Formal Verification of Cyber-Physical Systems

Marjan Sirjani

Cyber-Physical Systems Analysis Group

Mälardalen University

Västerås, Sweden

Feb. 19, 2024

NTNU: Norwegian University of Science and Technology

Trondheim, Norway

Acknowledgment: **Edward Lee, UC Berkeley**
Acknowledgment: **All the Rebeca Team**

Background

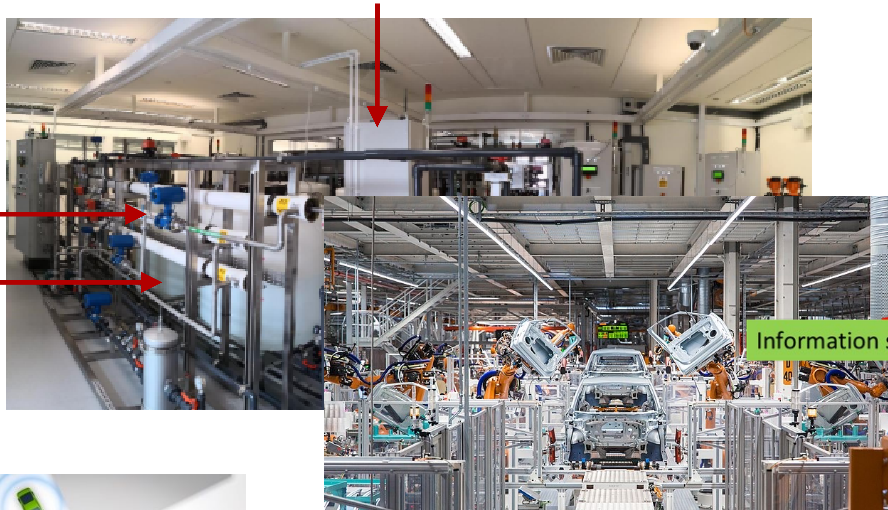
- Distributed Systems and Actors since 2000
  - Carolyn Talcott (SRI), Gul Agha (UIUC) since 2005
- Concurrency Theory and Formal verification since 2000
  - Mohammad Reza Mouasavi (King's College London), Christel Baier (UT Dresden) since 2003
- Coordination Languages since 2003
  - Farhad Arbab, Frank de Boer, Jan Rutten (CWI) since 2003
- Timed and Cyber-Physical Systems since 2007
  - Edward Lee (UC Berkeley) since 2015

Recent Projects and experience with industry

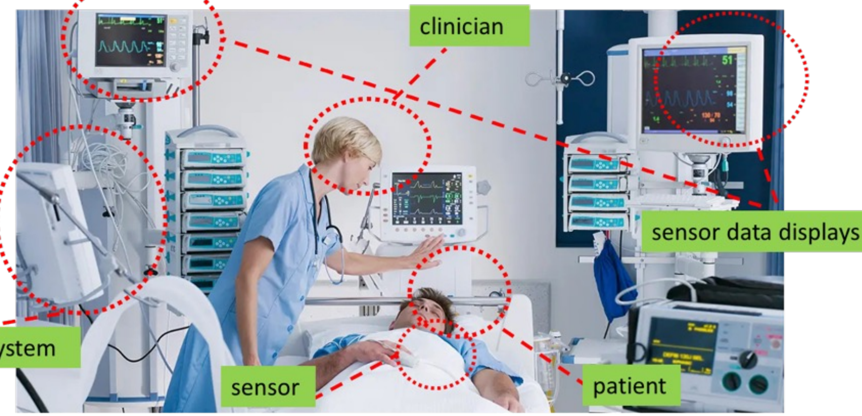
- Serendipity: Secure and Dependable Platforms for Autonomy (SSF- 2018-2024), VCE
- SACSys: Safe and Secure Adaptive Collaborative Systems (KKS - 2019-2024), VCE, Volvo GTO, Volvo Cars, ABB Robotics
- DPAC: Dependable Platforms for Autonomous systems and Control (KKS – 2015-2023), 12 companies …

Cyber-Physical Systems Everywhere!

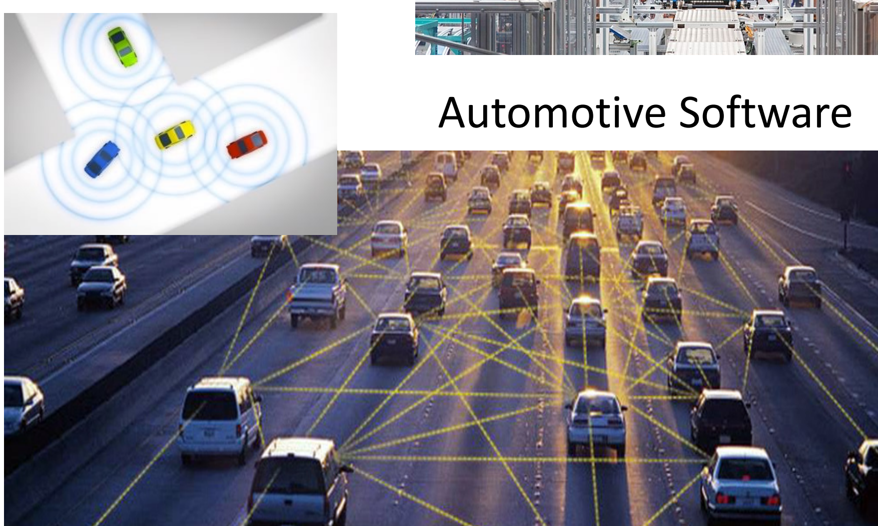
Industrial Automation



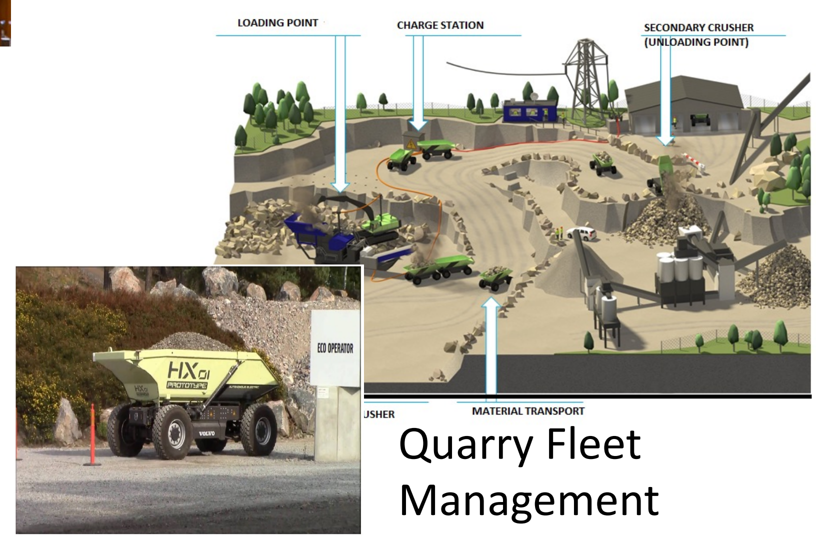
Interoperable Medical Devices



Automotive Software



Quarry Fleet Management



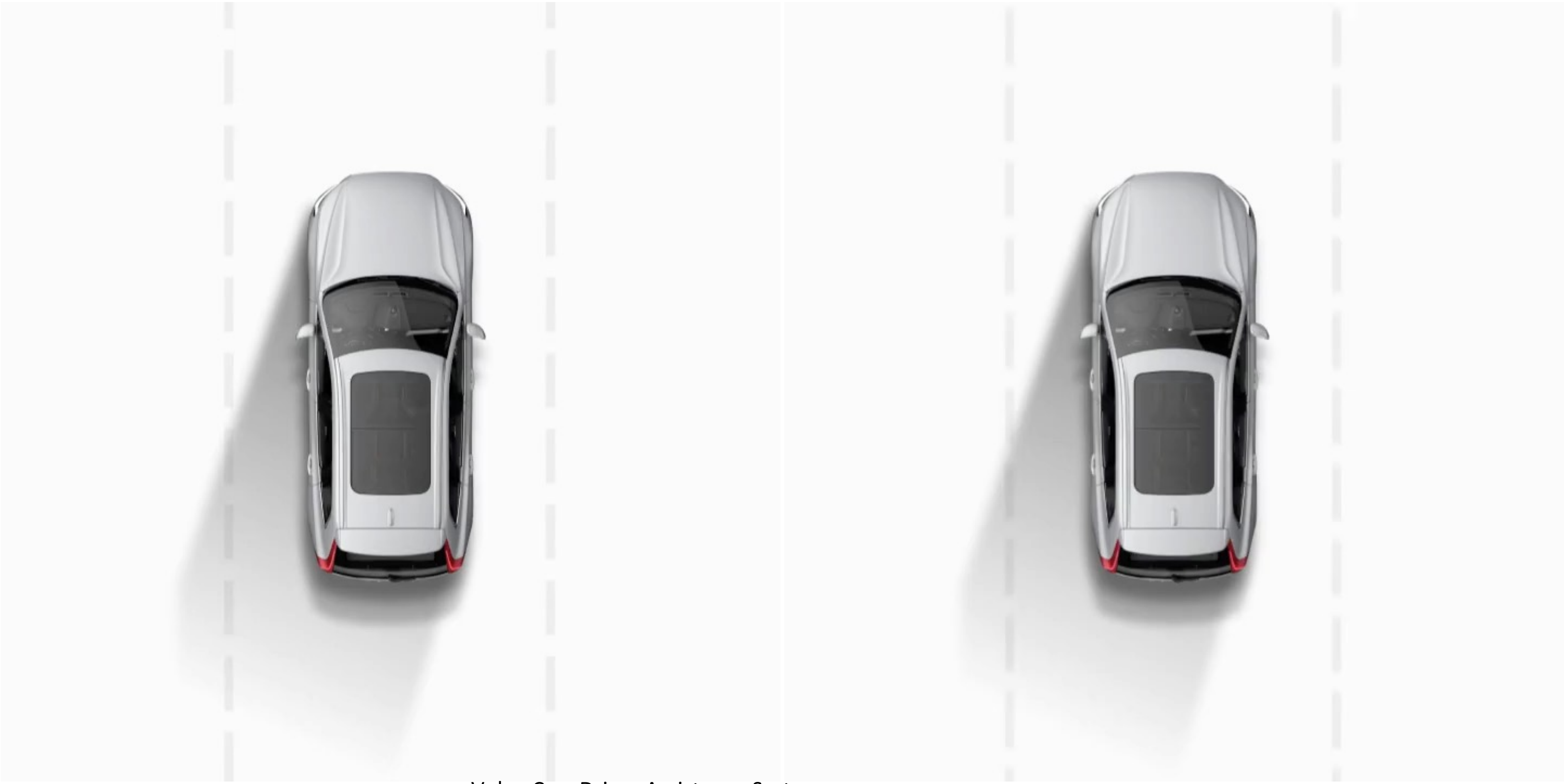
Complex Systems: Connected via network, and Time-Sensitive

Vehicle-2-Everything(V2X) Communication

Volvo Cars Driver Assistance Systems
https://www.volvocars.com/intl/v/car-safety/driver-assistance

Complex Systems: Connected via network, and Time-Sensitive

Vehicle-2-Everything(V2X) Communication



Volvo Cars Driver Assistance Systems
https://www.volvocars.com/intl/v/car-safety/driver-assistance

Complex Systems: Connected via network, and Time-Sensitive

Collaboration of Robots and Humans

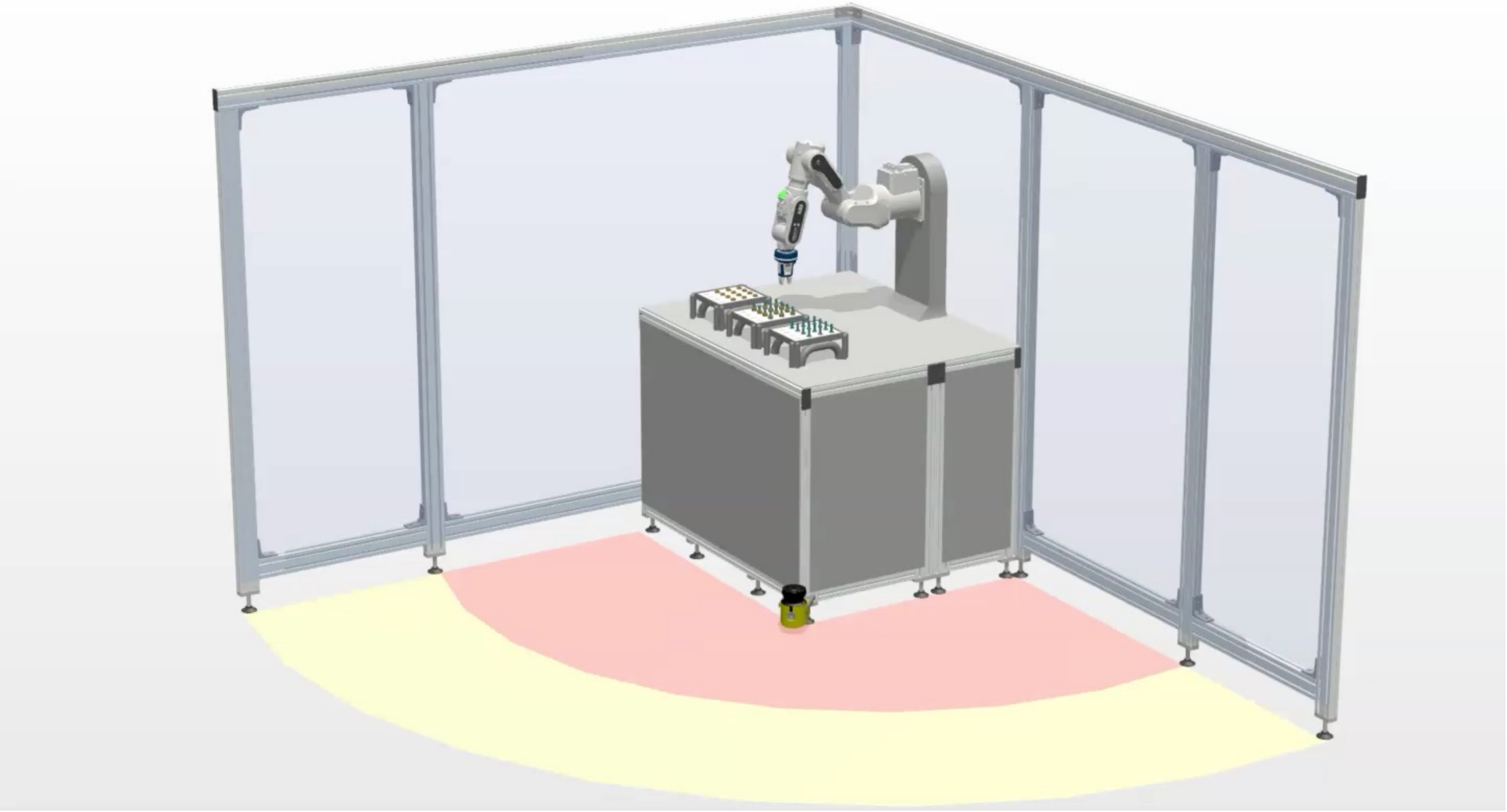


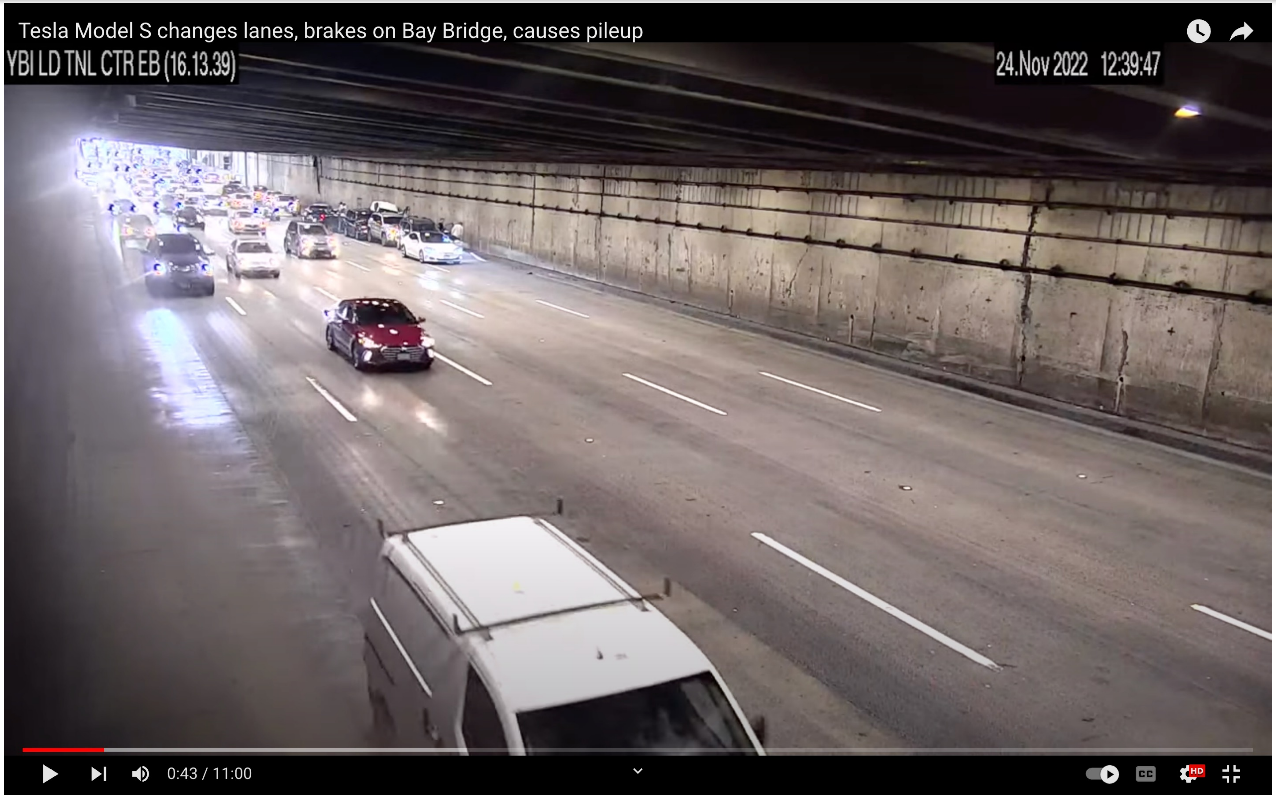
ABB Robotics
https://applicationbuilder.robotics.abb.com/en/home

Can We Trust Self-Driving Cars?

Tesla’s new “Full Self-Driving” feature decided to change lanes and then brakes and stops on the Bay Bridge

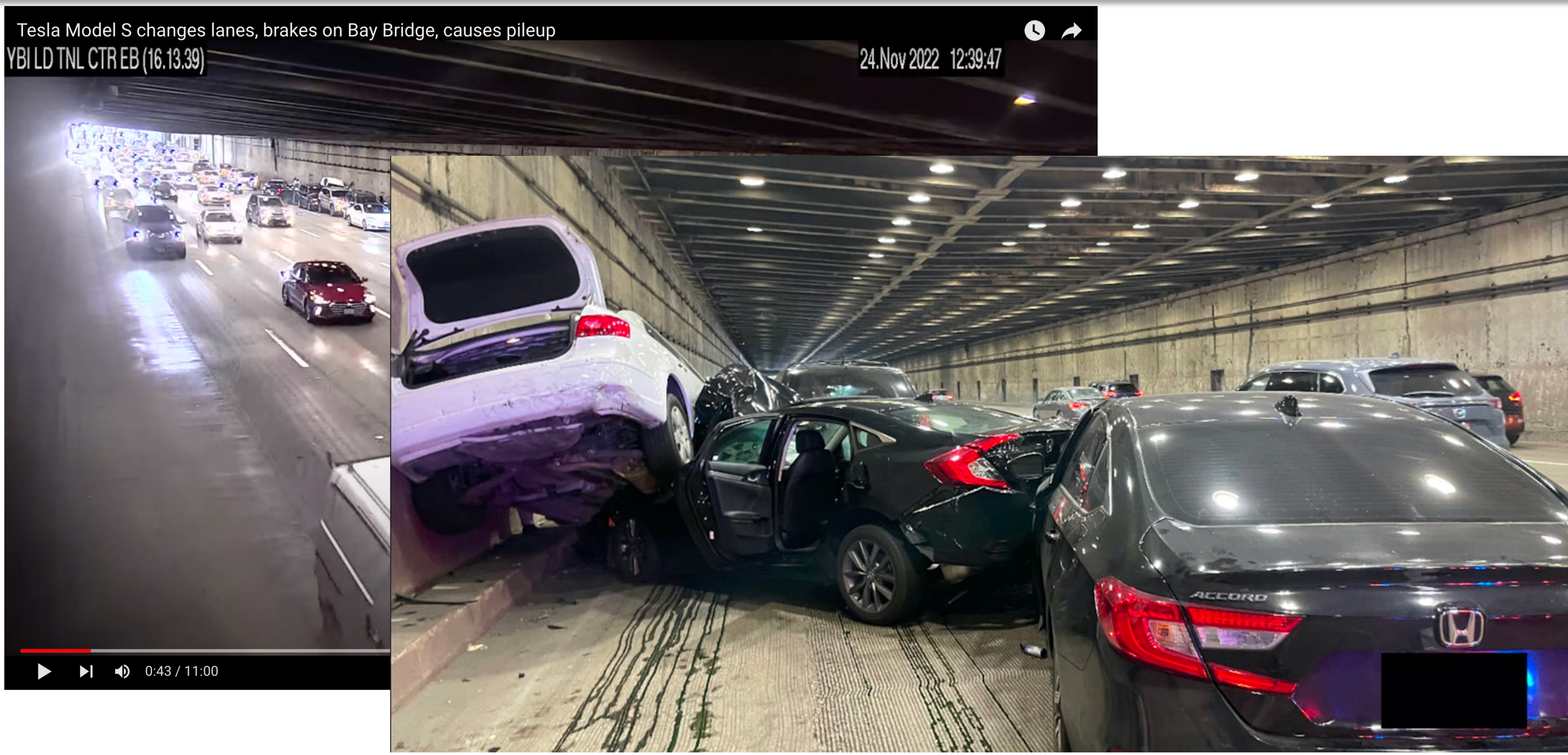
https://theintercept.com/2023/01/10/tesla-crash-footage-autopilot/
https://www.youtube.com/watch?v=WYpzk6TEViQ

Tesla's new "Full Self-Driving" feature decided to changes lanes and then brakes and stops on the Bay Bridge



TESLA CRASH - An eight-car pileup on Nov. 24, 2022, on San Francisco's Bay Bridge. Photo: California Highway Patrol
https://theintercept.com/2023/01/10/tesla-crash-footage-autopilot/
https://www.youtube.com/watch?v=WYpzk6TEViQ

Tesla’s new “Full Self-Driving” feature decided to changes lanes and then brakes and stops on the Bay Bridge



TESLA CRASH - An eight-car pileup on Nov. 24, 2022, on San Francisco’s Bay Bridge. Photo: California Highway Patrol

https://theintercept.com/2023/01/10/tesla-crash-footage-autopilot/

https://www.youtube.com/watch?v=WYpzk6TEViQ

Much older incidents

NASA's Toyota Study (US Dept. of Transportation, 2011) found that Toyota software was “untestable.”

Possible victim of unintended acceleration

Industrial robot crushes man to death in South Korean distribution centre

Nov. 10, 2023

The Guardian

Machine identified man inspecting it as one of the boxes it was stacking

BUT …
Cyber-Physical Systems are helping …

- Smart cars help!
- Our not very smart car prevented a few accidents already!

BUT …
Cyber-Physical Systems are helping …

- Smart cars help!
- Our not very smart car prevented a few accidents already!

We just need better methods to assure safety.

Example: What if you have two tasks where the order is important?

What happens when you forget to disarm the airplane doors!

[The Telegraph, 9 Sept. 2015](https://www.telegraph.co.uk/travel/news/What-happens-when-you-forget-to-disarm-the-plane-doors/)
https://www.telegraph.co.uk/travel/news/What-happens-when-you-forget-to-disarm-the-plane-doors/

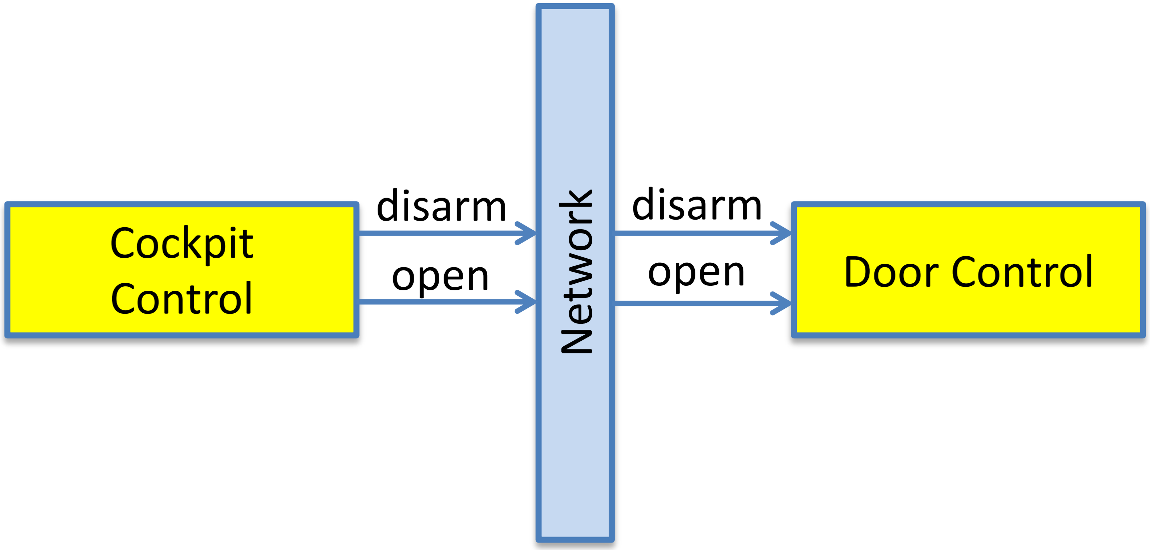
From Professor Edward Lee, UC Berkeley

Physics, Software, Network

Using Software instead of the pilot and the cabin crew, and a network in between.

Cyber-Physical Systems: Control Physical Components using Software through Network

Concurrency and timing problems.



