Integrating modeling checking and UML-based model-driven development for embedded systems

CMACS/AVACS Workshop

Zamira Daw[1], Rance Cleaveland[1], and Marcus Vetter[2]

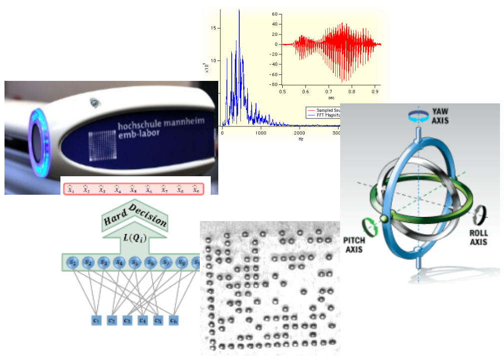
1.University of Maryland

2.Hochschule Mannheim - University of Applied Sciences

Carnegie Mellon University, November 20-22, 2013

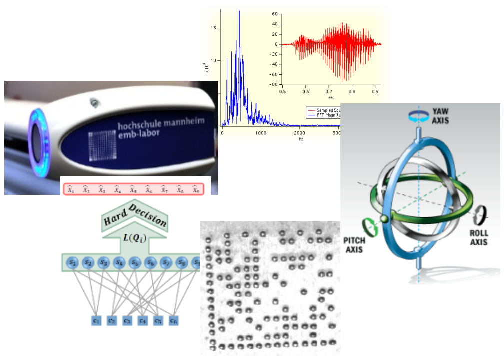
Motivation

- Increased use of embedded software in industries such as automotive, aerospace and medical-device
- Capability to implement richer and more sophisticated functionalities



Motivation

- Increased use of embedded software in industries such as automotive, aerospace and medical-device
- Capability to implement richer and more sophisticated functionalities



The increasingly **complex** and sophisticated **software** must be **written** and **verified** in order to ensure the **correct functionality** of the system and avoid system errors

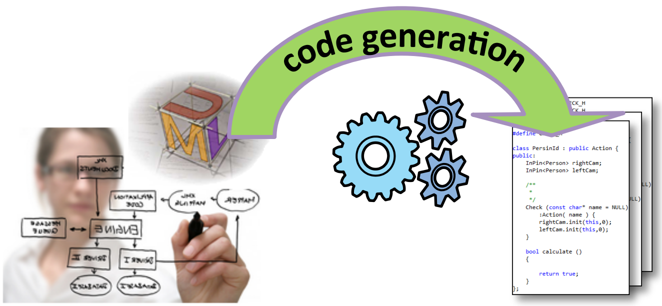
hochschule mannheim
emb-lab

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Model-driven development (MDD)

MDD reduces the complexity of the development of software using models



- High abstraction
- Model verification
- Code generation

Strong relationship between code and model

Integrating **model checking** into a **MDD** process, the **model/system** can be **verified** against the system requirement in **early phases** of the development

Unified Modeling Language (UML)

- UML is widely used in the software development
- UML provides 14 types of diagrams (behavioral and structural)
- However, UML does **not have a formal semantics** and therefore can not be directly verified by model checkers

- In order to allow formal verification, the behavior described by the UML models **has to be specified using mathematical well-defined languages**

Integrating model checking in UML-based MDD

Question: Which model checker and language should be used?

Problem: Existing work does not give a good answer

- Works are concentrated **only** on **one tool**
- Each work does **not** cover **the same properties** of the diagrams
- There are **not** enough **quantitative results** about the verification performance

Integrating model checking in UML-based MDD

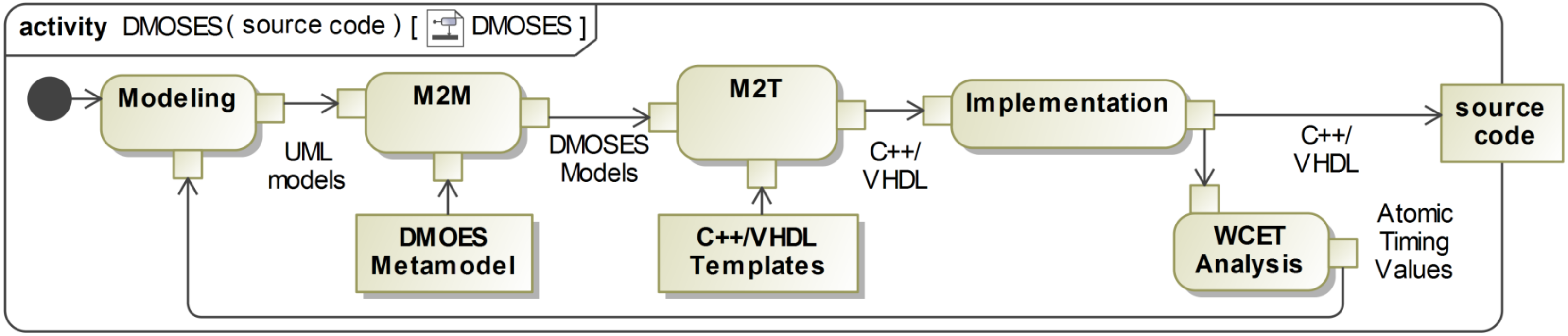
Question: Which model checker and language should be used?

Goals of this work are to:

- **Translate** UML activities into the mathematical well-defined languages *Timed Automata (TA)* and *NuSMV Language*
- **Compare** the verification performance between different tool chains (mapping + model checker)
- **Integrate** the toolchain into the DMOSES development process

DMOSES: Model-driven development for embedded systems

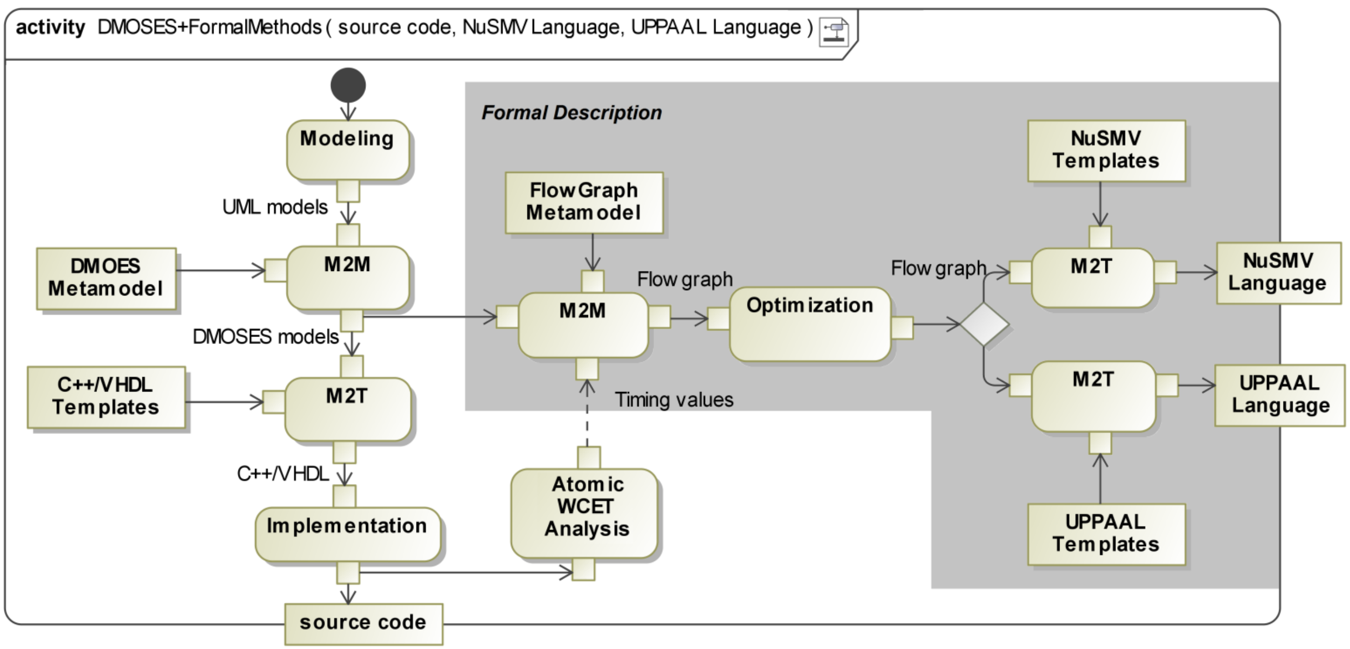
The behavior of embedded systems is modeled using extended UML activities and state machines, which are translated into source code for microcontroller and FPGAS.



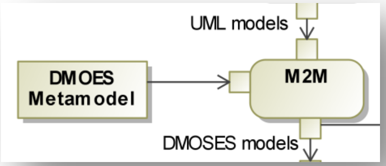
Integrate model checking in the DMOSES process in order to verify system requirements in early phases of the development.

Integration of formal verification into DMOSES

- DMOSES models are translated into well-formed mathematical languages using flow graphs.
- Flow graphs abstract information about the execution of the UML diagrams

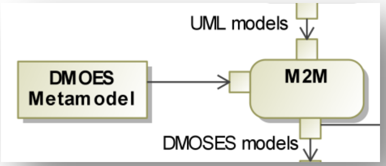


UML Activities enhanced with the DMOSES profile



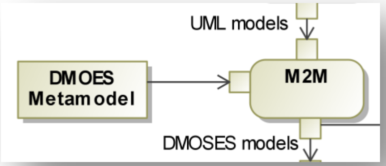
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding execution time (*WCET*), parallelism (*async*), resource distribution (*resource*) and priority (*priority*)

UML Activities enhanced with the DMOSES profile

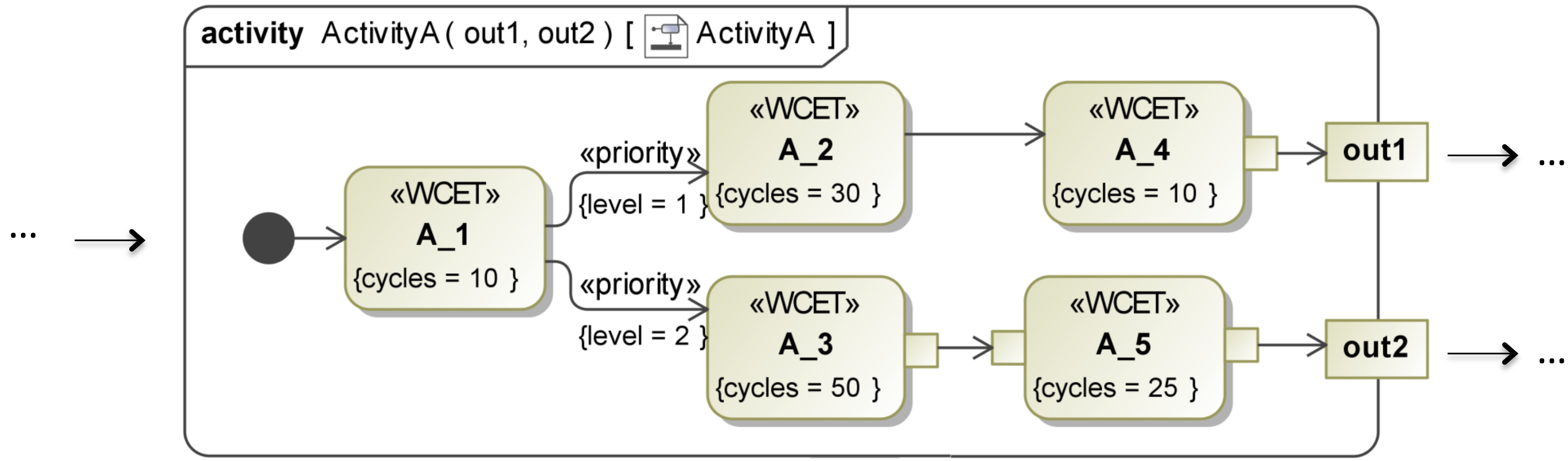


- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**

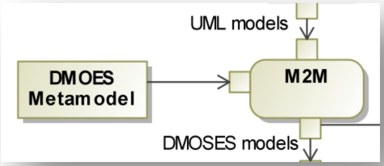
UML Activities enhanced with the DMOSES profile



