Worst-Case Execution Time Analysis from UML-based RT/E Applications

Chokri Mraidha, Sébastien Gérard, François Terrier, David Lugato

Chokri.Mraidha@cea.fr

Introduction

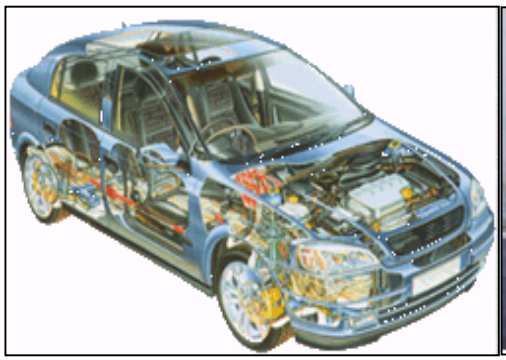
- PhD student in the Software for Process Safety Laboratory in the French Atomic Energy Agency (CEA)
  - ✓ Director: Judith Benzakki
  - ✓ Advisor: Sébastien Gérard
- PhD topic
  - ✓ Optimized software synthesis in model driven development of real-time embedded systems
  - ✓ Quantitative analysis of models
    - → Execution time

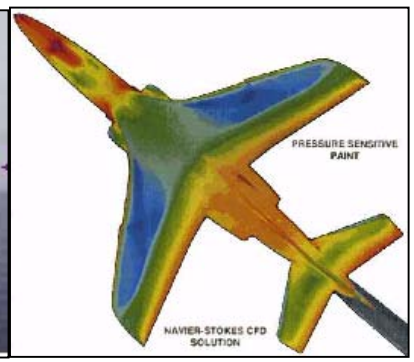
Plan of the presentation

- Context
- Related works on WCET analysis methods
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- Conclusions and perspectives

Context

- **Domain:** Distributed Real-time and Embedded Systems (DRES)

