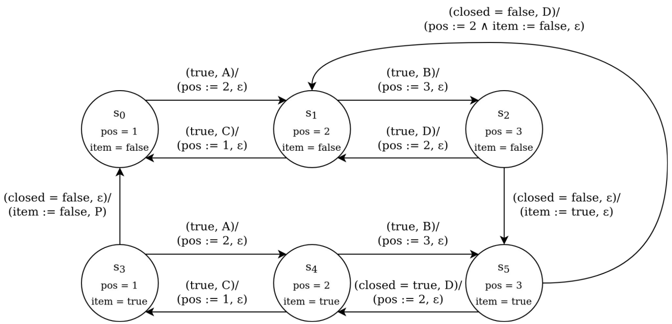
Friedrich-Alexander-Universität
School of Business,
Economics and Society

A RESTful Interaction Model for Semantic Digital Twins*



SeReCo Spring Workshop 2022 23.05.2022

Daniel Schraudner
Chair of Technical Information Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg

* This work has been funded by the German Federal Ministry of Education and Research through the MOSAIK project (Grant no. 01IS18070A).

Agenda

1. Motivation
2. Extended Finite State Machines
3. Interaction Model & REST Interface
4. Conclusion

1. Motivation

Semantic Digital Twin or Semantic Digital Shadow?

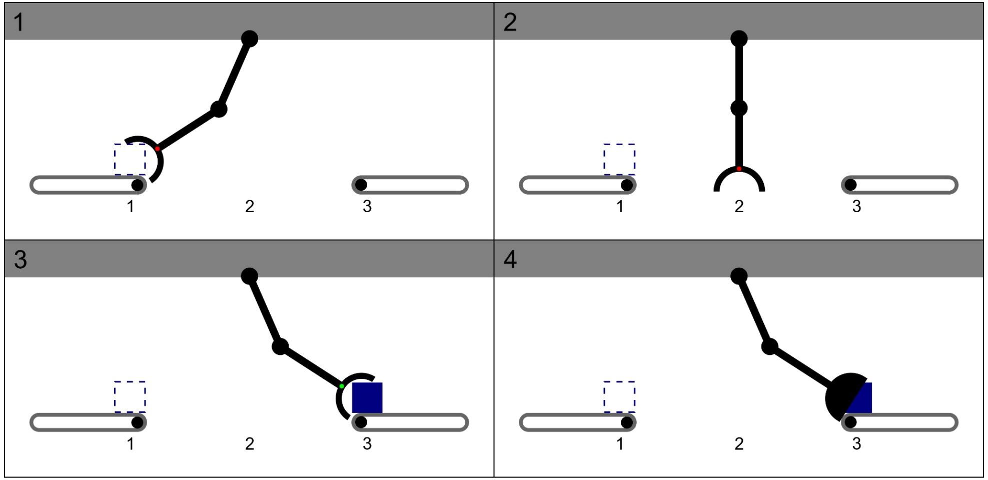
- Digital Twins and Digital Shadows are mixed up quite often
- Digital Shadows comprise only data (e.g. sensor data) that can be accessed (a digital read interface)
- Digital Twins add more functionality on top:
  - It must be possible to interact with the Digital Twin, i.e. send data to some actuators
  - Often additional services are also (e.g. predictive maintenance, simulation) are also part of a digital twin

Semantic Digital Twin or Semantic Digital Shadow?

- Digital Twins and Digital Shadows are mixed up quite often
- Digital Shadows comprise only data (e.g. sensor data) that can be accessed (a digital read interface)
- Digital Twins add more functionality on top:
  - It must be possible to interact with the Digital Twin, i.e. send data to some actuators
  - Often additional services are also (e.g. predictive maintenance, simulation) are also part of a digital twin

For our purpose we define a Semantic Digital Twin as a Read-Write Linked Data interface to an asset

Continuous Example: Robot Arm



Name	Possible Values	Description
pos	1, 2, 3	Current position of the arm
closed	true, false	Whether the clamp is closed
item	true, false	Whether an item can be sensed at the sensor

A Digital Shadow for the Robot Arm is easy!