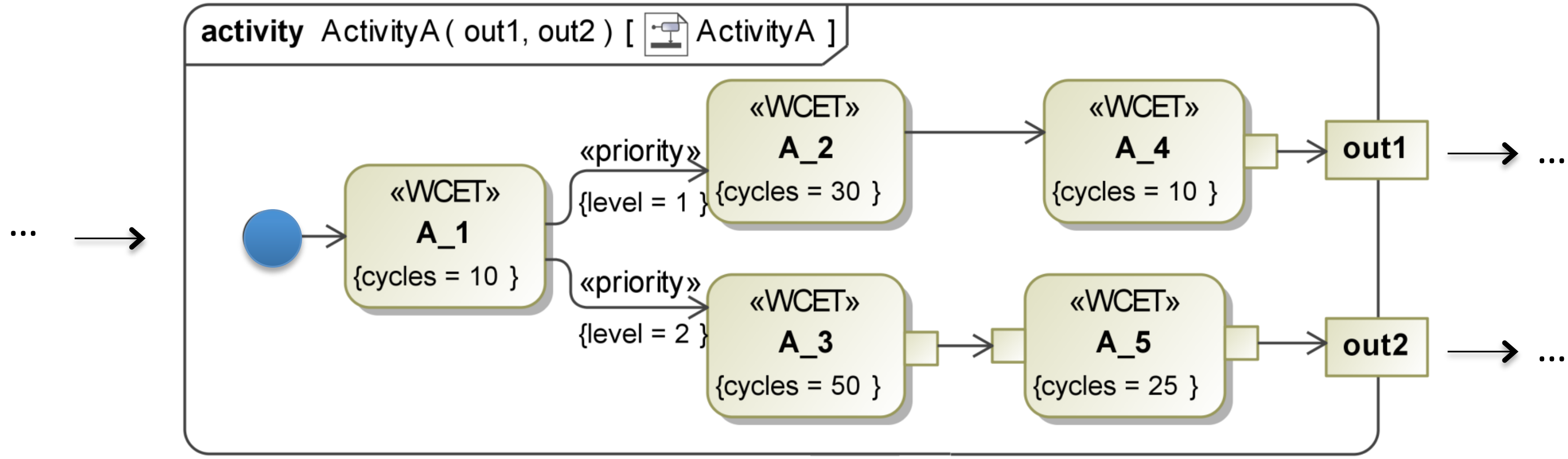
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**

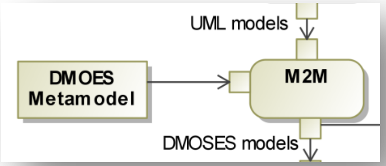


UML Activities enhanced with the DMOSES profile



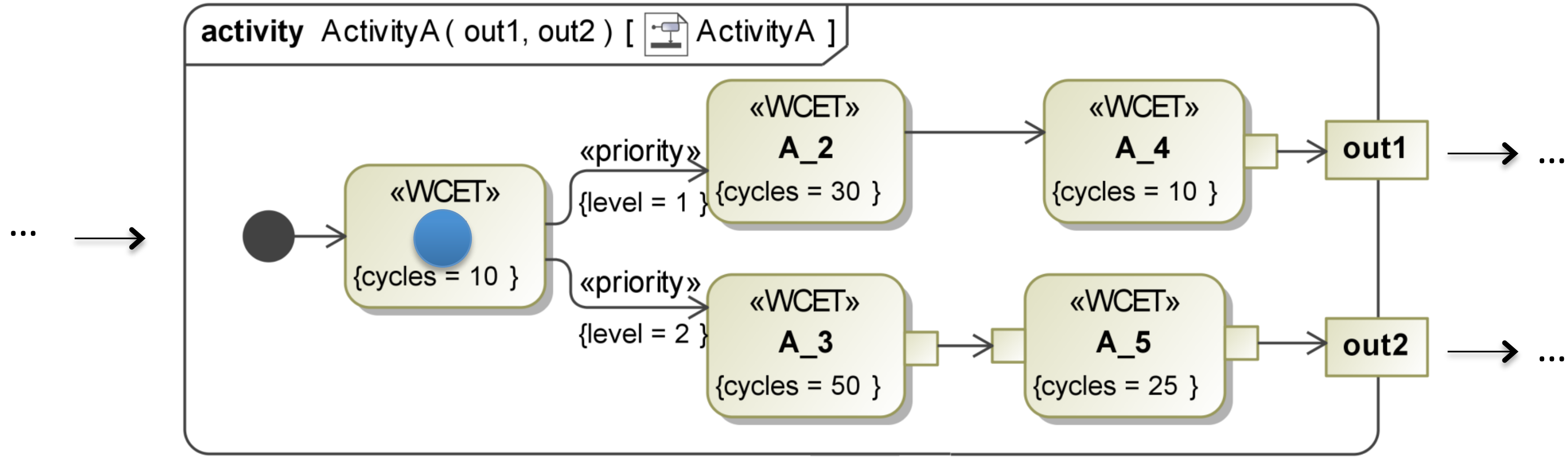
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**

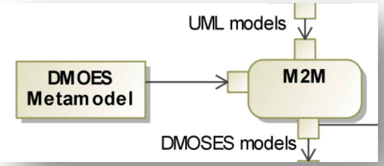




UML Activities enhanced with the DMOSES profile

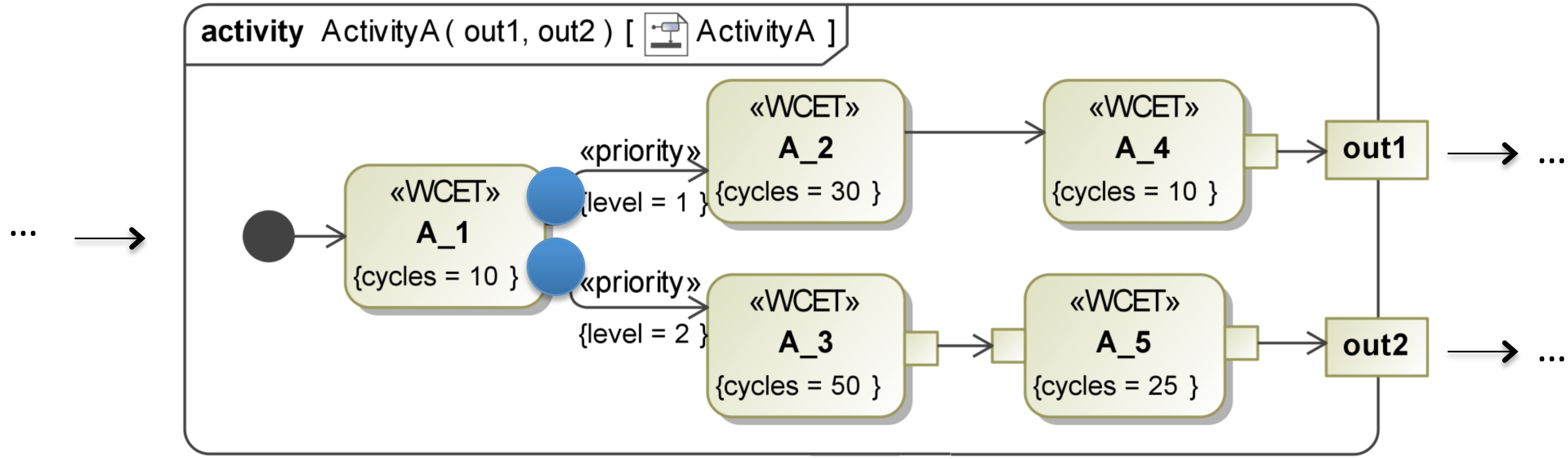
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**

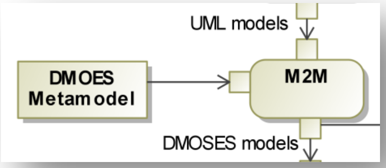




UML Activities enhanced with the DMOSES profile

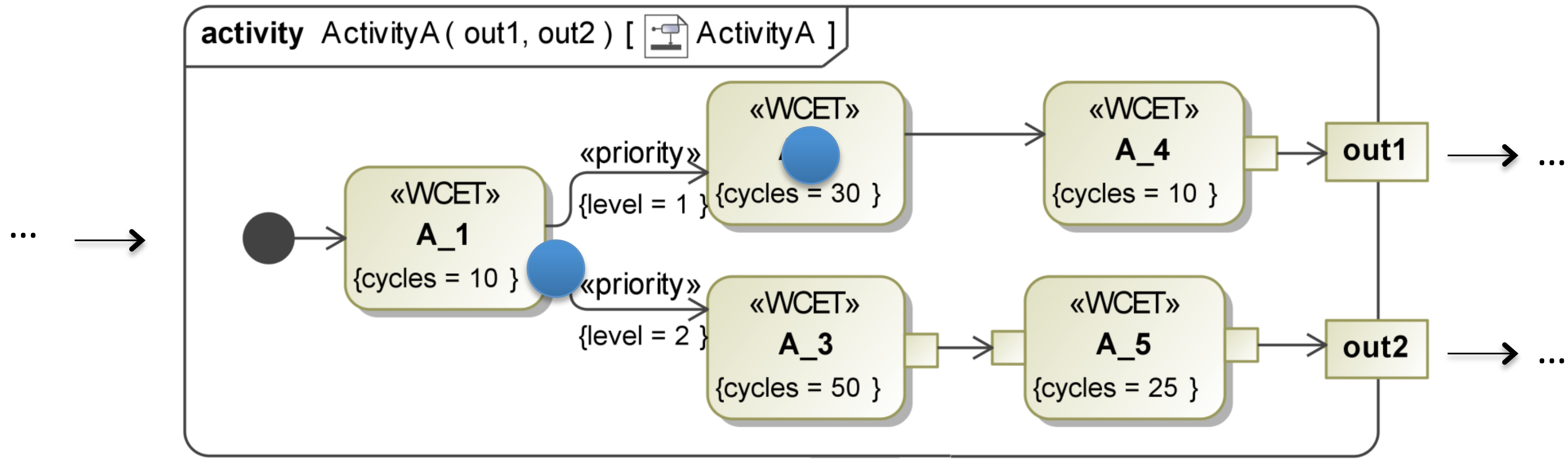
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**





UML Activities enhanced with the DMOSES profile

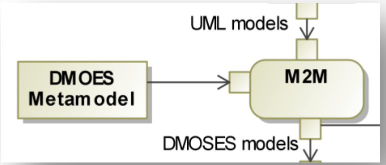
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



hochschule mannheim
emb-lab

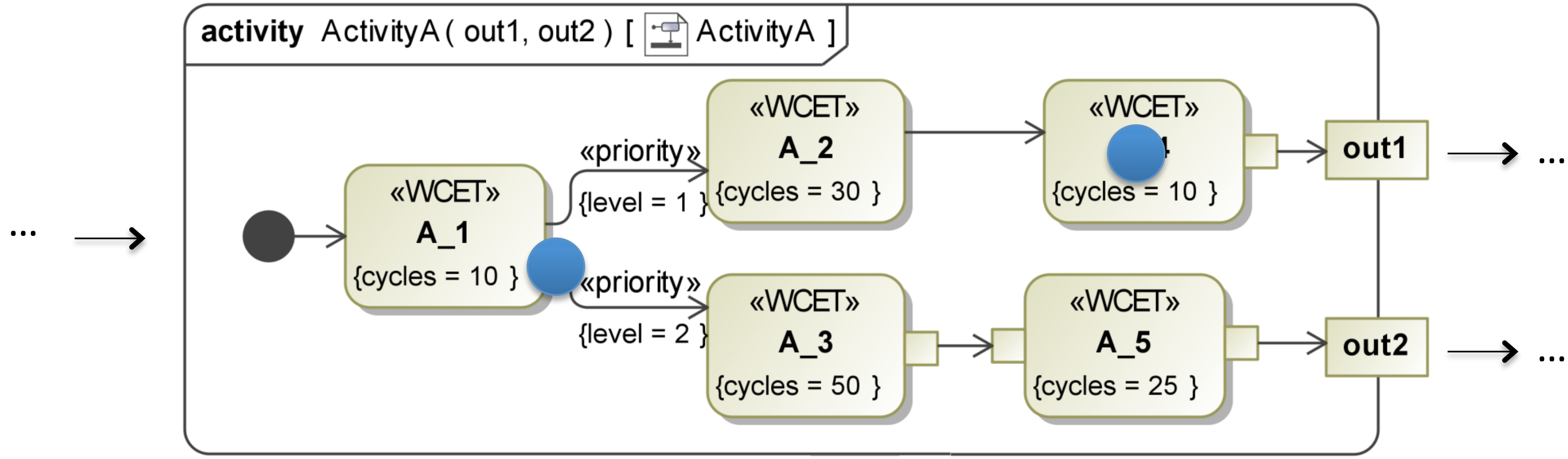
Fraunhofer USA Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