A module that can receive either of two messages:

1. “open”
2. “disarm”

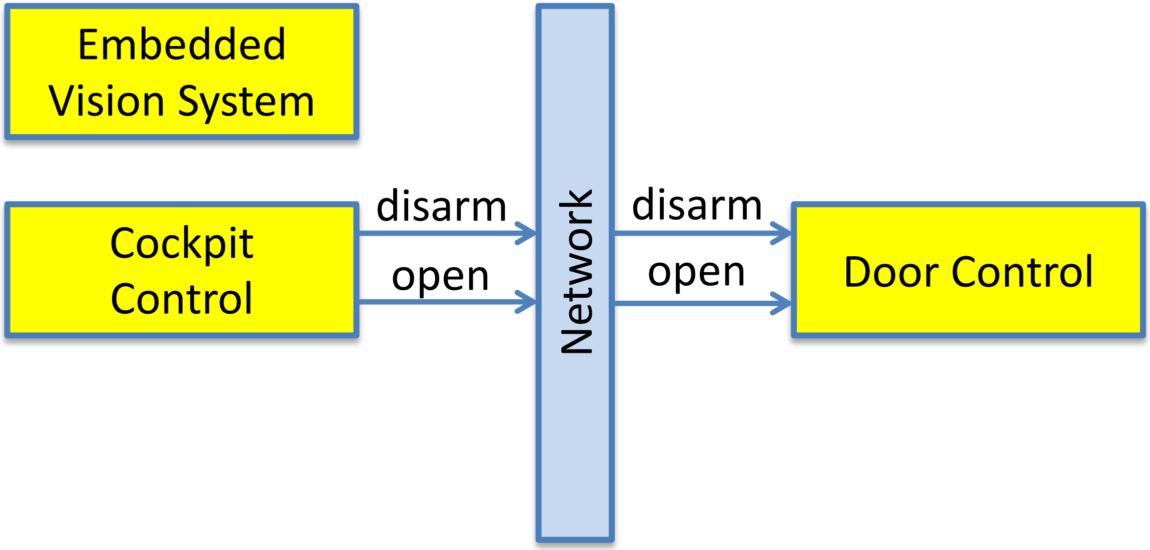
Assume the state is closed and armed.

Physics, Software, Network

Using Software instead of the pilot and the cabin crew, and a network in between.

Cyber-Physical Systems: Control Physical Components using Software through Network

Concurrency and timing problems.



A module that can receive either of two messages:

1. “open”
2. “disarm”

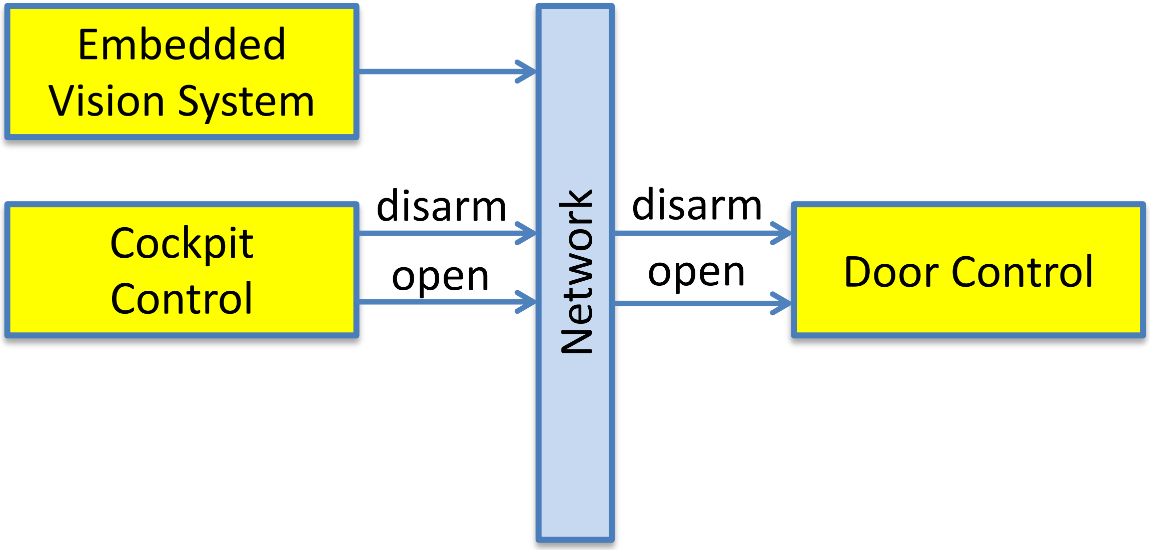
Assume the state is closed and armed.

Physics, Software, Network

Using Software instead of the pilot and the cabin crew, and a network in between.

Cyber-Physical Systems: Control Physical Components using Software through Network

Concurrency and timing problems.



A module that can receive either of two messages:

1. “open”
2. “disarm”

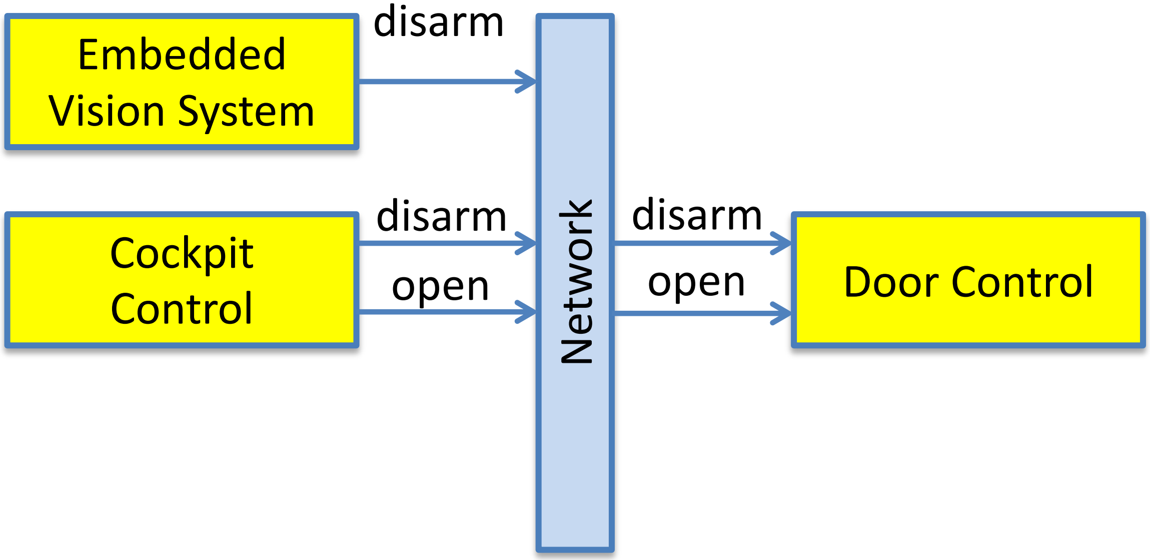
Assume the state is closed and armed.

Physics, Software, Network

Using Software instead of the pilot and the cabin crew, and a network in between.

Cyber-Physical Systems: Control Physical Components using Software through Network

Concurrency and timing problems.



A module that can receive either of two messages:

1. “open”
2. “disarm”

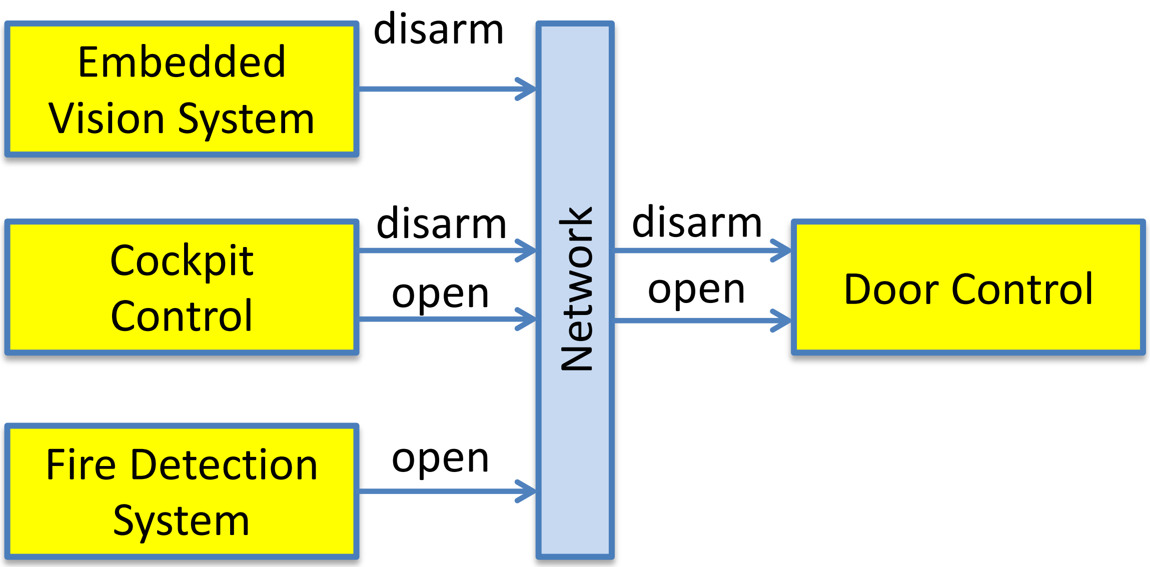
Assume the state is closed and armed.

Physics, Software, Network

Using Software instead of the pilot and the cabin crew, and a network in between.

Cyber-Physical Systems: Control Physical Components using Software through Network

Concurrency and timing problems.



A module that can receive either of two messages:

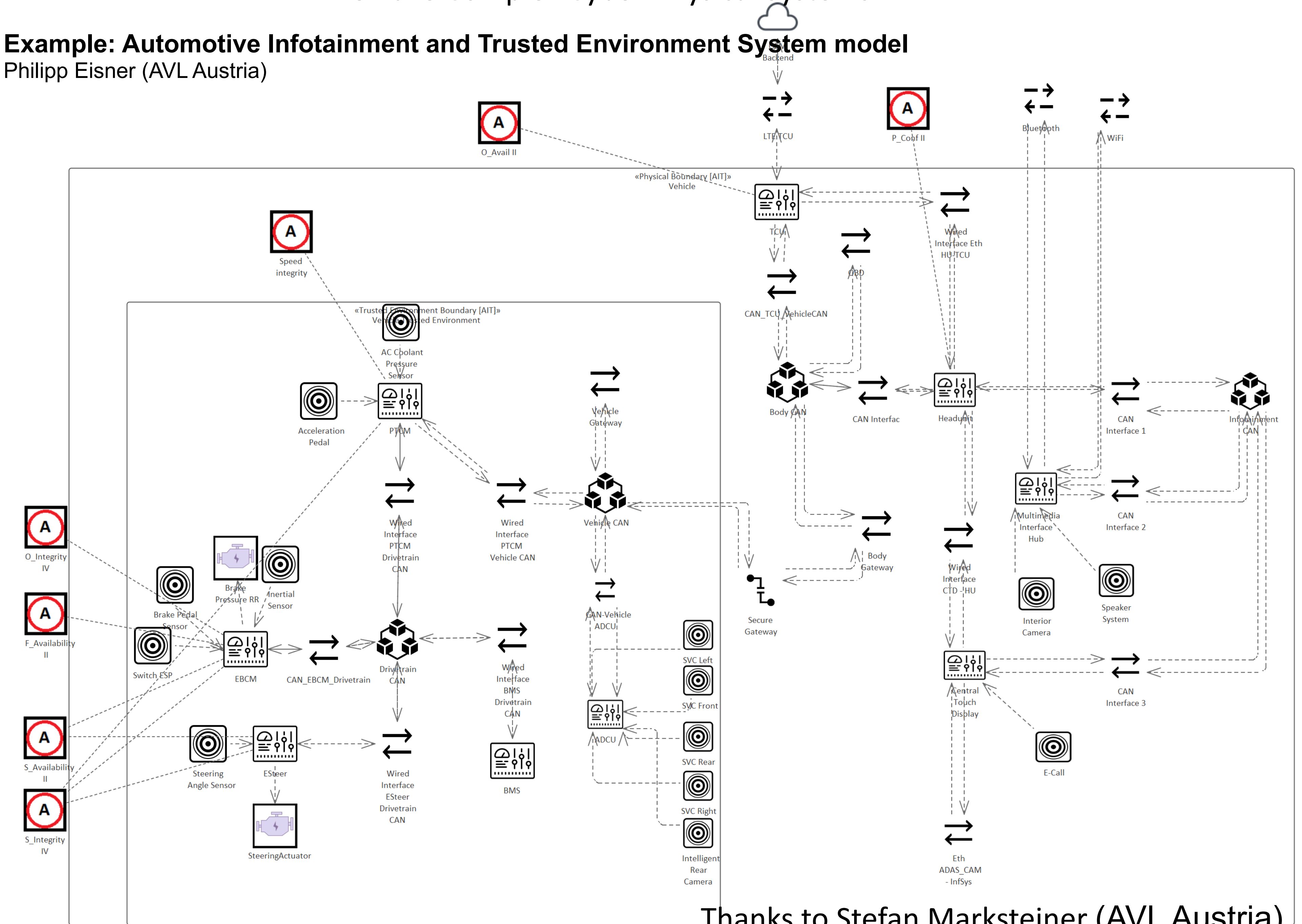
1. "open"
2. "disarm"

Assume the state is closed and armed.

We have Complex Cyber-Physical Systems

Example: Automotive Infotainment and Trusted Environment System model

Philipp Eisner (AVL Austria)

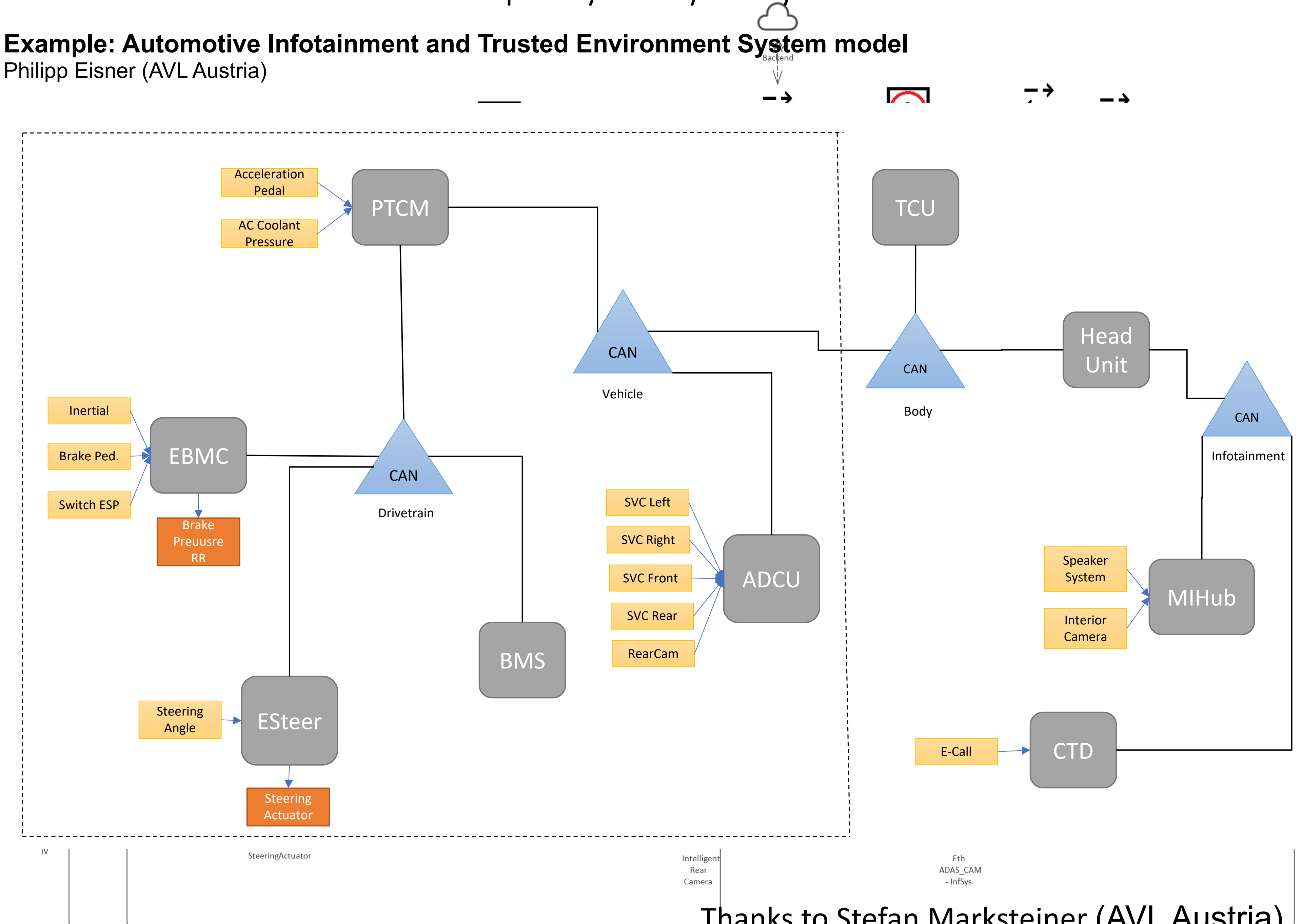


Thanks to Stefan Marksteiner (AVL Austria) object SystemModel

We have Complex Cyber-Physical Systems

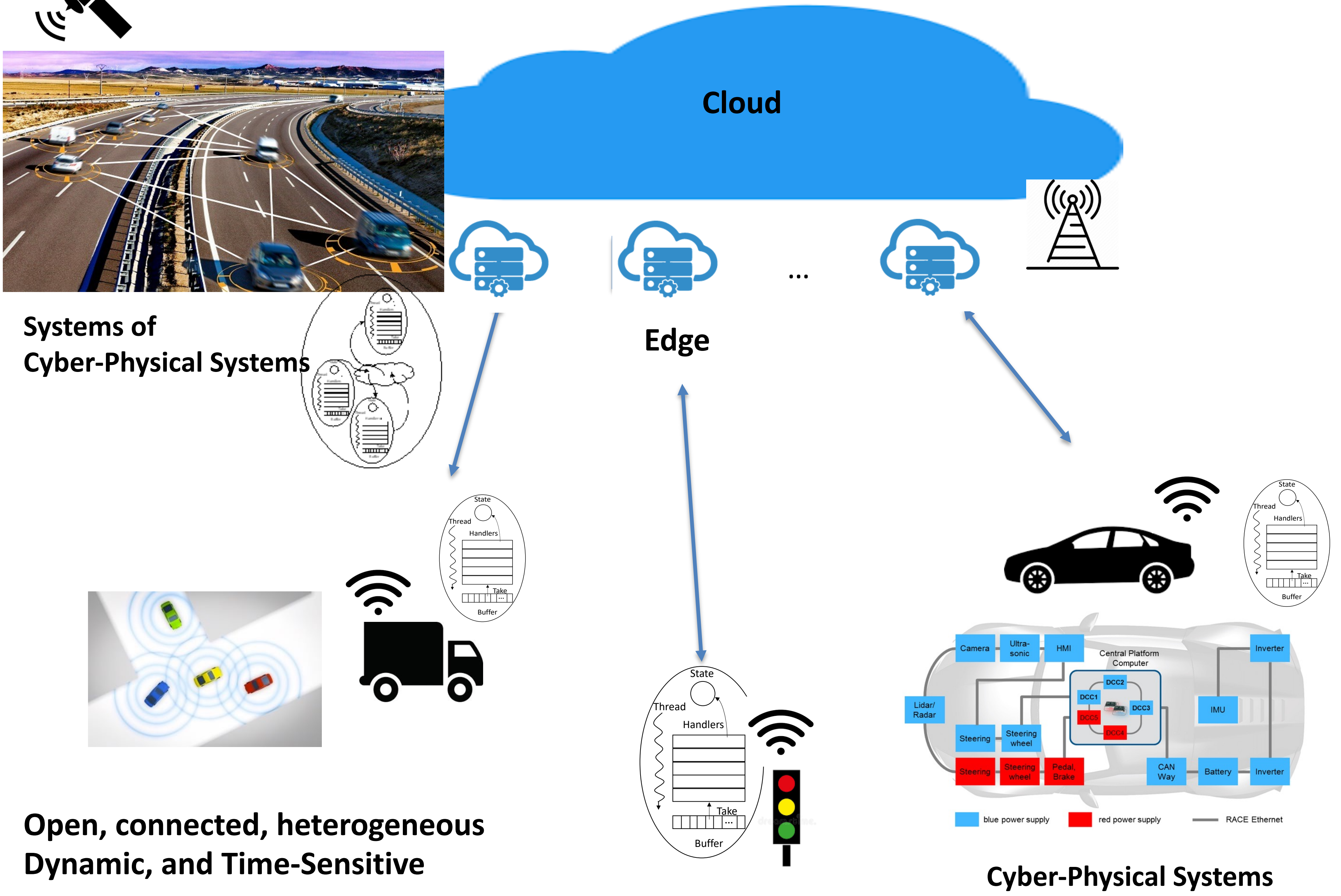
Example: Automotive Infotainment and Trusted Environment System model

Philipp Eisner (AVL Austria)



Thanks to Stefan Marksteiner (AVL Austria)

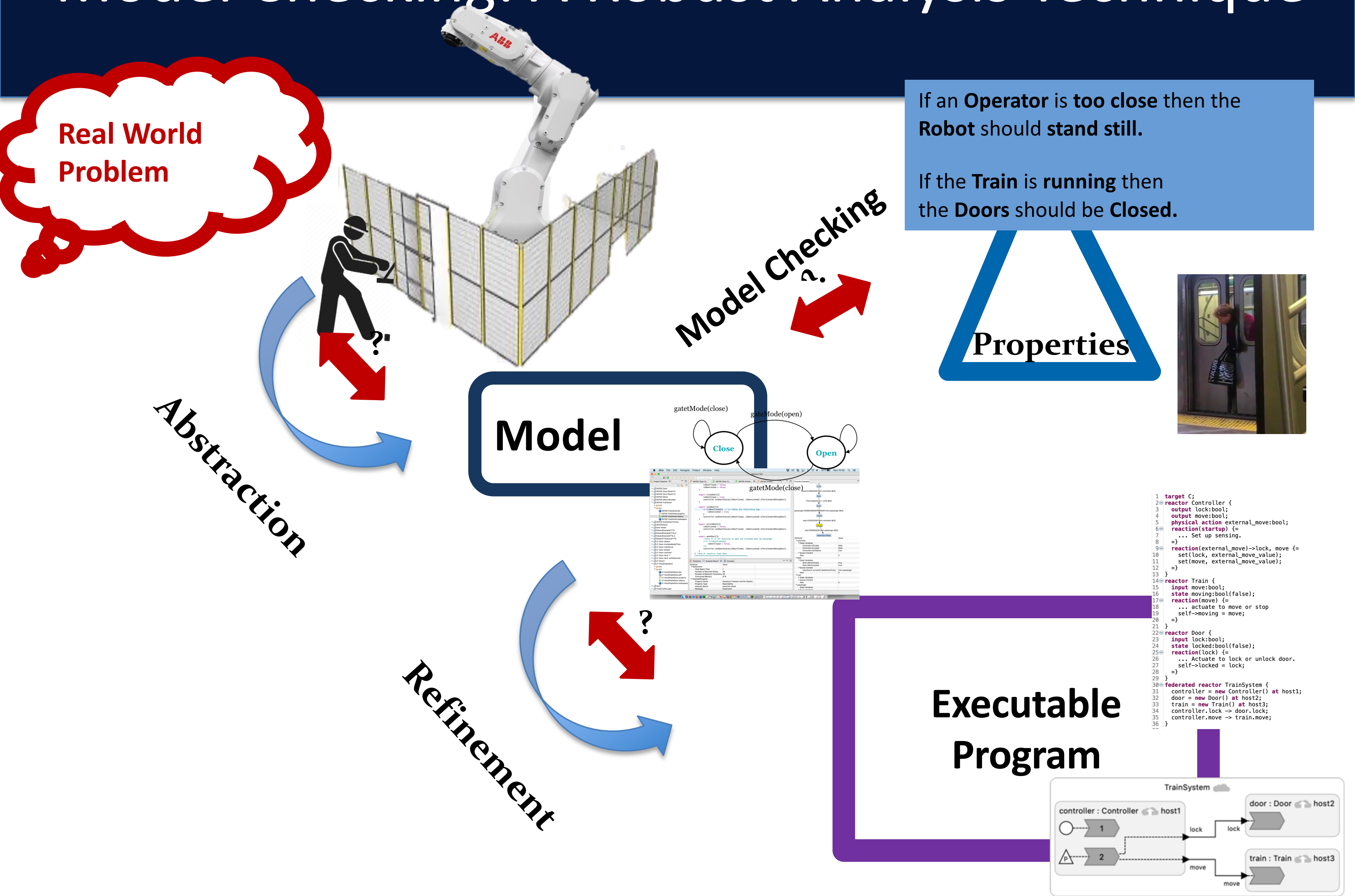
We have Complex Cyber-Physical Systems Nowadays