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix robotarm: <https://solid.ti.rw.fau.de/public/ns/robotArm#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost:8080/#robotArm>
    rdf:type             robotarm:RobotArm ;
    robotarm:pos         1 ;
    robotarm:closed      "true"^^xsd:boolean ;
    robotarm:item        "false"^^xsd:boolean .
```

A Digital Shadow for the Robot Arm is easy!

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix robotarm: <https://solid.ti.rw.fau.de/public/ns/robotArm#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "false"^^xsd:boolean .
```

But how can we control the Robot Arm? With a PUT Request? With a POST Request?

2. Extended Finite State Machines

Extended Finite State Machines (EFSMs)

- First proposed by Cheng & Krishnakumar [1]
- Based on classical Finite State Machines (FSMs)
- EFSMs add variables, transition enabling functions, and variable update transformations to FMSs
- We add an admissibility function for states on top

[1] K.-T. Cheng, A. Krishnakumar, Automatic functional test generation using the extended finite state machine model, in: 30th ACM/IEEE Design Automation Conference, 1993, pp. 86–91. doi:10.1109/DAC.1993.203924 .

EFSM Example

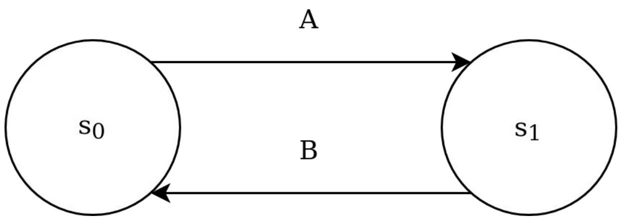
$S = \{s_0, s_1\}$

EFSM Example

$S = \{s_0, s_1\}$

$T = \{(s_0) \rightarrow (s_1), (s_1) \rightarrow (s_0)\}$

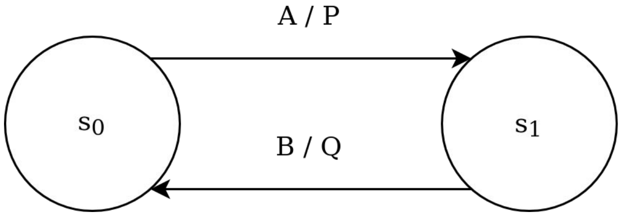
EFSM Example



$S = \{s_0, s_1\}$ $I = \{A, B\}$

$T = \{(s_0, A) \rightarrow (s_1), (s_1, B) \rightarrow (s_0)\}$

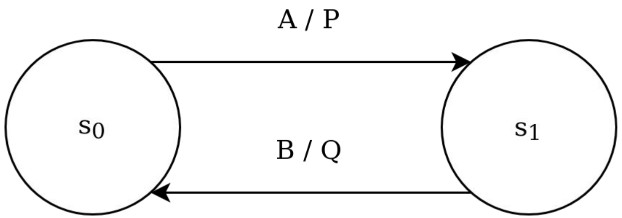
EFSM Example



$S = \{s_0, s_1\}$ $I = \{A, B\}$ $O = \{P, Q\}$

$$T = \{(s_0, A) \rightarrow (s_1, P), (s_1, B) \rightarrow (s_0, Q)\}$$

EFSM Example



$S = \{s_0, s_1\}$

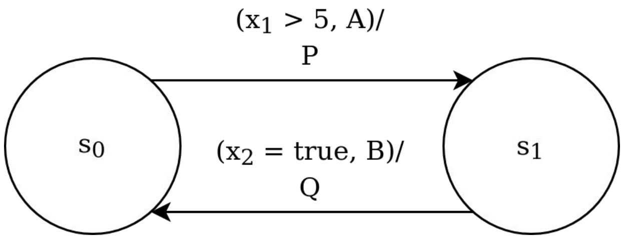
$I = \{A, B\}$

$O = \{P, Q\}$

$D = \mathbb{N} \times \{true, false\}$

$T = \{(s_0, A) \rightarrow (s_1, P), (s_1, B) \rightarrow (s_0, Q)\}$

EFSM Example



$S = \{s_0, s_1\}$

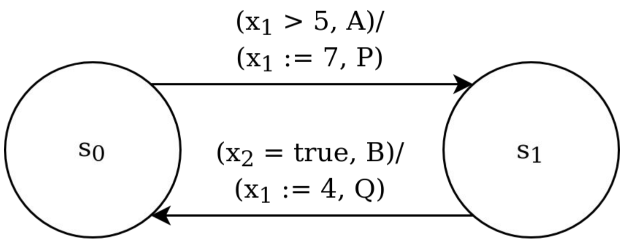
$I = \{A, B\}$

$O = \{P, Q\}$

$D = \mathbb{N} \times \{true, false\}$

$T = \{(s_0, x_1 > 5, A) \rightarrow (s_1, P), (s_1, x_2 = true, B) \rightarrow (s_0, Q)\}$

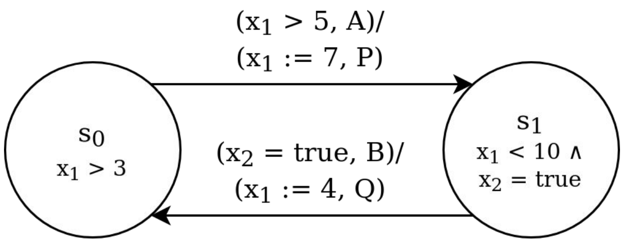
EFSM Example



$S = \{s_0, s_1\}$ $I = \{A, B\}$ $O = \{P, Q\}$ $D = \mathbb{N} \times \{true, false\}$

