Discrete Event Modeling and Simulation for IoT Efficient Design

Asynchronous software component M&S for simulation-based properties testing: Application to the modeling of FSM execution machine.

L. Capocchi, S. Sehili, J.F. Santucci
University of Corsica - SPE UMR CNRS 6134
capocchi@univ-corse.fr

Workshop on Actuation conflict in IoT
salle du conseil Templiers 1
I3S Lab, Sophia-Antipolis, Nice
Nov. 9, 2018

Outline

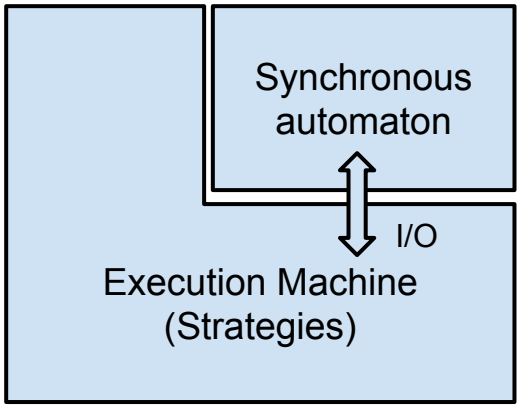
1. Introduction & Context
2. Objectives
3. DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine
4. Case study: The double click mouse system
5. Summary
6. Publications

Introduction & Context (1)

- Ubiquitous application developpement brings issues:
  - Conflicts of execution related to the sharing of the resources and competing access conflicts
  - Interference between components
  - Integration of functionalities in the assembly (appearance / disappearance of the components)
  - …
- Moreover, any application executed in an ambient environment must respect the constraints imposed by the software infrastructure.
- **The challenge in ubiquitous computing applications is to dynamically adapt the behaviors of components in a dynamic execution environment.**

Introduction & Context (2)

- The WCOMP Bean class components include the behavior of the device and its execution machine).
- The compilation allows to obtain a set of library components which are used in a given Assembly.
- However, eventual conflicts due to the connections involved by the Assembly can be detected only after execution:
  - The Designer has to modify the execution machine of some components and restart the design.
- **The main challenge is to define the strategies employed by the execution machine encapsulating the synchronous automaton and handling asynchronous Input/Output.**

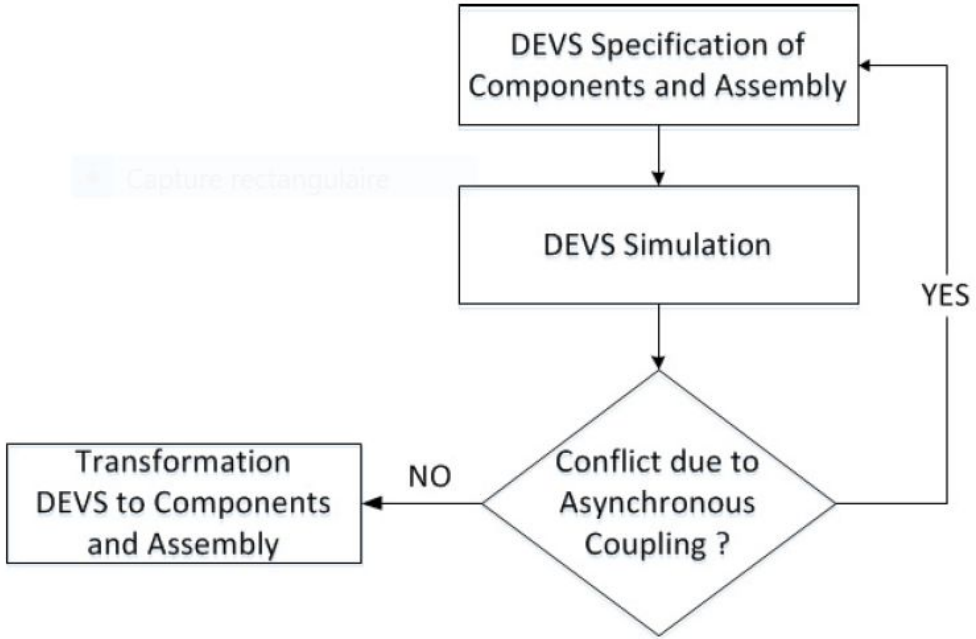


Outline

1. Introduction & Context
2. Objectives
3. DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine
4. Case study: The double click mouse system
5. Summary
6. Publications

Objectives

- We propose a DEVS-based process which consists of helping the designer to:
  - Validate different strategies for execution machines involved in an Assembly.