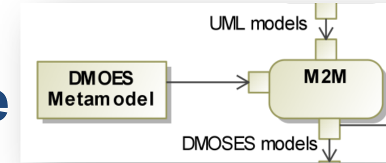


UML Activities enhanced with the DMOSES profile

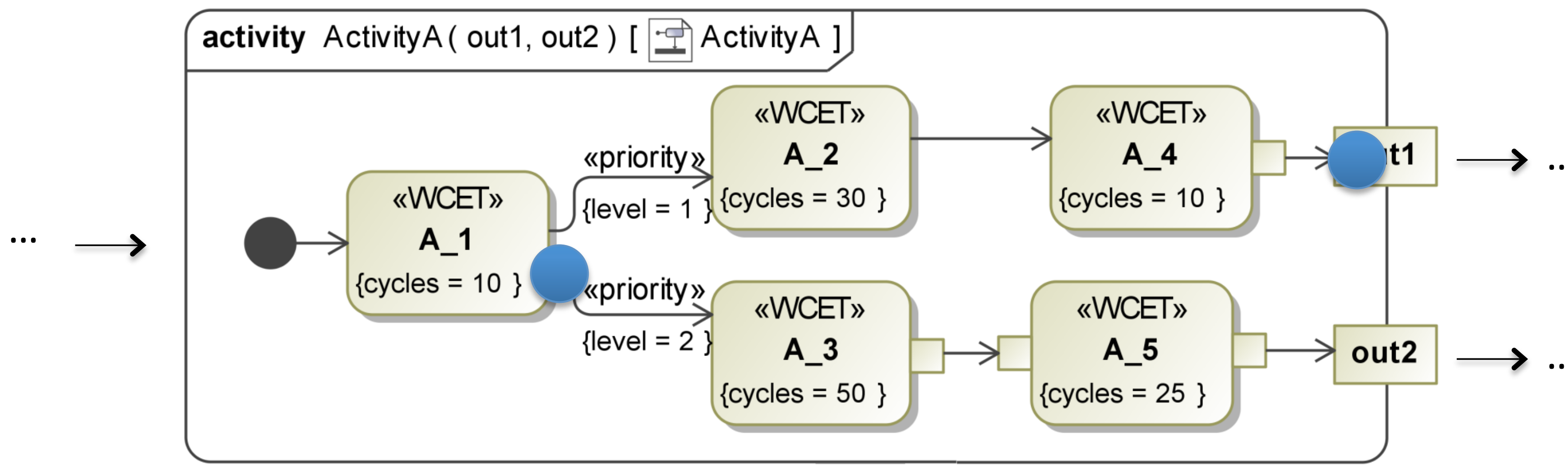
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



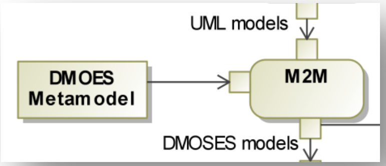
UML Activities enhanced with the DMOSES profile



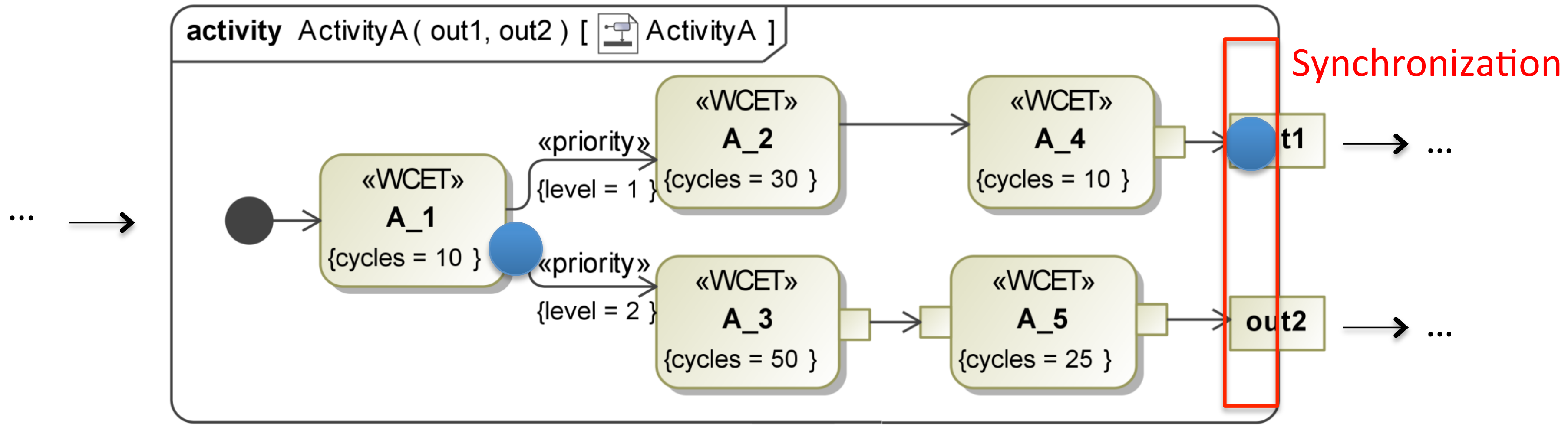
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**

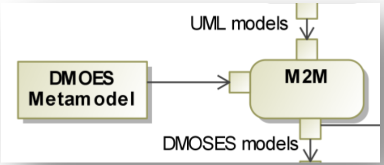


UML Activities enhanced with the DMOSES profile



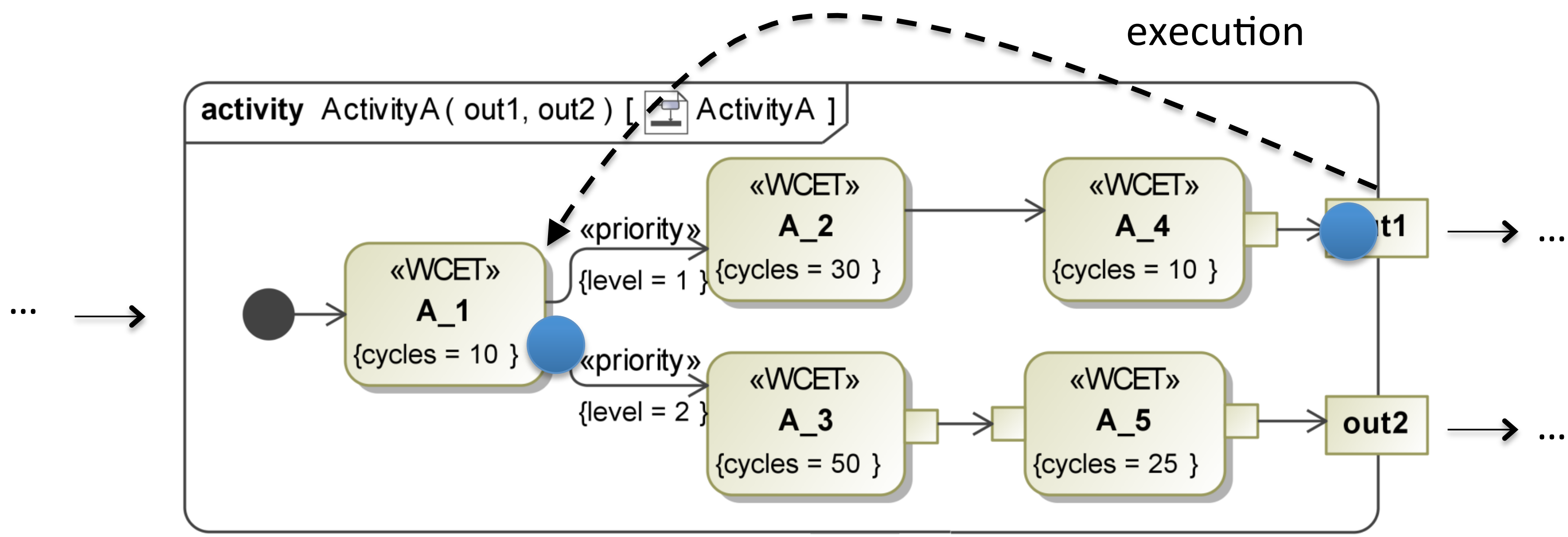
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



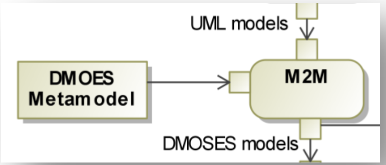


UML Activities enhanced with the DMOSES profile

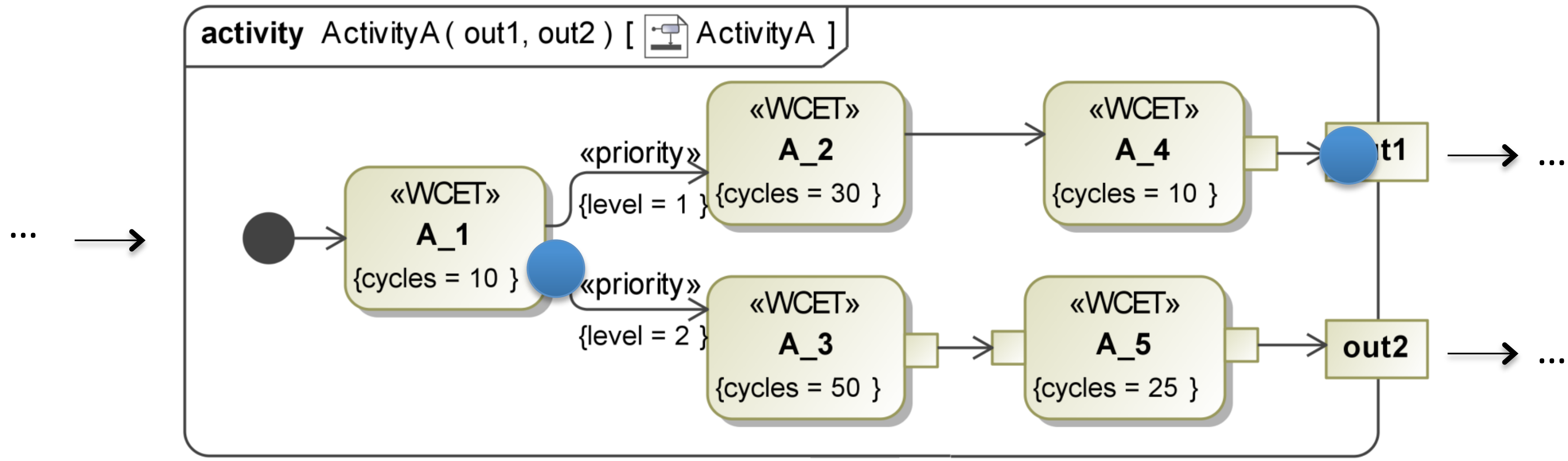
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



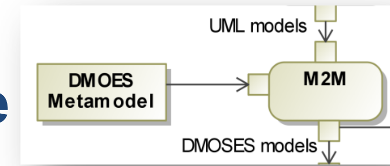
UML Activities enhanced with the DMOSES profile



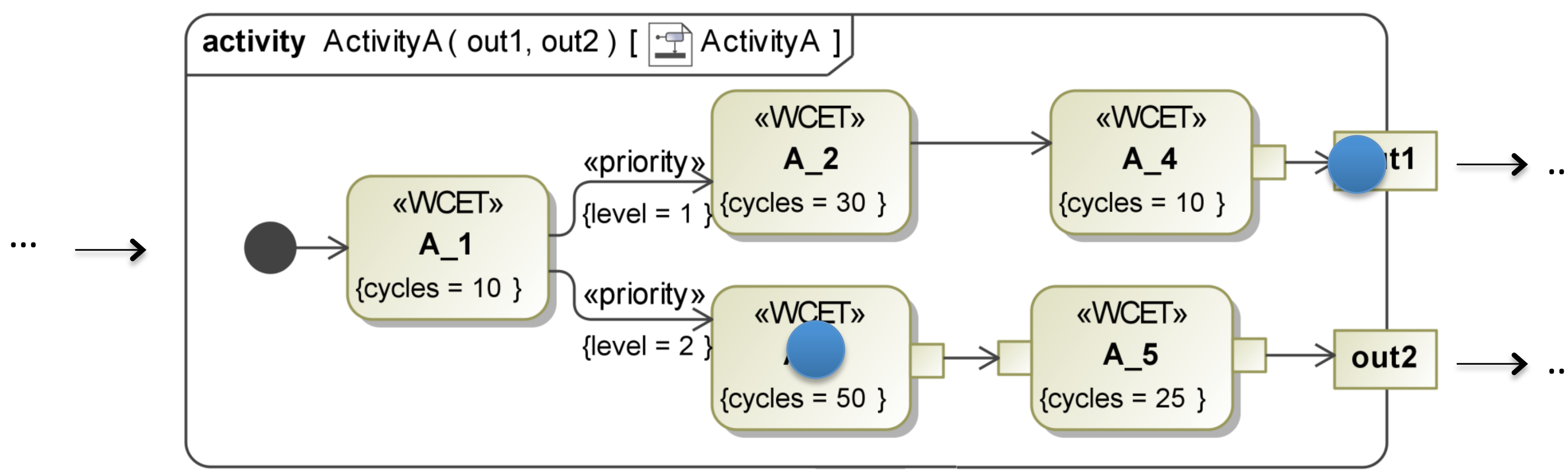
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