$T = \{(s_0, x_1 > 5, A) \rightarrow (s_1, x_1 := 7\ P), (s_1, x_2 = true, B) \rightarrow (s_0, x_1 := 4, Q)\}$

EFSM Example



$S = \{s_0, s_1\}$ $I = \{A, B\}$ $O = \{P, Q\}$ $D = \mathbb{N} \times \{true, false\}$

$T = \{(s_0, x_1 > 5, A) \to (s_1, x_1 := 7\ P), (s_1, x_2 = true, B) \to (s_0, x_1 := 4, Q)\}$

$A(s_0) = x_1 > 3$ $A(s_1) = x_1 < 10 \wedge x_2 = true$

Extended Finite State Machines (EFSMs)

We define an EFSM as a 8-tuple $E = (S, I, O, D, F, U, T, A)$, where

- S is the set of symbolic states,
- I is a set of input symbols,
- O is a set of output symbols,
- D is an n-dimensional space $D_1 \times \cdots \times D_n$ (i.e. the value space of all variables),
- F is a set of enabling functions f_i such that $f_i : D \rightarrow \{true, false\}$,
- U is a set of variable update transformations ui such that $u_i : D \rightarrow D$,
- T is a transition relation such that $T : S \times F \times I \rightarrow S \times U \times O$ and
- A is the admissibility function such that $A : S \times D \rightarrow \{true, f\,alse\}$.

3. Interaction Model & REST Interface

Interaction Model

- Interactions with the digital twin can happen by using
  - Properties
  - Actions
  - Events
- Well-established abstractions that are used e.g. in the Web of Things
- Properties can be read and optionally written; when written they must change immediately
- Actions initiate a state change that takes more time
- Events give the information to the outside world that a state change has happened

Properties

Properties are represented using the variables of the EFSM

Our robot arm has three variables that can always be read, but only for *closed* it makes sense to be writable!
Also: Not every integer should be allowed for *pos.*

Name	Possible Values	Description	Interactions
pos	1, 2, 3	Current position of the arm	Read
closed	true, false	Whether the clamp is closed	Read & Write
item	true, false	Whether an item can be sensed at the sensor	Read

Properties

Properties are represented using the variables of the EFSM

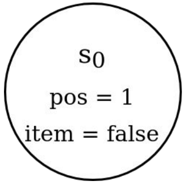
Our robot arm has three variables that can always be read, but only for *closed* it makes sense to be writable!
Also: Not every integer should be allowed for *pos.*

How can we define in which ways properties are allowed to be changed?