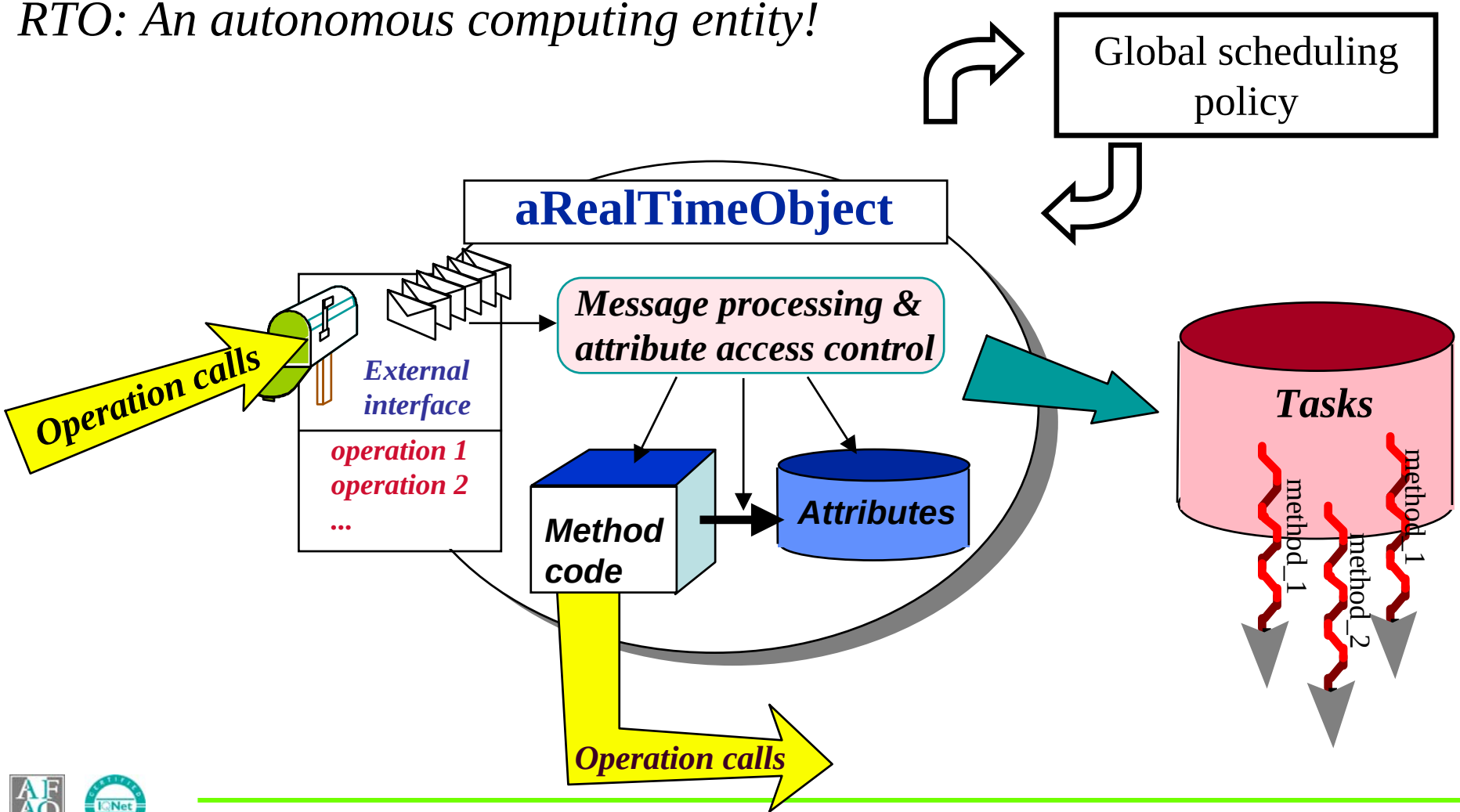




- **Accord/UML**
  - ✓ An MDD approach for DRES
    - → Based on the UML
    - → RTE profile specializing the SPT profile
    - → Set of modeling rules for RT applications development
  - ✓ Accord/UML tool support
    - → Implementation of the Accord/UML profile
    - → Accord framework
      - » Kernel and virtual machine (UNIX, Linux, VxWorks)

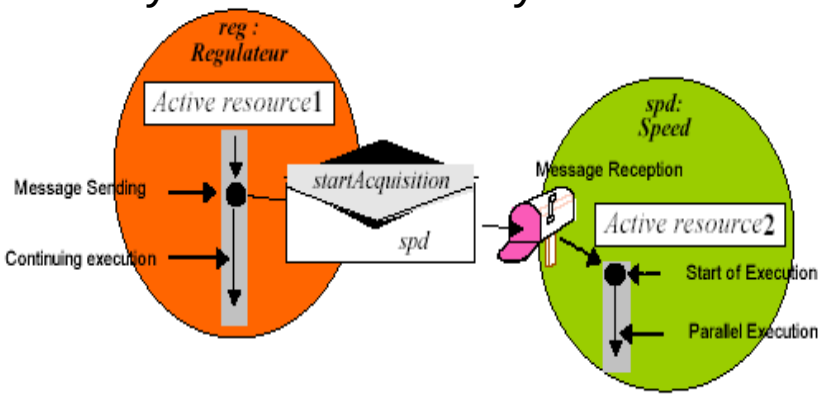
Highlight: an execution model based on RTOs

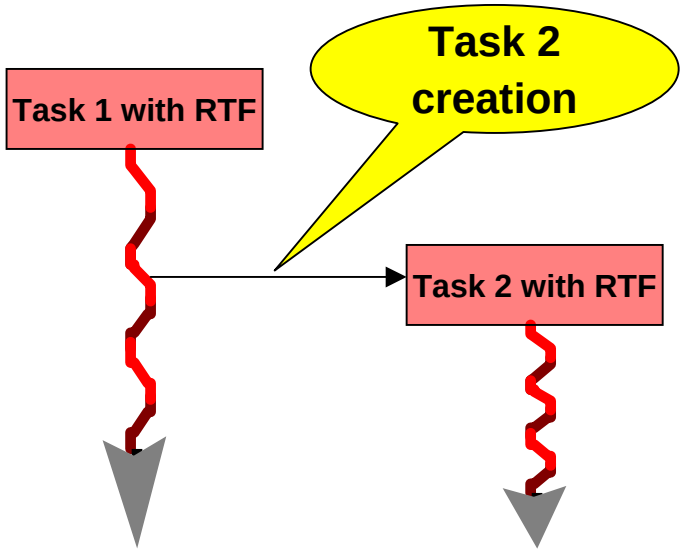
RTO: An autonomous computing entity!