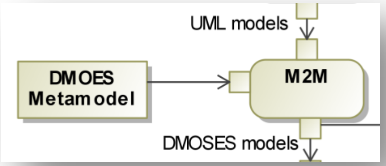
UML Activities enhanced with the DMOSES profile



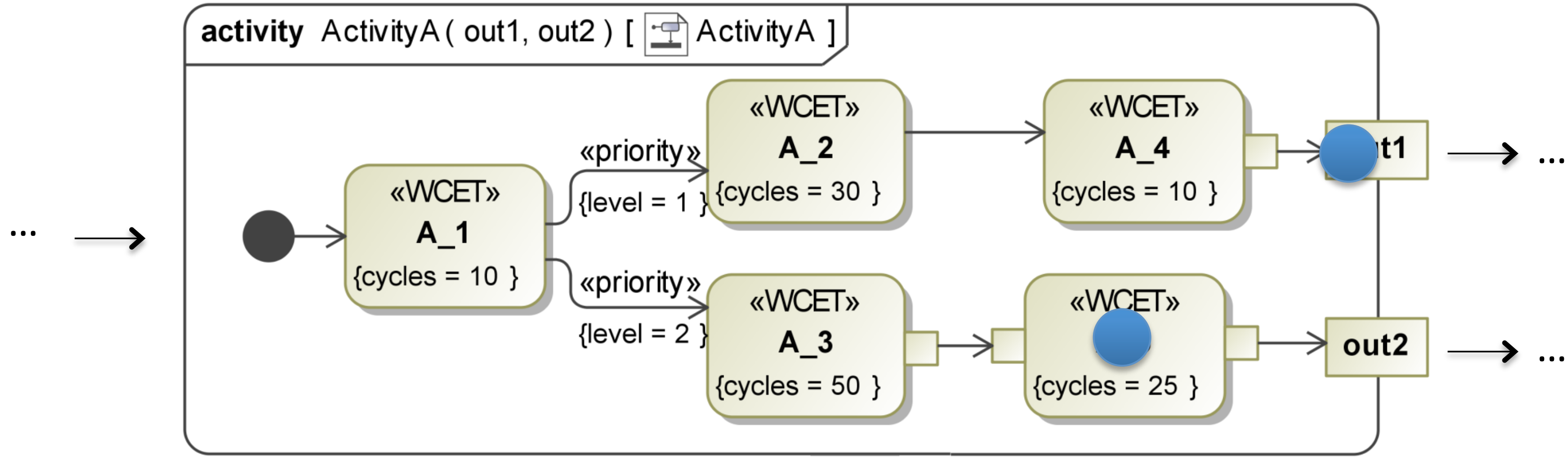
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



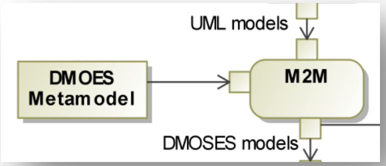
UML Activities enhanced with the DMOSES profile



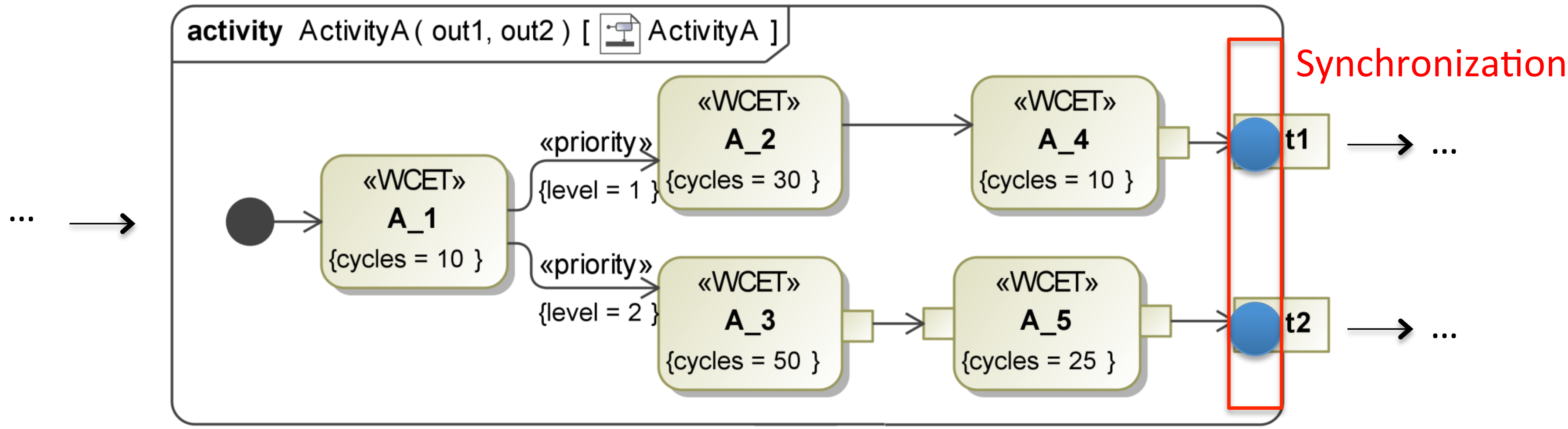
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



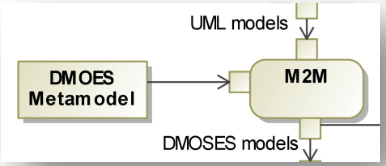
UML Activities enhanced with the DMOSES profile



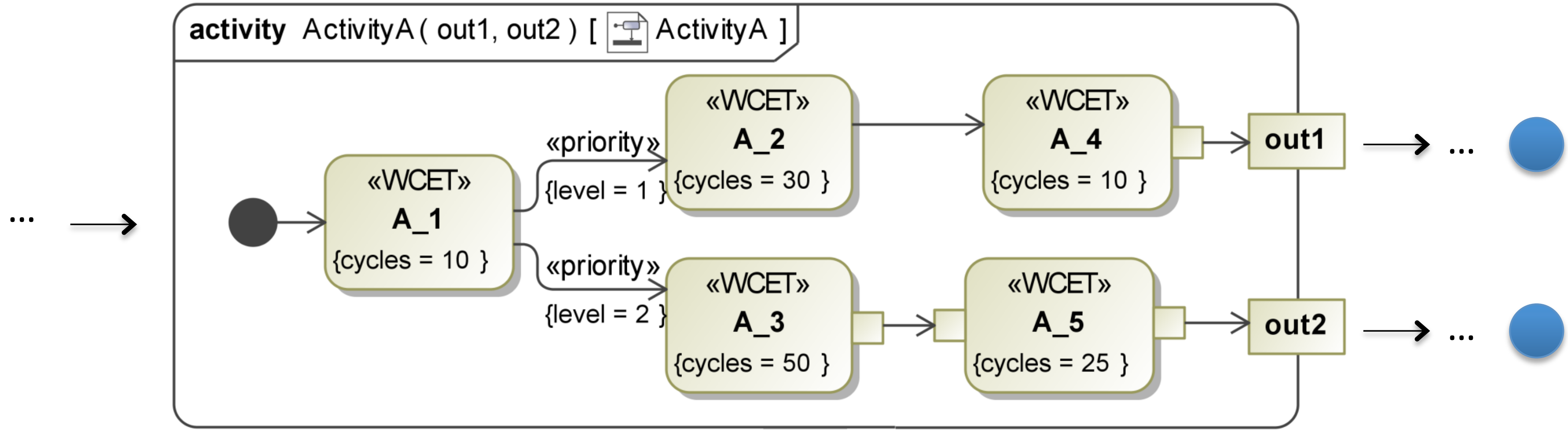
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



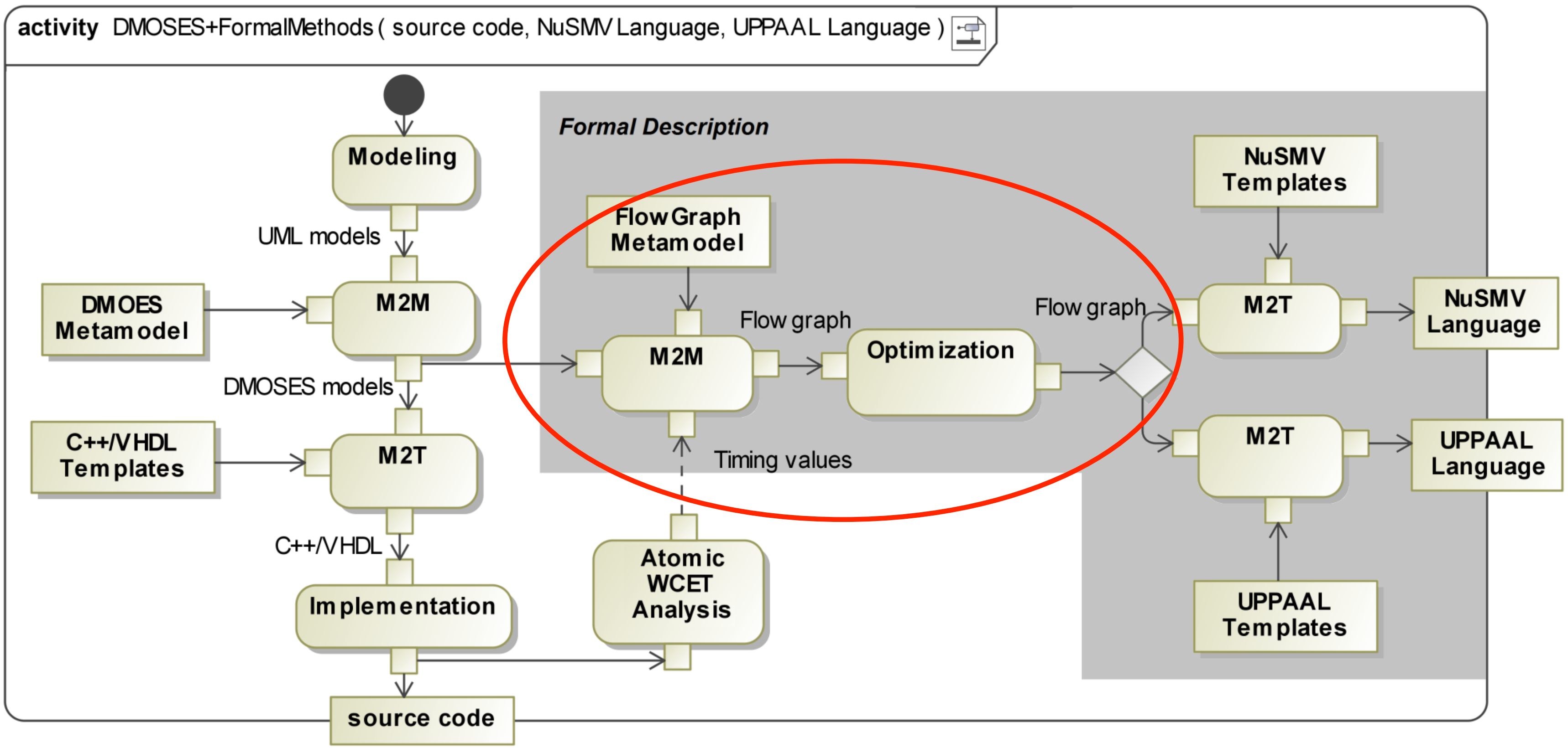
UML Activities enhanced with the DMOSES profile



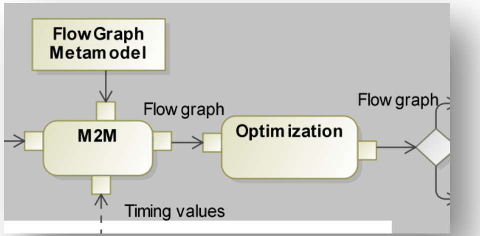
- Semantics is based on Petri nets
- DMOSES profile introduces information regarding **execution time (*WCET*)**, parallelism (*async*), resource distribution (*resource*) and **priority (*priority*)**