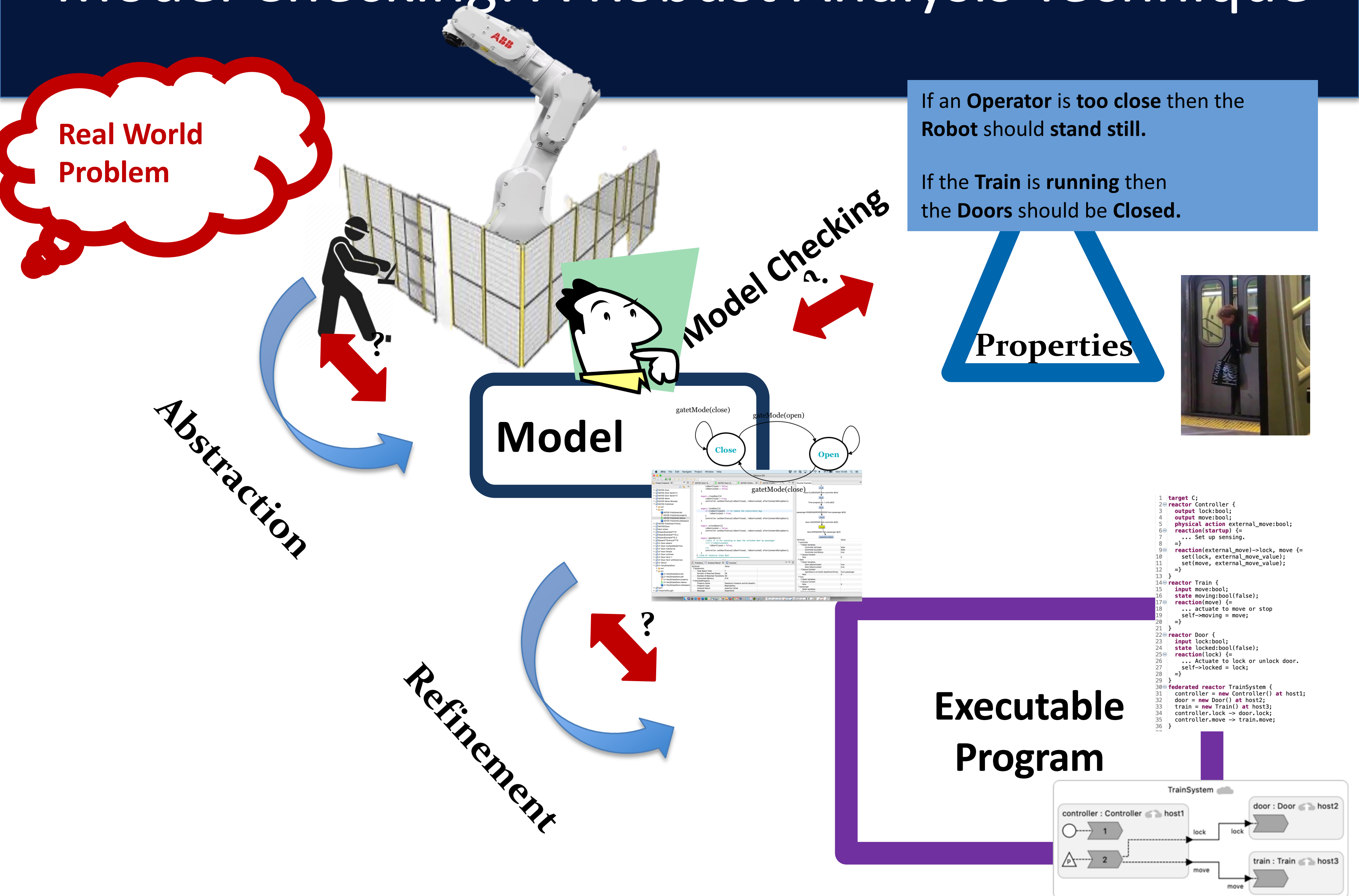
We need Robust Development Methods

Formal Verification of Cyber Physical Systems

Model Checking: A Robust Analysis Technique



Model Checking: A Robust Analysis Technique



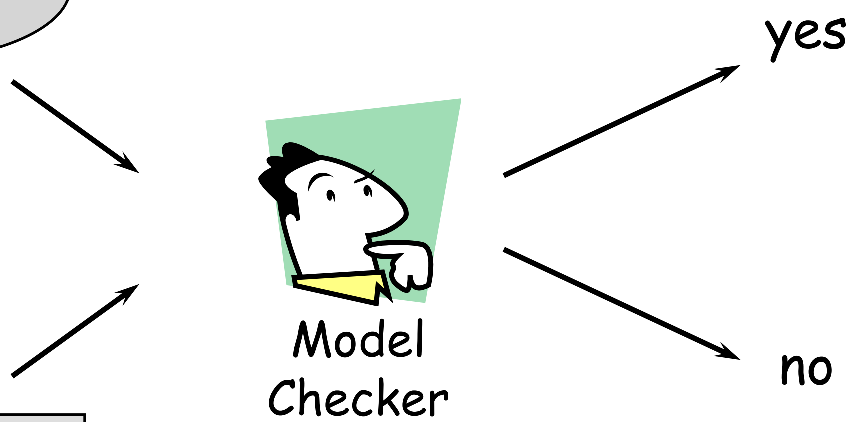
Model Checking: Prove Properties

If an **Operator** is **too close** then the **Robot** should **stand still.**

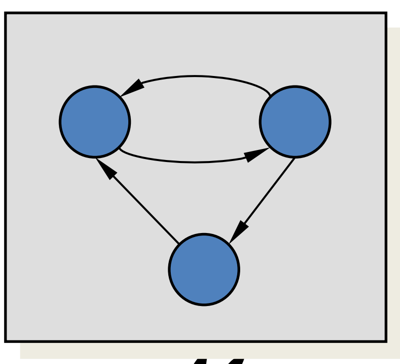
If the **Train** is **running** then the **Doors** should be **Closed.**

$p \rightarrow F\, q$

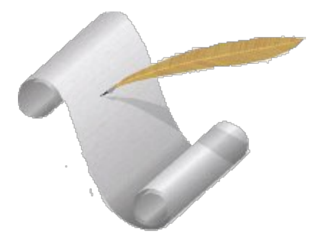
φ




```
reactiveclass Controller(5) {
knownrebecs {
Door door;
Train train;
}
statevars { boolean moveP; }
Controller() {
self.external();
}
msgsrv external() {
boolean oldMoveP = moveP;
moveP = ?(true,false);
if(moveP != oldMoveP) {
door.lock(moveP);
train.move(moveP);
}
self.external() after(1);
}
}
reactiveclass Train(5) {
statevars { boolean moving; }
Train() {
moving = false;
}
msgsrv move(boolean tmove) {
if (tmove) {
moving = true;
} else {
moving = false;
}
}
}
reactiveclass Door(5) {
statevars { boolean is_locked; }
Door() {
is_locked = false;
}
msgsrv lock (boolean lockPar) {
is_locked = lockPar;
}
}
main {
@priority(1) Controller controller(door,
train):();
@priority(2) Train train():();
@priority(2) Door door():();
}
```

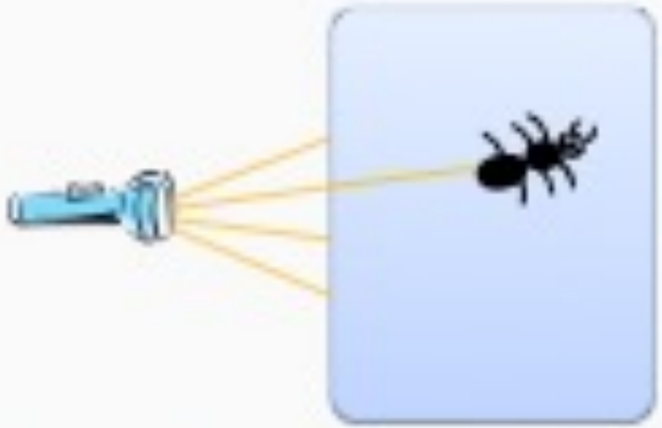


$\mathcal{M}$



Error Trace

(Formal) Software Verification is the act of proving/disproving that a program is bug-free using mathematics



Testing and simulation can only check a few cases

Software verification checks **all** possible behaviors

Different approaches for Modeling and Verification

Abstract

Mathematical

Modeling languages

CCS CSP
Petri net
RML
Timed Automata

FDR

UPPAAL

NuSMV

Verification Techniques:

- **Deduction**
  needs high expertise
- **Model checking**
  causes state explosion

SMV
Promela

Spin

Java PathFinder

Programming languages

Too heavy
Not always
formal

Java
C

Bandera

SLAM

Our choice for modeling: Actors

- A reference model for concurrent computation
- Consisting of concurrent, distributed active objects

- Proposed by Hewitt as an agent-based language (MIT, 1971)
- Developed by Agha as a concurrent object-based language (Illinois, since 1984)
- Formalized by Talcott (with Agha, Mason and Smith): Towards a Theory of Actor Computation (CONCUR 1992)

Our choice for modeling: Actors

– A reference model for concurrent computation
– Consisting of concurrent, distributed active objects

Friendly to the modeler and to the network systems

- Proposed by Hewitt as an agent-based language (MIT, 1971)
- Developed by Agha as a concurrent object-based language (Illinois, since 1984)
- Formalized by Talcott (with Agha, Mason and Smith): Towards a Theory of Actor Computation (CONCUR 1992)

Actor-based Language Rebeca

Rebeca: **Re**active o**bject** l**a**nguage (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008

Based on Hewitt actors

Concurrent reactive objects

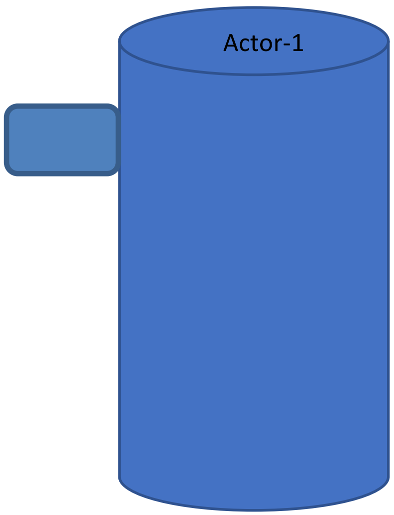
Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

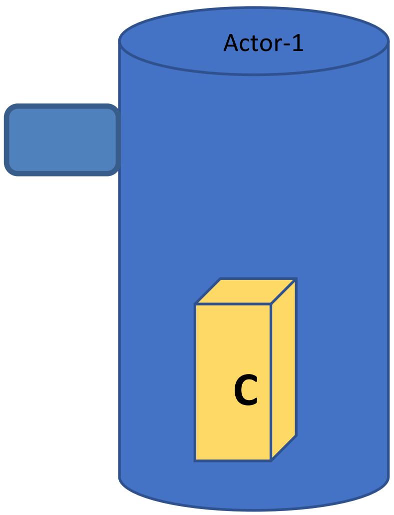
Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive o**bjec**t l**a**nguage (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

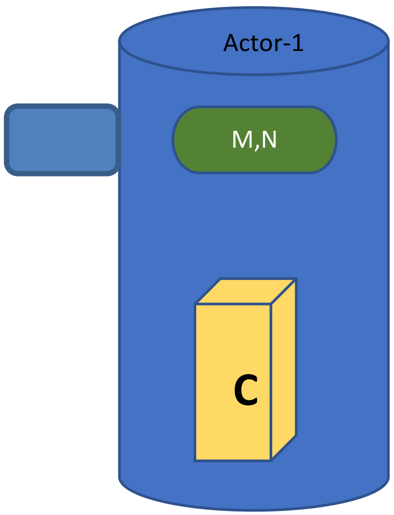
Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

Concurrent reactive objects

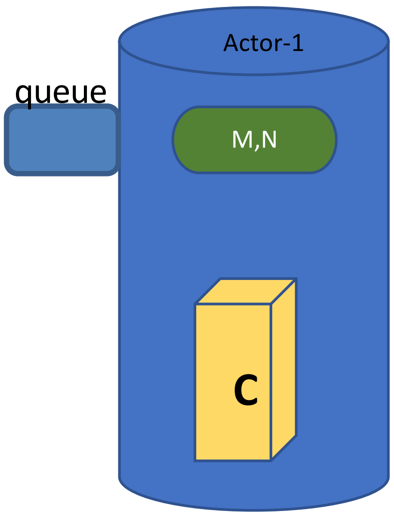
Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive o**bj**ect l**a**nguage (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

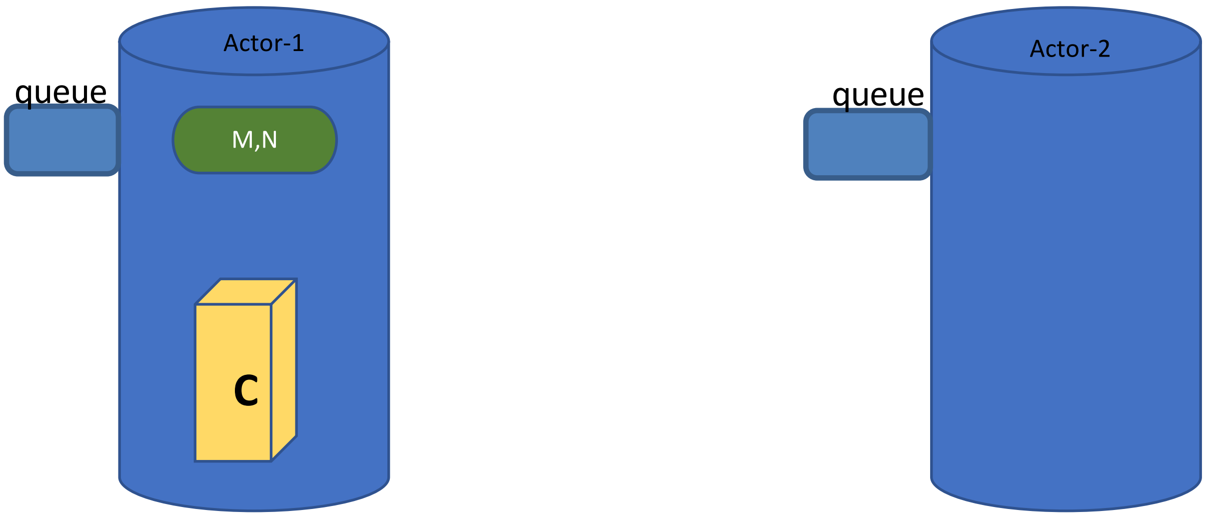
Actor-based Language Rebeca

Rebeca: <u>Re</u>active o<u>bj</u><u>ec</u>t l<u>a</u>nguage (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

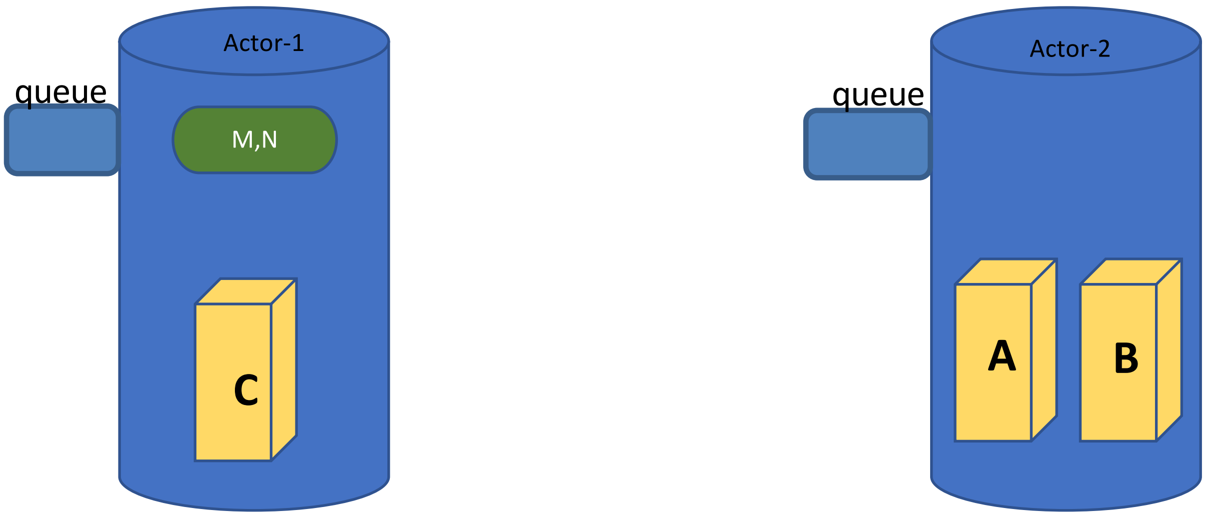
Actor-based Language Rebeca

Rebeca: Reactive o**bject** l**a**nguage (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

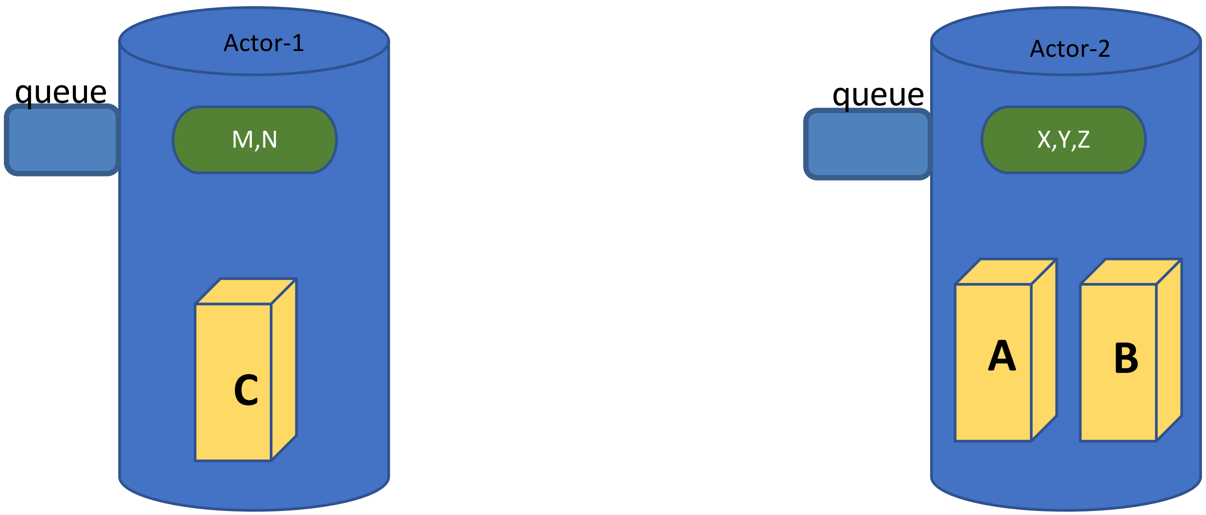
Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

Concurrent reactive objects

Java like syntax

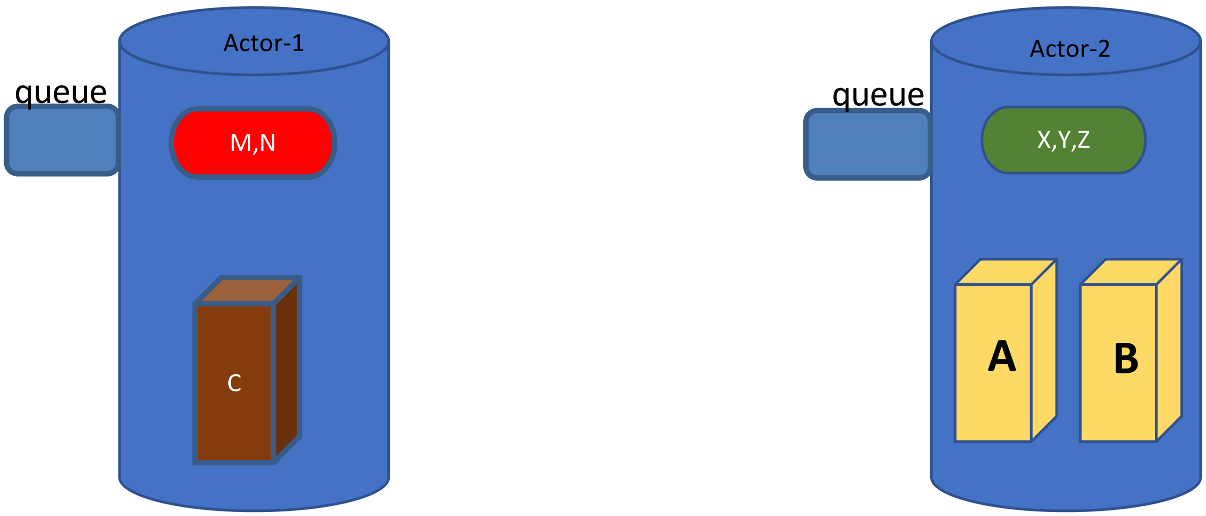
- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

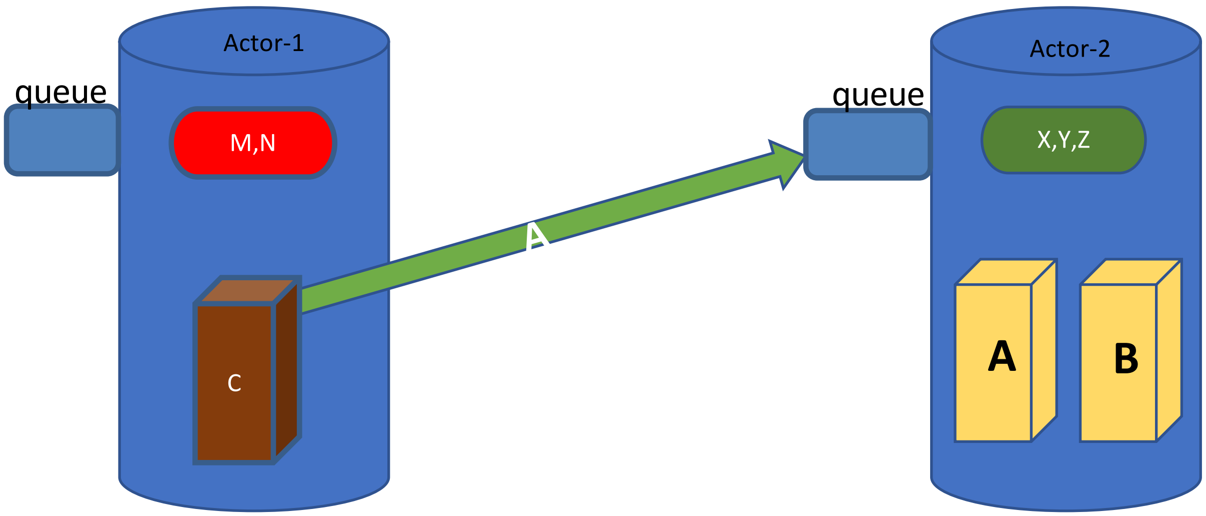
Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



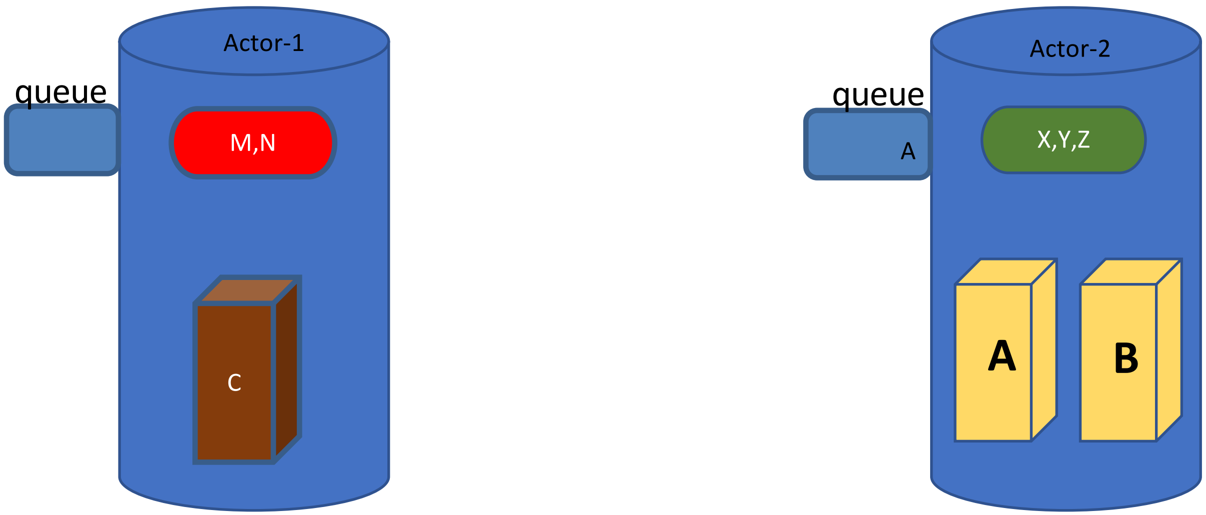
- Based on Hewitt actors
- Concurrent reactive objects
- Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

Concurrent reactive objects

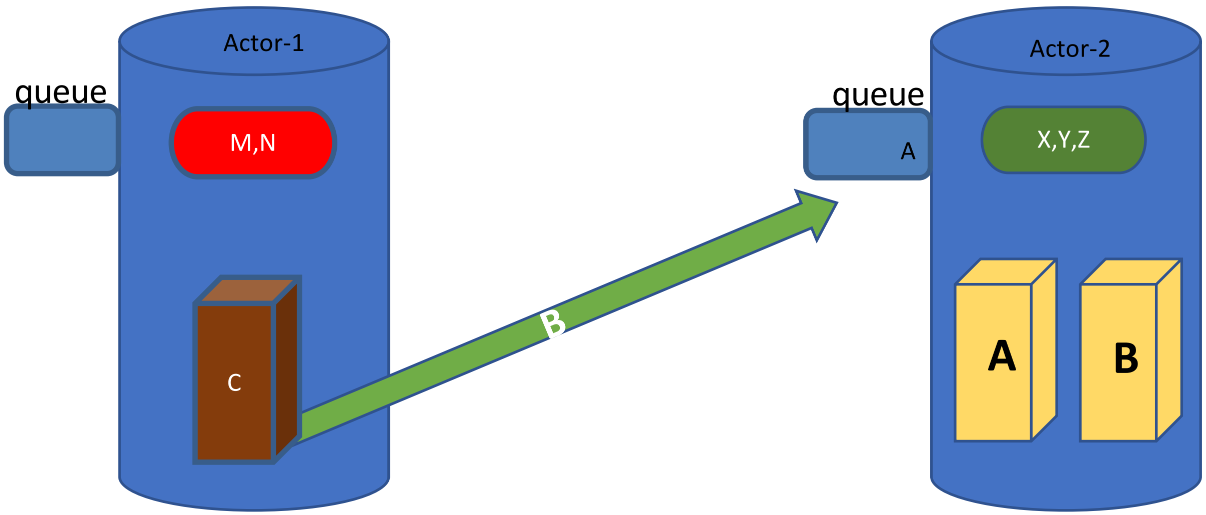
Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive **obj**ect **la**nguage (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

Concurrent reactive objects

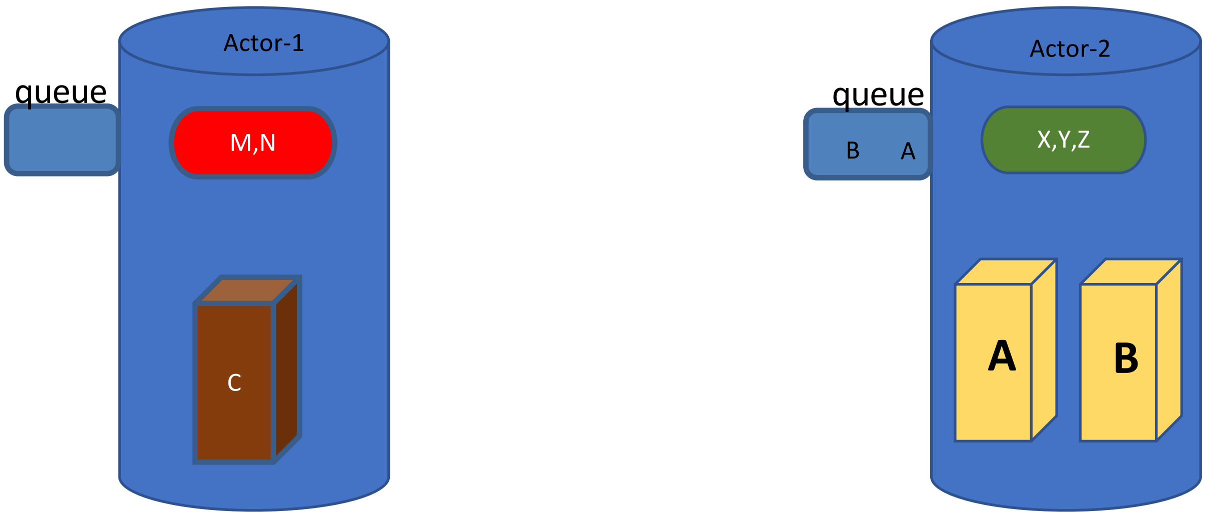
Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: <u>Re</u>active o<u>bjec</u>t l<u>a</u>nguage (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