Communication mechanisms and task model

- **Two main schemes of communication**
  - Synchronous or asynchronous



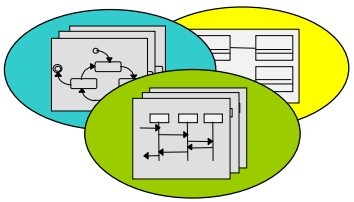


- **Accord/UML tasking model underlying Real-Time Object concept**
  - General computational model:
    - ✓ One task per operation call
    - ✓ Real-time features
      - → TimeRef, Deadline, ready time, period and period number
  - Scheduling policy
    - ✓ EDF

Schedulability analysis on UML models

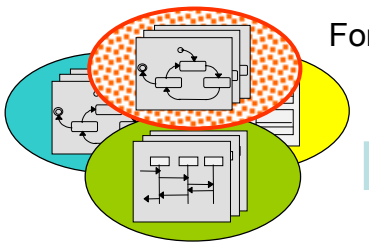
➔ *Task model calculus & schedulability analysis*

If not schedulable system:
- Model correction
- Real-time constraints adaptation



Accord/UML-models with RT specification

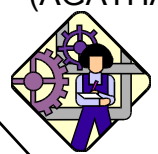
1



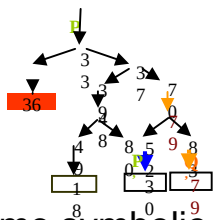
Adaptated application models
+ all required RT specification for schedulability analysis

2

Formal behavioral analyser tool (AGATHA)



3



Real-time symbolic execution tree of the application

4

Mapping between UML models and AGATHA' formal language with clear modelling of the scheduling policy

➔ *Need the Worst Case Execution Time of each operation (~ task)*

Plan of the presentation

- Context
- **Related works on WCET analysis methods**
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- Conclusions and perspectives

Analysis-based methods (static)

- No execution of the program
- A two-phases process
  - ✓ High-level analysis: computes all execution paths
  - ✓ Low-level analysis: estimates execution time of these paths
- Requires
  - ✓ Task specification
  - ✓ Timing model of the microprocessor
- Result
  - ✓ An upper bound of the WCET: safe
- Main issues
  - ✓ Avoid overestimation of the WCET
  - ✓ Requires an elaborated timing model of the microprocessor
  - ✓ Not easily retargetable (timing model)

Measurement-based methods (dynamic)

- Measurement of the execution time at runtime
  - ✓ Find input values having the longest execution time
- A two-phases process
  - ✓ Find input values covering all execution paths
  - ✓ Measure execution time for every input value at runtime
- Requires
  - ✓ A set of input values
  - ✓ A binary of the task to measure its execution time
- Result
  - ✓ A lower-bound of the WCET
- Main issues
  - ✓ Find all input values for measurement
    - → Impossible for infinite domains

Outcome

	Analysis-based approaches	Measurement-based approaches
++	High-level analysis: execution path computing ➔exhaustive	Execution time measurement for given input values ➔more precise results than analysis ones
- -	Low-level analysis: execution time estimation ➔overestimations	Find input values for measurement ➔how to find input values covering all execution paths?

Plan of the presentation

- Context
- Related works on WCET analysis methods
- **A hybrid method for WCET estimation**
- An analysis-based method for WCET estimation
- Conclusions and perspectives

