB concepts
- Has been developed as a pattern sequence
- The structure of the solution (i.e. the code sets) facilitates
  - The development of model transformations
  - The development of other frameworks, for applications with different requirements
- Nevertheless, it is just a first step

6.- Future Work

- Correct current limitations of the framework:
  - Increase the granularity of concurrency → leaf states
  - Deal with sporadic activities → probably in an specialised sporadic **COMMAND PROCESSOR**
  - Testing and adding heuristics for activities allocation and task grouping
  - Perform schedulability analysis
  - Component distribution using middleware
  - Adopt the *Ravenscar* profile, as it suits many requirements
- Develop other frameworks with different requirements
- Extend the modelling language (V^3CMM) in order to incorporate timing requirements (timed automata or petri nets)
- Generate **CS3** through a model transformation, since it is the main design driver behind the framework

Towards the Definition of a Pattern Sequence for RT Applications using a MDE Approach

Universidad Politécnica de Cartagena

Juan Ángel Pastor, **Diego Alonso**,
Pedro Sánchez, Bárbara Álvarez

DSIE
DIVISIÓN DE SISTEMAS E INGENIERÍA ELECTRÓNICA

5.- A Sample Execution Scenario