Name	Possible Values	Description	Interactions
pos	1, 2, 3	Current position of the arm	Read
closed	true, false	Whether the clamp is closed	Read & Write
item	true, false	Whether an item can be sensed at the sensor	Read

Properties

The admissibility functions defines which variable values are allowed in which state




```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "false"^^xsd:boolean ;
    robotarm:item       "false"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

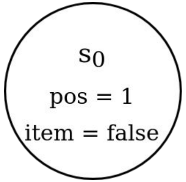
```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "false"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

HTTP PUT
http://localhost:8080/

HTTP GET
http://localhost:8080/

Properties

The admissibility functions defines which variable values are allowed in which state




```
<http://localhost:8080/#robotArm>
    rdf:type             robotarm:RobotArm ;
    robotarm:pos         1 ;
    robotarm:closed      "true"^^xsd:boolean ;
    robotarm:item        "false"^^xsd:boolean ;
    robotarm:tasks       <http://localhost:8080/tasks/> ;
    robotarm:events      <http://localhost:8080/events/> .
```

HTTP GET
http://localhost:8080/

```
<http://localhost:8080/#robotArm>
    rdf:type             robotarm:RobotArm ;
    robotarm:pos         1 ;
    robotarm:closed      "false"^^xsd:boolean ;
    robotarm:item        "false"^^xsd:boolean ;
    robotarm:tasks       <http://localhost:8080/tasks/> ;
    robotarm:events      <http://localhost:8080/events/> .
```

→

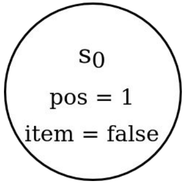
HTTP PUT
http://localhost:8080/

```
<http://localhost:8080/#robotArm>
    rdf:type             robotarm:RobotArm ;
    robotarm:pos         1 ;
    robotarm:closed      "false"^^xsd:boolean ;
    robotarm:item        "false"^^xsd:boolean ;
    robotarm:tasks       <http://localhost:8080/tasks/> ;
    robotarm:events      <http://localhost:8080/events/> .
```

200 OK

Properties

The admissibility functions defines which variable values are allowed in which state




```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "true"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

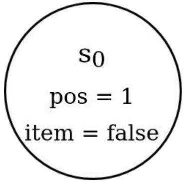
```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "false"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

HTTP PUT
http://localhost:8080/

HTTP GET
http://localhost:8080/

Properties

The admissibility functions defines which variable values are allowed in which state




```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "false"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

HTTP GET
http://localhost:8080/

```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "true"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

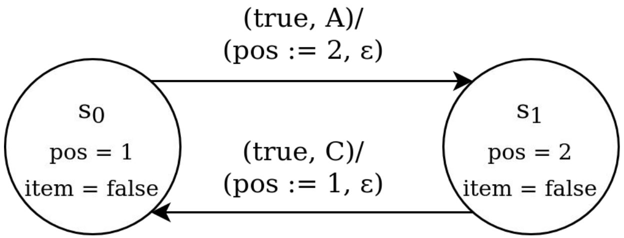
HTTP PUT
http://localhost:8080/

```
<http://localhost:8080/#robotArm>
    rdf:type            robotarm:RobotArm ;
    robotarm:pos        1 ;
    robotarm:closed     "true"^^xsd:boolean ;
    robotarm:item       "false"^^xsd:boolean ;
    robotarm:tasks      <http://localhost:8080/tasks/> ;
    robotarm:events     <http://localhost:8080/events/> .
```

409 Conflict

Actions

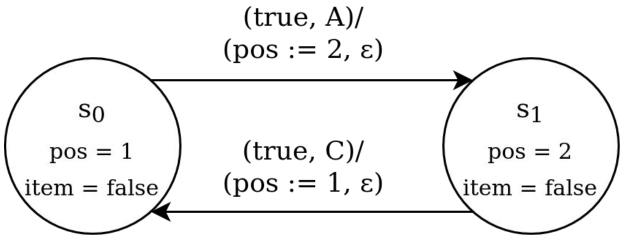
Changing the position of the robot arm cannot happen instantaneously because the arm has to move physically



Actions map directly to one (or multiple) input symbols. Input symbols activate transitions that execute a variable update transformation.