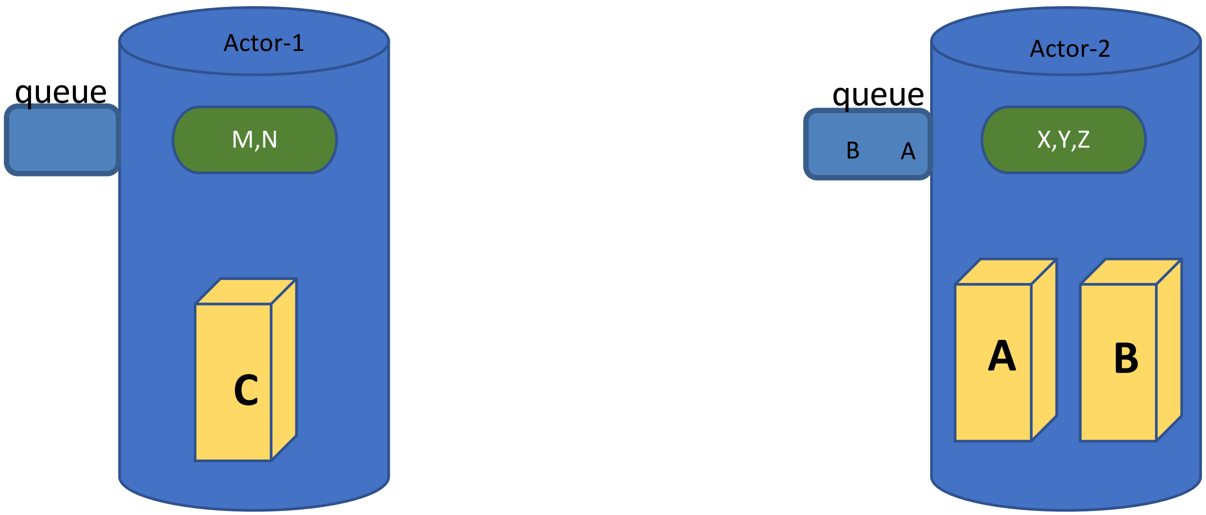
Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

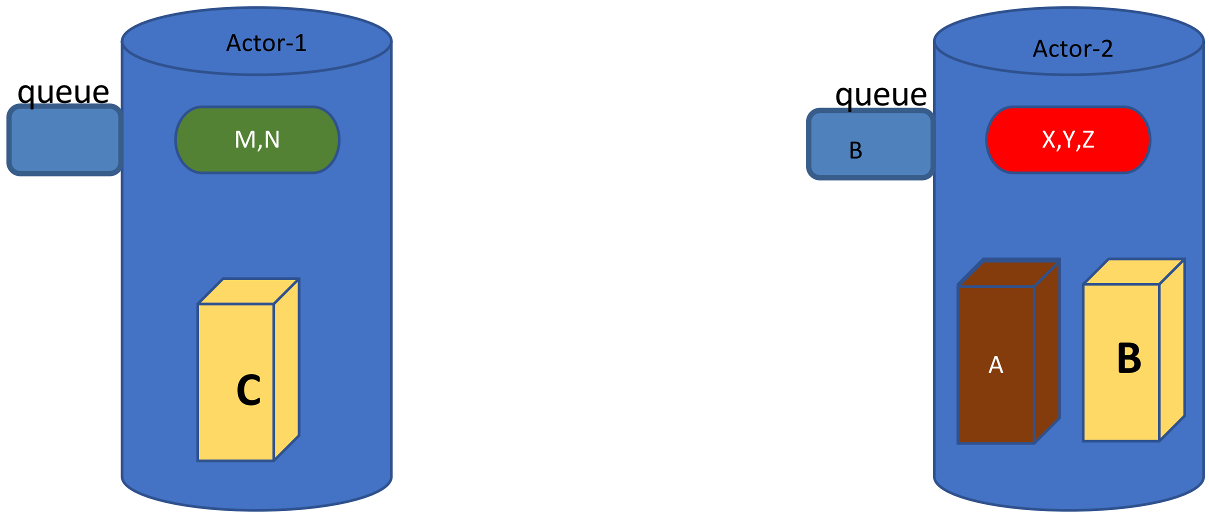
Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

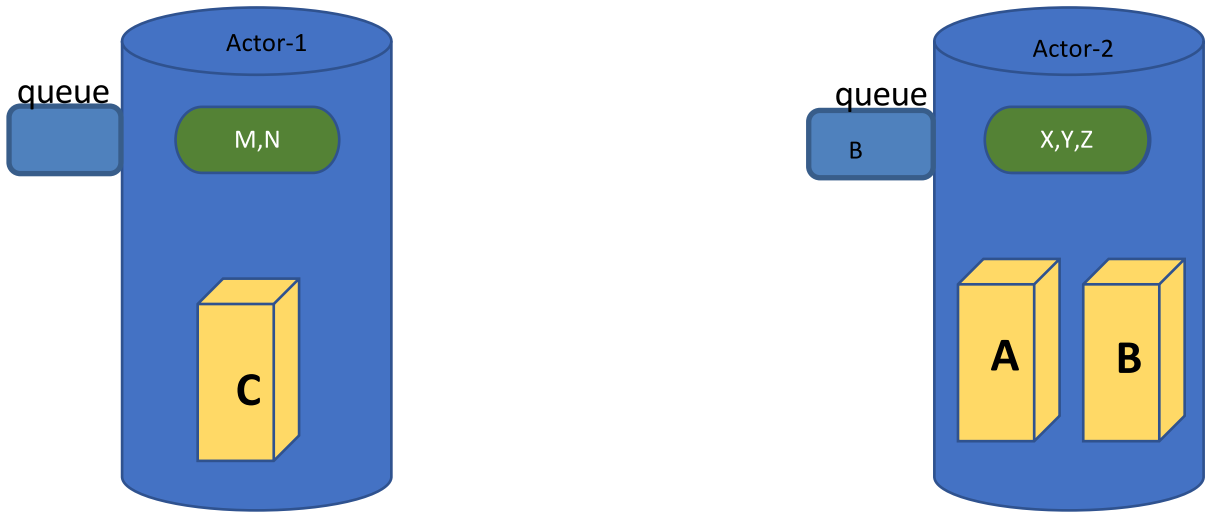
Actor-based Language Rebeca

Rebeca: <u>Re</u>active o<u>bjec</u>t l<u>a</u>nguage (Sirjani, Movaghar, **2001**)

Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

Concurrent reactive objects

Java like syntax

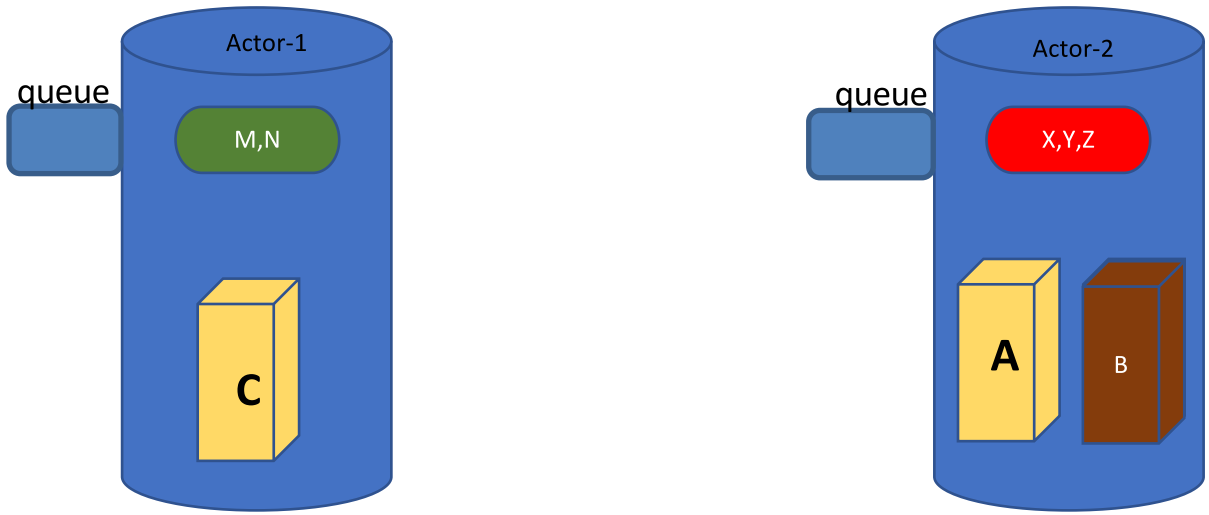
- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008

An actor:

- Message servers
- State Variables
- A message queue



Based on Hewitt actors

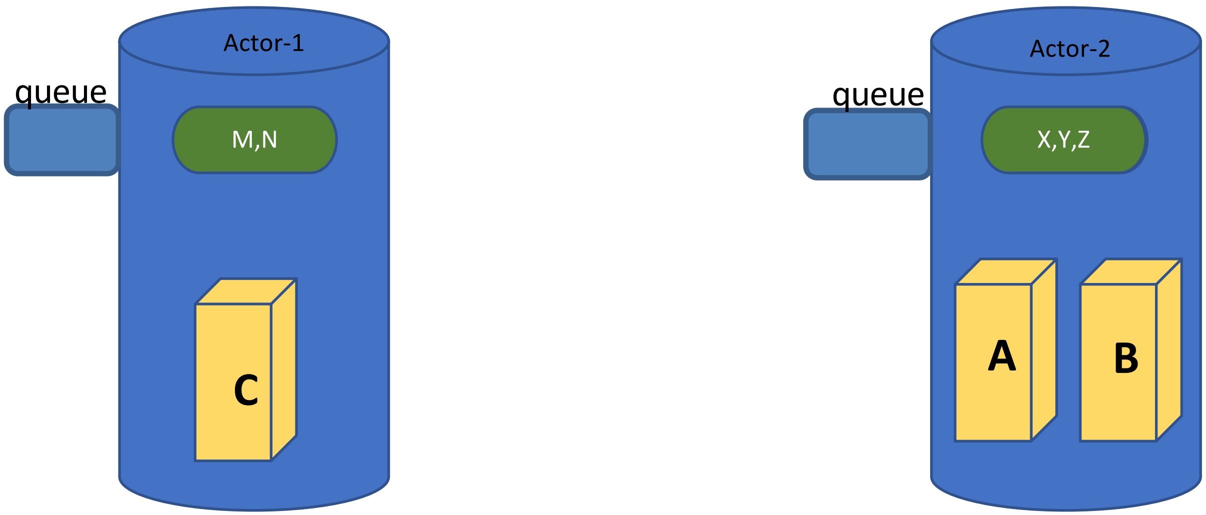
Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Actor-based Language Rebeca

Rebeca: Reactive object language (Sirjani, Movaghar, **2001**)
Timed Rebeca: 2008



An actor:

- Message servers
- State Variables
- A message queue

Based on Hewitt actors

Concurrent reactive objects

Java like syntax

- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec (in theory)
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven

Timed Rebeca (2008)

- An extension of Rebeca for real time systems modeling
  - Computation time (delay)
  - Message delivery time (after)
  - Periods of occurence of events (after)

  - Message expiration (deadline)

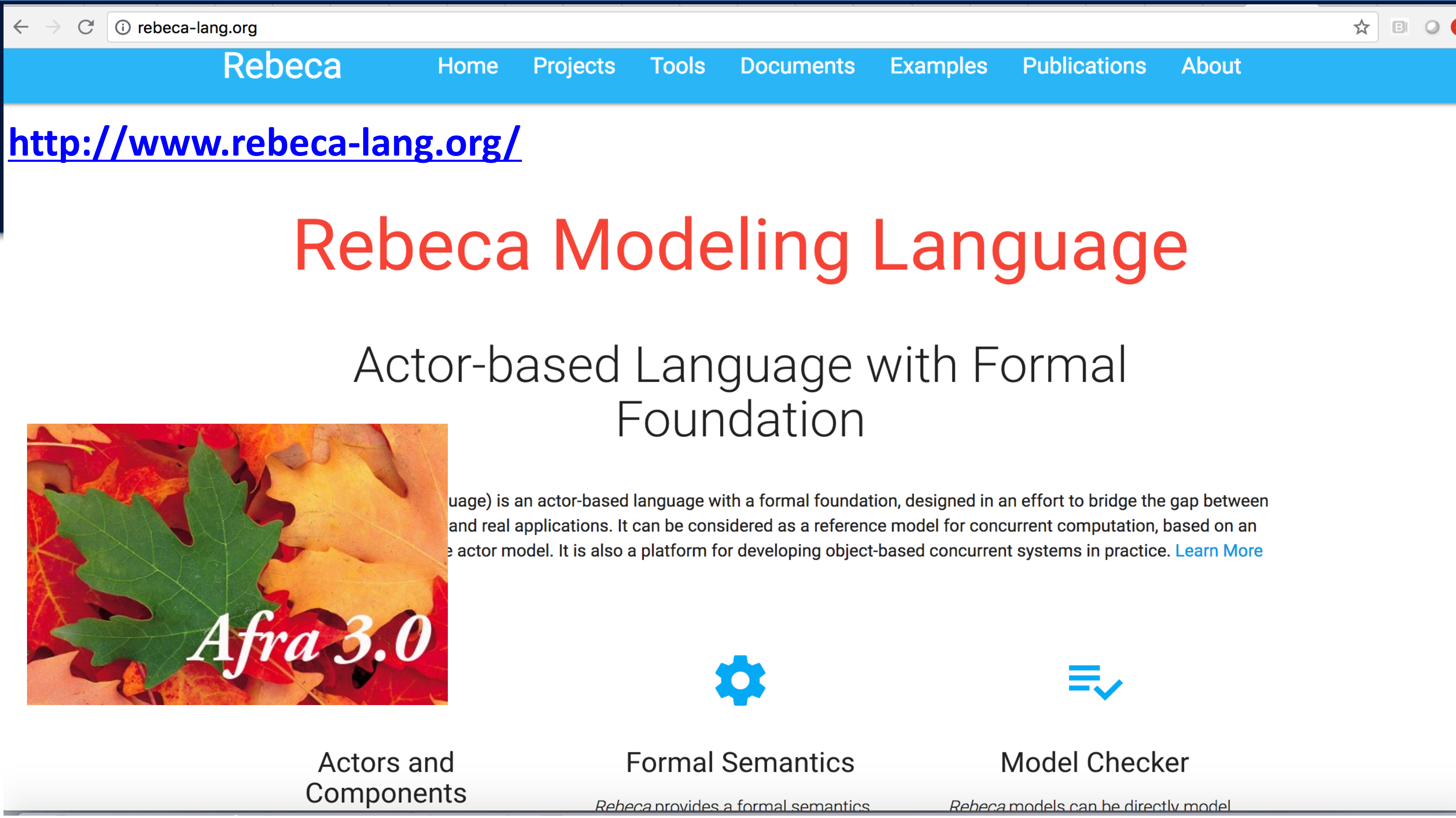
FIFO message queues become message bags containing tagged messages

A simple Timed-Rebeca Model

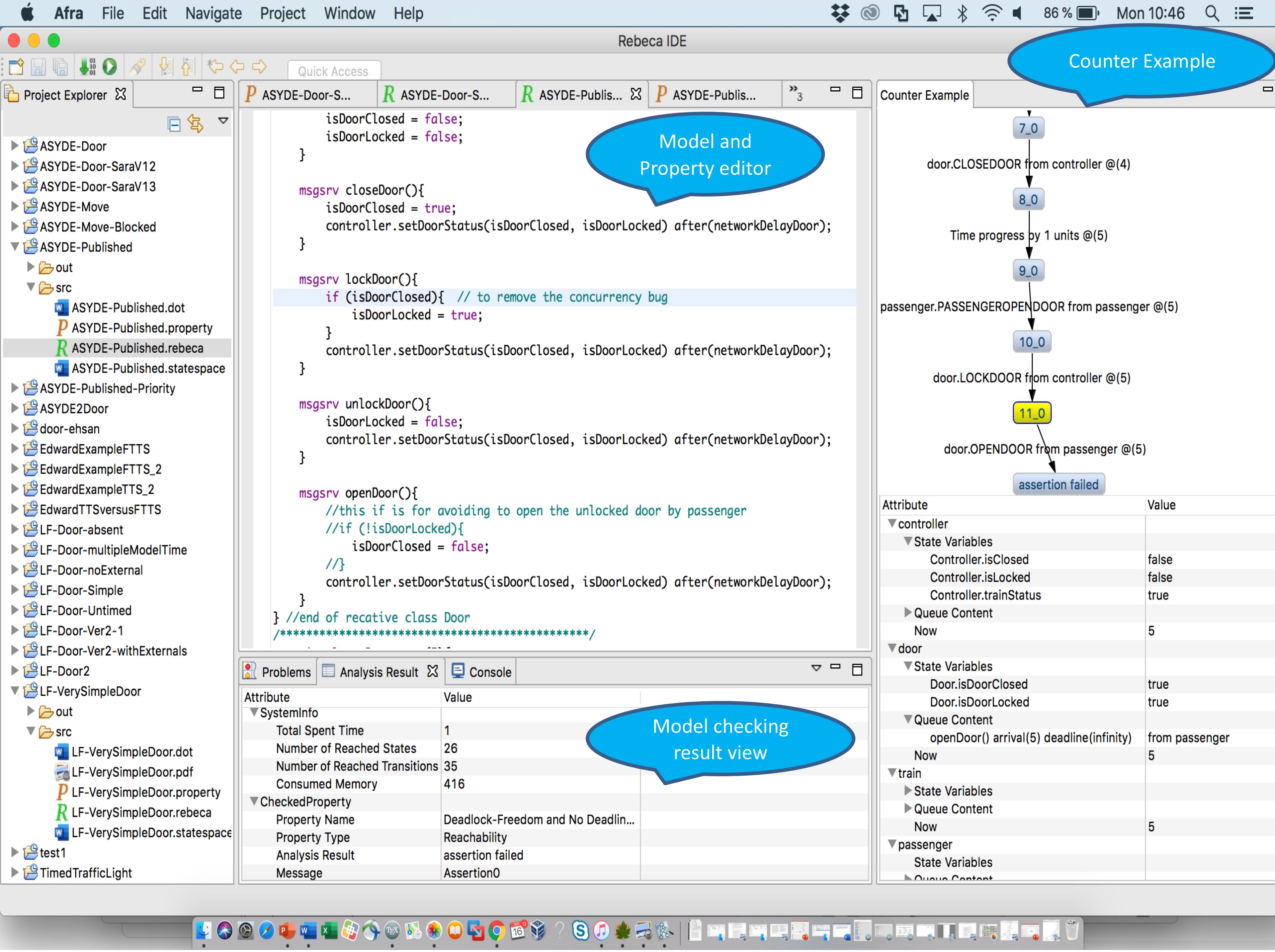
```
reactiveclass RC1 (3) {
    knownrebecs {
     RC2 r2;
    }
    RC1() {
     self.m1();
    }
    msgsrv m1() {
     delay(2);
     r2.m2();
     delay(2);
      r2.m3() after (5);
     self.m1() after (10);
    }
}
```

```
reactiveclass RC2 (4) {
    knownrebecs {
     RC1 r1;
    }
    RC2() { }
    msgsrv m2() { }

    msgsrv m3() { }
}

main {
    RC1 r1(r2):();
    RC2 r2(r1):();
}
```

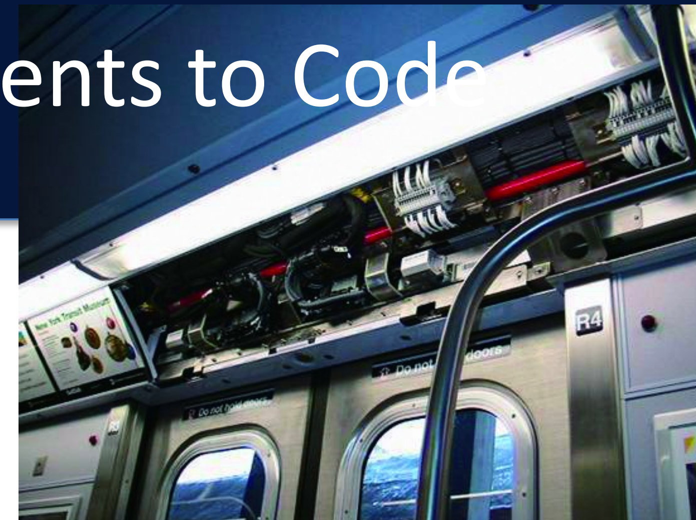
http://www.rebeca-lang.org/



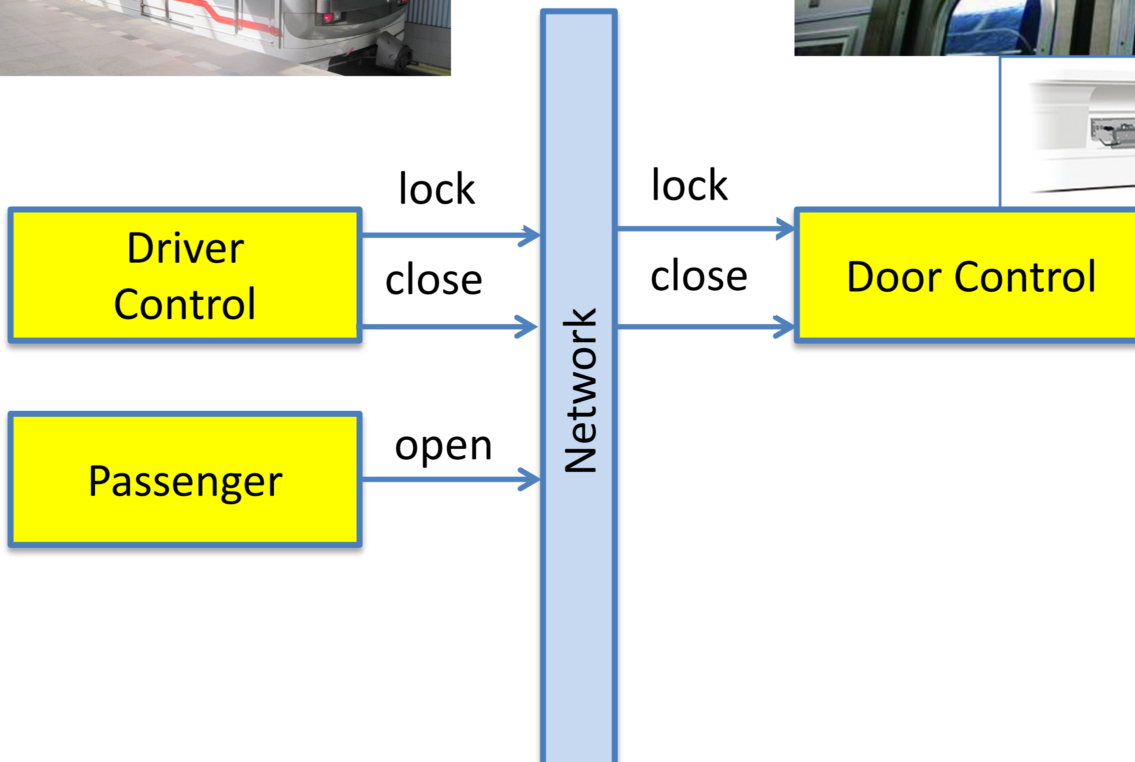
- **Ten years of Analyzing Actors: Rebeca Experience** (Sirjani, Jaghouri), Carolyn Talcott Festschrift, 70th birthday, LNCS 7000, 2011
- **On Time Actors** (Sirjani, Khamespanah), Theory and Practice of Formal Methods, Frank de Boer Festschrift, 2016
- **Power is Overrated, Go for Friendliness! Expressiveness, Faithfulness and Usability in Modeling - The Actor Experience**, Edward Lee Festschrift, 2017


Afra File Edit Navigate Project Window Help
Mon 10:46
Rebeca IDE
Quick Access
Counter Example
Model and Property editor
Model checking result view
Project Explorer
ASYDE-Door
ASYDE-Door-SaraV12
ASYDE-Door-SaraV13
ASYDE-Move
ASYDE-Move-Blocked
ASYDE-Published
out
src
ASYDE-Published.dot
ASYDE-Published.property
ASYDE-Published.rebeca
ASYDE-Published.statespace
ASYDE-Published-Priority
ASYDE2Door
door-ehsan
EdwardExampleFTTS
EdwardExampleFTTS_2
EdwardExampleTTS_2
EdwardTTSversusFTTS
LF-Door-absent
LF-Door-multipleModelTime
LF-Door-noExternal
LF-Door-Simple
LF-Door-Untimed
LF-Door-Ver2-1
LF-Door-Ver2-withExternals
LF-Door2
LF-VerySimpleDoor
out
src
LF-VerySimpleDoor.dot
LF-VerySimpleDoor.pdf
LF-VerySimpleDoor.property
LF-VerySimpleDoor.rebeca
LF-VerySimpleDoor.statespace
test1
TimedTrafficLight
ASYDE-Door-S...
ASYDE-Door-S...
ASYDE-Publis...
ASYDE-Publis...
isDoorClosed = false;
isDoorLocked = false;
}
msgsrv closeDoor(){
isDoorClosed = true;
controller.setDoorStatus(isDoorClosed, isDoorLocked) after(networkDelayDoor);
}
msgsrv lockDoor(){
if (isDoorClosed){ // to remove the concurrency bug
isDoorLocked = true;
}
controller.setDoorStatus(isDoorClosed, isDoorLocked) after(networkDelayDoor);
}
msgsrv unlockDoor(){
isDoorLocked = false;
controller.setDoorStatus(isDoorClosed, isDoorLocked) after(networkDelayDoor);
}
msgsrv openDoor(){
//this if is for avoiding to open the unlocked door by passenger
//if (!isDoorLocked){
isDoorClosed = false;
//}
controller.setDoorStatus(isDoorClosed, isDoorLocked) after(networkDelayDoor);
}
} //end of recative class Door
/***/
Problems
Analysis Result
Console
Attribute
Value
SystemInfo
Total Spent Time
1
Number of Reached States
26
Number of Reached Transitions
35
Consumed Memory
416
CheckedProperty
Property Name
Deadlock-Freedom and No Deadlin...
Property Type
Reachability
Analysis Result
assertion failed
Message
Assertion0
Counter Example
7_0
door.CLOSEDOOR from controller @(4)
8_0
Time progress by 1 units @(5)
9_0
passenger.PASSENGEROPENDOOR from passenger @(5)
10_0
door.LOCKDOOR from controller @(5)
11_0
door.OPENDOOR from passenger @(5)
assertion failed
Attribute
Value
controller
State Variables
Controller.isClosed
false
Controller.isLocked
false
Controller.trainStatus
true
Queue Content
Now
5
door
State Variables
Door.isDoorClosed
true
Door.isDoorLocked
true
Queue Content
openDoor() arrival(5) deadline(infinity)
from passenger
Now
5
train
State Variables
Queue Content
Now
5
passenger
State Variables