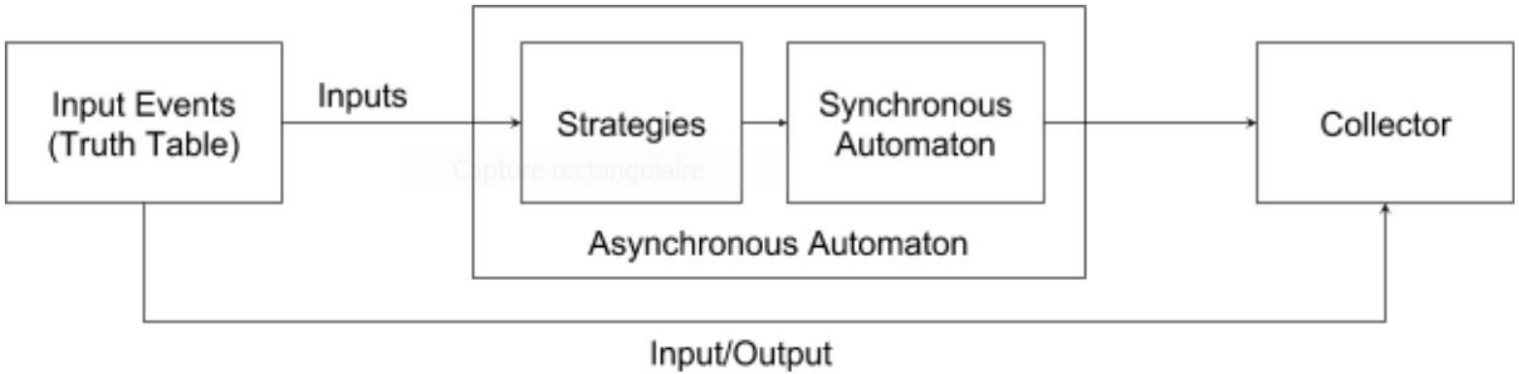


Outline

1. Introduction & Context
2. Objectives
3. DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine
4. Case study: The double click mouse system
5. Summary
6. Publications

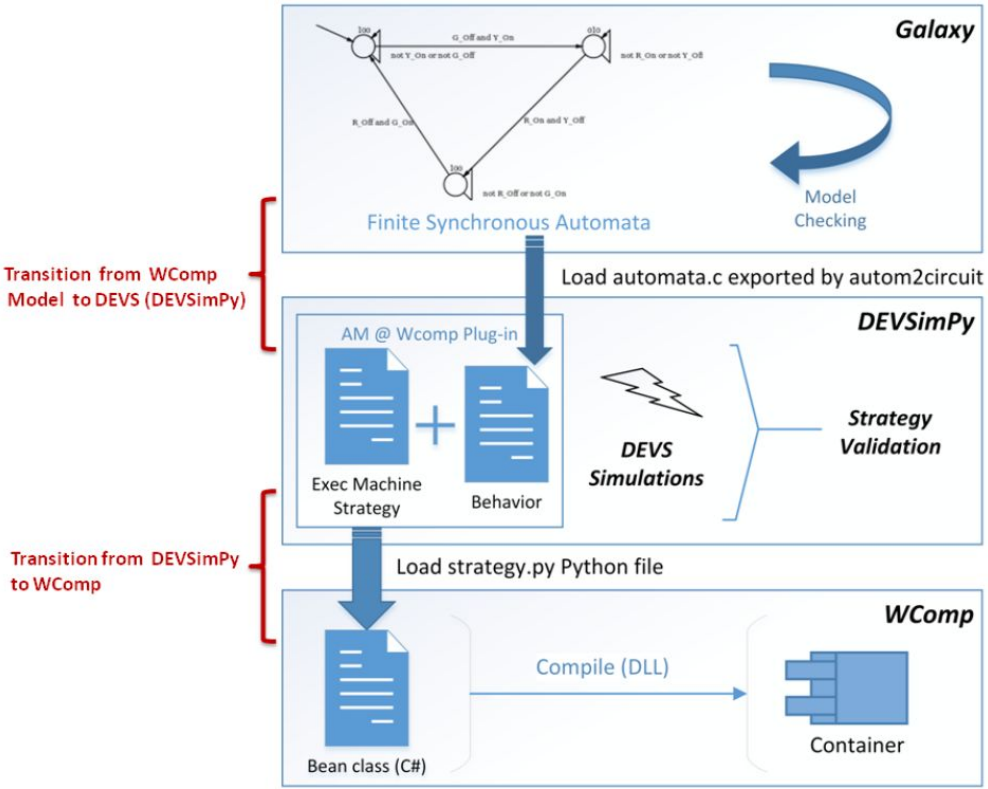
DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine (1)