Actions

Actions can be requested by submitting a Task to the interface



Input Symbol	URI
A	robotarm:MoveFrom1To2Task
C	robotarm:MoveFrom2To1Task

```
<http://localhost:8080/tasks/>
    rdf:type            robotarm:TaskContainer .
```

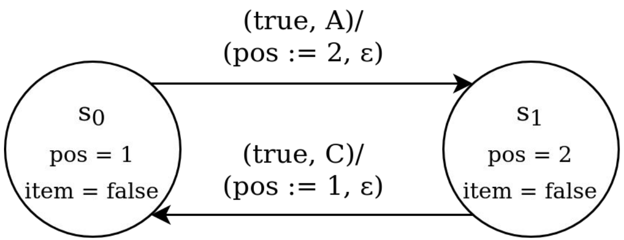
HTTP GET
http://localhost:8080/tasks/

```
[]
   rdf:type            robotarm:MoveFrom1To2Task .
```

HTTP POST
http://localhost:8080/tasks/

Actions

Actions can be requested by submitting a Task to the interface



Input Symbol	URI
A	robotarm:MoveFrom1To2Task
C	robotarm:MoveFrom2To1Task

```
<http://localhost:8080/tasks/>
    rdf:type              robotarm:TaskContainer .
```

HTTP GET
http://localhost:8080/tasks/

```
[]
    rdf:type              robotarm:MoveFrom1To2Task .
```

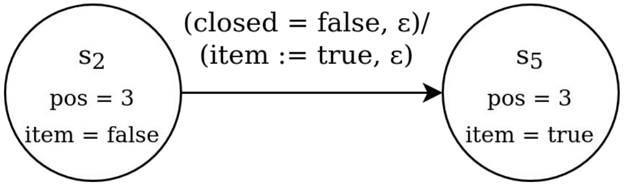
HTTP POST
http://localhost:8080/tasks/

```
<http://localhost:8080/tasks/0>
    rdf:type              robotarm:MoveFrom1To2Task ;
    robotarm:taskNumber 0 ;
    robotarm:taskState  robotarm:running .
```

201 Created

Epsilon Transitions

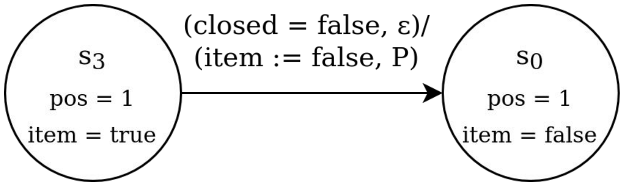
An EFSM can also contain transitions with the empty input symbol ε. Those transitions can happen spontaneously anytime they are activated.



Epsilon transitions can be used to model state changes from the outside (that are not part of the interface), e.g. when a conveyor belt delivers an item to the robot arm.

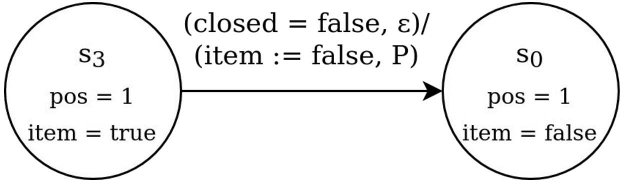
Events

Output symbols map directly to an event that will be made available by the interface



Events

Events can be retrieved from the event container



Output Symbol	URI
P	robotarm:DeliveredItemEvent

```
<http://localhost:8080/events/>
    rdf:type              robotarm:EventContainer ;
    ldp:contains          :0 .
```