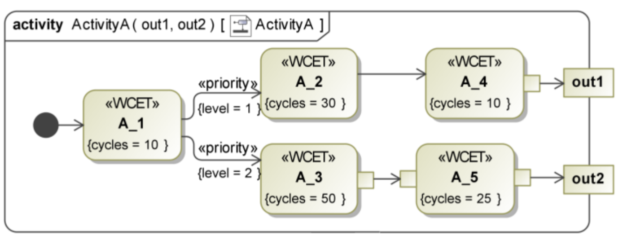
Integration of formal verification into DMOSES



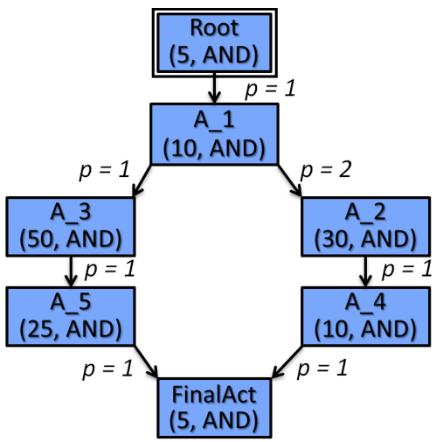
Transforming UML models into flow graphs



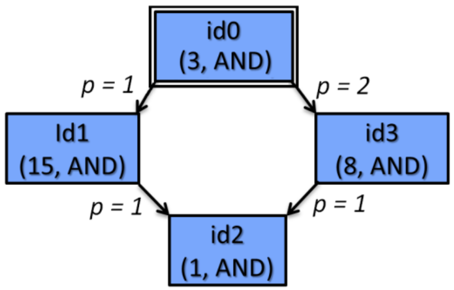
Enhanced UML Activity



Flow graph



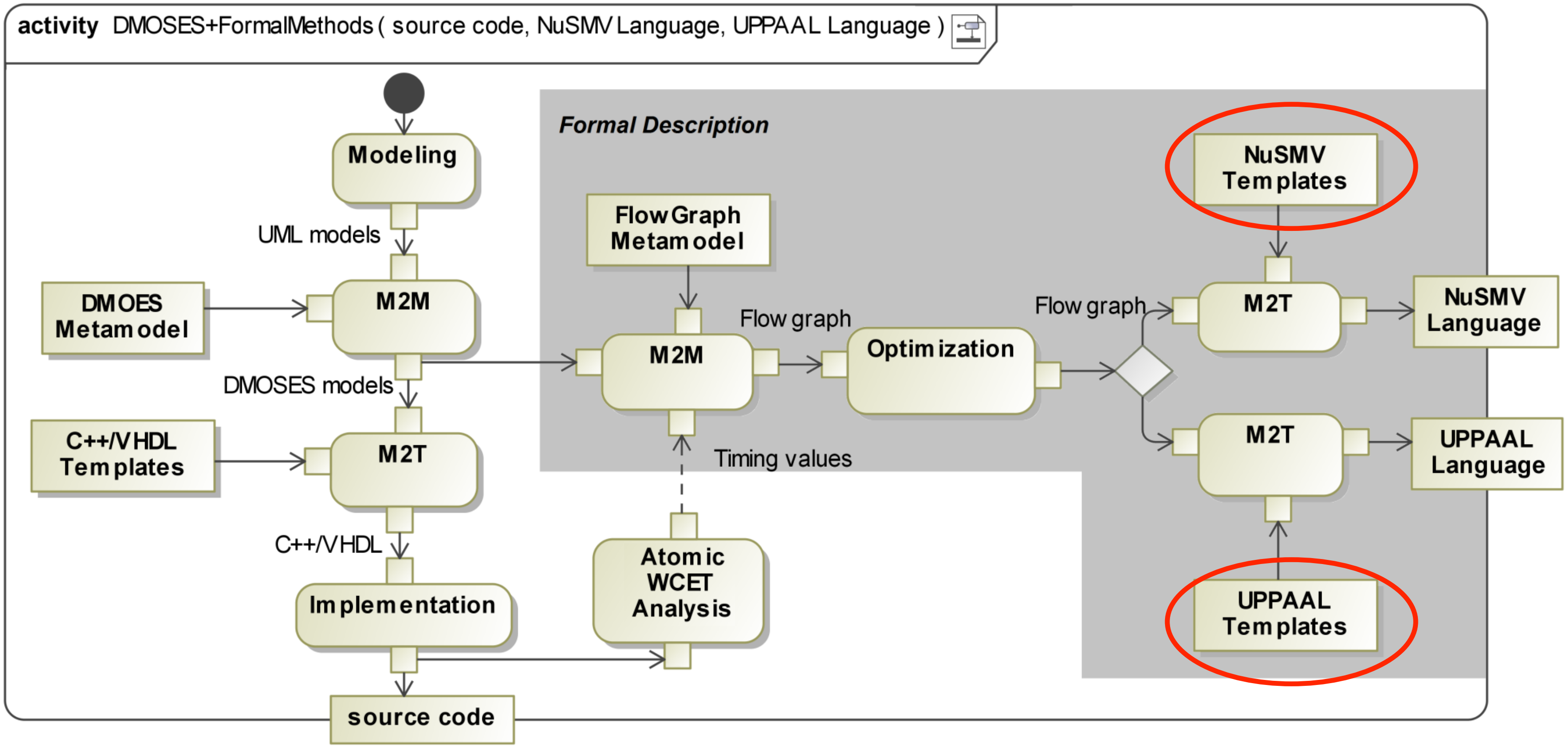
Optimized flow graph



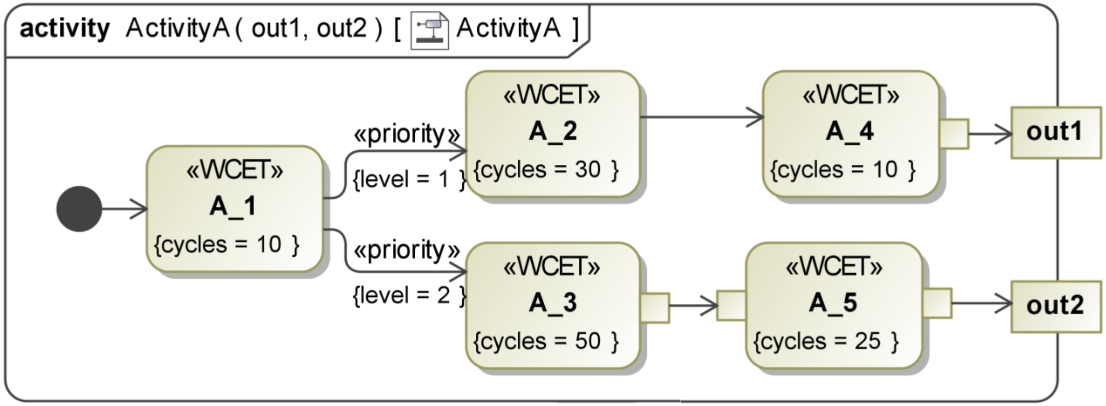
Extract the required information

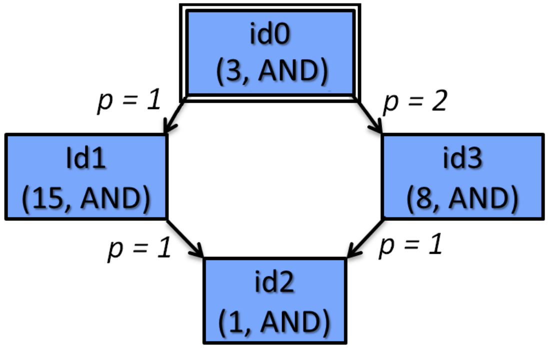
Merge sequential vertices, rescale execution time and flatten of hierarchal models

Integration of formal verification into DMOSES



Translating UML Activities into well-defined mathematical languages





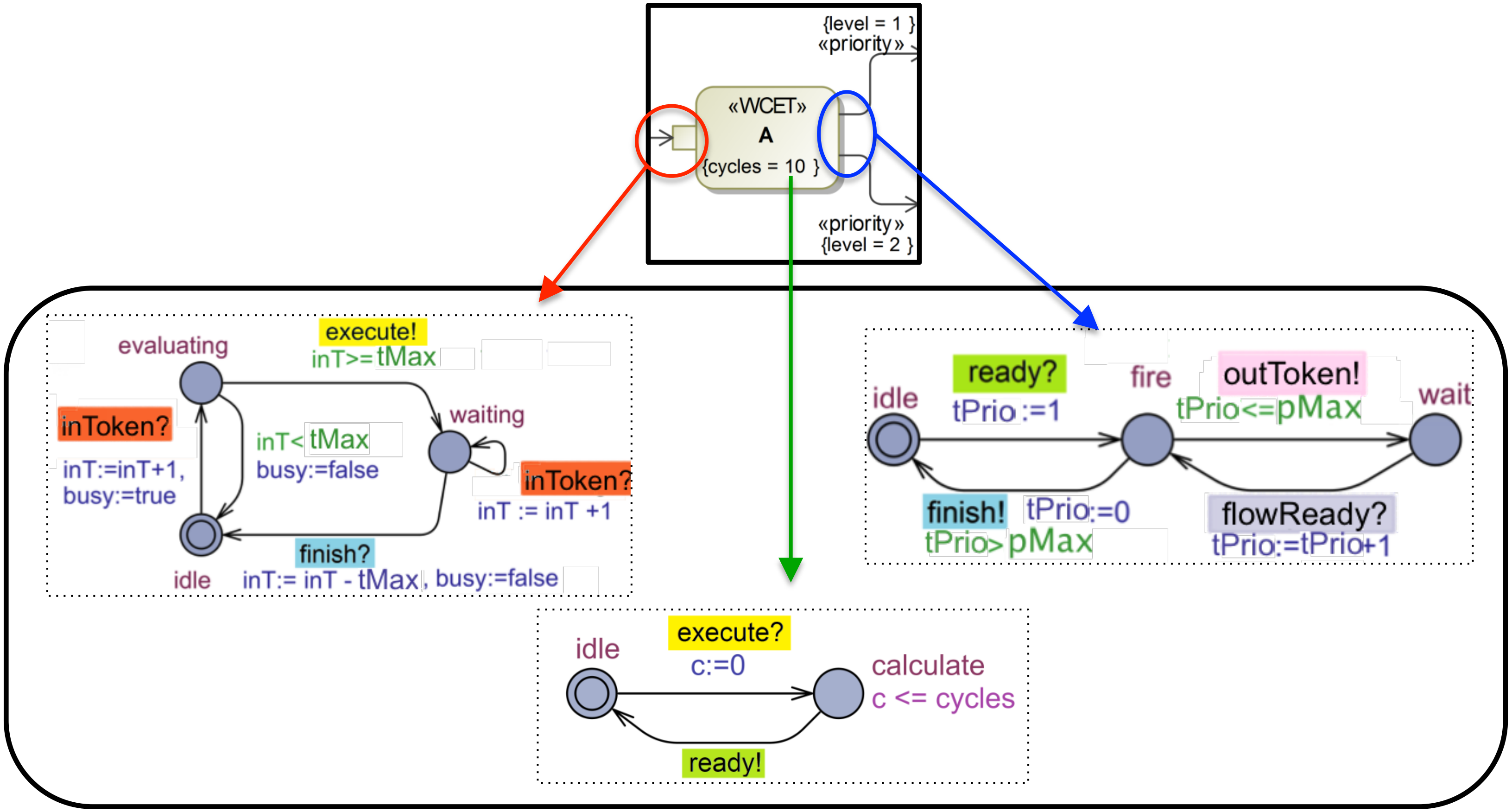
Activity behavior is divided into:

Atomic execution: specifies the **beginning**, the **end** and the **result** of the execution of an atomic element