- DEVS-based Approach



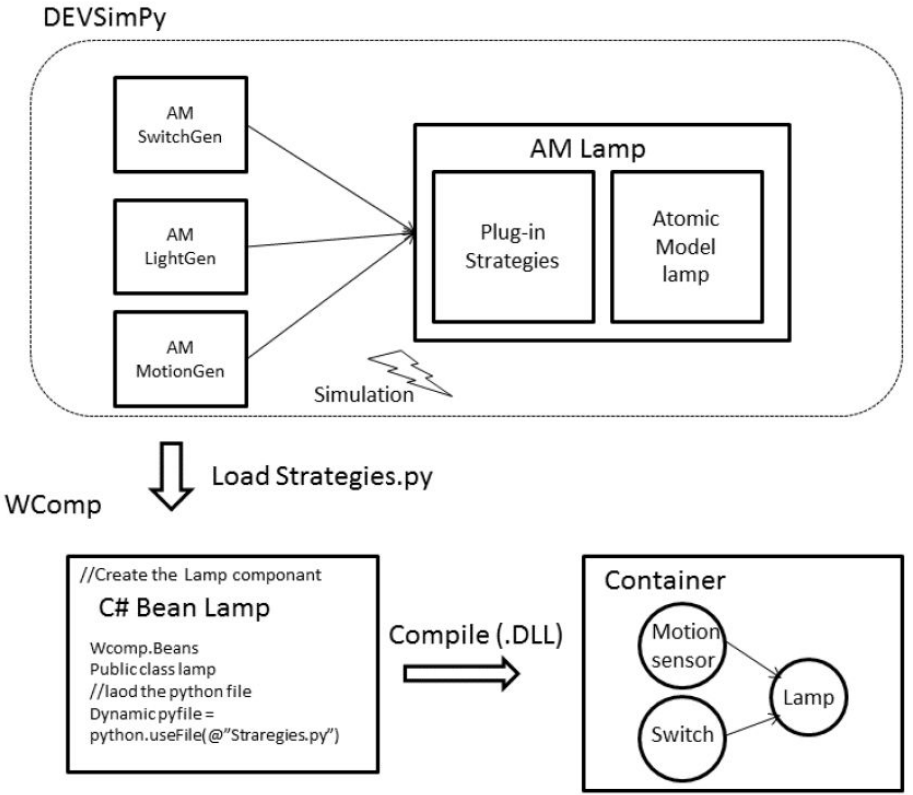
DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine (2)

- Global Scheme of the transition between DEVS and WComp



DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine (3)

- Example of integration of strategies validated with DEVSimPy into WComp.

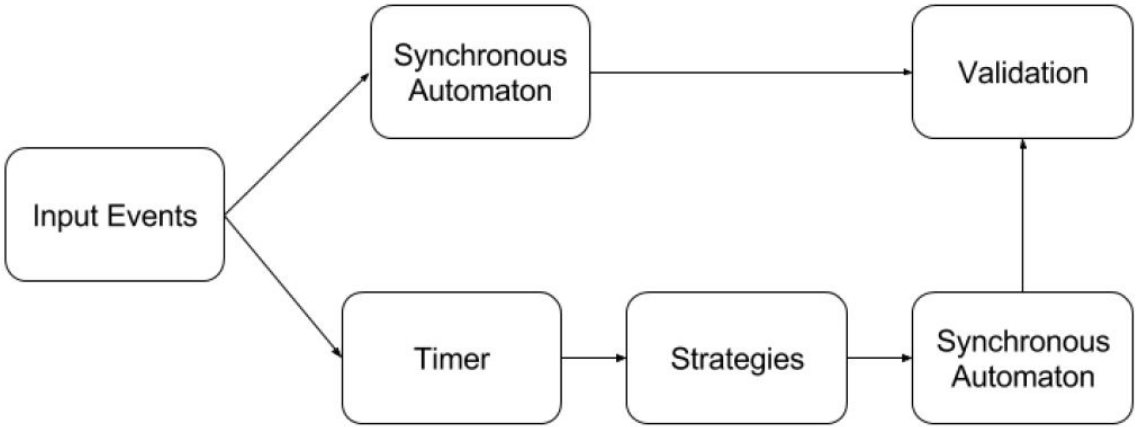


Outline

1. Introduction & Context
2. Objectives
3. DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine
4. Case study: The double click mouse system
5. Summary
6. Publications