An example: from Requirements to Code



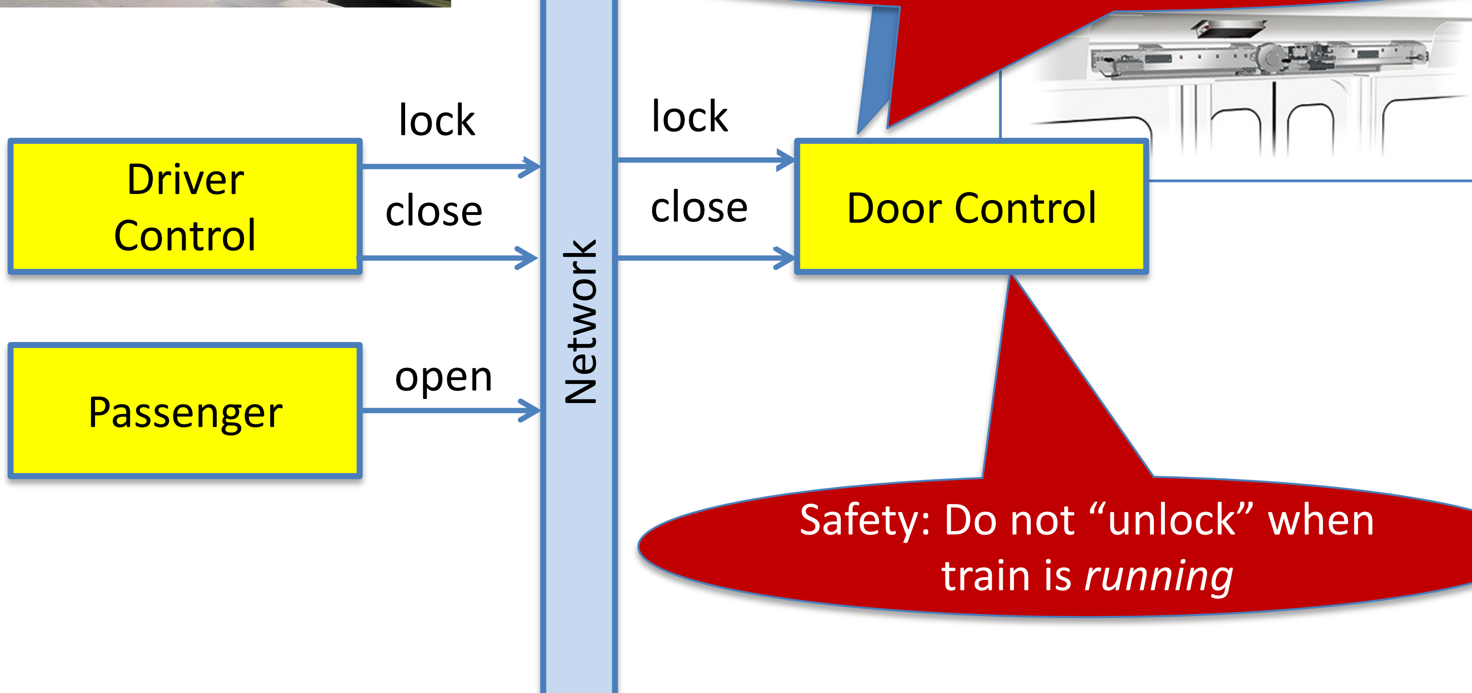
Train Door Controller



Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam, Mehrdad Saadatmand: Towards a Verification-Driven Iterative Development of Software for Safety-Critical Cyber-Physical Systems, Journal of Internet Services and Applications, 2021

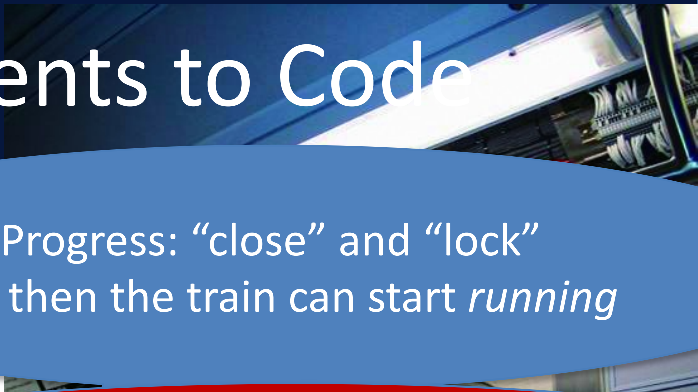
An example: from Requirements to Code

Train Door Controller



Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam, Mehrdad Saadatmand: Towards a Verification-Driven Iterative Development of Software for Safety-Critical Cyber-Physical Systems, Journal of Internet Services and Applications, 2021

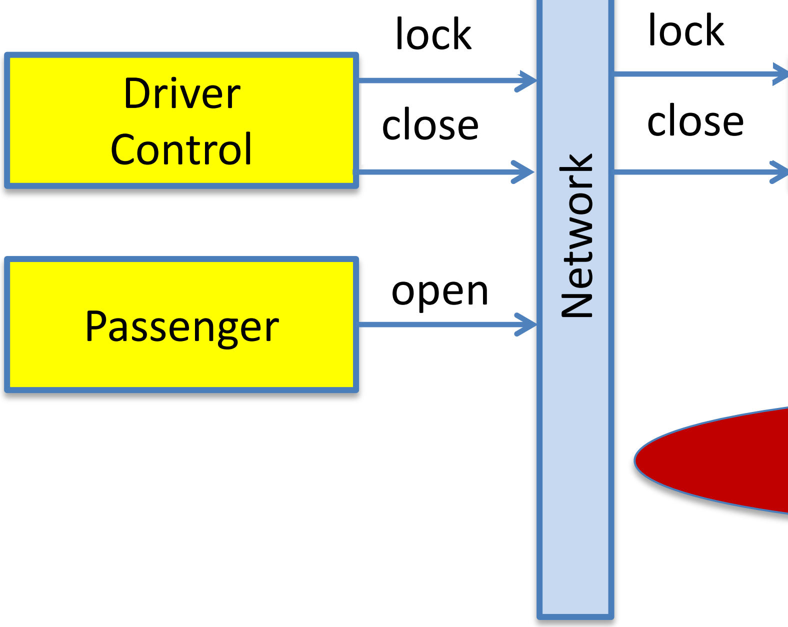
An example: from Requirements to Code



Train Door Controller

Progress: “close” and “lock” and then the train can start *running*

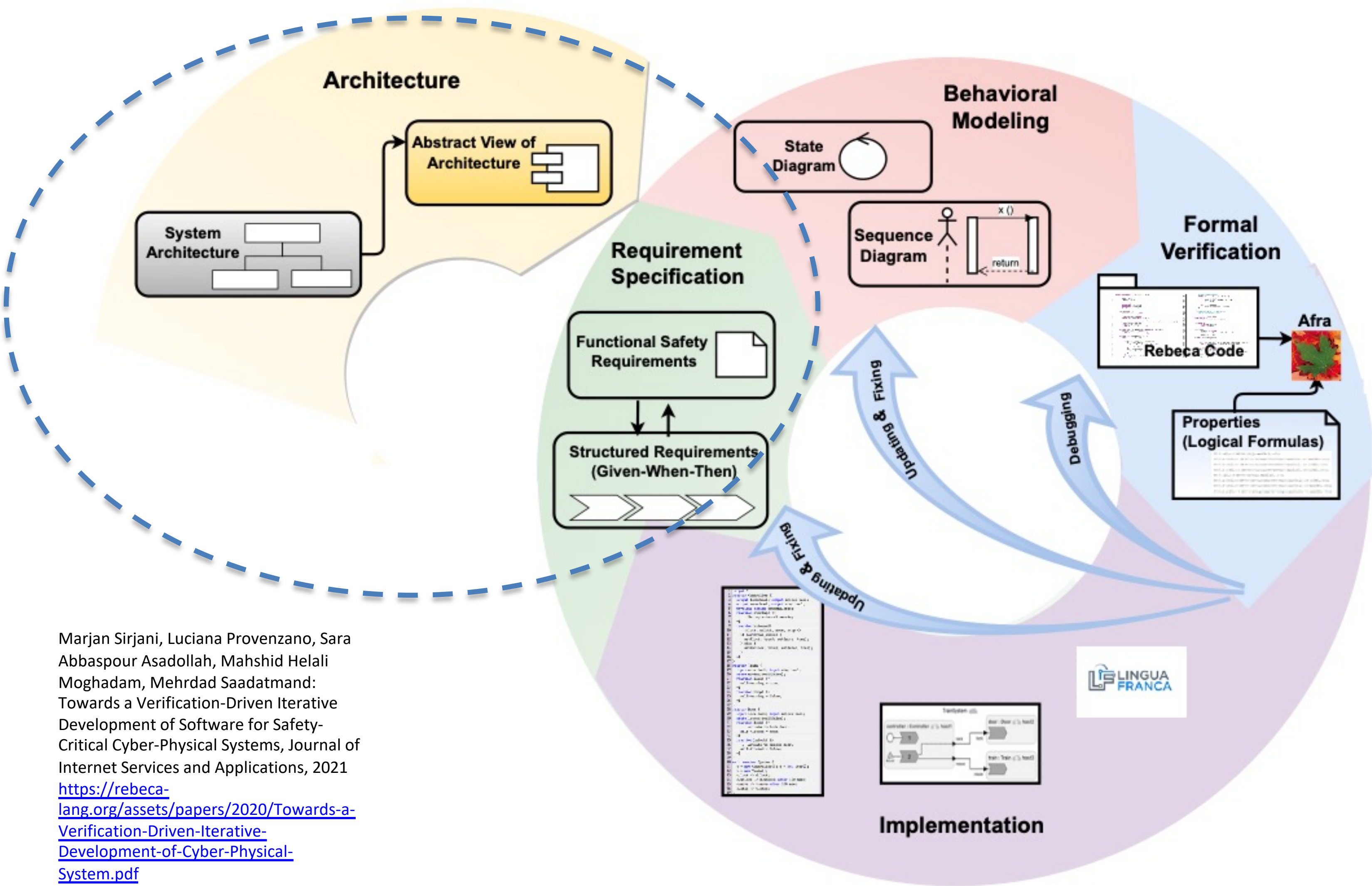
Safety: Do not “open” a *locked* door



Safety: Do not “unlock” when train is *running*

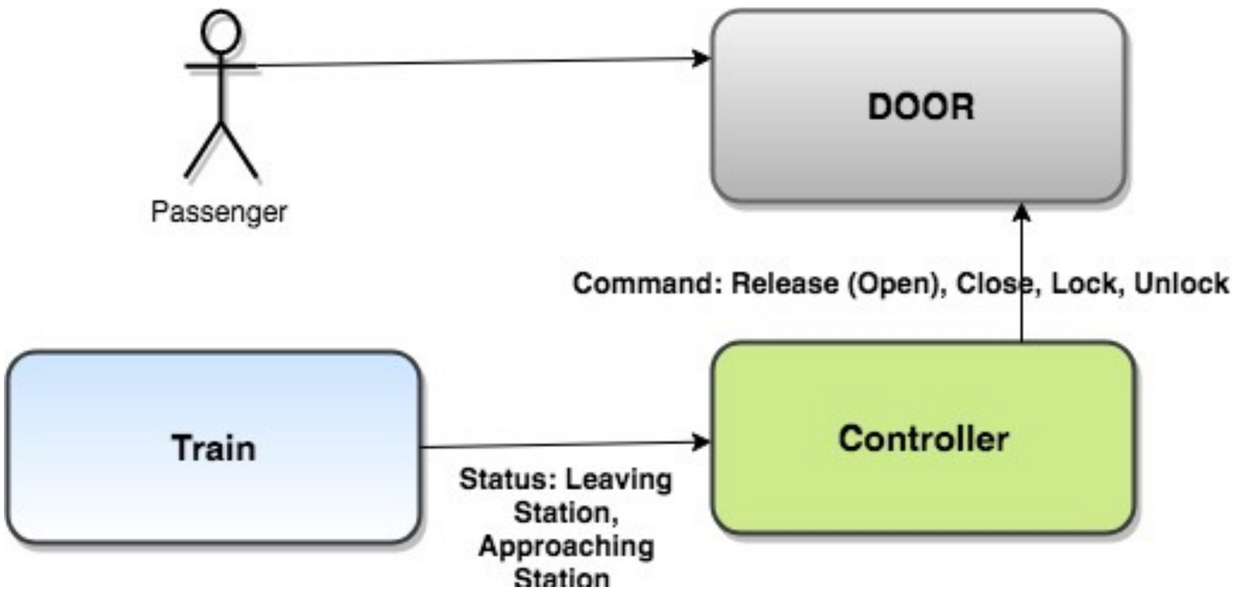
Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam, Mehrdad Saadatmand: Towards a Verification-Driven Iterative Development of Software for Safety-Critical Cyber-Physical Systems, Journal of Internet Services and Applications, 2021

Process: Start from the Requirements

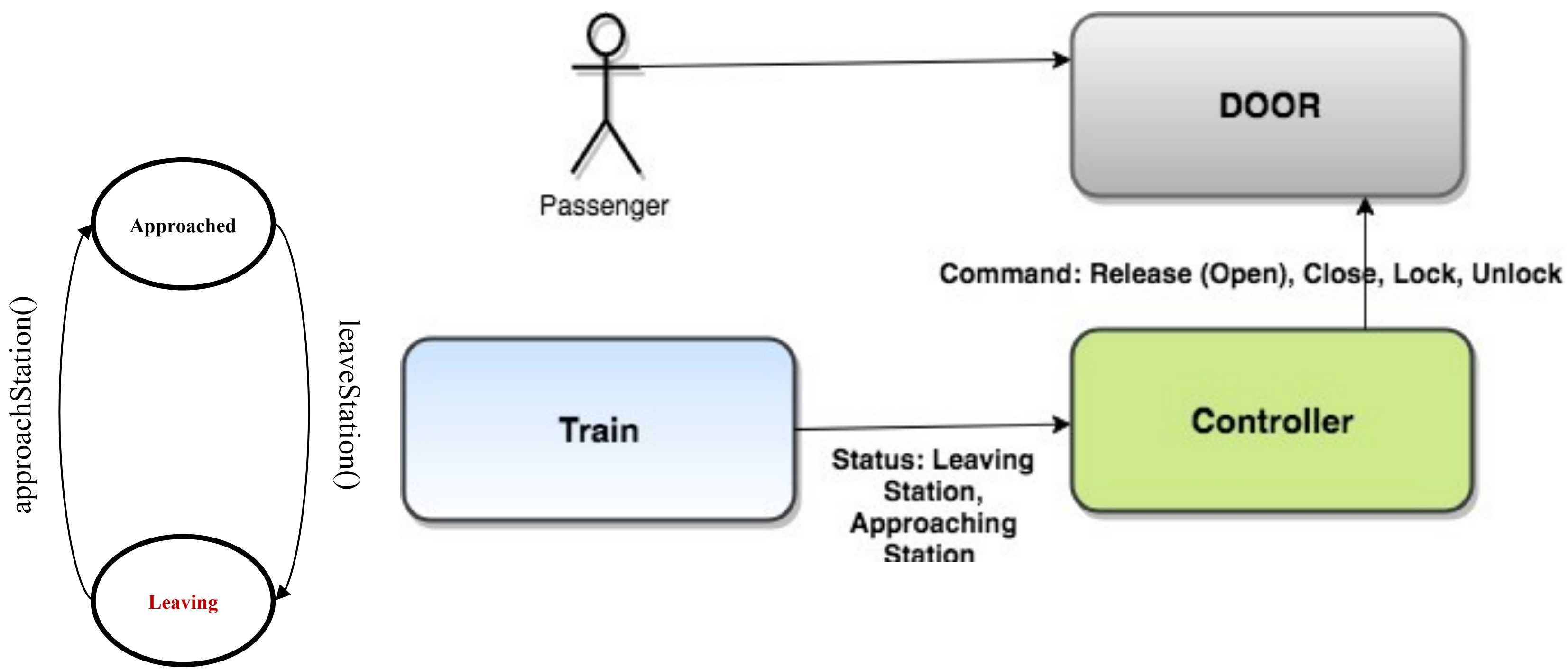


Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam, Mehrdad Saadatmand: Towards a Verification-Driven Iterative Development of Software for Safety-Critical Cyber-Physical Systems, Journal of Internet Services and Applications, 2021 https://rebeca-lang.org/assets/papers/2020/Towards-a-Verification-Driven-Iterative-Development-of-Cyber-Physical-System.pdf