Token flow: defines the **interaction** between the components

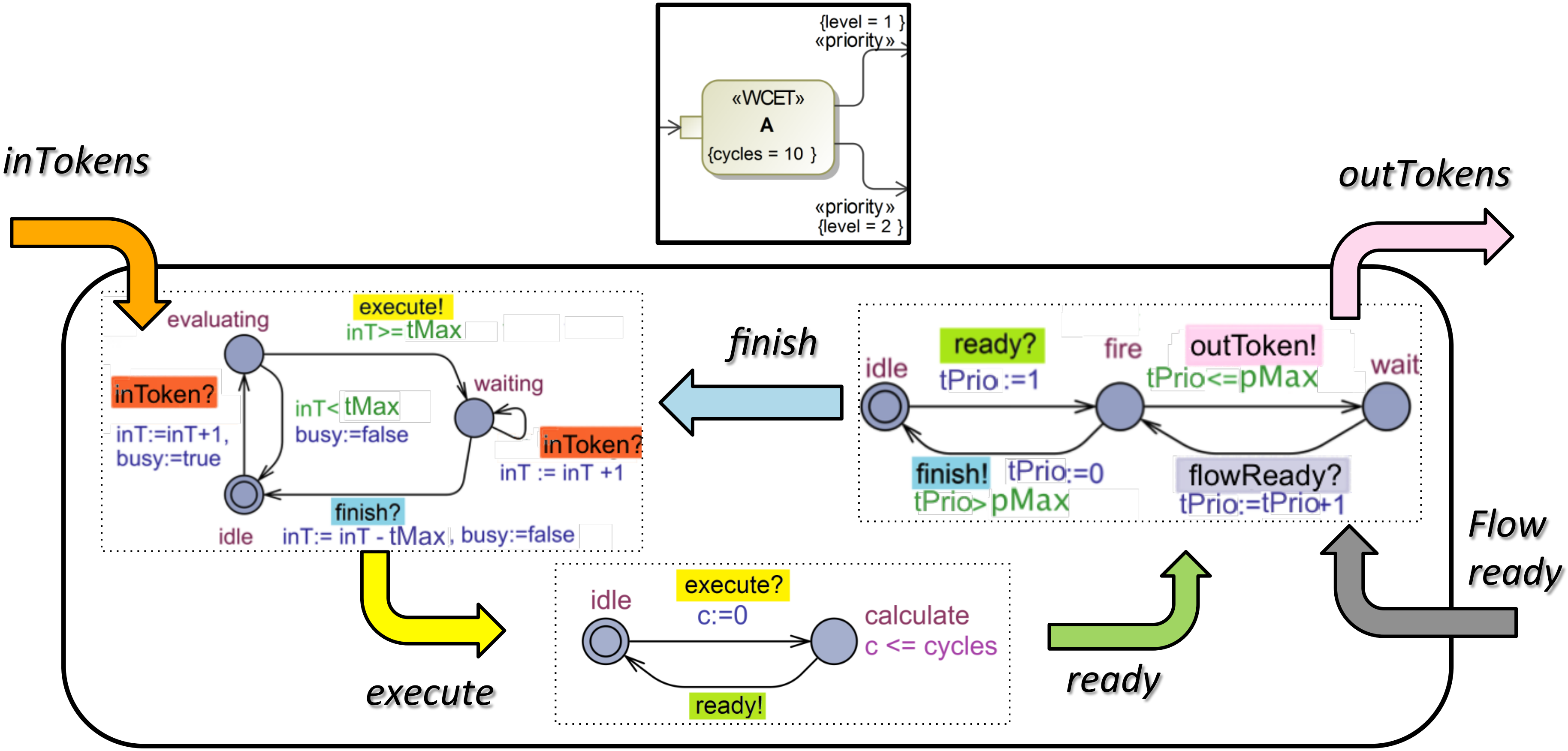
Translating UML Activities into Timed Automata (TA)

Atomic execution: specifies the **beginning**, the **end** and the **result**



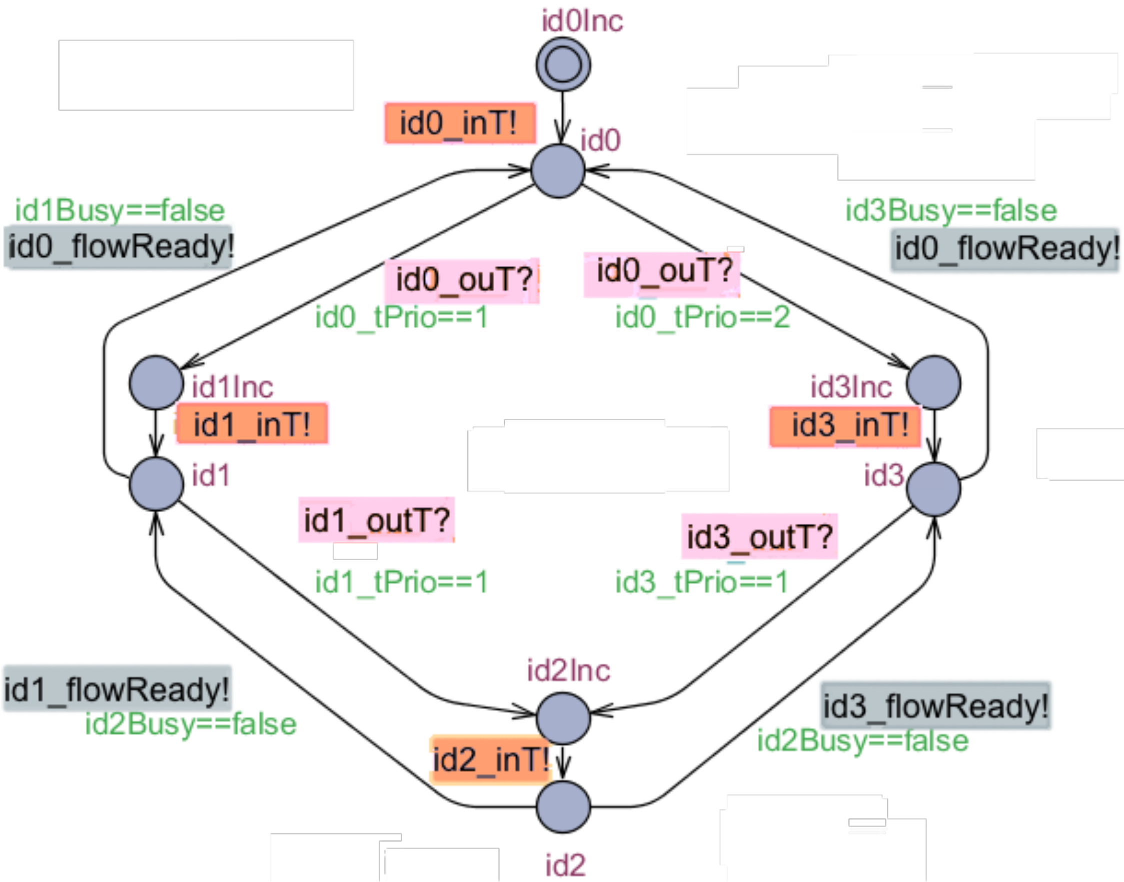
Translating UML Activities into Timed Automata (TA)

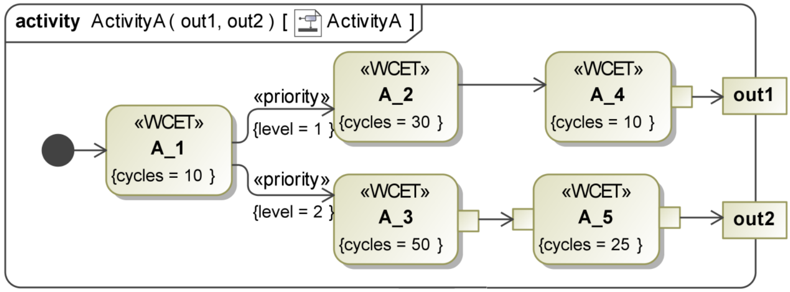
Atomic execution: specifies the **beginning**, the **end** and the **result**

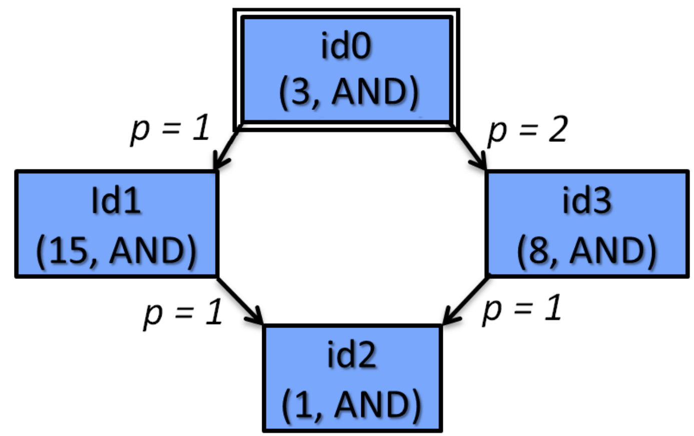


Translating UML Activities into Timed Automata (TA)

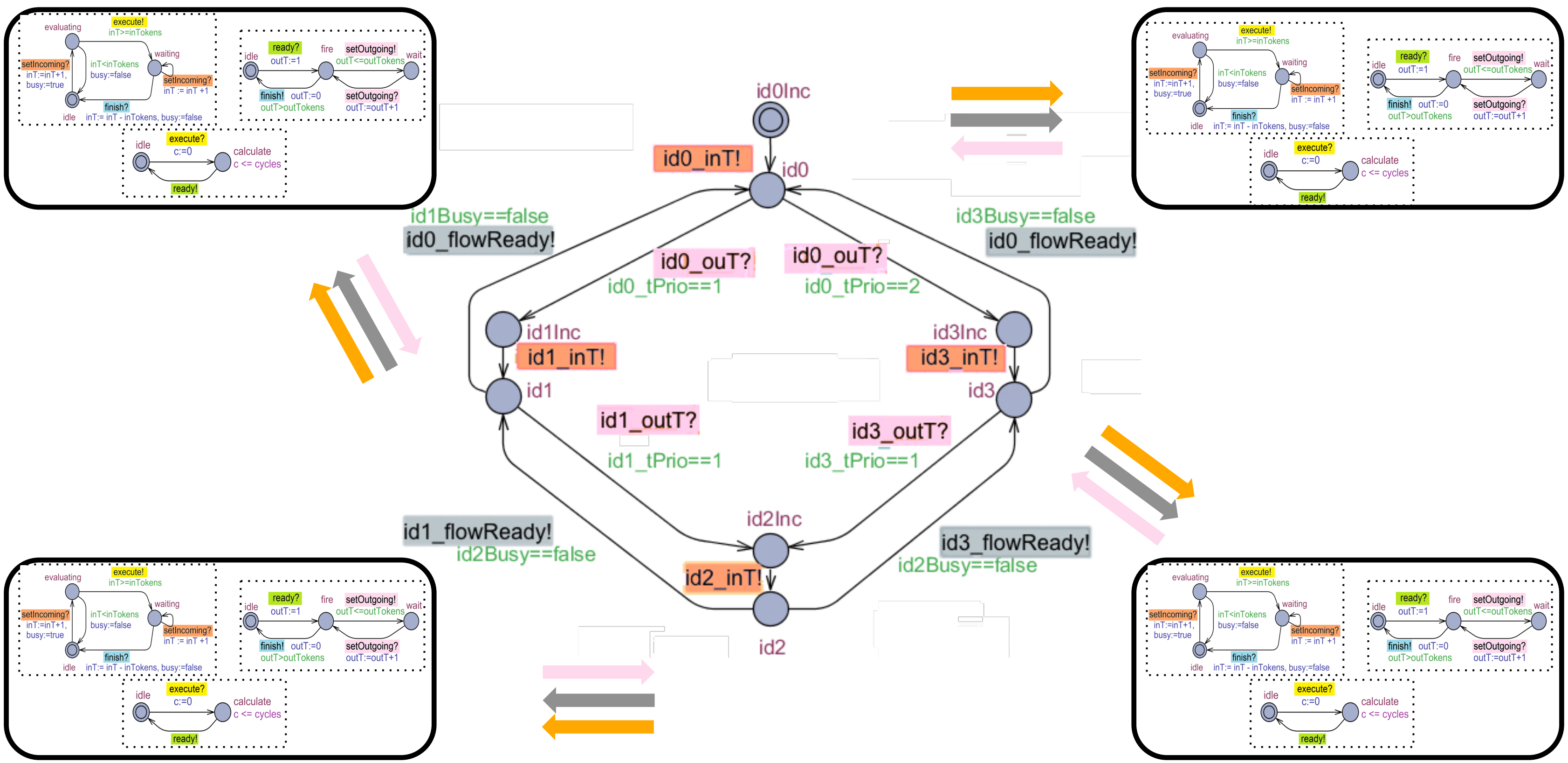
Token flow: defines the **interaction** between the components







Translating UML Activities into Timed Automata (TA)



One *atomic execution* (3 TAs) is assigned to **each double-state**

Translating UML Activities into NuSMV language

Atomic execution: specifies the **beginning**, the **end** and the **result**

Beginning

For every $x \in Vertex$
ASSIGN
 next($xInT$) :=
  case
   -- **"finish"**
   tokenFlow $= x$ & $xOut = 1 : xInT - ReqTokens$
   For edge $z = \{y \in Vertex | z = (y, x)\}$
   -- **"receiving tokens"**
   tokenFlow $= z$ & $zOutT = 1 : xInT + 1$
  esac:

xOut

Result

For every $x \in Vertex$
ASSIGN
 next($xOutT$) :=
  case
   tokenFlow $= x$ & $xC = xR : 1$ -- **"start"**
   -- **"finish"**
   tokenFlow $= x$ & $xOut = xMaxOutgoings + 1 : 0$
   For edge $z = \{y \in Vertex | z = (x, y)\}$
   -- **"firing"**
   tokenFlow $= z$ & $zReady$ & $zOutT = xP: xP + 1$
  esac;