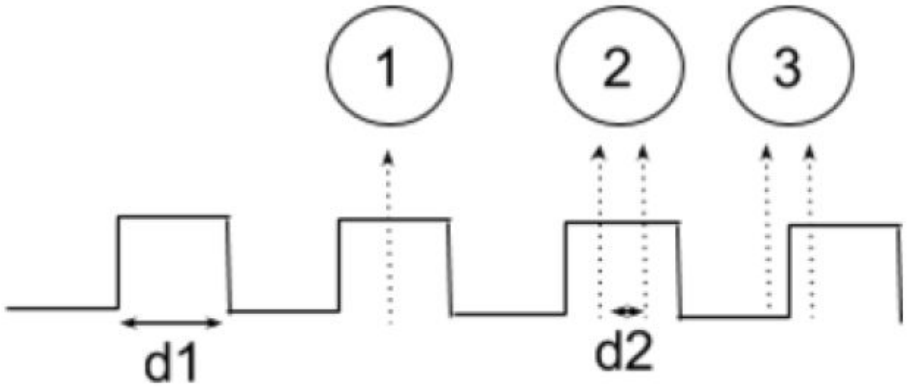
Case study: The double click mouse system (1)

- The proposed approach consists in encapsulating synchronous automata into an asynchronous environment using the DEVS formalism involves
  - Definition of a Mealy automata into DEVS with DEVSimPy environment.
  - Insertion of the synchronous model into an asynchronous environment.
  - Analysis of the results allowing to validate the potential strategies.



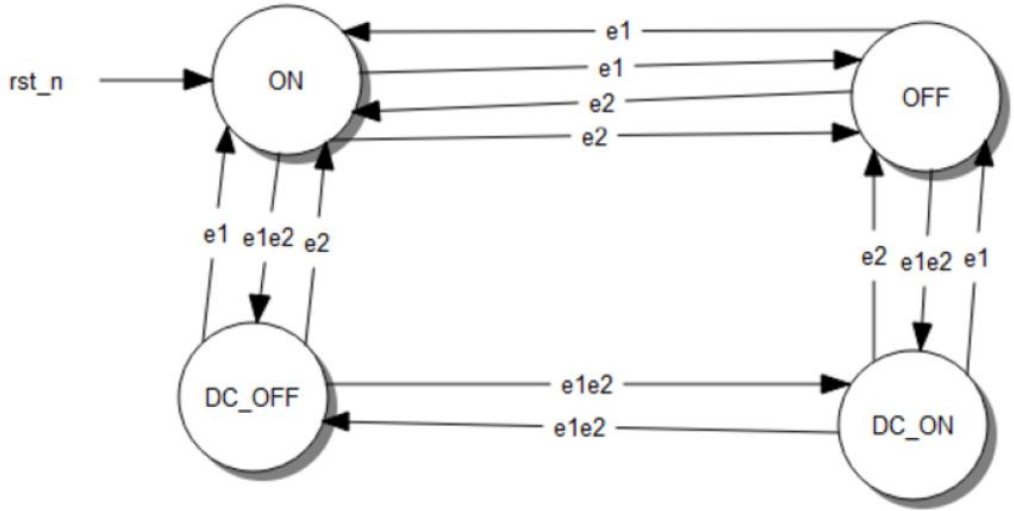
Case study: The double click mouse system (2)

- **Description**
  - (1) the detection of a simple click during the first cycle (with a d1 duration)
  - (2) double click has been detected on the second cycle (with d2<d1).
  - (3) the double click is not detected and will be considered as a simple click.