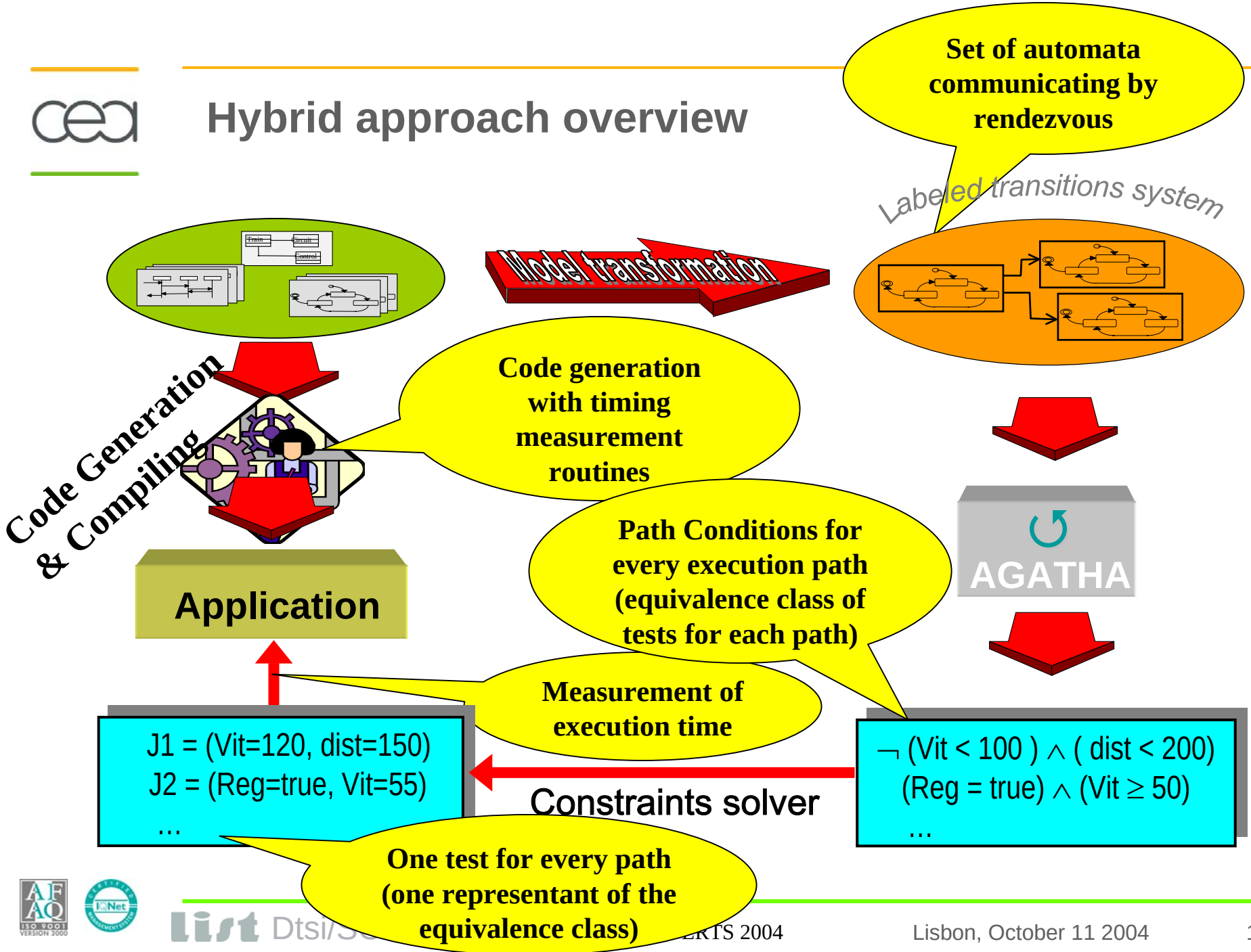
A hybrid method for WCET estimation

- **A two-phases approach**

  - A static analysis phase
    - ✓ High-level analysis of static analysis methods
      - → Automatic test cases generation
        - » Tests covering all possible execution paths
        - » Analysis techniques: model checking, constraint-based techniques, symbolic execution…

  - A measurement-based phase
    - ✓ Runtime measurement
      - → Measurement using tests of the previous phase

Hybrid approach overview



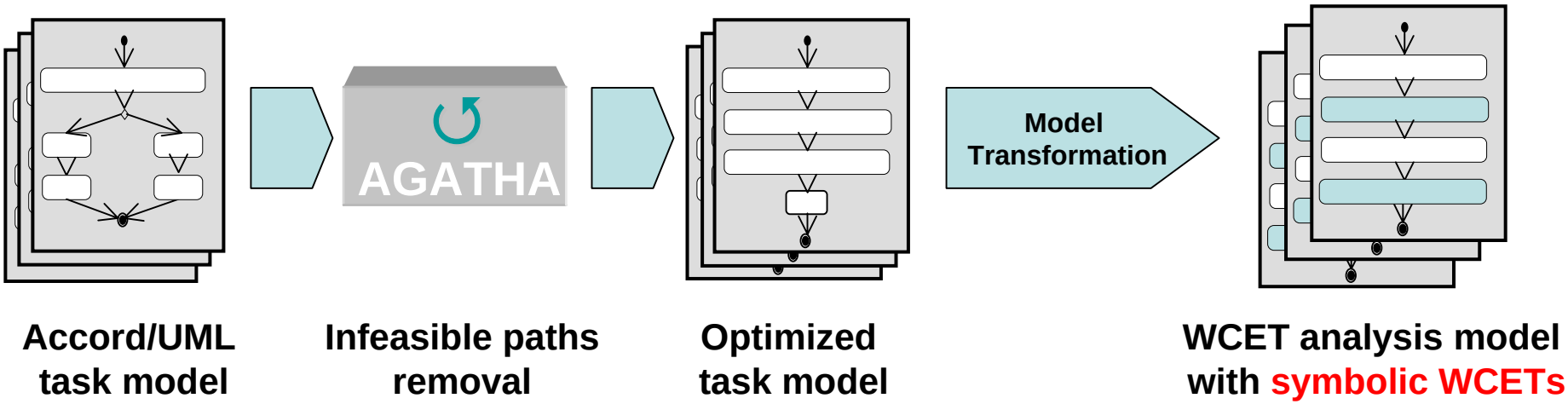
Advantages and limitations of the approach