End

xInT

For every $x \in Vertex$
ASSIGN
 next(xC) :=
  case
   $xC = xR$: -1 -- **"idle"**
   tokenFlow $= x$ & $xC != -1: xC + 1$ -- **"execution"**
   -- **"start"**
   tokenFlow $= x$ & $xInT \geq ReqTokens$ & $xOutT$: 0
  esac;

xC

Translating UML Activities into NuSMV language

***Atomic execution*:** specifies the **beginning**, the **end** and the **result**

Beginning

```
For every x ∈ Vertex
ASSIGN
  next(xInT) :=
    case
     -- "finish"
     tokenFlow = x & xOut = 1 : xInT − ReqTokens
     For edge z = {y ∈ Vertex | z = (y,x)}
      -- "receiving tokens"
      tokenFlow = z & zOutT = 1 : xInT + 1
    esac;
```

Result

```
For every x ∈ Vertex
ASSIGN
  next(xOutT) :=
    case
     tokenFlow = x & xC = xR : 1 -- "start"
       -- "finish"
     tokenFlow = x & xOut = xMaxOutgoings + 1 : 0
       For edge z = {y ∈ Vertex | z = (x,y)}
        -- "firing"
      tokenFlow = z & zReady & zOutT = xP: xP + 1
    esac;
```

xOut

End

```
For every x ∈ Vertex
ASSIGN
  next(xC) :=
    case
      xC = xR : -1 -- "idle"
     tokenFlow = x & xC != -1: xC + 1 -- "execution"
      -- "start"
     tokenFlow = x & xInT ≥ ReqTokens & xOutT: 0
    esac;
```

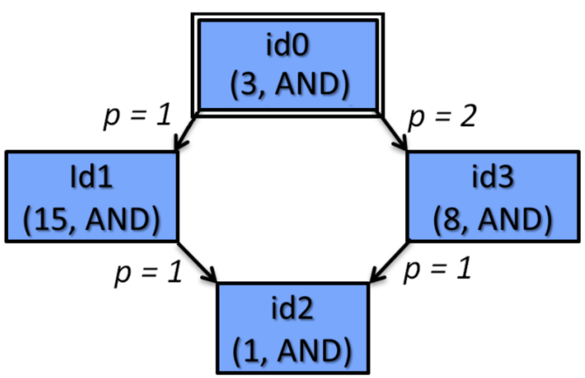
xInT

xC

Translating UML Activities into NuSMV language

Token flow: defines the **interaction** between the components

```
ASSIGN
  next(tokenFlow) :=
   case
    tokenFlow = root : x0; -- "Initial state"
For edge z ∈ Egde z = (x,y,p)
    tokenFlow = x & xOut = p: y -- "FT"
    tokenFlow = y & yReady & xOut = p: x -- "BT"
   esac;
```




```
tokenflow :{ root, id0, id1, id2, id3 };

next(tokenflow ):= case
  tokenflow = root : id0;
  tokenflow = id0 & id0OutT = 1: id1;
  tokenflow = id0 & id0OutT = 2: id3;
  tokenflow = id1 & id1OutT = 1: id2;
  tokenflow = id3 & id3OutT = 1: id2;

  tokenflow = id1 & id1Ready & id0OutT = 1: id0;
  tokenflow = id3 & id3Ready & id0OutT = 2: id0;
  tokenflow = id2 & id2Ready & id1OutT = 1: id1;
  tokenflow = id2 & id2Ready & id3OutT = 1: id3;
  TRUE :tokenflow ;
esac;
```

Forward Transitions

Backward Transitions

Translating UML Activities into NuSMV language

Token flow

```
ASSIGN
  next(tokenFlow) :=
   case
    tokenFlow = root : x0; -- "Initial state"
For edge z ∈ Egde z = (x, y, p)
    tokenFlow = x & xOut = p: y -- "FT"
    tokenFlow = y & yReady & xOut = p: x -- "BT"
   esac;
```

Atomic execution

End

```
For every x ∈ Vertex
ASSIGN
  next(xC) :=
   case
    xC = xR : -1 -- "idle"
    tokenFlow = x & xC != -1: xC + 1 -- "execution"
    -- "start"
    tokenFlow = x & xInT ≥ ReqTokens & xOutT : 0
   esac;
```

Beginning

```
For every x ∈ Vertex
ASSIGN
  next(xInT) :=
   case
    -- "finish"
    tokenFlow = x & xOut = 1 : xIntT − ReqTokens
    For edge z = {y ∈ Vertex|z = (y, x)}
     -- "receiving tokens"
     tokenFlow = z & zOutT = 1 : xIntT + 1
   esac:
```

Result

```
For every x ∈ Vertex
ASSIGN
  next(xOutT) :=
   case
    tokenFlow = x & xC = xR : 1 -- "start"
      -- "finish"
    tokenFlow = x & xOut = xMaxOutgoings + 1 : 0
      For edge z = {y ∈ Vertex|z = (x, y)}
       -- "firing"
     tokenFlow = z & zReady & zOutT = xP: xP + 1
   esac;
```

Performance evaluation

- The performance is evaluated by measuring the **verification time** of **deadlock freedom**

$$\mathbf{E} <> id_{final}Cal.calculate$$

UPPAAL

$$\mathbf{EF}(tokenFlow = id_{final} \ \& \ id_{final}C = 0)$$

NuSMV

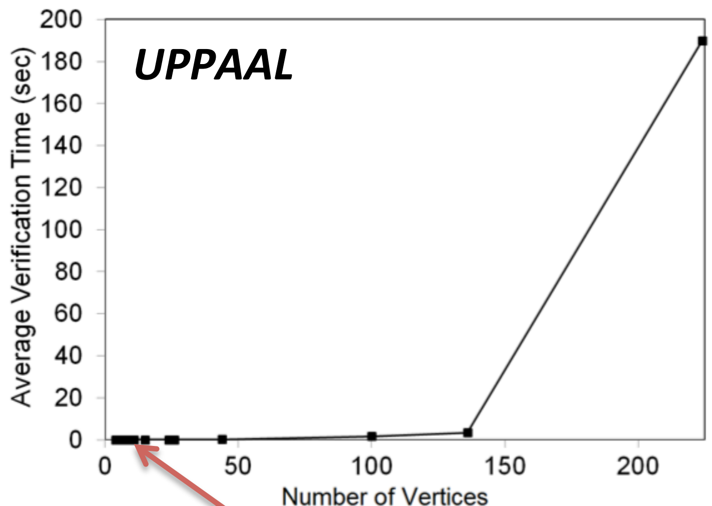
- Influence of the verification time against:

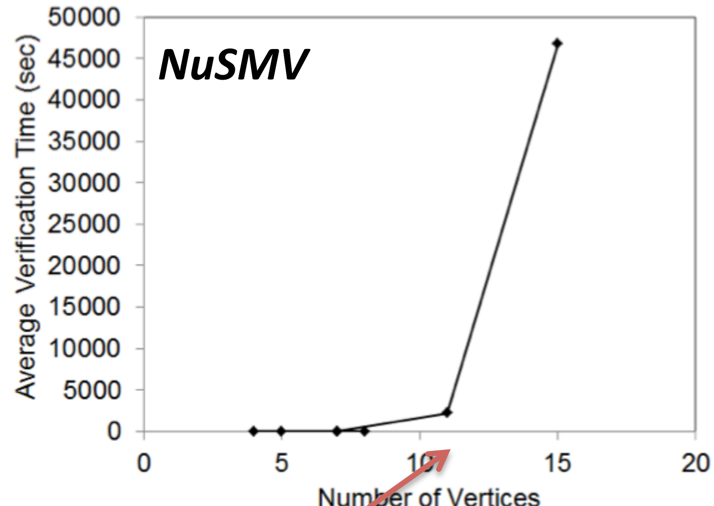
  1) The **number of elements and connections**. A set of 30 activities (number of vertices [1, 224], number of edges [0,290], hierarchical levels [1, 5] and with or without deadlocks).

  2) The **execution time** (WCET) of the elements

Performance evaluation

Different number of vertices (UML elements) and the same execution time

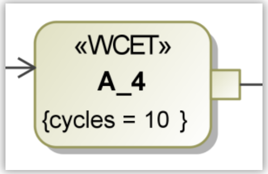




0.08 sec

2221.77 sec

The verification time is directly **proportional increased** by the number of vertices



Performance evaluation

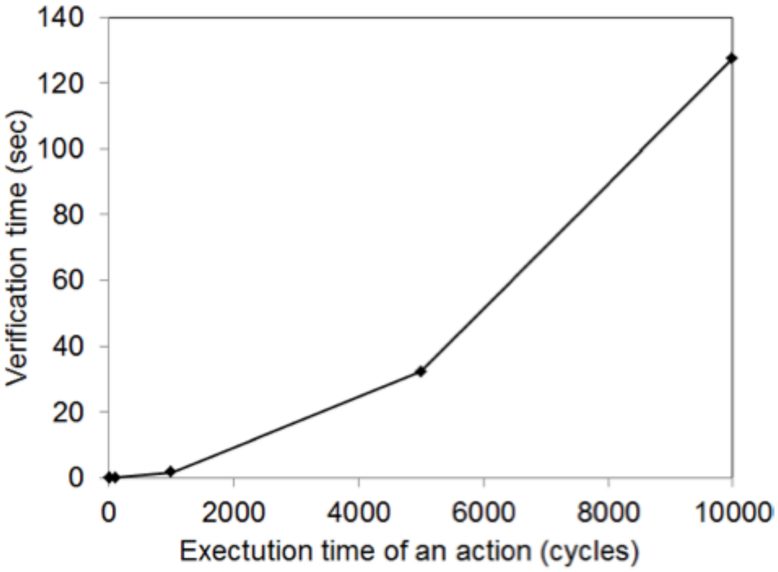
Different execution times of an action and the same number of elements

UPPAAL

No changes in the performance were evident

NuSMV

The state space is directly proportional increased by the number of the cycles of an action

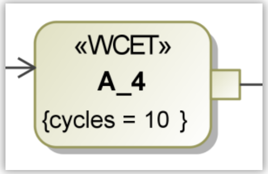


hochschule mannheim
emb-lab

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Performance evaluation



The state space and performance of NuSMV is **strongly influenced** by variables

Variable are required to count **time** (*Real-Time deadlines*) and **tokens** (*Lost of data*)

End

For every $x \in Vertex$
ASSIGN
next(xC) :=
case
$xC = xR$: -1 -- **"idle"**
tokenFlow$= x$ & $xC != -1$: $xC + 1$ -- **"execution"**
-- **"start"**
tokenFlow$= x$ & $xInT \geq ReqTokens$ & $xOutT$: 0
esac;

Beginning

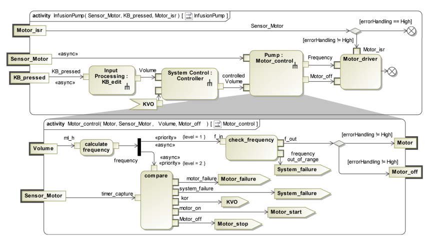
For every $x \in Vertex$
ASSIGN
next($xInT$) :=
case
-- **"finish"**
tokenFlow$= x$ & $xOut = 1$: $xIntT - ReqTokens$
For edge $z = \{y \in Vertex | z = (y,x)\}$
-- **"receiving tokens"**
tokenFlow $= z$ & $zOutT = 1$: $xIntT + 1$
esac:

Result

For every $x \in Vertex$
ASSIGN
next($xOutT$) :=
case
tokenFlow$= x$ & $xC = xR$: 1 -- **"start"**
-- **"finish"**
tokenFlow$= x$ & $xOut = xMaxOutgoings + 1$: 0
For edge $z = \{y \in Vertex | z = (x,y)\}$
-- **"firing"**
tokenFlow$= z$ & $zReady$ & $zOutT = xP$: $xP + 1$
esac;

Verification of system requirements using UPPAAL

- An **infusion pump** is used to administer medicaments or nutrients into a patient's circulatory system. **Errors** in the system can lead to **degradation of the patient's health** or even his death.