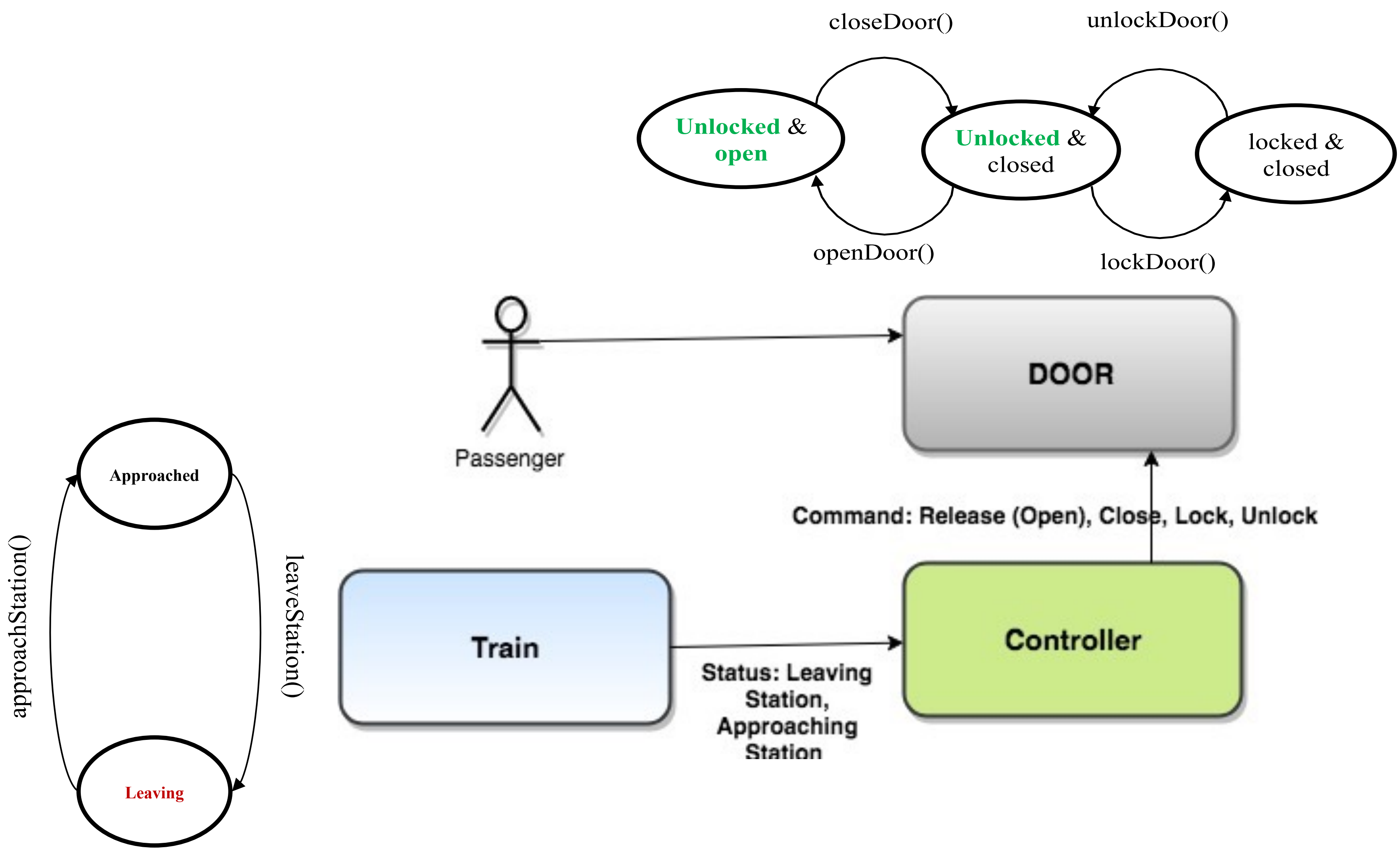
Architecture



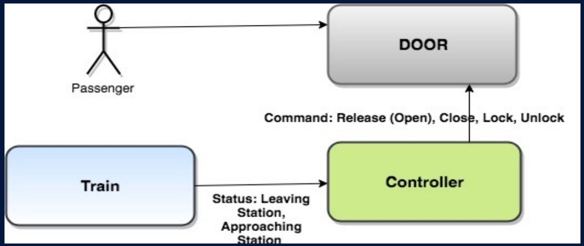
Architecture

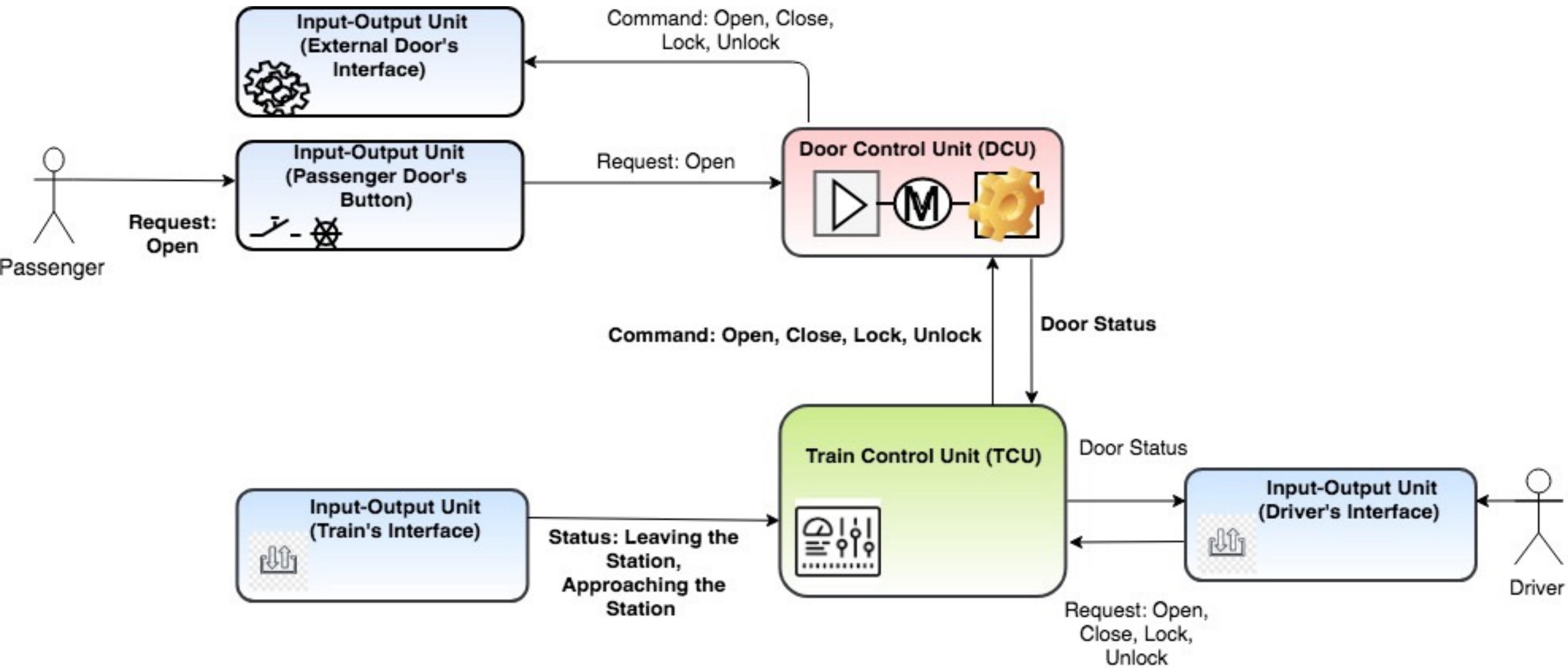


Architecture

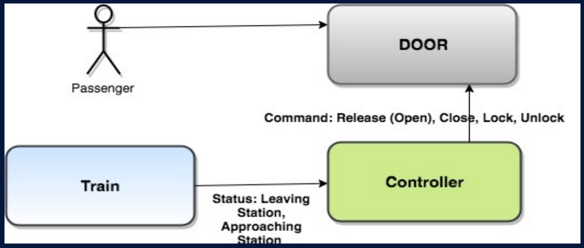


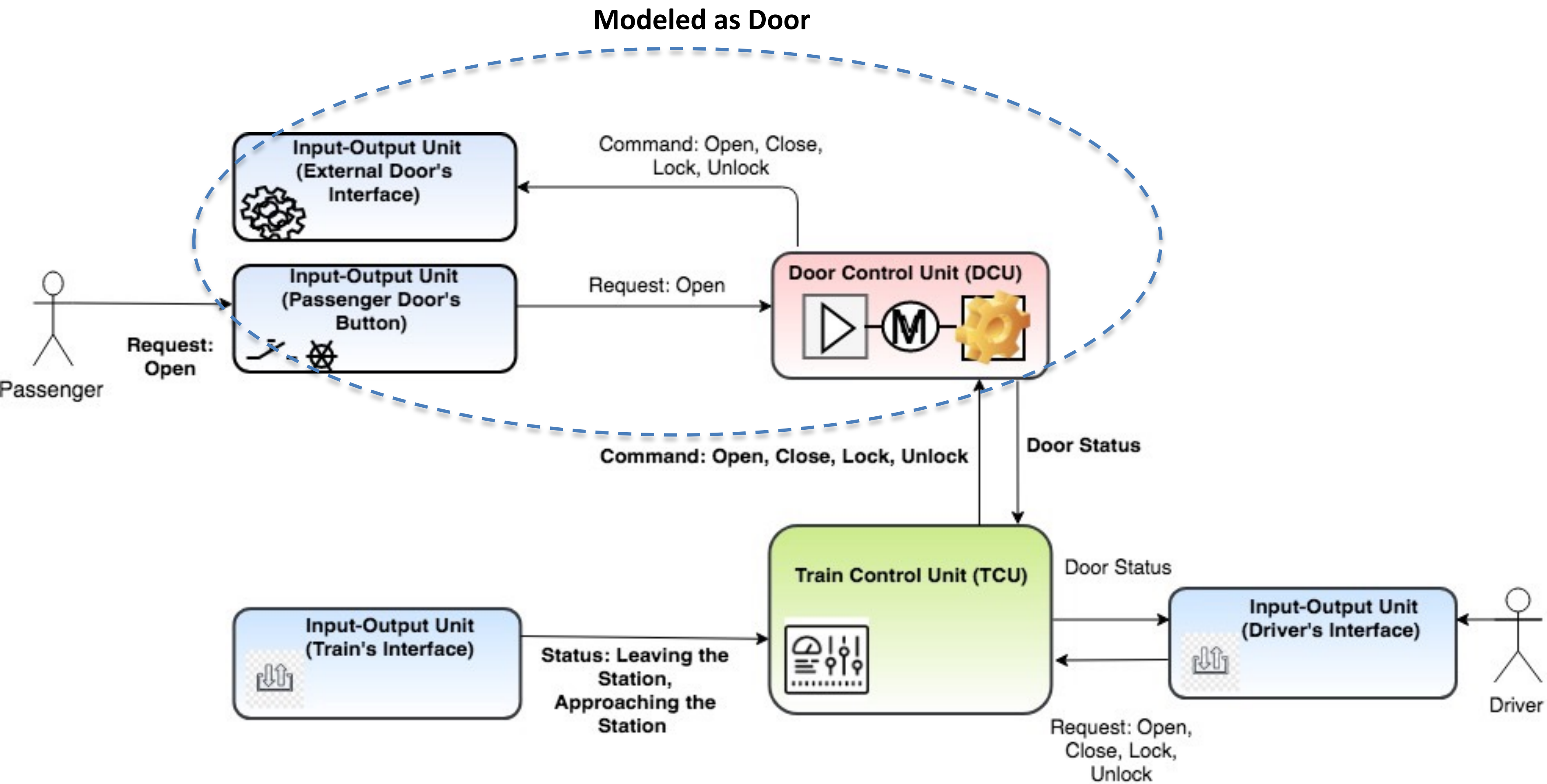
Architecture as Actors



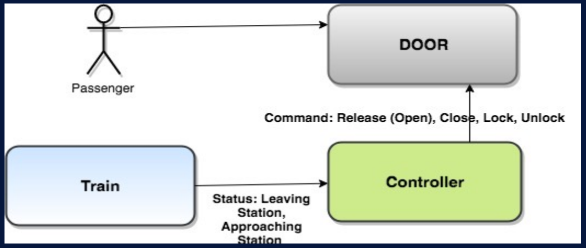


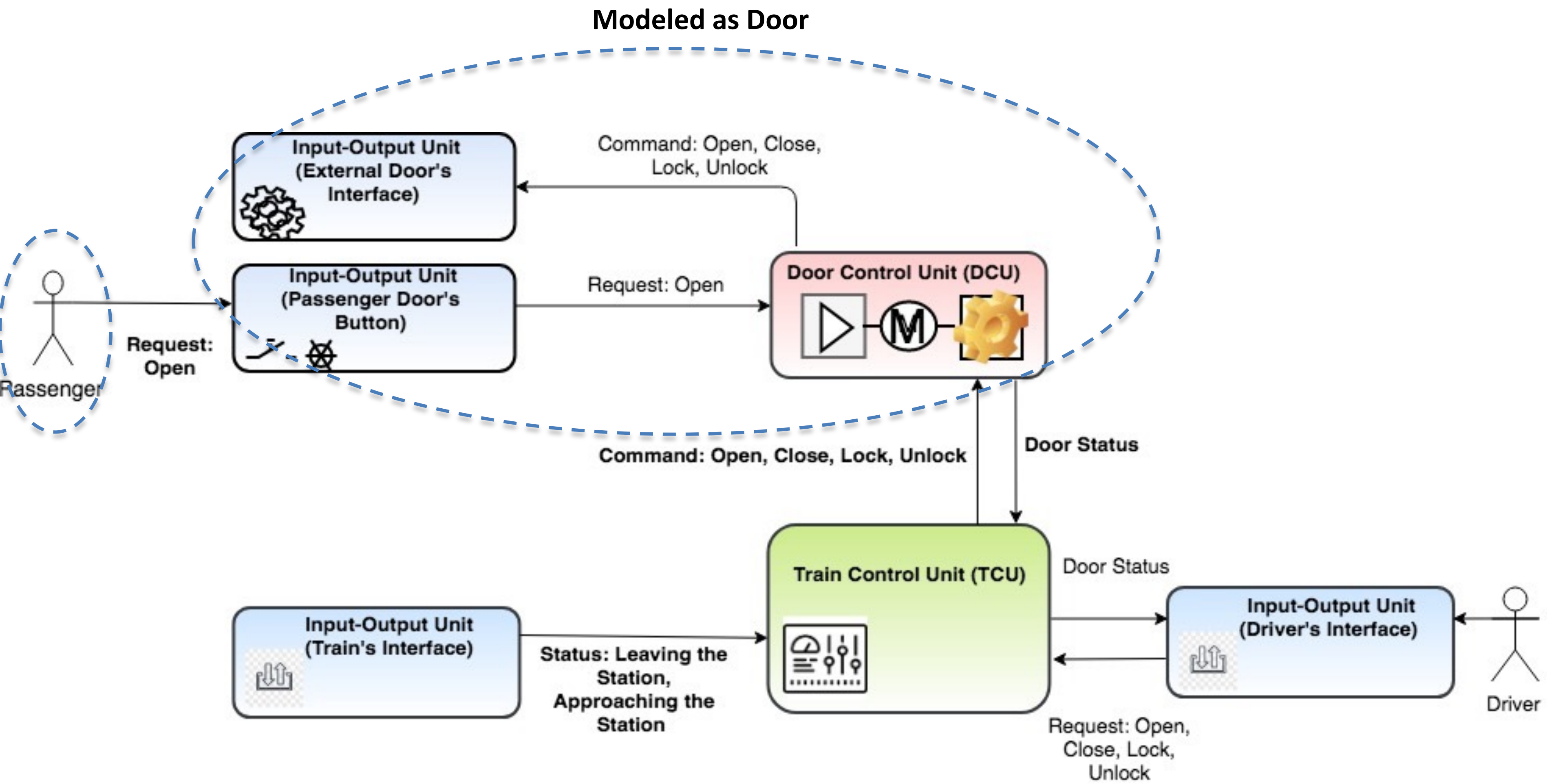
Architecture as Actors



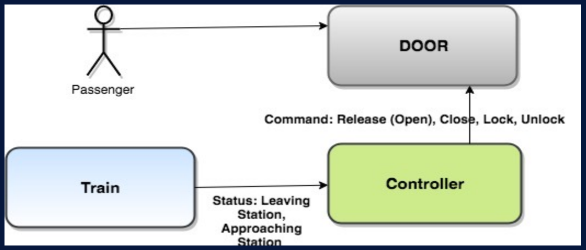


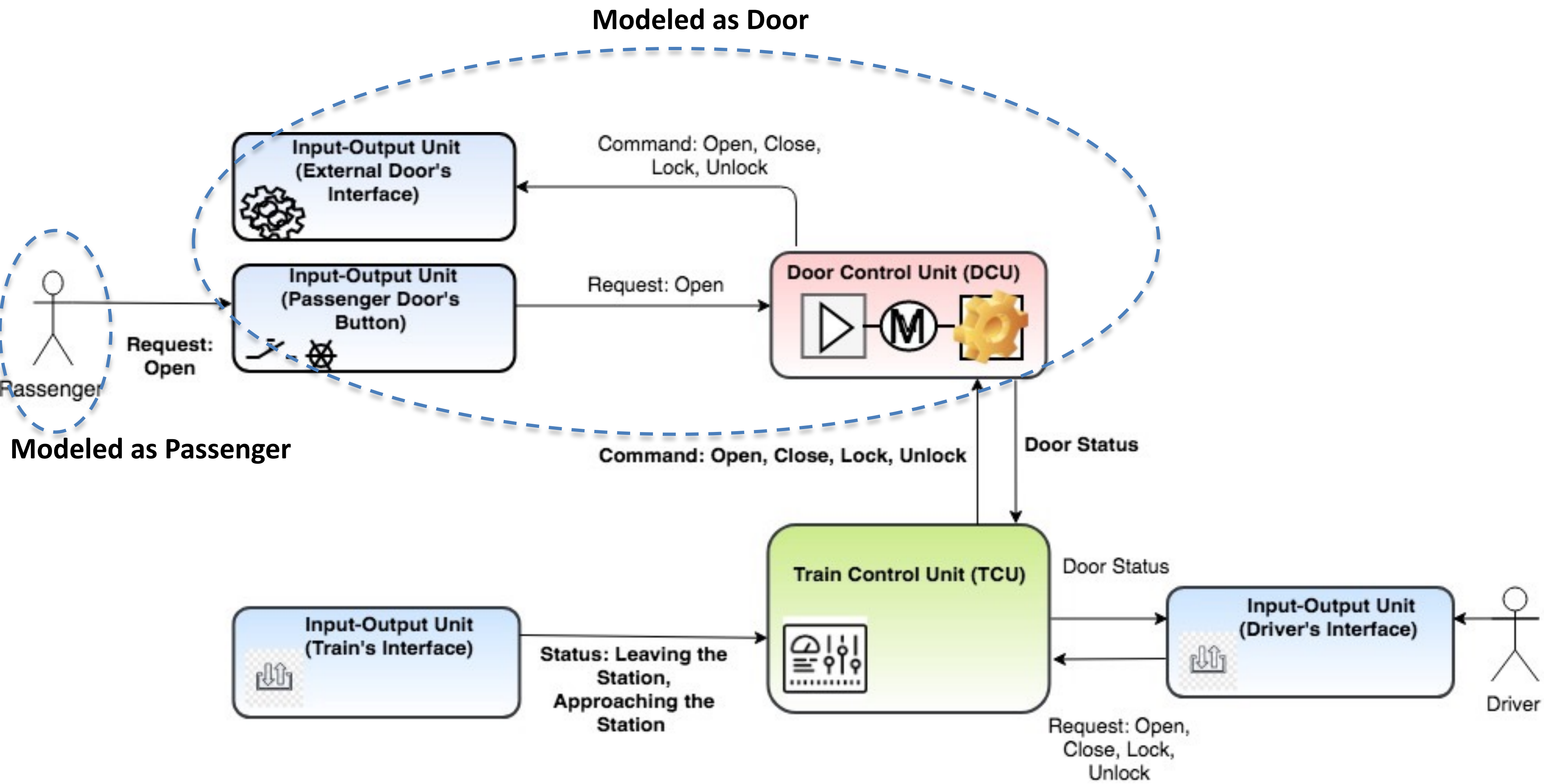
Architecture as Actors



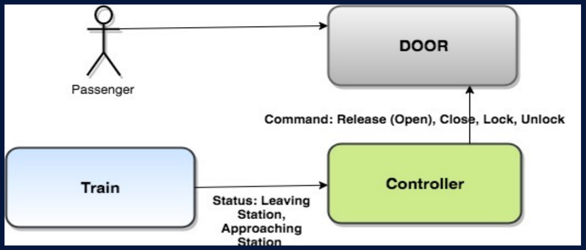


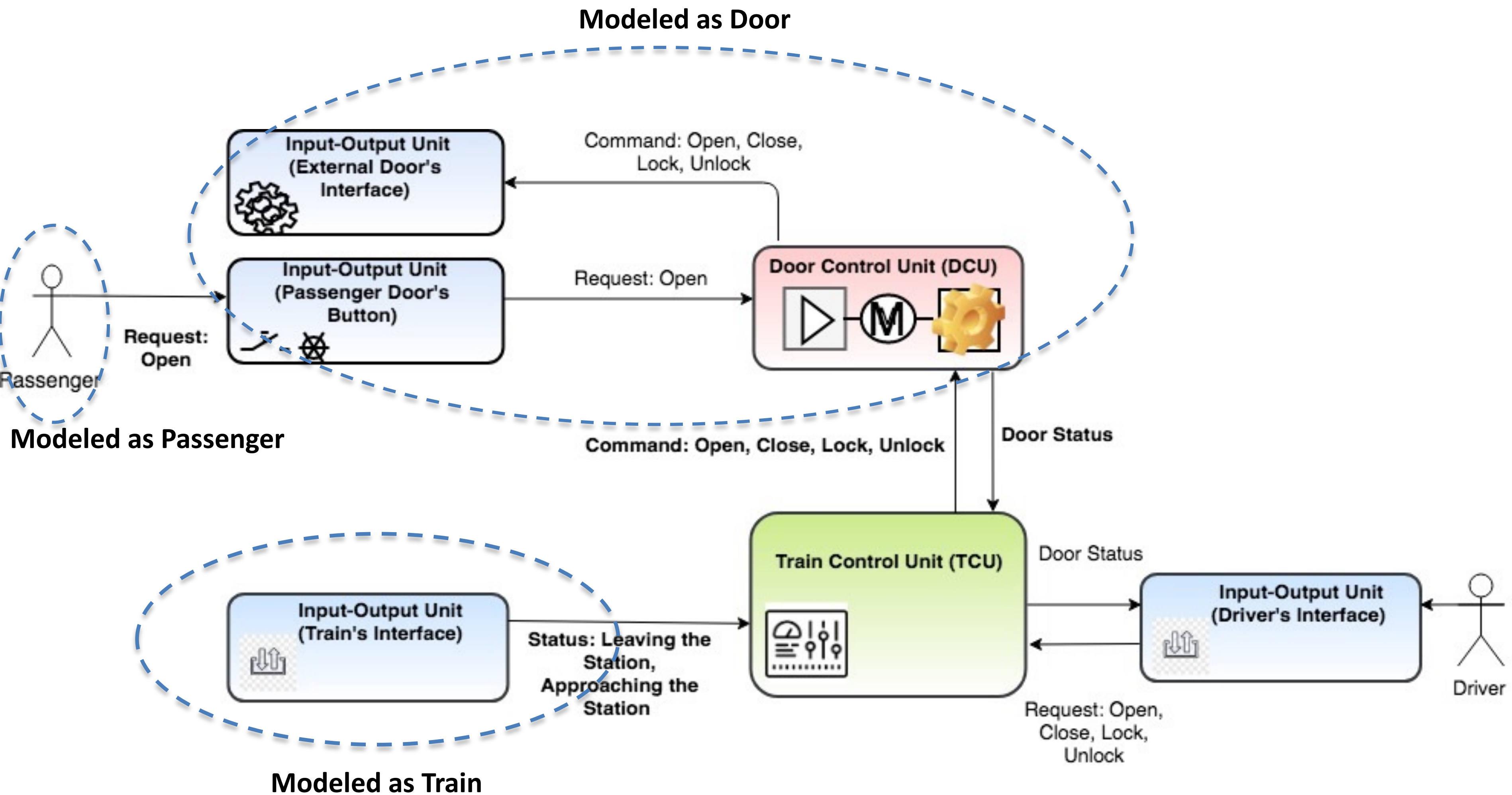
Architecture as Actors



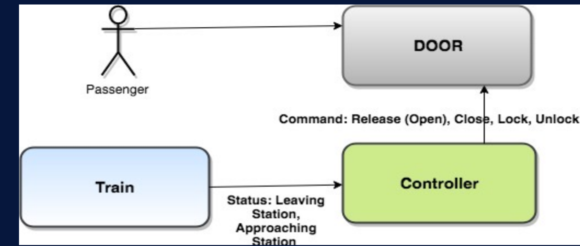


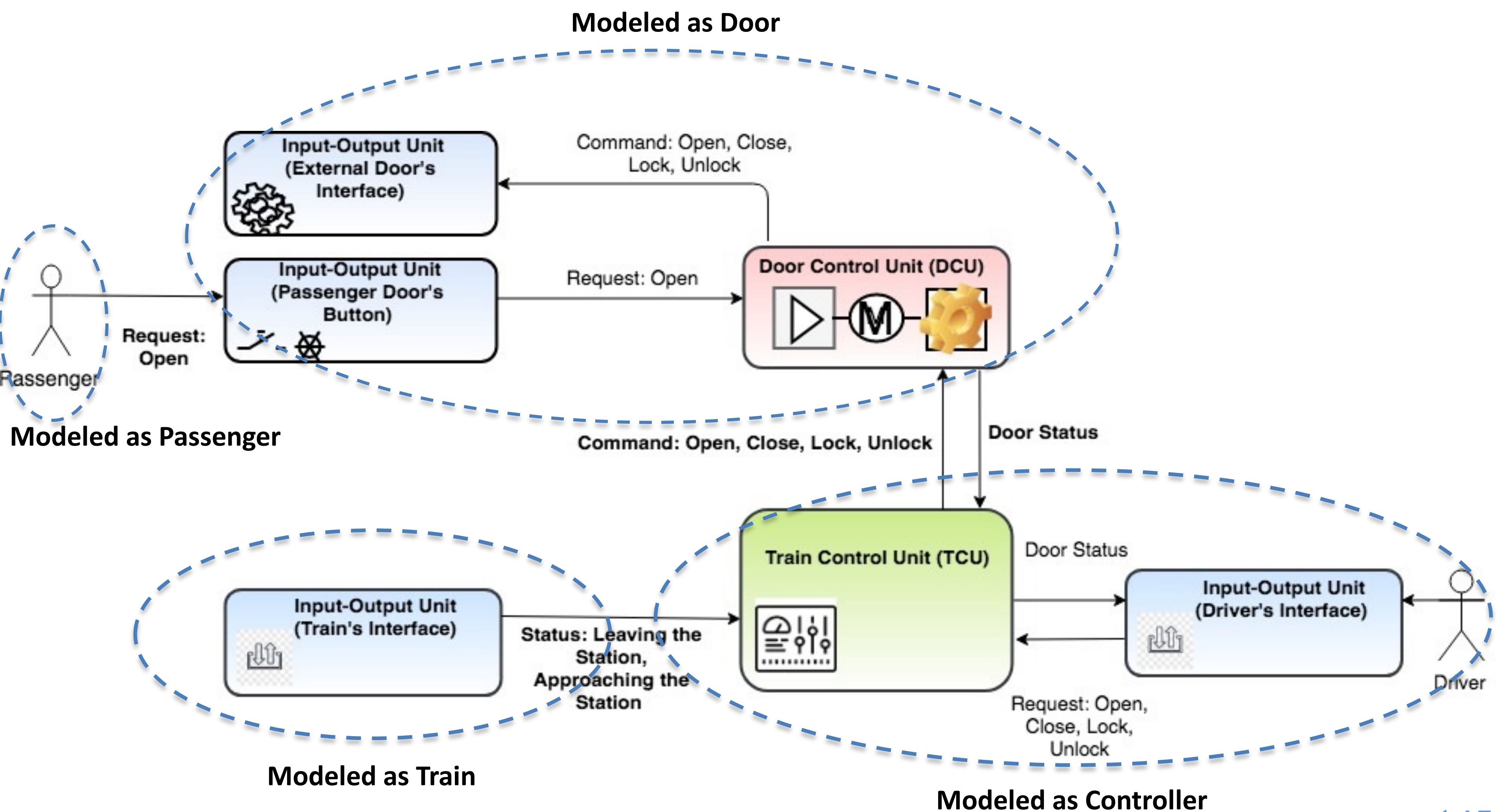
Architecture as Actors





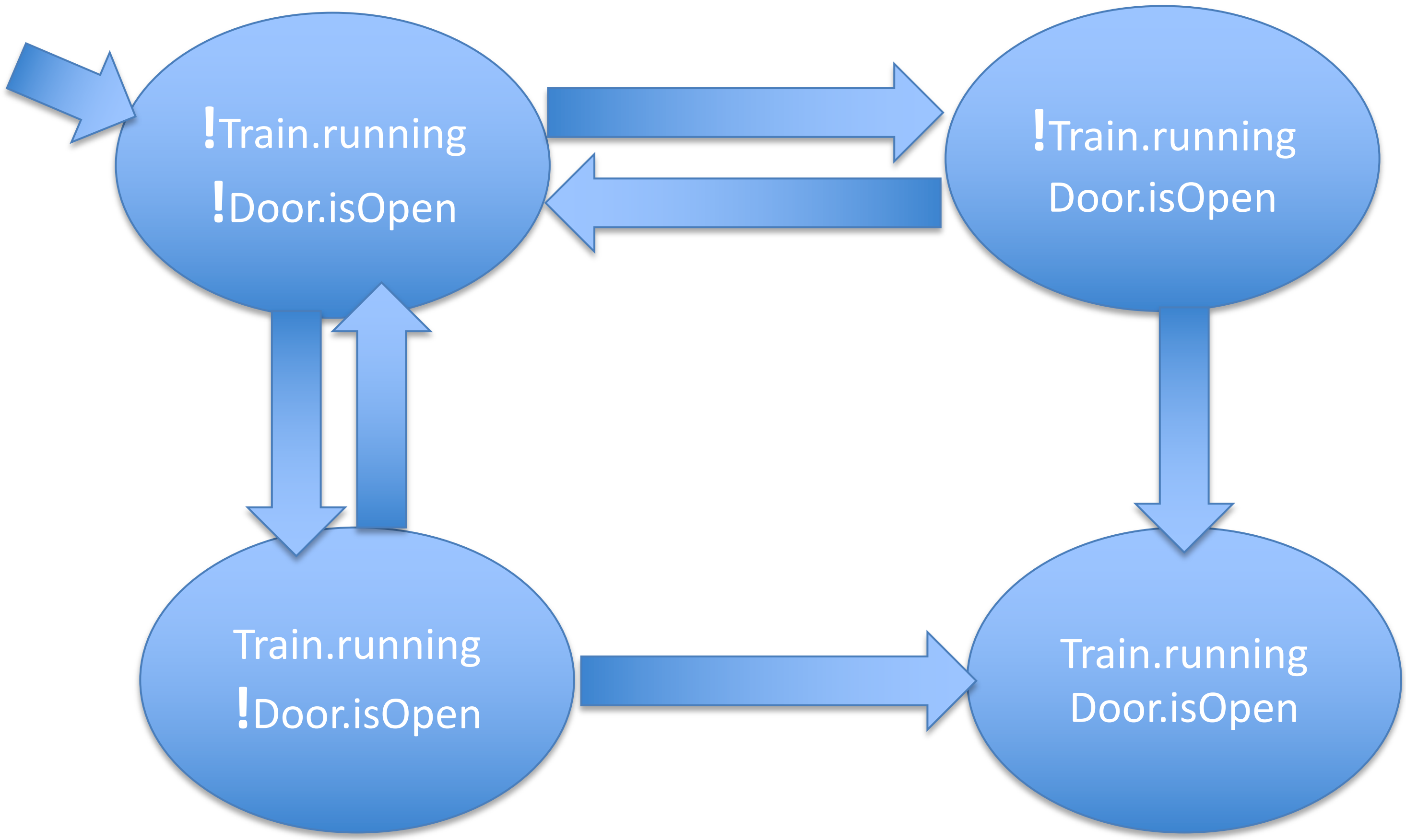
Architecture as Actors





Properties

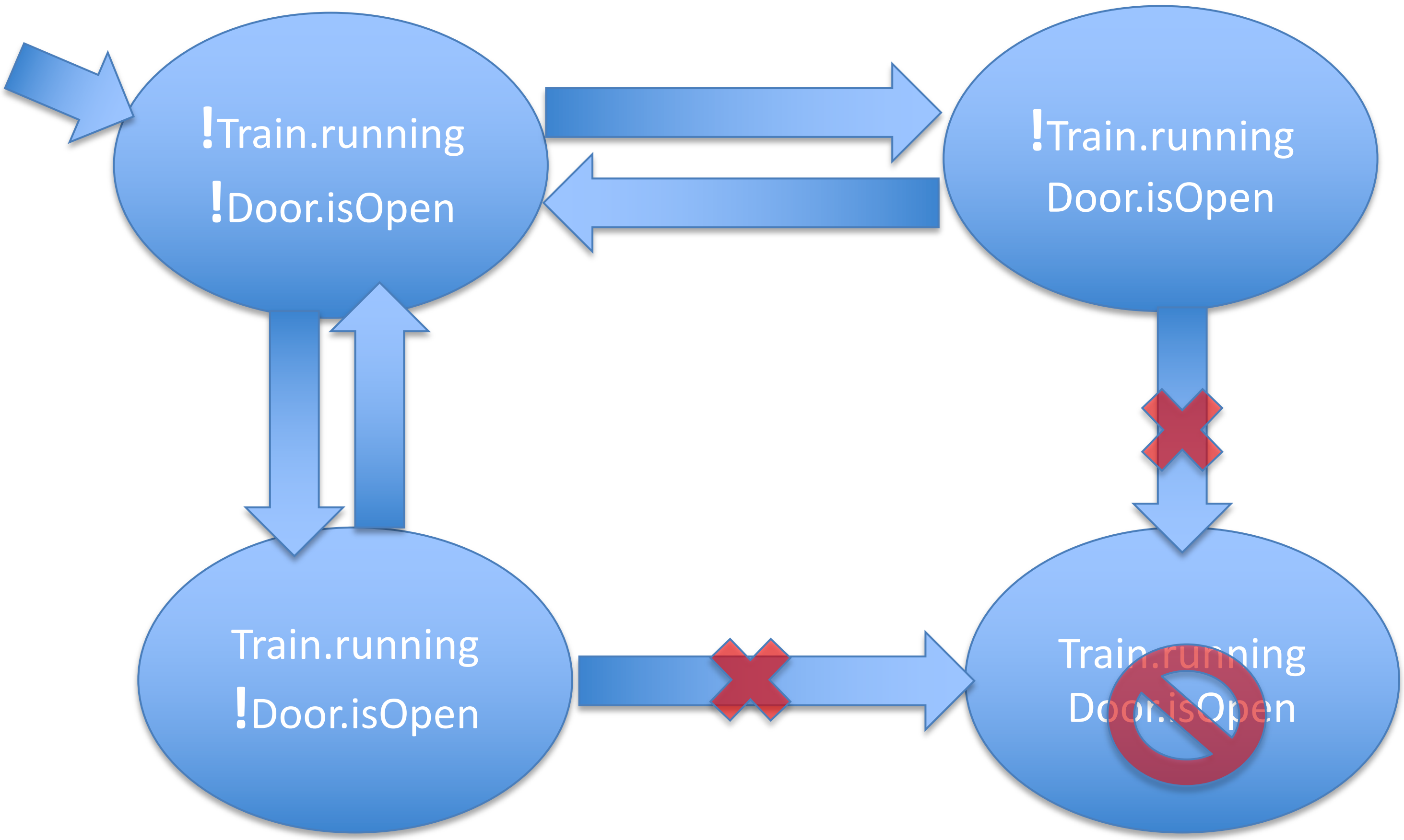
REQ ID	REQ DESCRIPTION	Elicited REQ ID
SSysSpecReq1	GIVEN the train is ready to run WHEN the driver requests to lock the external doors THEN all the external doors in the train shall be closed and locked	SSysReq1



Doors must not be open while the train is running.