- **Advantages**
  - Automatic process
    - ✓ No code or model annotation required from the user
    - ✓ Integrated into a development toolchain
  - No timing model required
    - ✓ Easily retargetable to new sw/hw components
  - Good scalability with symbolic execution
- **Limitations**
  - First order logic models (Presburger arithmetic)
    - ✓ Limitation due to the Agatha tool
    - ✓ Ongoing work to compute more complex models
  - Possible scalability improvements
    - ✓ Explore the longest path first ➔ execution tree prunning
    - ✓ Test selection criteria
    - ✓ Apply the method to sub models ➔ iterative process
  - Gives only a lower bound of the WCET (measurement)!

Plan of the presentation

- Context
- Related works on WCET analysis methods
- A hybrid method for WCET estimation
- **An analysis-based method for WCET estimation**
- Conclusions and perspectives

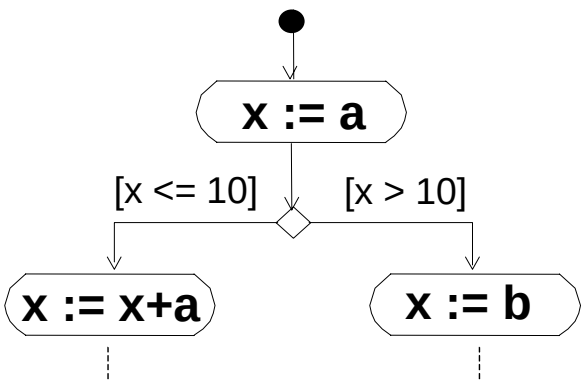
Model-based WCET static analysis overview