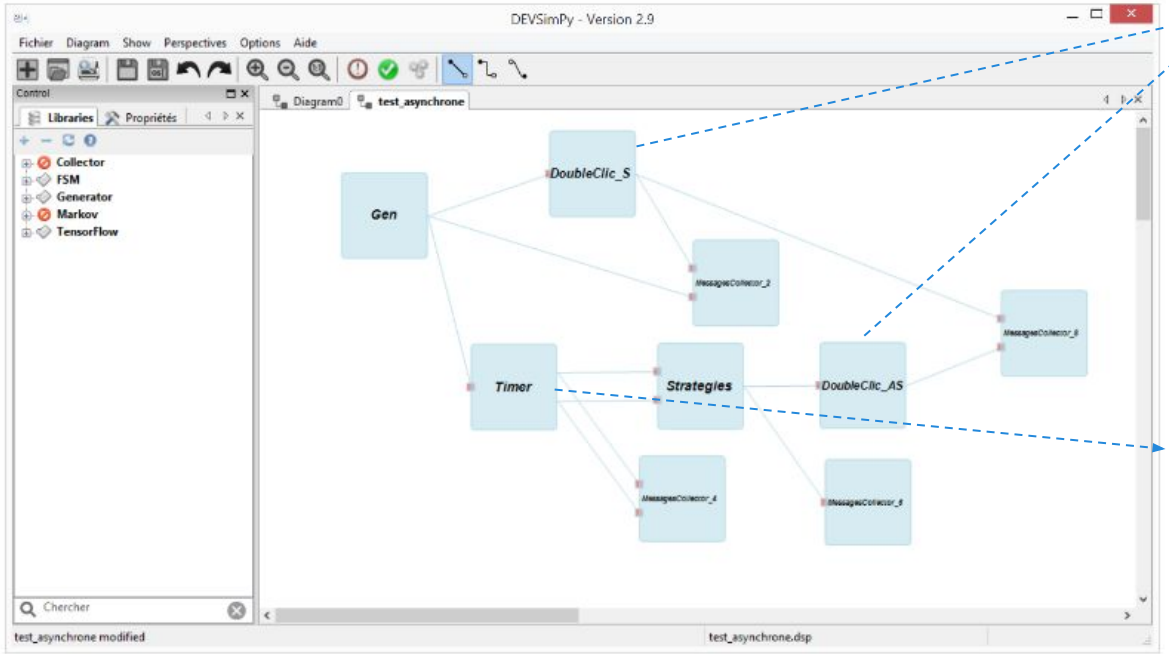
Case study: The double click mouse system (3)

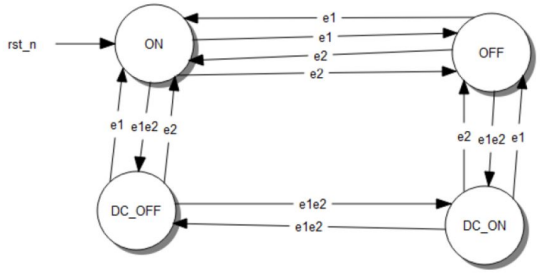
- **DEVS Modeling of FSM**
  - ON and OFF are the two states reached using simple click (e1 or e2);
  - DC_ON and DC_OFF are the two states used to represent double click (e1e2).

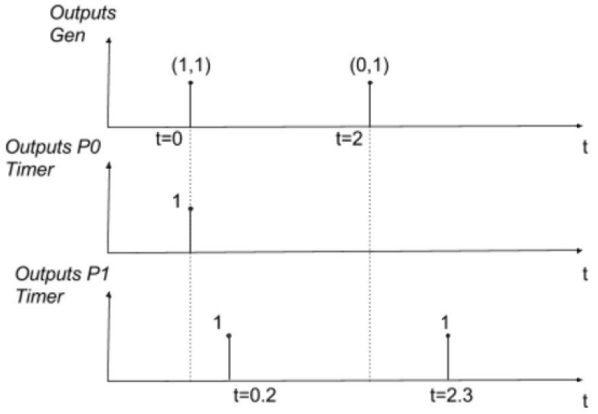


Case study: The double click mouse system (4)

- **DEVS Modeling of FSM**







Case study: The double click mouse system (5)

- **DEVS Simulation results**
  - synchronous simulation results after 10 cycles of simulation and frequency 1