Properties

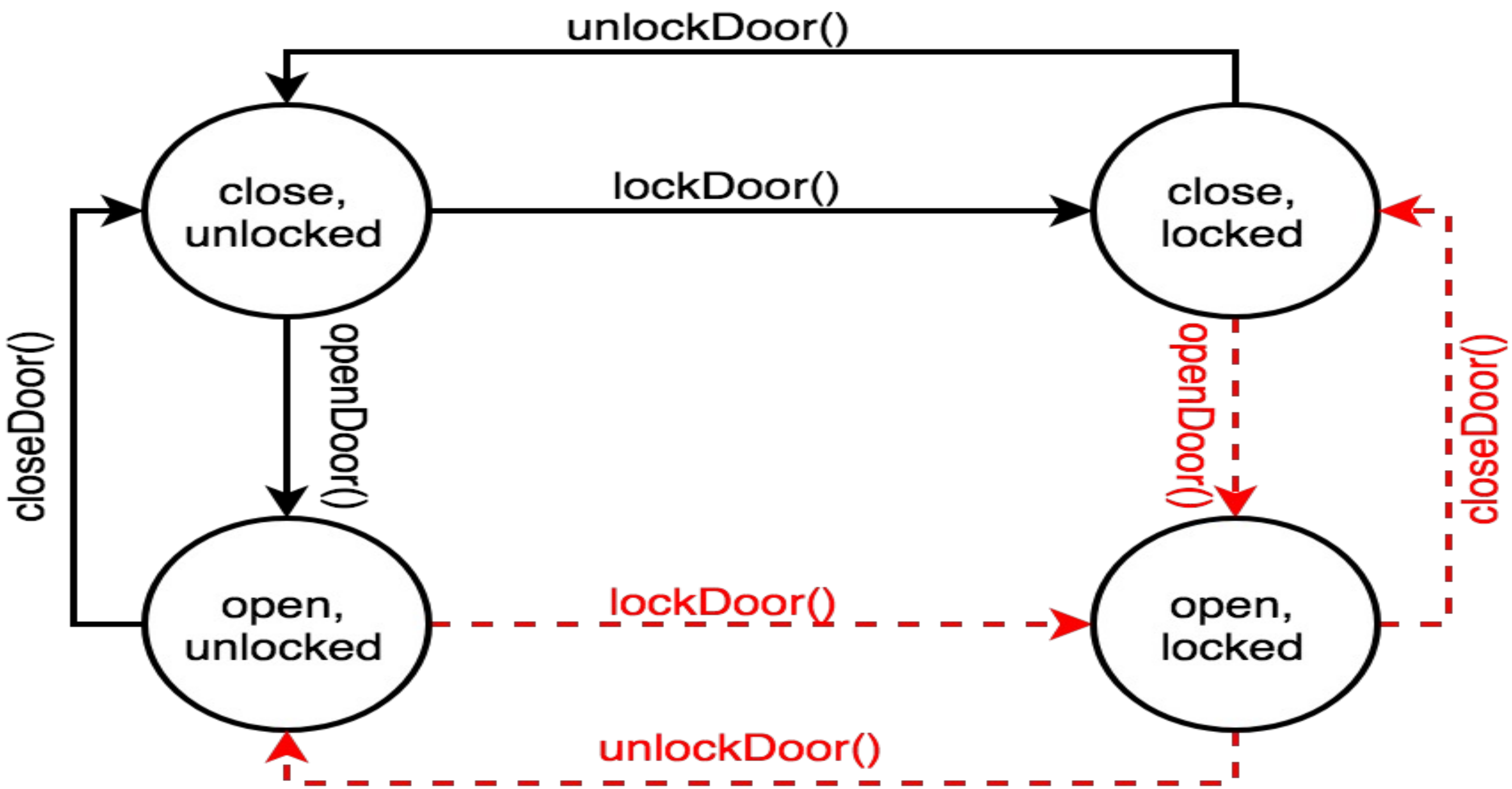
REQ ID	REQ DESCRIPTION	Elicited REQ ID
SSysSpecReq1	GIVEN the train is ready to run WHEN the driver requests to lock the external doors THEN all the external doors in the train shall be closed and locked	SSysReq1



Doors must not be open while the train is running.

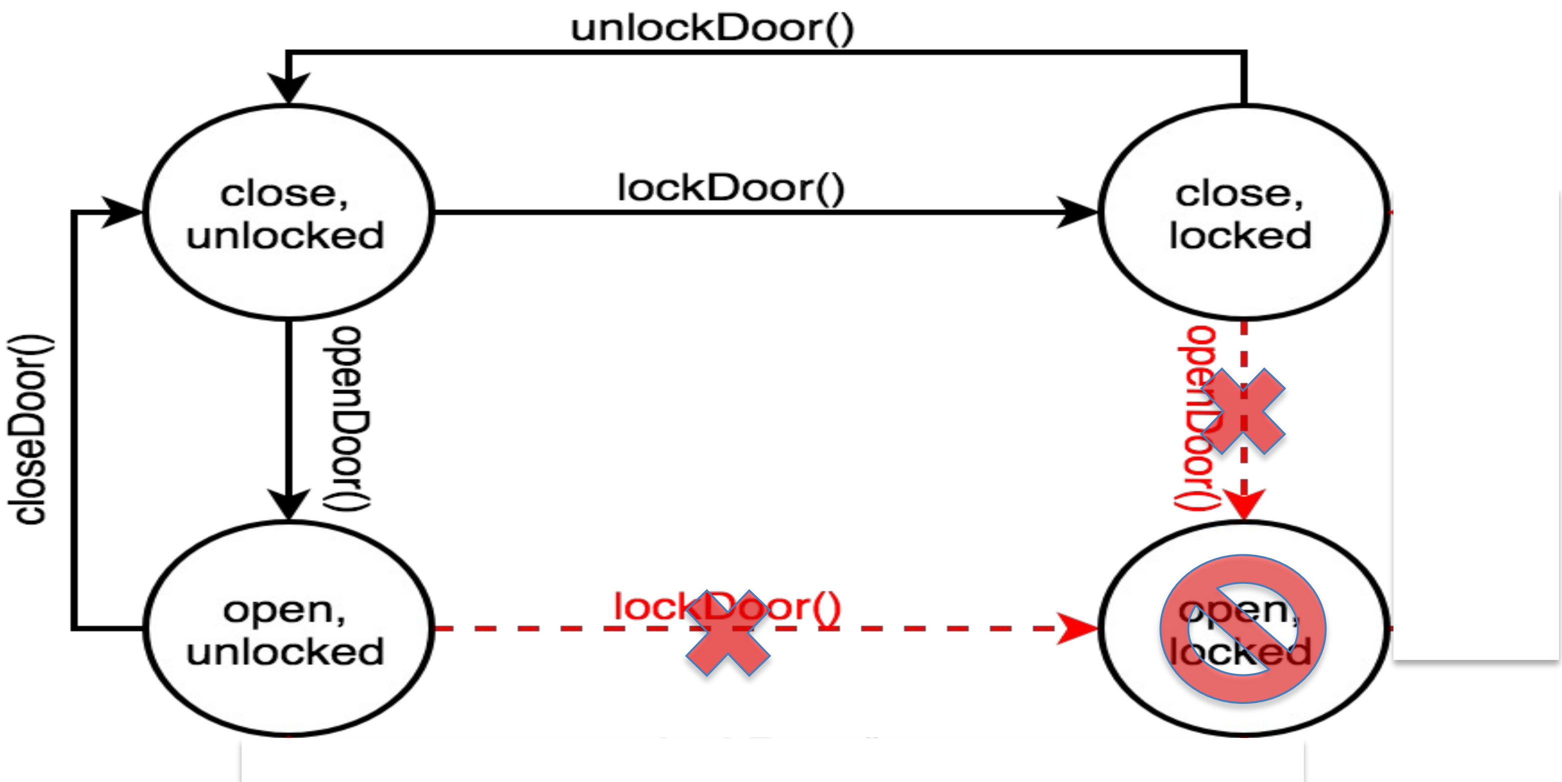
Properties

We want to verify that it is not possible to open a locked door or lock an open door.

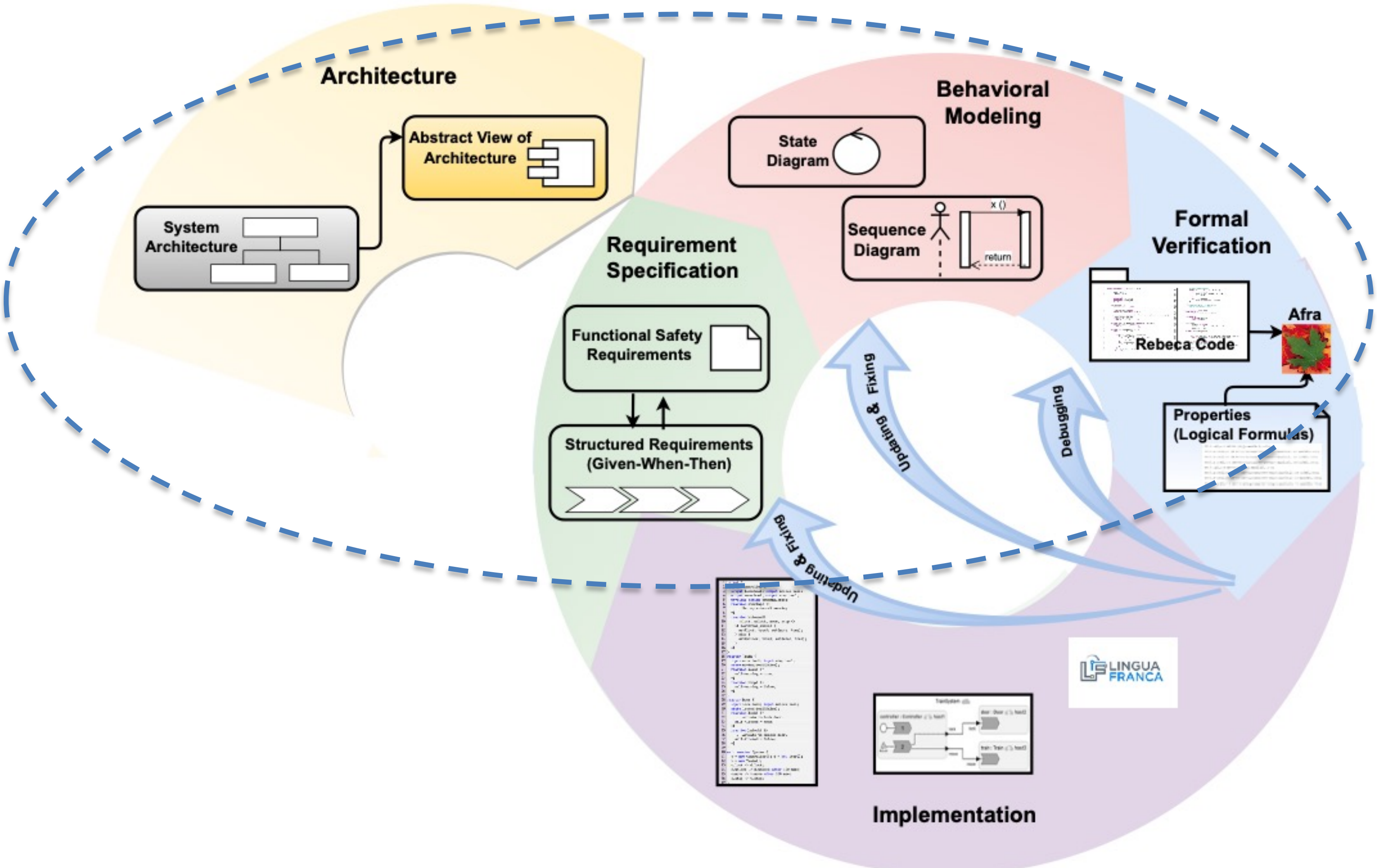


Properties

We want to verify that it is not possible to open a locked door or lock an open door.

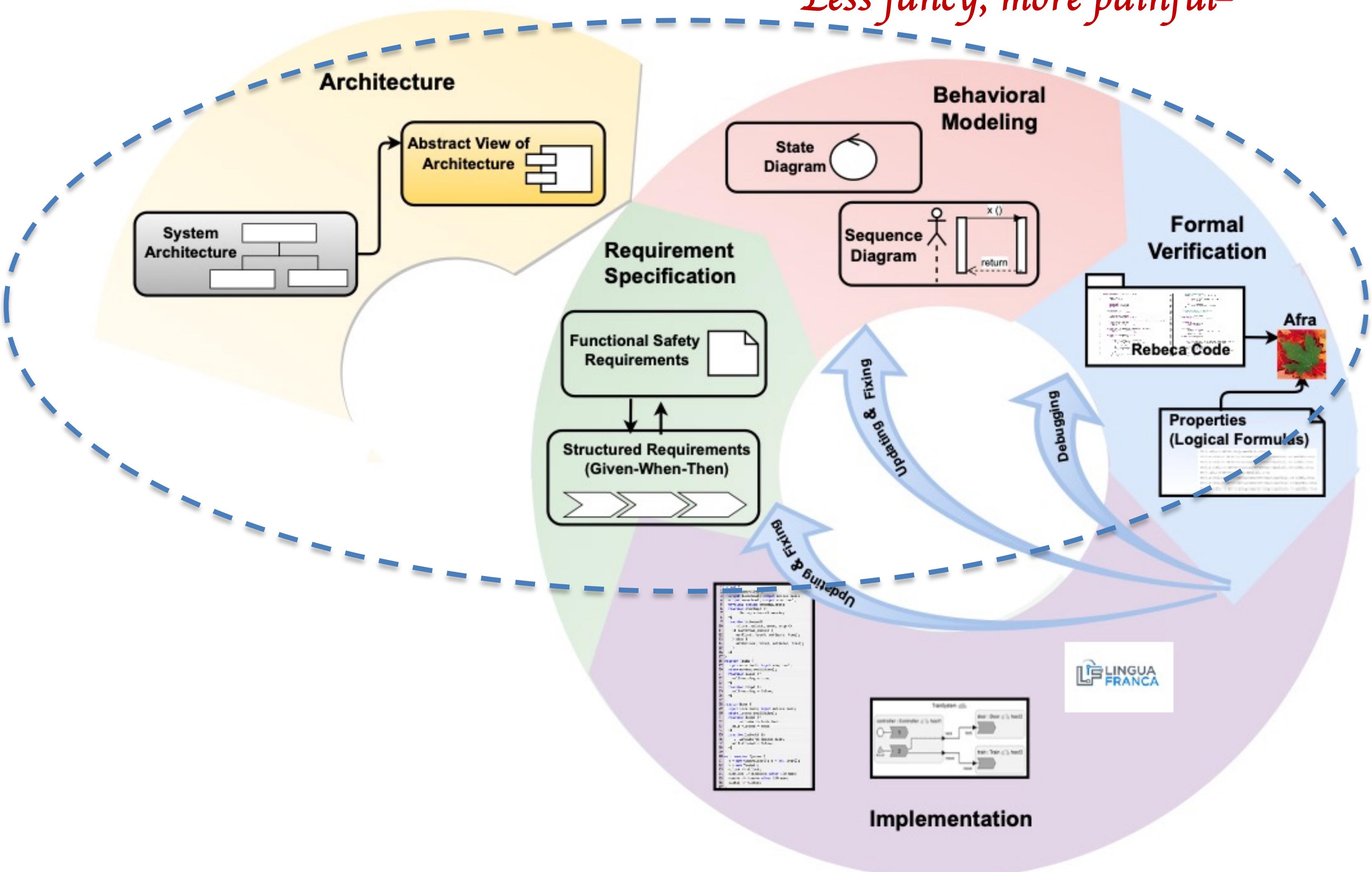


Reality:
Iterative and Incremental Process



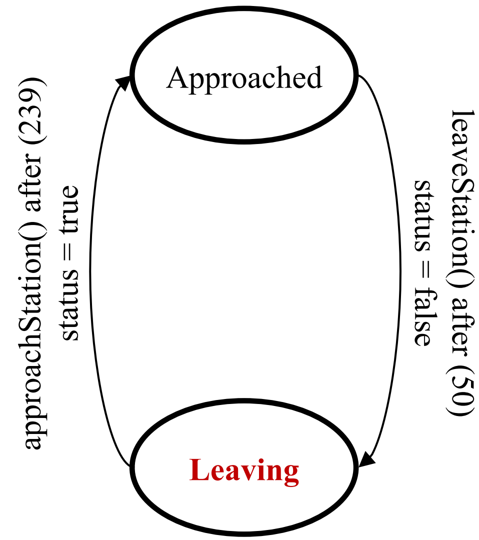
Reality:
Iterative and Incremental Process

~~Less fancy, more painful~~

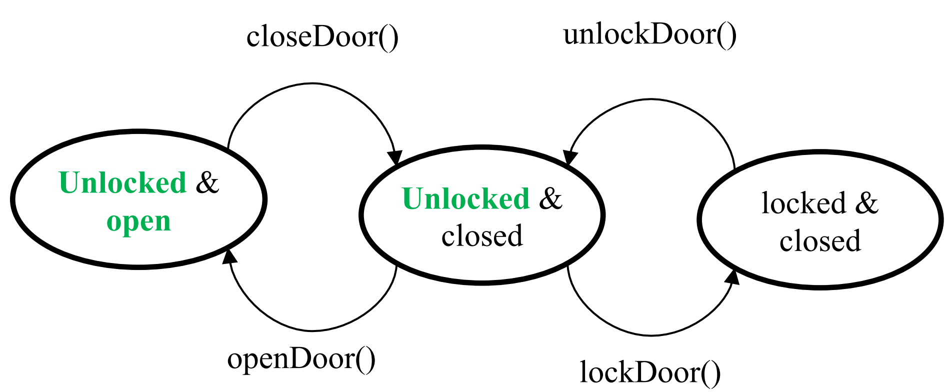


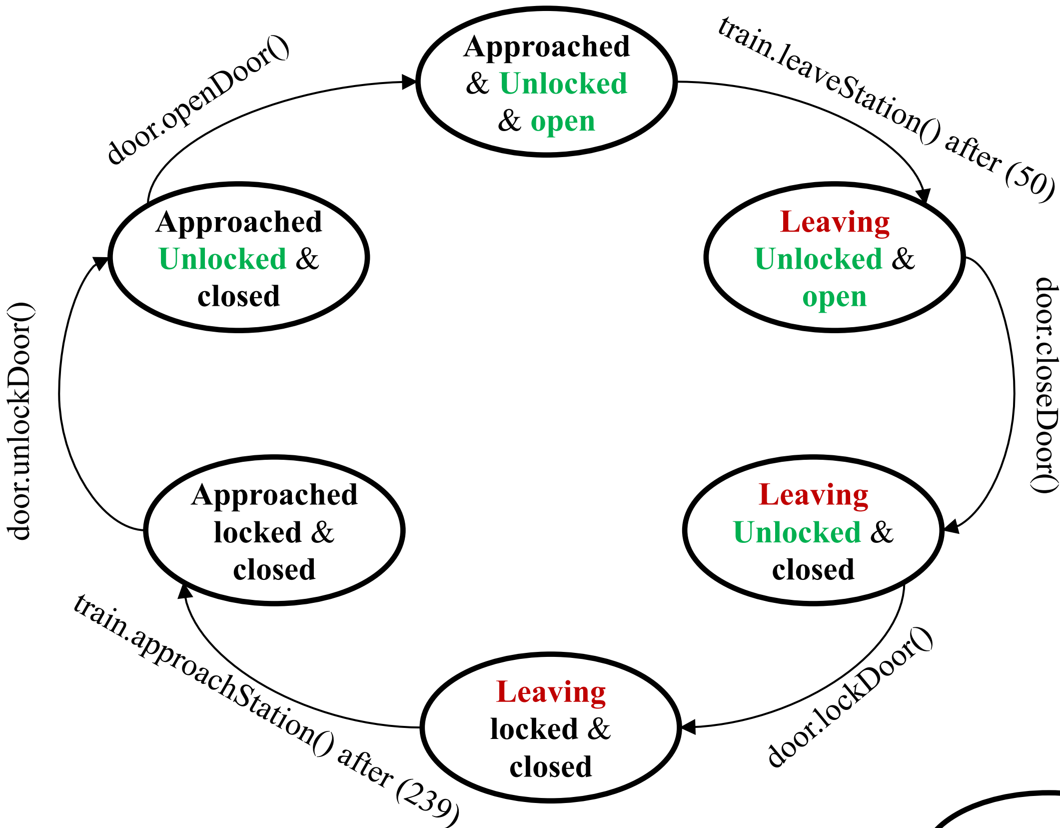
State Diagrams

Train



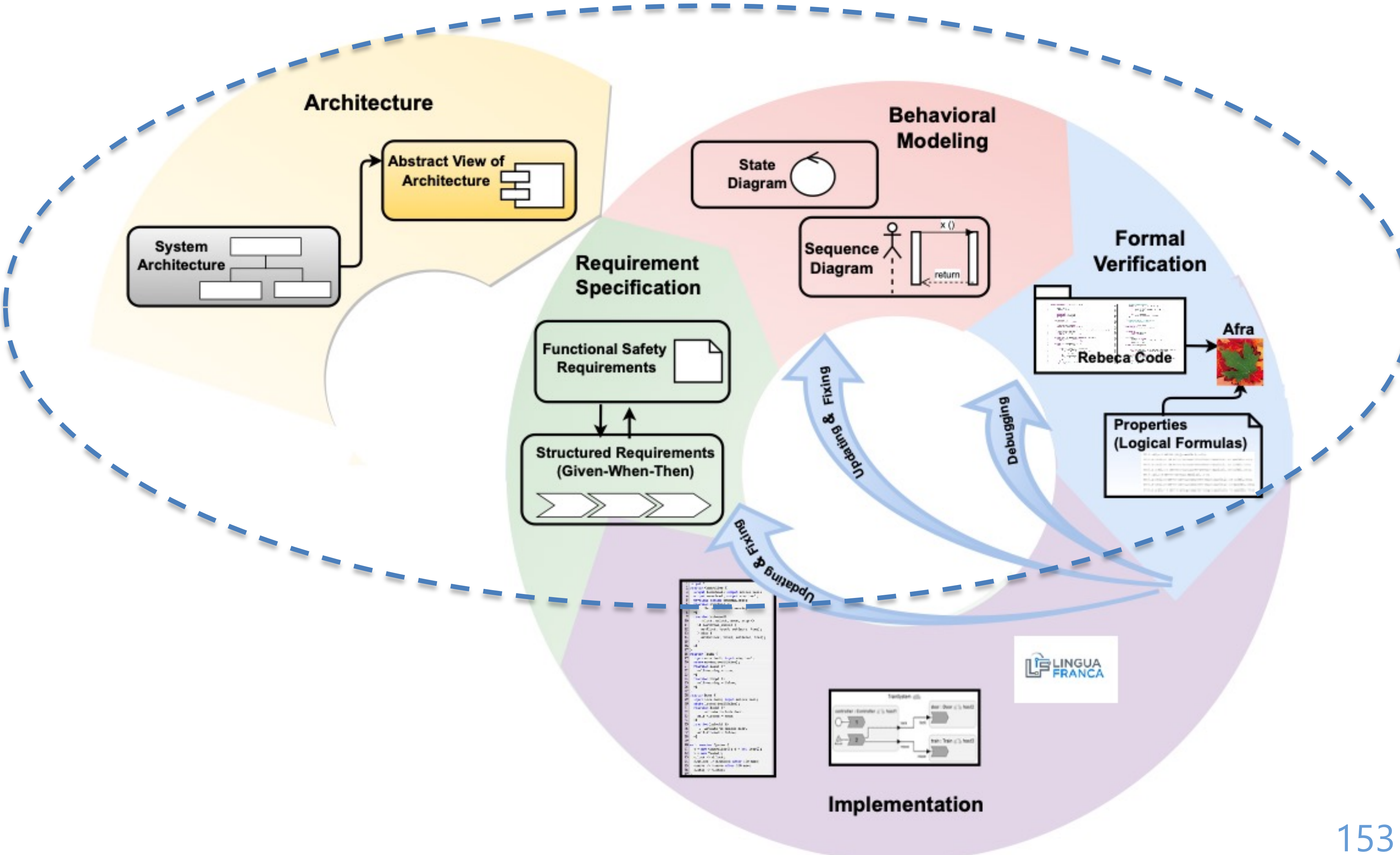
Door





Controller

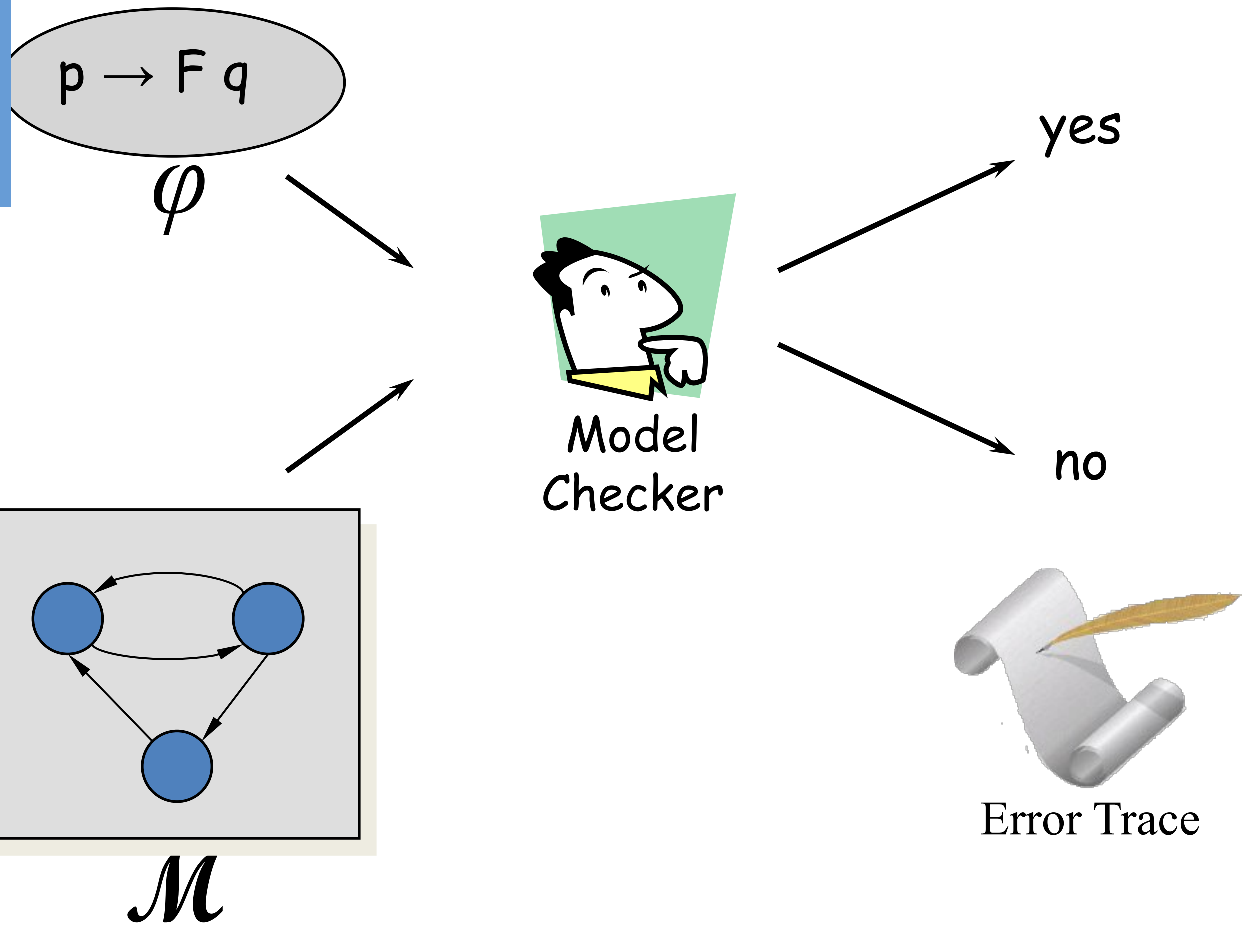
Process: Continue to Formal Verification