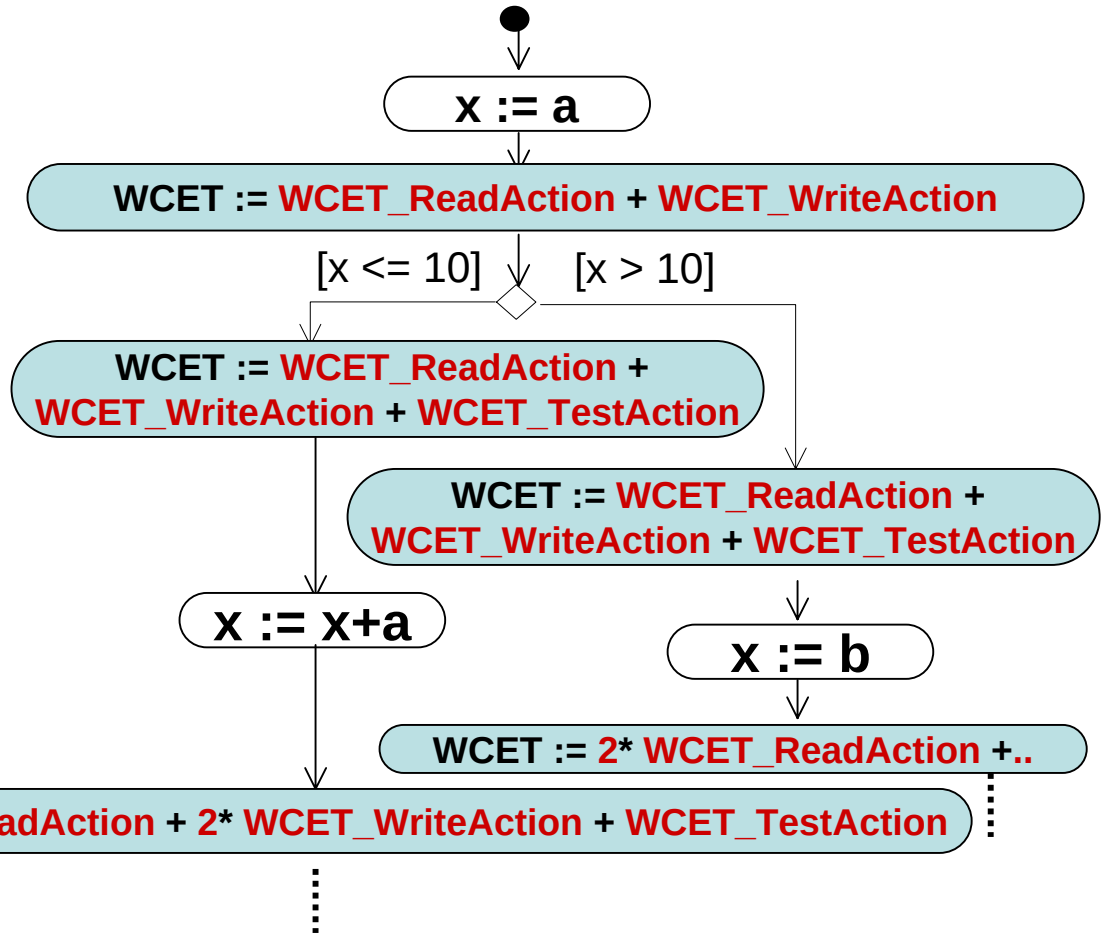


- Path analysis with symbolic execution (optimization phase)
  - ✓ Infeasible paths removal ➔ execution tree prunning
- Model transformation (computation phase)
  - ✓ Assign symbolic WCET values to primitive actions of the action langage
    - → WCET_ReadAction; WCET_WriteAction; WCET_AddAction…
  - ✓ Compute symbolic WCET for every execution path
- WCET is obtained by replacing symbolic values by numerical values

Illustration with an example

Model transformation to add symbolic timing values of elementary actions





Results

- **Advantages**
  - Good results for systems using simple microprocessors
    - ✓ Sequential execution: no pipeline
    - ✓ No cache
    - ✓ Systems where architectural mechanisms harming time predictability are switched off (critical systems)
- **Drawback**
  - Overestimated WCET for systems using these architectural features (increasingly frequent)

➔ **Need to take into account architectural features to reduce overestimation**

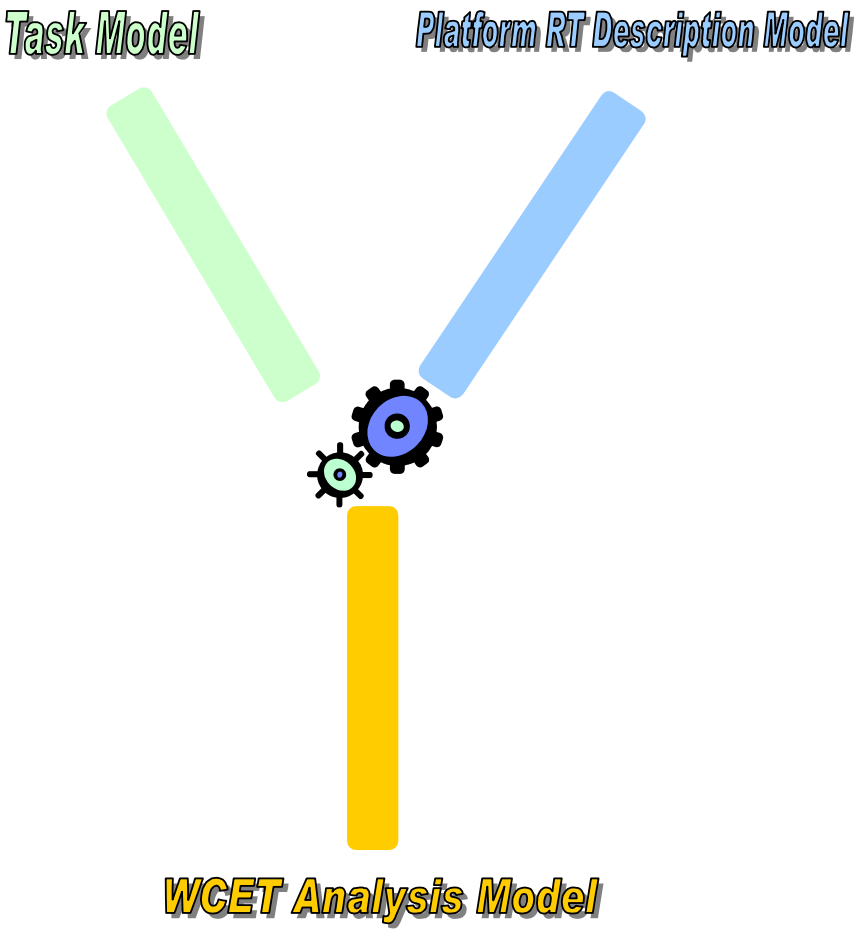
  - Pipeline
  - Cache
  - Branch predictors
  - Out of order execution