MessagesCollector_2

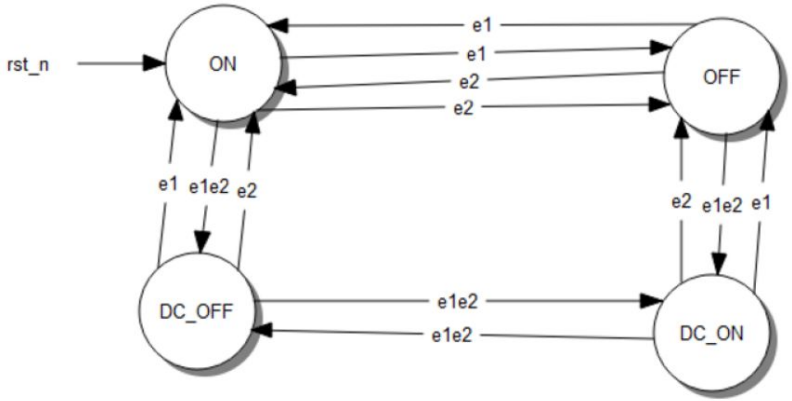
DoubleClic (Port 0) | Gen (Port 0)

	Event	Message
1	0	<< value = (1, 1), time = 0.0>>
2	1	<< value = (1, 1), time = 1.0>>
3	2	<< value = (0, 1), time = 2.0>>
4	3	<< value = (1, 1), time = 3.0>>
5	4	<< value = (1, 0), time = 4.0>>
6	5	<< value = (1, 0), time = 5.0>>
7	6	<< value = (1, 1), time = 6.0>>
8	7	<< value = (1, 0), time = 7.0>>
9	8	<< value = (1, 1), time = 8.0>>
10	9	<< value = (0, 1), time = 9.0>>
11	10	<< value = (0, 0), time = 10.0>>

MessagesCollector_2

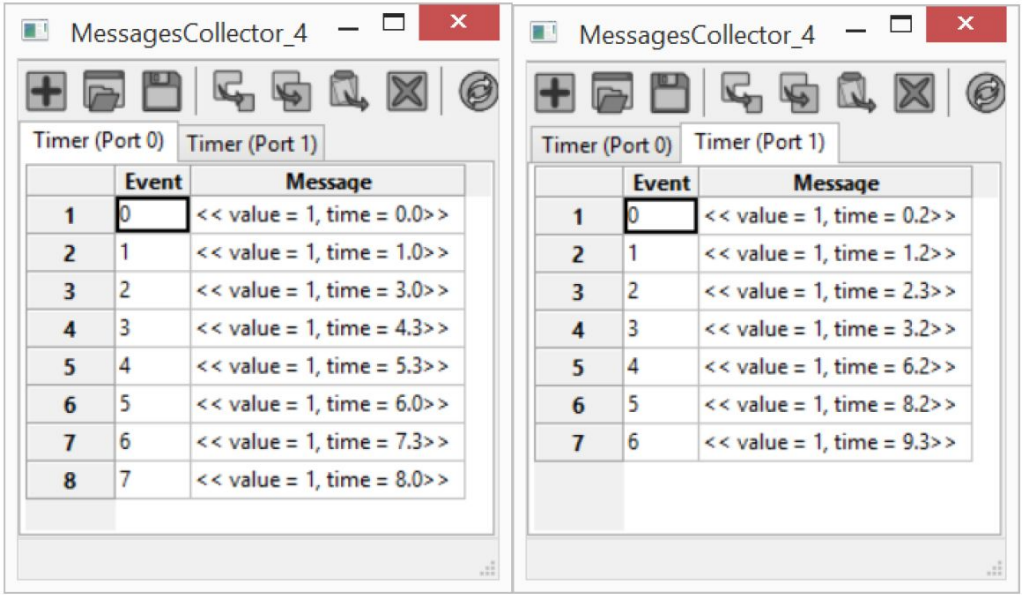
DoubleClic (Port 0) | Gen (Port 0)

	Event	Message
1	0	<< value = DC_OFF, time = 0.0>>
2	1	<< value = DC_ON, time = 1.0>>
3	2	<< value = OFF, time = 2.0>>
4	3	<< value = DC_ON, time = 3.0>>
5	4	<< value = OFF, time = 4.0>>
6	5	<< value = ON, time = 5.0>>
7	6	<< value = DC_OFF, time = 6.0>>
8	7	<< value = ON, time = 7.0>>
9	8	<< value = DC_OFF, time = 8.0>>
10	9	<< value = ON, time = 9.0>>



Case study: The double click mouse system (6)

- **DEVS Simulation results**
  - asynchronous simulation: outputs of the Timer model that introduce time lapse equals to 0.2 between the two events e1 and e2.