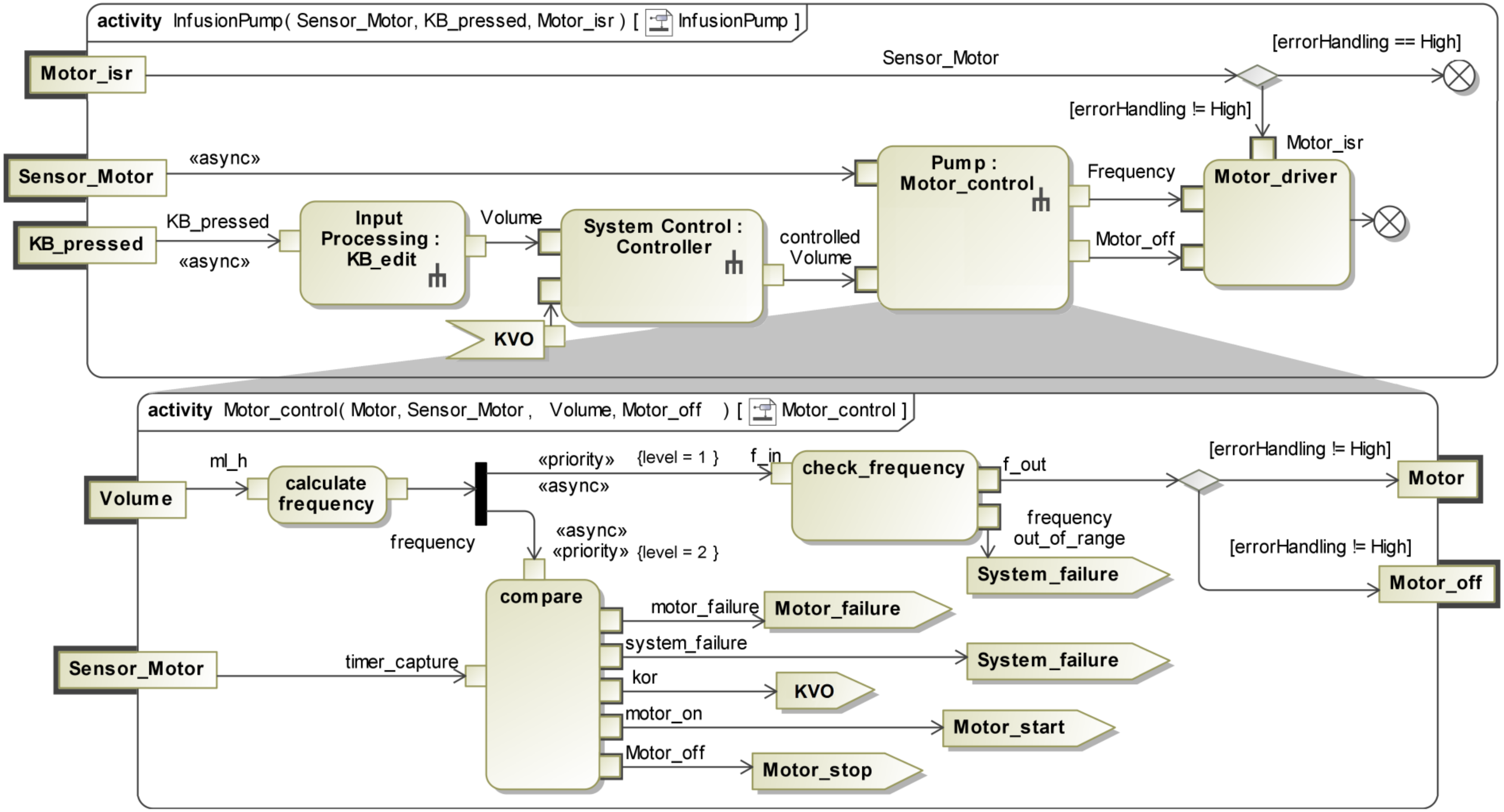
- The control system of the infusion pump is modeled using **UML models** and generated using the **DMOSES** tool. The design is targeted to an ARM7 processor (LPC2368).
- Liveness and **safety requirements** have been verified for the Infusion Pump example.
- The flow graph of this system contains **200 vertices** and **288 edges**.

hochschule mannheim
emb-lab

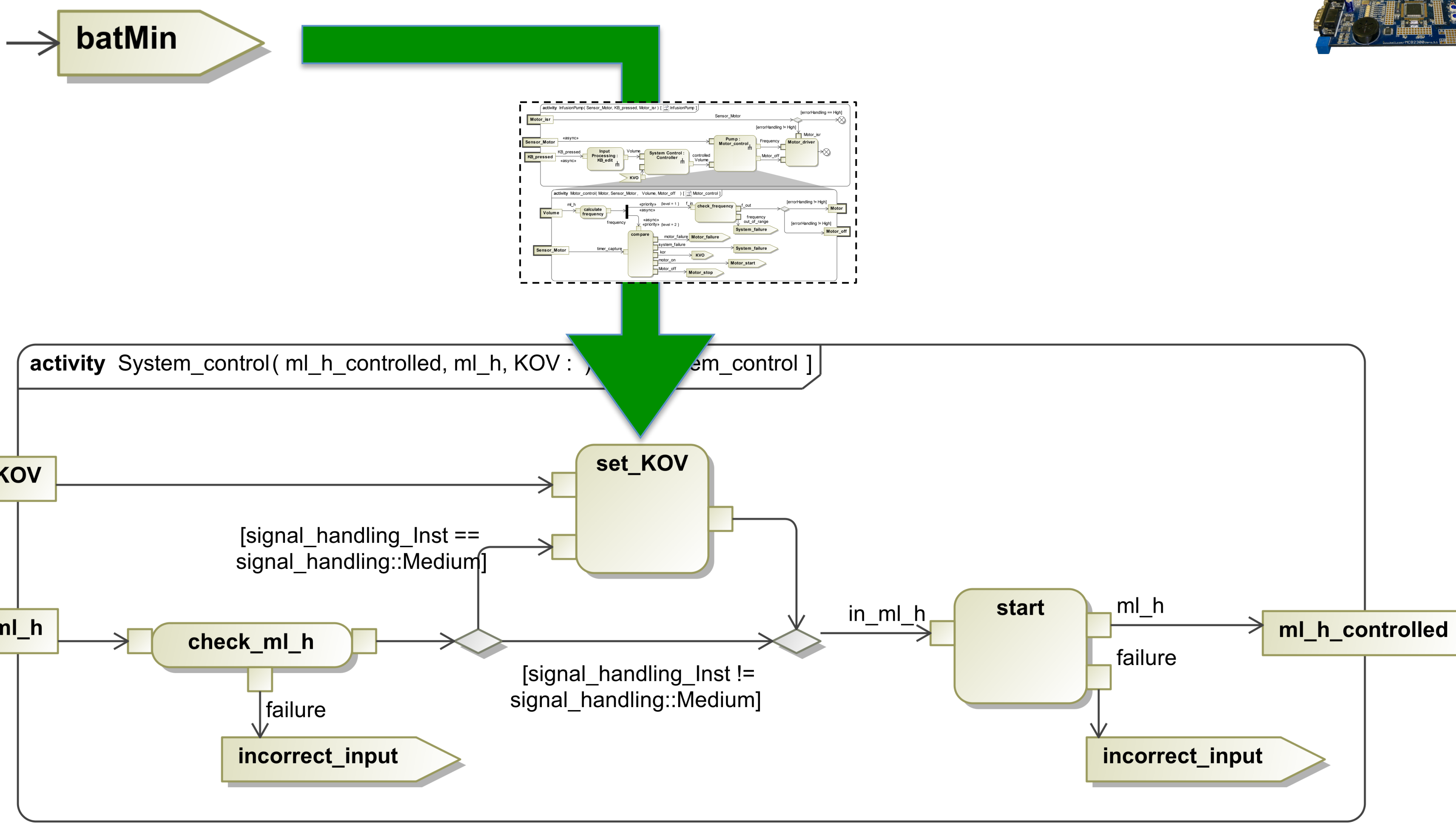
Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

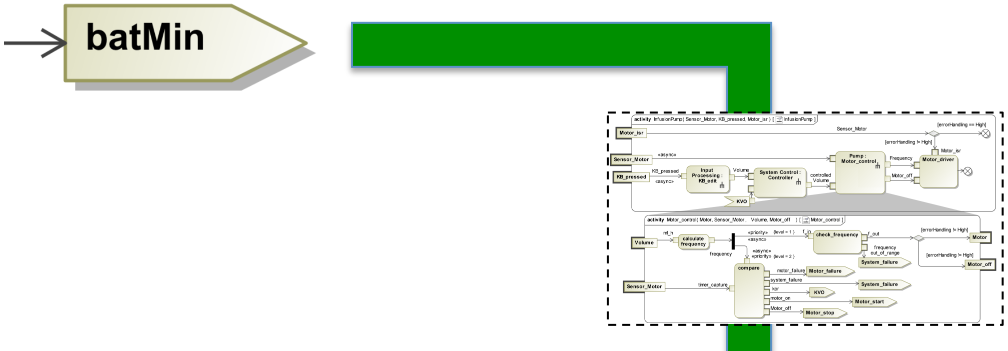
Verification of system requirements using UPPAAL



Verification of system requirements using UPPAAL

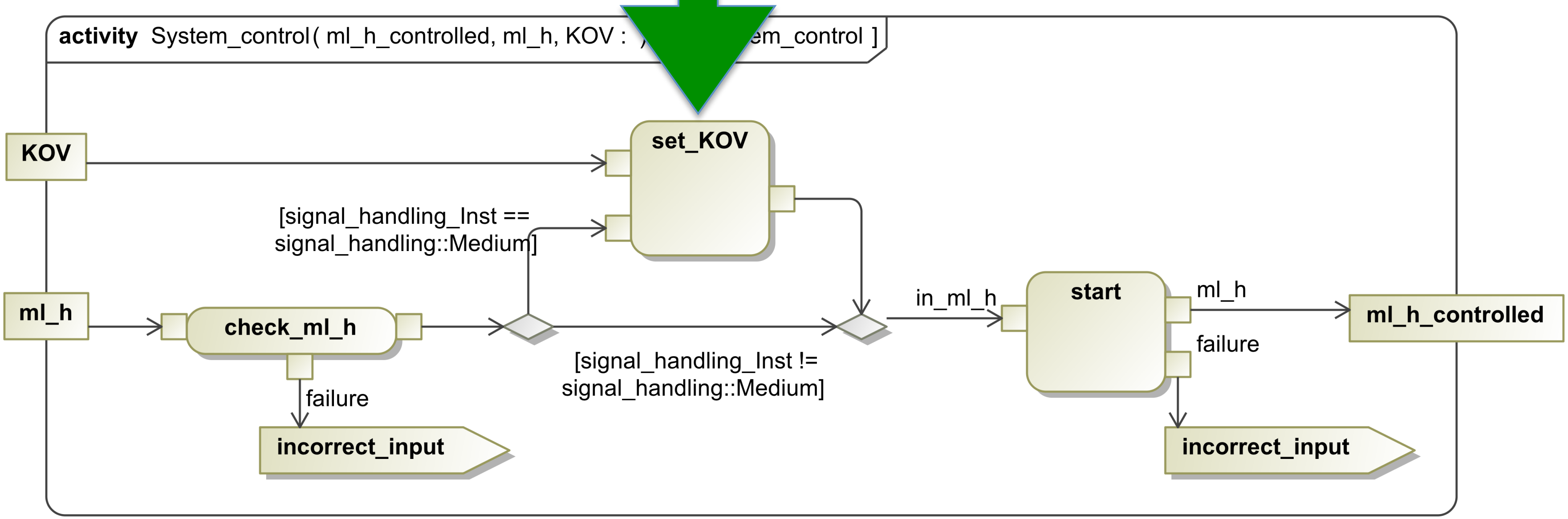


Verification of system requirements using UPPAAL



CTL Formula

$\boxed{\mathbf{A[\,]}\ (\ batMin\ \Rightarrow\ \mathbf{A\langle\rangle}\ setKVO\ )}$



Conclusion

- Automatic **integration** of model checking into a model-driven development
- Description of **UML activitie**s using well-defined mathematic languages
- **Comparison** between **NuSMV** and **UPPAAL** tool chains
- Verification a real case study such as a **infusion pump** developed with DMOSES

Future works

- Verification of **interconnected UML activities and state machines**
- Improve the optimization for hierarchical models
- Inclusion of **best execution time** and **multicore**

hochschule mannheim
emb-lab

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Thank you for your attention

CMACS/AVACS Workshop

Zamira Daw[1], Rance Cleaveland[1], and Marcus Vetter[2]

1.University of Maryland

2.Hochschule Mannheim - University of Applied Sciences

Carnegie Mellon University, November 20-22, 2013

EU & ERDF
Baden-Württemberg