Perspectives: improvements

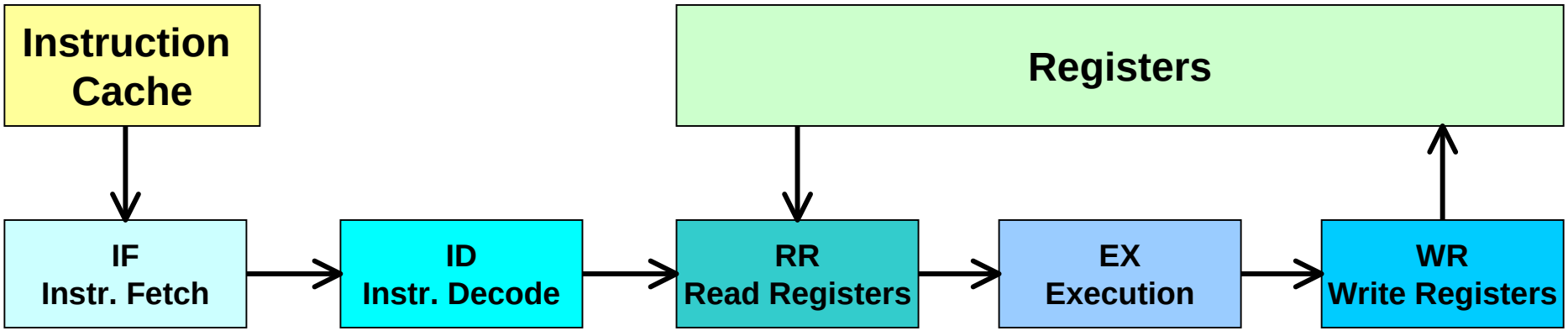
- **How to take into account architectural features?**
  - Timing models of microprocessors
    - ✓Implement this model in a static analysis tool (Agatha in our case)
      - → Not interesting because not flexible, not portable…
    - ✓Extract architectural elements timing properties in a dedicated model
      - → Platform Real-time Description Model
  - What should this model contain?
    - ✓Timing properties of architectural elements
      - → pipeline, cache…
    - ✓Accuracy of results increases with number of modeled elements
    - ✓Stay at a some level of abstraction in order to be usable by tools

Perspectives: Model-driven WCET analysis

- **Apply Model Driven Engineering tenets for WCET analysis**
  - Task model
  - Platform RT Description Model
  - Model transformations to map the platform model on the task model ➔ WCET Analysis Model
- **Benefits**
  - Portability
  - Reusablity
  - Maintainability
  - Model-level automated process
    - ✓ Model transformations
    - ✓ We can analyse WCET for several platforms and choose the best suited one before implementation



Example: a simple pipelined architecture



- **No memory, no data cache**
  - All operands are in registers or in the instruction itself (constants)
- **No branch prediction mechanisms**
- **No out of order execution**
- **No short-circuit mechanisms**
  - A data dependency costs a pipeline stall of 2 cycles
  - Only RAW (Read After Write) data dependencies are possible
    - ✓ Occurs when $instr_{n+1}$ needs to read data processed by $instr_n$