MessagesCollector_4

Timer (Port 0) | Timer (Port 1)

	Event	Message
1	0	<< value = 1, time = 0.0>>
2	1	<< value = 1, time = 1.0>>
3	2	<< value = 1, time = 3.0>>
4	3	<< value = 1, time = 4.3>>
5	4	<< value = 1, time = 5.3>>
6	5	<< value = 1, time = 6.0>>
7	6	<< value = 1, time = 7.3>>
8	7	<< value = 1, time = 8.0>>

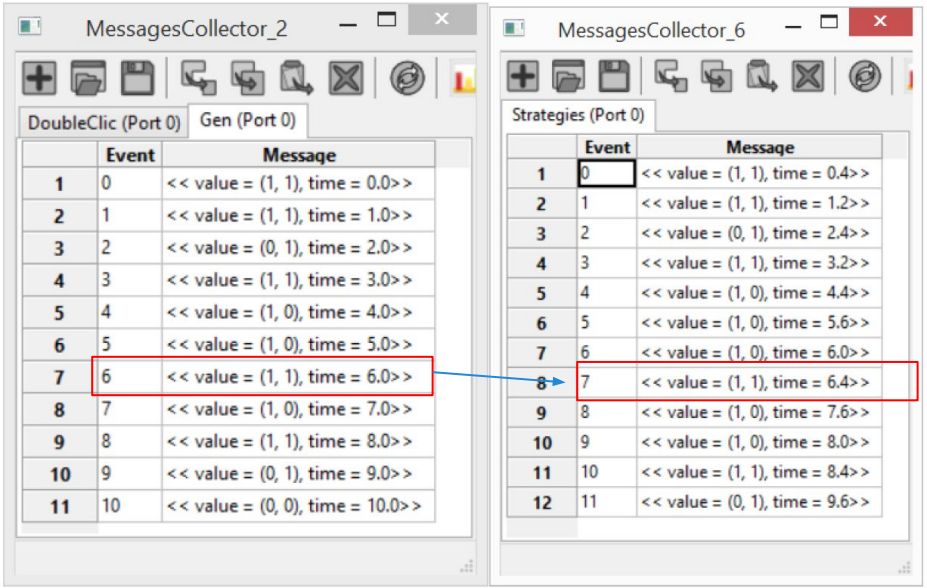
MessagesCollector_4

Timer (Port 0) | Timer (Port 1)

	Event	Message
1	0	<< value = 1, time = 0.2>>
2	1	<< value = 1, time = 1.2>>
3	2	<< value = 1, time = 2.3>>
4	3	<< value = 1, time = 3.2>>
5	4	<< value = 1, time = 6.2>>
6	5	<< value = 1, time = 8.2>>
7	6	<< value = 1, time = 9.3>>

Case study: The double click mouse system (7)

- **DEVS Simulation results**
  - asynchronous simulation: outputs of the Gen and Strategies models after a simulation with the strategy 1 (clk of Strategies model is set to 0.4).



MessagesCollector_2

DoubleClic (Port 0) | Gen (Port 0)

	Event	Message
1	0	<< value = (1, 1), time = 0.0>>
2	1	<< value = (1, 1), time = 1.0>>
3	2	<< value = (0, 1), time = 2.0>>
4	3	<< value = (1, 1), time = 3.0>>
5	4	<< value = (1, 0), time = 4.0>>
6	5	<< value = (1, 0), time = 5.0>>
7	6	<< value = (1, 1), time = 6.0>>
8	7	<< value = (1, 0), time = 7.0>>
9	8	<< value = (1, 1), time = 8.0>>
10	9	<< value = (0, 1), time = 9.0>>
11	10	<< value = (0, 0), time = 10.0>>

MessagesCollector_6

Strategies (Port 0)

	Event	Message
1	0	<< value = (1, 1), time = 0.4>>
2	1	<< value = (1, 1), time = 1.2>>
3	2	<< value = (0, 1), time = 2.4>>
4	3	<< value = (1, 1), time = 3.2>>
5	4	<< value = (1, 0), time = 4.4>>
6	5	<< value = (1, 0), time = 5.6>>
7	6	<< value = (1, 0), time = 6.0>>
8	7	<< value = (1, 1), time = 6.4>>
9	8	<< value = (1, 0), time = 7.6>>
10	9	<< value = (1, 0), time = 8.0>>
11	10	<< value = (1, 1), time = 8.4>>
12	11	<< value = (0, 1), time = 9.6>>

This fault appears due to the lacks of the strategy 1 when a double click event occurs at the end of one analysis cycle.