HTTP GET
http://localhost:8080/events/

HTTP GET
http://localhost:8080/events/0

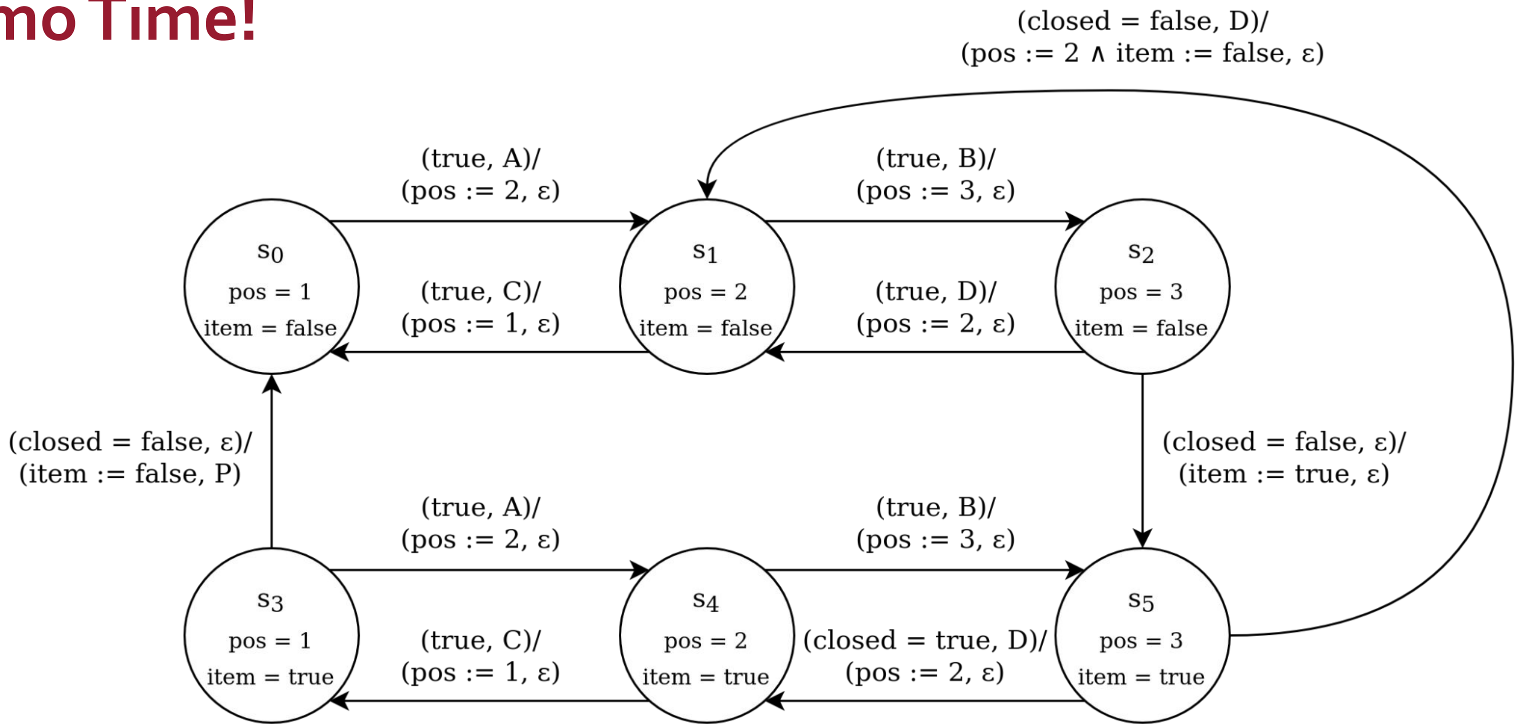
```
<http://localhost:8080/events/0>
    rdf:type              robotarm:DeliveredItemEvent ;
    robotarm:eventNumber  0 ;
    robotarm:eventTime    "2022-05-16T09:41:33.054Z"^^xsd:dateTimeStamp .
```

4. Conclusion

Conclusion

- We provide a Read-Write Linked Data interaction model for Semantic Digital Twins based on a formal model
- We extended EFSMs by an admissibility function to describe how properties can change in a certain state
- We provided a clearly defined REST API for properties, actions, and events and a direct mapping to the EFSM formalism
- We have a working implementation that creates a REST interface for any declaratively defined EFSM

Demo Time!


(closed = false, D)/
(pos := 2 ∧ item := false, ε)
(true, A)/
(pos := 2, ε)
(true, B)/
(pos := 3, ε)
s0
pos = 1
item = false
s1
pos = 2
item = false
s2
pos = 3
item = false
(true, C)/
(pos := 1, ε)
(true, D)/
(pos := 2, ε)
(closed = false, ε)/
(item := false, P)
(closed = false, ε)/
(item := true, ε)
(true, A)/
(pos := 2, ε)
(true, B)/
(pos := 3, ε)
s3
pos = 1
item = true
s4
pos = 2
item = true
s5
pos = 3
item = true
(true, C)/
(pos := 1, ε)
(closed = true, D)/
(pos := 2, ε)

Thank you for your attention!