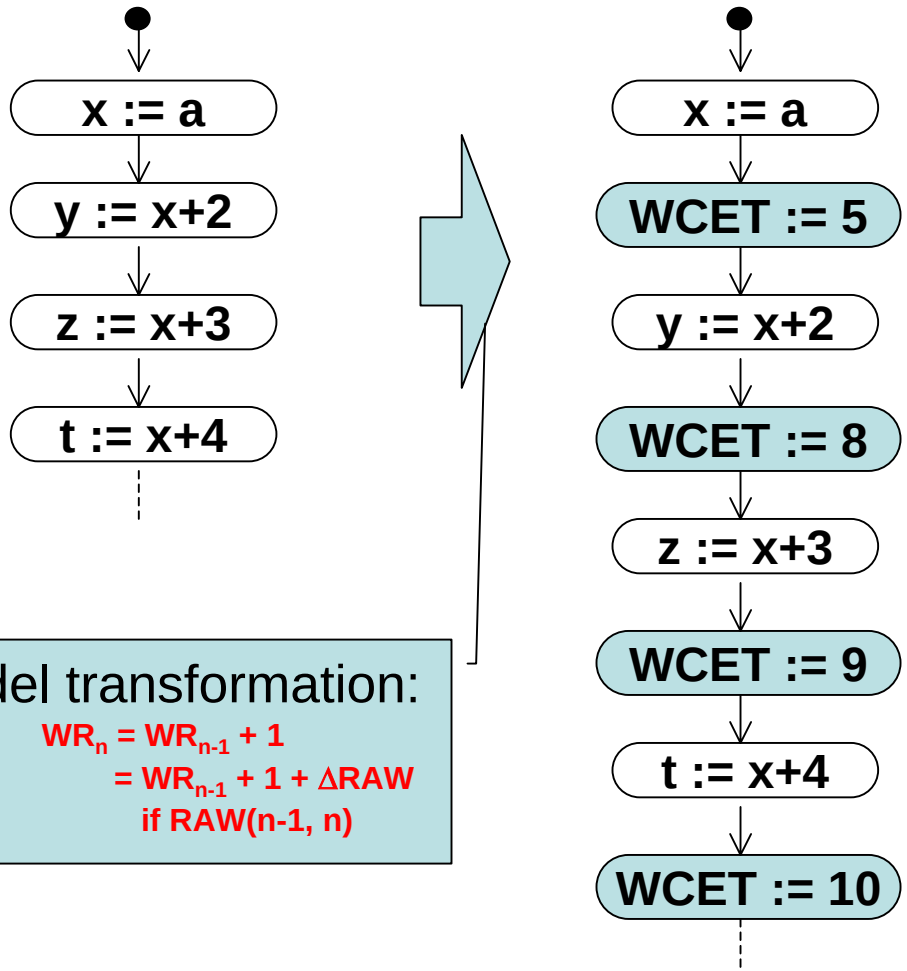
Example: a simple pipelined architecture (cont.)

	1	2	3	4	5	6	7	8	9	10
x:=a	IF	ID	RR	EX	WR					
y:=x+2		IF	ID	stall	stall	RR	EX	WR		
z:=x+3			IF	stall	stall	ID	RR	EX	WR	
t:=x+4						IF	ID	RR	EX	WR

- **WCET**
  - The cycle number of WR unit of the last instruction
- **Model transformation rules**
  - Computes cycle of WR for every instruction
    - ✓ WR_n: cycle of WR of instruction n

$$\boxed{\begin{aligned} WR_n &= WR_{n-1} + 1 \\ &= WR_{n-1} + 1 + \Delta RAW \quad \text{if } RAW(n-1, n) \end{aligned}}$$

Model transformation



Model transformation:

$WR_n = WR_{n-1} + 1$

$= WR_{n-1} + 1 + \Delta RAW$

if RAW(n-1, n)

- **Without taking into account the pipeline: sequential execution**
  - WCET = 4 x 5 = 20 cycles
- **A saving of 50%**
  - For 4 instructions only!
  - For a very simple architecture
- **Taking into account the architecture improves accuracy!**

Plan of the presentation

- Context
- Worst-Case Execution Time methods
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- **Conclusions and perspectives**

Conclusions

- **Estimation of WCET from UML-based RT models**
  - A hybrid approach
    - ✓Provides lower-bound of WCET
    - ✓Precise results useful for soft real-time systems
  - An analysis approach
    - ✓Safe bounds
    - ✓Model-driven analysis technique
      - →Task model + Platform RT Description model ➔ WCET analysis model
        - » Flexible, portable technique
      - →Model Transformation process
      - →The architecture becomes a parameter of the analysis (SoC)
- **Perspectives**
  - Take into account more architectural features
    - ✓Refine the model transformation rules by taking into account the data cache