Case study: The double click mouse system (8)

- **DEVS Simulation results**
  - The strategy 1 is not efficient to resolve the proposed problem of the asynchronous execution of the double click synchronous automata.
  - However, the goal is to show how the DEVS formalism gives a formal framework to implement strategies allowing to deal with critical system computer design problems.

Strategies.py

```
38    def extTransition(self):
39        ''' DEVS external transition function.
40        '''
41
42        ### take inputs
43        e1 = self.peek(self.IPorts[0])
44        e2 = self.peek(self.IPorts[1])
45
46        ### update state
47        self.state['status'] = self.Strategie1(self.delta_double_clic, e1, e2)
48
49        ### update time advance of the new state
50        self.state['sigma'] -= round(self.elapsed,2)
51
52    def outputFnc(self):
57
58    def intTransition(self):
59        ''' DEVS internal transition function.
60        '''
61        self.state['status'] = 'LISTEN'
62        self.state['sigma'] = self.clk
63
64    def Strategie1(self, dc, e1, e2):
65
66        ### append buffer
67        if e1: self.buffer.append((e1.value, 0, e1.time))
68        if e2: self.buffer.append((e2.value, 1, e2.time))
69
70        ### update state
71        if len(self.buffer) > 1:
72            if str(self.buffer[-2][-1]+dc) == str(self.buffer[-1][-1]):
73                new_state = (1,1)
74            else:
75                new_state = (1,0) if self.buffer[-1][1] == 0 else (0,1)
76        elif len(self.buffer) == 1:
77            new_state = (1,0) if self.buffer[0][1] == 0 else (0,1)
78        else:
79            new_state = (0,0)
80
81        return new_state
```

Outline

1. Introduction & Context
2. Objectives
3. DEVS-based M&S of Synchronous Automaton with its Asynchronous Execution Machine
4. Case study: The double click mouse system
5. Summary
6. Publications

Summary

- IoT components result of composition/fusion of synchronous automata that imply concurrent input events (complex system).
- A synchronous automaton is disassociated from its asynchronous execution machine that manages inputs/outputs according to strategies.
- The problem is to know if the behavior of the execution machine will remain in conformity with what the synchronous automaton models while considering additional constraints (properties) in a dynamic environment.
- DEVS is a good candidate to model the asynchronous automaton and its execution machine strategies in order to validate them using discrete event simulation.
- DEVS simulation can be used in the conflict actuation management at the Operational Model level.

Publications

1. S. Souhila, L. Capocchi, J.F. Santucci, S. Lavirotte, J.Y. Tigli, *IoT Efficient Design Using WComp Framework and Discrete Event Modeling and Simulation*, Simulation and Modeling Methodologies - Technologies and Applications, Advances in Intelligent Systems and Computing, Springer, 442:4, 2016, pp. 49-69..
2. S. Souhila, L. Capocchi, J.F. Santucci, *Design and Implementation of Ambient Intelligent Systems using Discrete Event Simulations*, International Journal On Advances in Systems and Measurements, v8, n1-2, 2015, pp. 92-102.
3. S. Sehili, L. Capocchi, J.F. Santucci, S. Lavirotte, J.Y. Tigli, "Discrete Event Modeling and Simulation for IoT Efficient Design Combining WComp and DEVSimPy Framework", in Proc. of 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), July 21-23, 2015, ISBN 978-989-758-120-5, Colmar, Alsace, France, pp. 44-52.
4. S. Sehili, L. Capocchi, J.F. Santucci, "Management of Ubiquitous Systems with a Mobile Application Using Discrete Event Simulations (WIP)", in Proc. of ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, June 10-12, 2015, ISBN 978-1-4503-3583-6, London, UK, pp. 169-170.
5. S. Sehili, L. Capocchi, J. F. Santucci, "IoT Component Design and Implementation using Discrete Event Specification Simulations",, in Proc. of the Sixth International Conference on Advances in System Simulation (SIMUL2014), Oct. 12-16, 2014, Nice (France), ISBN 978-1-61208-371-1, pp. 71-76.
6. S. Sehili, L. Capocchi, J.F. Santucci, Aide à la conception d'architectures IoT sous le Framework WComp par la simulation DEVSimPy, poster à la Journée Des Doctorants, SPE UMR CNRS 6134, 25 juin 2015.