Model Checking: Prove Properties

If an **Operator** is **too close** then the **Robot** should **stand still.**

If the **Train** is **running** then the **Doors** should be **Closed.**

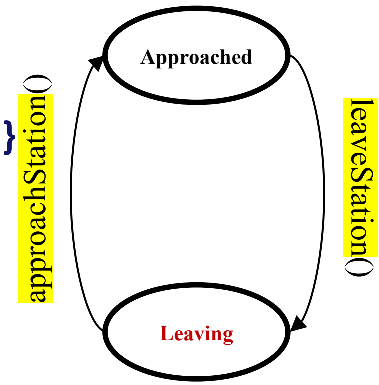



```
reactiveclass Controller(5) {
knownrebecs {
Door door;
Train train;
}
statevars { boolean moveP; }
Controller() {
self.external();
}
msgsrv external() {
boolean oldMoveP = moveP;
moveP = ?(true,false);
if(moveP != oldMoveP) {
door.lock(moveP);
train.move(moveP);
}
self.external() after(1);
}
}
reactiveclass Train(5) {
statevars { boolean moving; }
Train() {
moving = false;
}
msgsrv move(boolean tmove) {
if (tmove) {
moving = true;
} else {
moving = false;
}
}
}
reactiveclass Door(5) {
statevars { boolean is_locked; }
Door() {
is_locked = false;
}
msgsrv lock (boolean lockPar) {
is_locked = lockPar;
}
}
main {
@priority(1) Controller controller(door,
train):();
@priority(2) Train train():();
@priority(2) Door door():();
}
```

```
reactiveclass Train(10){
   knownrebecs{
      Controller controller; }
   statevars{
      boolean status;}

   Train(){
     status = true;
     self.leaveStation();
    }
    msgsrv leaveStation(){
       status = true;
       controller.setTrainStatus(status)
                           after(networkDelayTrain);
       self.approachStation() after (runningTime);
    }
     msgsrv approachStation(){
        status = false;
        controller.setTrainStatus(status)
                           after(networkDelayTrain);
        self.leaveStation() after(atStationTime);
     }
}
```

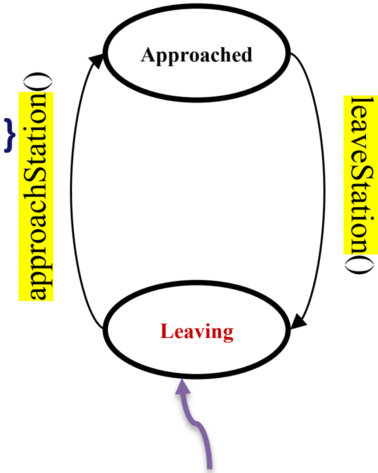
```
reactiveclass Door(15){
   knownrebecs{
      Controller controller;}
   statevars{
      boolean isDoorClosed, isDoorLocked;}

    Door(){
      isDoorClosed = false;  isDoorLocked = false;
     }
    msgsrv closeDoor(){
      isDoorClosed = true;
      controller.setDoorStatus(isDoorClosed,
          isDoorLocked) after(networkDelayDoor);
    }
    msgsrv lockDoor(){
       isDoorLocked = true;
       controller.setDoorStatus(…);
    }
    msgsrv unlockDoor(){…}
    msgsrv openDoor(){…}
  }
```

```
reactiveclass Train(10){
  knownrebecs{
    Controller controller; }
  statevars{
    boolean status;}

  Train(){
   status = true;
   self.leaveStation();
  }
  msgsrv leaveStation(){
    status = true;
    controller.setTrainStatus(status)
                    after(networkDelayTrain);
    self.approachStation() after (runningTime);
  }
  msgsrv approachStation(){
    status = false;
    controller.setTrainStatus(status)
                    after(networkDelayTrain);
    self.leaveStation() after(atStationTime);
  }
}
```



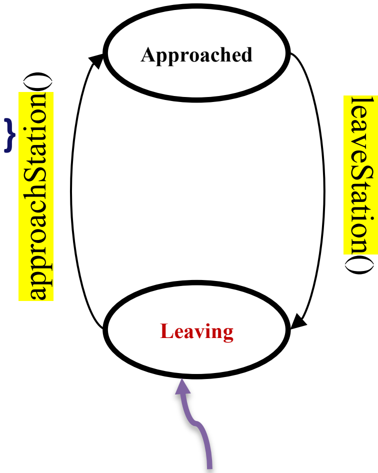

```
reactiveclass Door(15){
  knownrebecs{
    Controller controller;}
  statevars{
    boolean isDoorClosed, isDoorLocked;}

  Door(){
    isDoorClosed = false;  isDoorLocked = false;
  }
  msgsrv closeDoor(){
    isDoorClosed = true;
    controller.setDoorStatus(isDoorClosed,
       isDoorLocked) after(networkDelayDoor);
  }
  msgsrv lockDoor(){
    isDoorLocked = true;
    controller.setDoorStatus(...);
  }
  msgsrv unlockDoor(){...}
  msgsrv openDoor(){...}
}
```

```
reactiveclass Train(10){
  knownrebecs{
    Controller controller; }
  statevars{
    boolean status;}

  Train(){
   status = true;
   self.leaveStation();
  }
  msgsrv leaveStation(){
    status = true;
    controller.setTrainStatus(status)
                        after(networkDelayTrain);
    self.approachStation() after (runningTime);
  }
  msgsrv approachStation(){
     status = false;
     controller.setTrainStatus(status)
                        after(networkDelayTrain);
     self.leaveStation() after(atStationTime);
   }
}
```

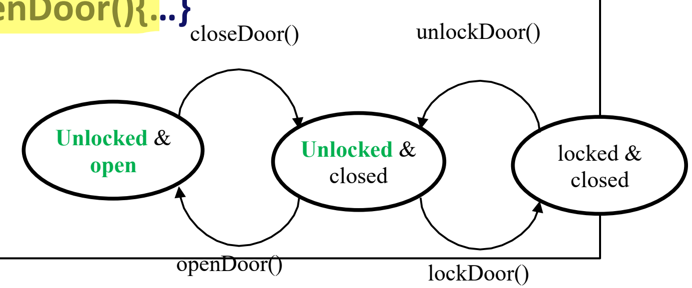



```
reactiveclass Door(15){
  knownrebecs{
    Controller controller;}
  statevars{
    boolean isDoorClosed, isDoorLocked;}

  Door(){
    isDoorClosed = false;  isDoorLocked = false;
   }
  msgsrv closeDoor(){
    isDoorClosed = true;
    controller.setDoorStatus(isDoorClosed,
        isDoorLocked) after(networkDelayDoor);
  }
  msgsrv lockDoor(){
     isDoorLocked = true;
     controller.setDoorStatus(…);
  }
  msgsrv unlockDoor(){…}
  msgsrv openDoor(){…}
}
```

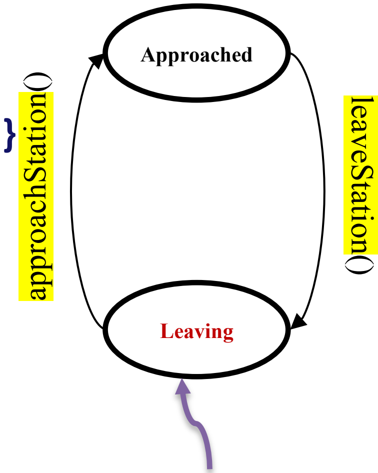
```
reactiveclass Train(10){
    knownrebecs{
        Controller controller; }
    statevars{
        boolean status;}

    Train(){
      status = true;
      self.leaveStation();
    }
    msgsrv leaveStation(){
        status = true;
        controller.setTrainStatus(status)
                                after(networkDelayTrain);
        self.approachStation() after (runningTime);
    }
    msgsrv approachStation(){
        status = false;
        controller.setTrainStatus(status)
                                after(networkDelayTrain);
        self.leaveStation() after(atStationTime);
    }
}
```



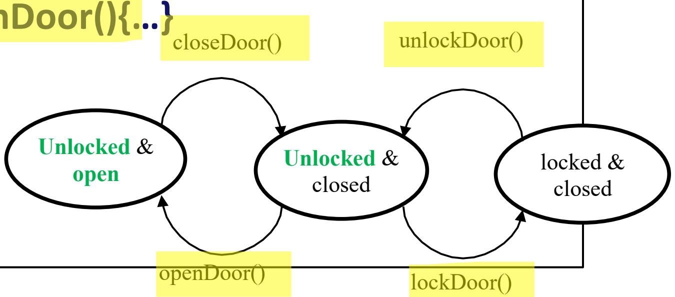

```
reactiveclass Door(15){
    knownrebecs{
        Controller controller;}
    statevars{
        boolean isDoorClosed, isDoorLocked;}

    Door(){
        isDoorClosed = false;  isDoorLocked = false;
    }
    msgsrv closeDoor(){
        isDoorClosed = true;
        controller.setDoorStatus(isDoorClosed,
            isDoorLocked) after(networkDelayDoor);
    }
    msgsrv lockDoor(){
        isDoorLocked = true;
        controller.setDoorStatus(...);
    }
    msgsrv unlockDoor(){...}
    msgsrv openDoor(){...}
}
```



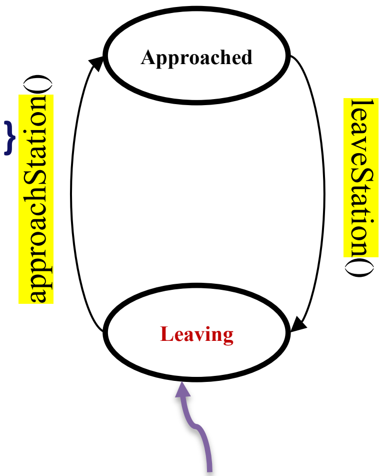
```
reactiveclass Train(10){
    knownrebecs{
        Controller controller; }
    statevars{
        boolean status;}

    Train(){
      status = true;
      self.leaveStation();
    }
    msgsrv leaveStation(){
        status = true;
        controller.setTrainStatus(status)
                                after(networkDelayTrain);
        self.approachStation() after (runningTime);
    }
    msgsrv approachStation(){
        status = false;
        controller.setTrainStatus(status)
                                after(networkDelayTrain);
        self.leaveStation() after(atStationTime);
    }
}
```



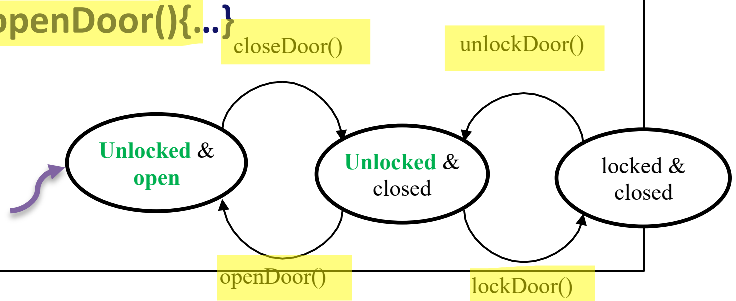

```
reactiveclass Door(15){
    knownrebecs{
        Controller controller;}
    statevars{
        boolean isDoorClosed, isDoorLocked;}

    Door(){
        isDoorClosed = false;  isDoorLocked = false;
    }
    msgsrv closeDoor(){
        isDoorClosed = true;
        controller.setDoorStatus(isDoorClosed,
            isDoorLocked) after(networkDelayDoor);
    }
    msgsrv lockDoor(){
        isDoorLocked = true;
        controller.setDoorStatus(…);
    }
    msgsrv unlockDoor(){…}
    msgsrv openDoor(){…}
}
```



```
reactiveclass Train(10){
    knownrebecs{
        Controller controller; }
    statevars{
        boolean status;}

    Train(){
      status = true;
      self.leaveStation();
    }
    msgsrv leaveStation(){
        status = true;
        controller.setTrainStatus(status)
                                after(networkDelayTrain);
        self.approachStation() after (runningTime);
    }
    msgsrv approachStation(){
        status = false;
        controller.setTrainStatus(status)
                                after(networkDelayTrain);
        self.leaveStation() after(atStationTime);
    }
}
```



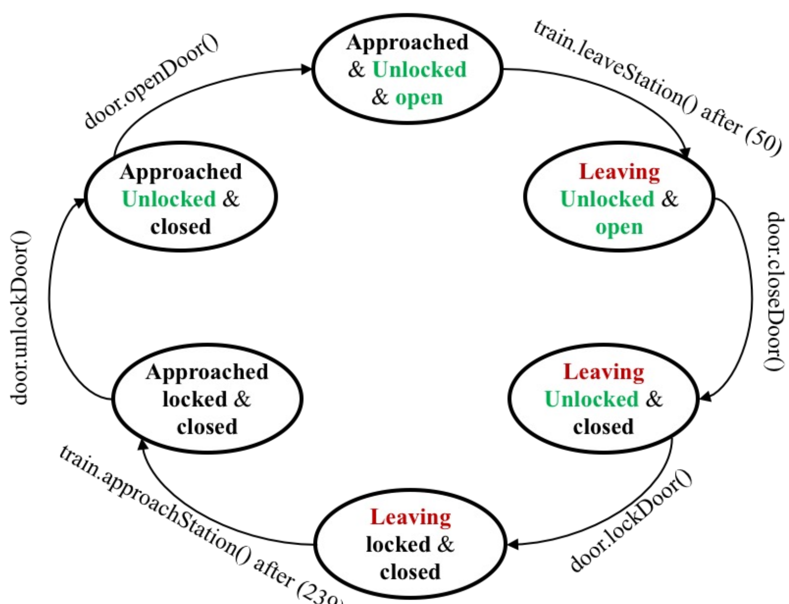

```
reactiveclass Door(15){
    knownrebecs{
        Controller controller;}
    statevars{
        boolean isDoorClosed, isDoorLocked;}

    Door(){
        isDoorClosed = false;  isDoorLocked = false;
    }
    msgsrv closeDoor(){
        isDoorClosed = true;
        controller.setDoorStatus(isDoorClosed,
            isDoorLocked) after(networkDelayDoor);
    }
    msgsrv lockDoor(){
        isDoorLocked = true;
        controller.setDoorStatus(...);
    }
    msgsrv unlockDoor(){...}
    msgsrv openDoor(){...}
}
```



```
reactiveclass controller(10){
    knownrebecs{
        Door door; }
    statevars{
        boolean isClosed, isLocked, trainStatus;}

    Controller(){
      trainStatus = true; isClosed, isLocked = false;
     }
     msgsrv setDoorStatus(boolean close, lock){
       isClosed = close; isLocked = lock;
     }
      msgsrv setTrainStatus(boolean status){
          trainStatus = status;
          self.driveController();
       }
}
```

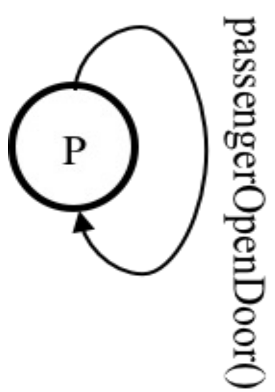
```
msgsrv driveController(){
    if(trainStatus){   // leave the station
        if(!isClosed || !isLocked) {
            if(!isClosed) {
                door.closeDoor() after(nd);
                delay(reactionDelay);
             }
            if(!isLocked) {
                door.lockDoor() after(nd);
             }
        }
    }// end of if(trainStatus)
    else if(!trainStatus){  // arrive the station
       if(isClosed || isLocked) {
          if (isLocked) {
              door.unlockDoor() after(nd);
              delay(reactionDelay);
           }
          if (isClosed) {
             door.openDoor() after(nd);
           } }  ...
}
```

```
reactiveclass controller(10){
    knownrebecs{
        Door door; }
    statevars{
        boolean isClosed, isLocked, trainStatus;}

    Controller(){
      trainStatus = true; isClosed, isLocked = false;
    }
    msgsrv setDoorStatus(boolean close, lock){
      isClosed = close; isLocked = lock;
    }
}
```

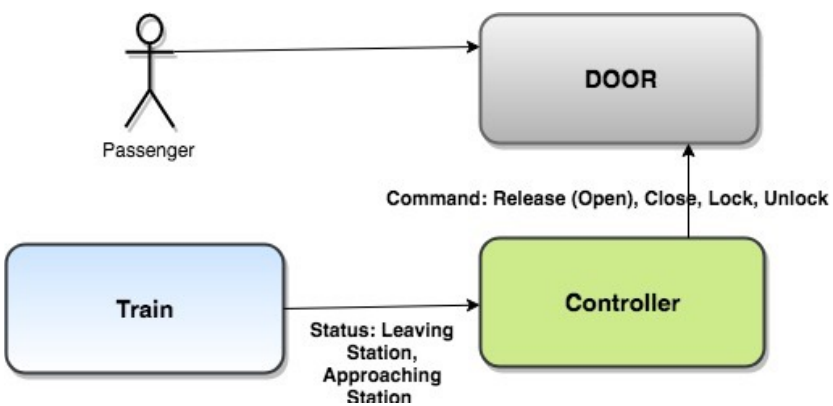



```
msgsrv driveController(){
  if(trainStatus){   // leave the station
    if(!isClosed || !isLocked) {
      if(!isClosed) {
          door.closeDoor() after(nd);
          delay(reactionDelay);
      }
      if(!isLocked) {
          door.lockDoor() after(nd);
      }
    }
  }// end of if(trainStatus)
  else if(!trainStatus){  // arrive the station
    if(isClosed || isLocked) {
      if (isLocked) {
          door.unlockDoor() after(nd);
          delay(reactionDelay);
      }
      if (isClosed) {
          door.openDoor() after(nd);
      } } ...
}
```

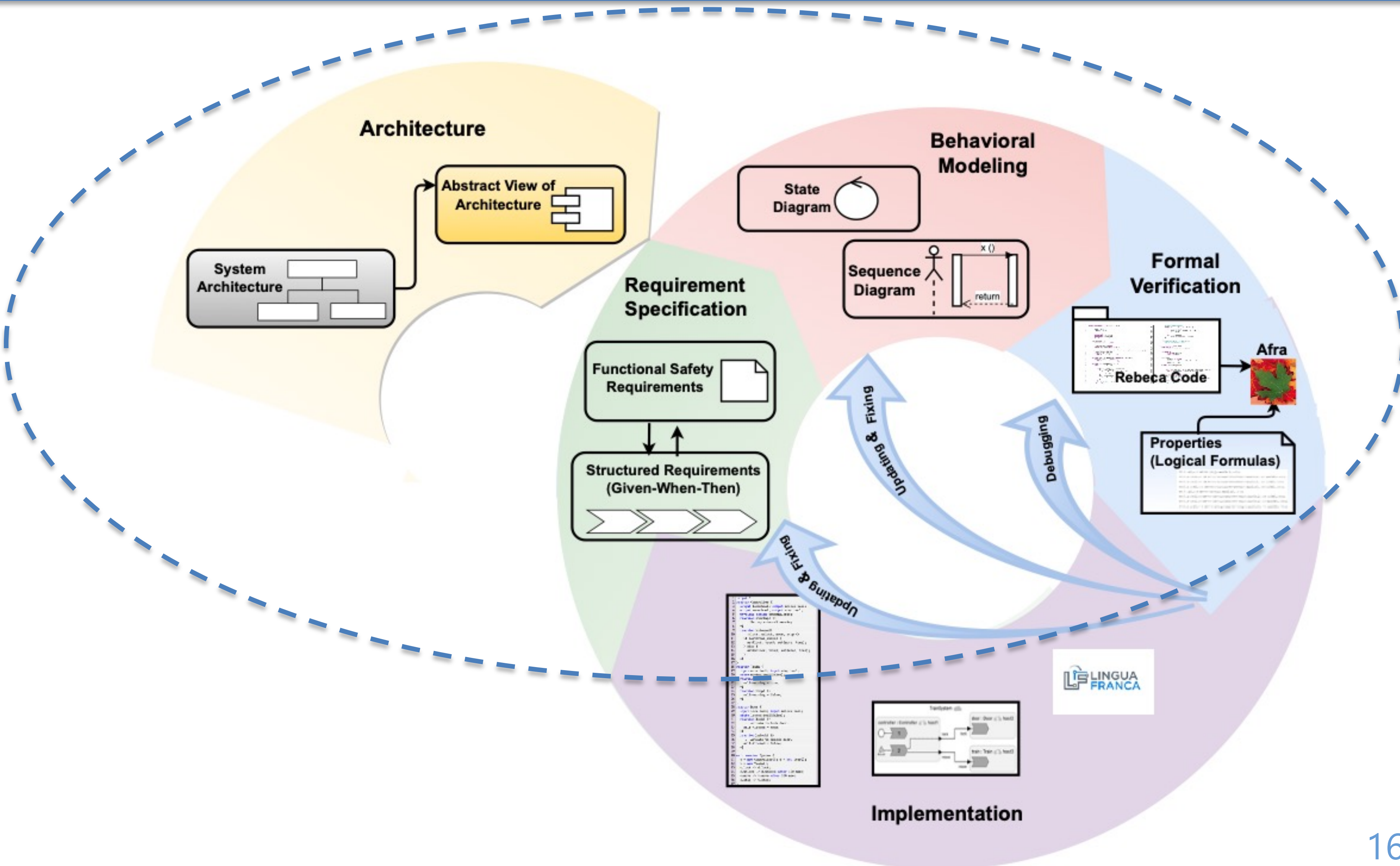
```
reactiveclass passenger(10){
    knownrebecs{
        Door door; }
    Passenger(){
        self.passengerOpenDoor() after(passP);
    }
     msgsrv passengerOpenDoor(){
         door.openDoor();
         self.passengerOpenDoor() after(passP);
     }
}
```




```
main {
      Controller controller(door):();
      Door door(controller):();
      Train train(controller):();
      Passenger passenger(door):();
}
```

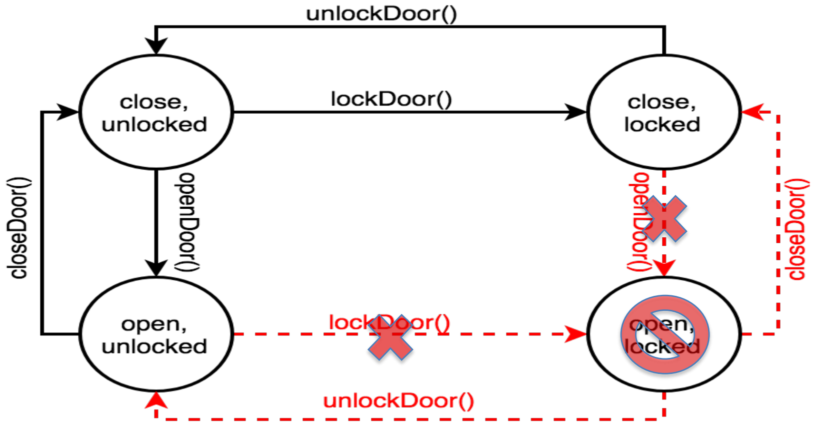


Process: Model Check and Debug



Properties

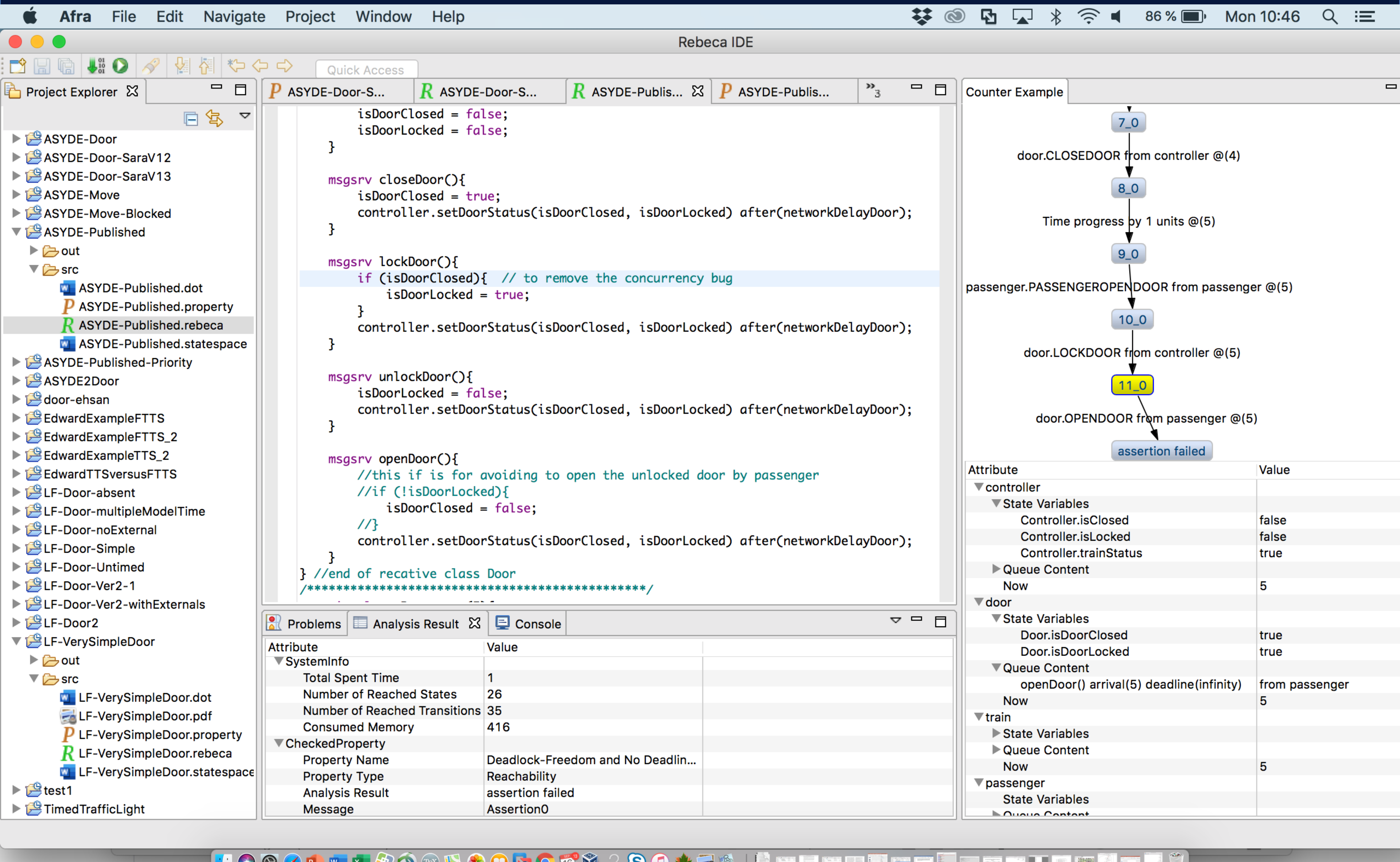
REQ ID	REQ DESCRIPTION	Elicited REQ ID
SSysSpecReq1	GIVEN the train is ready to run WHEN the driver requests to lock the external doors THEN all the external doors in the train shall be closed and locked	SSysReq1



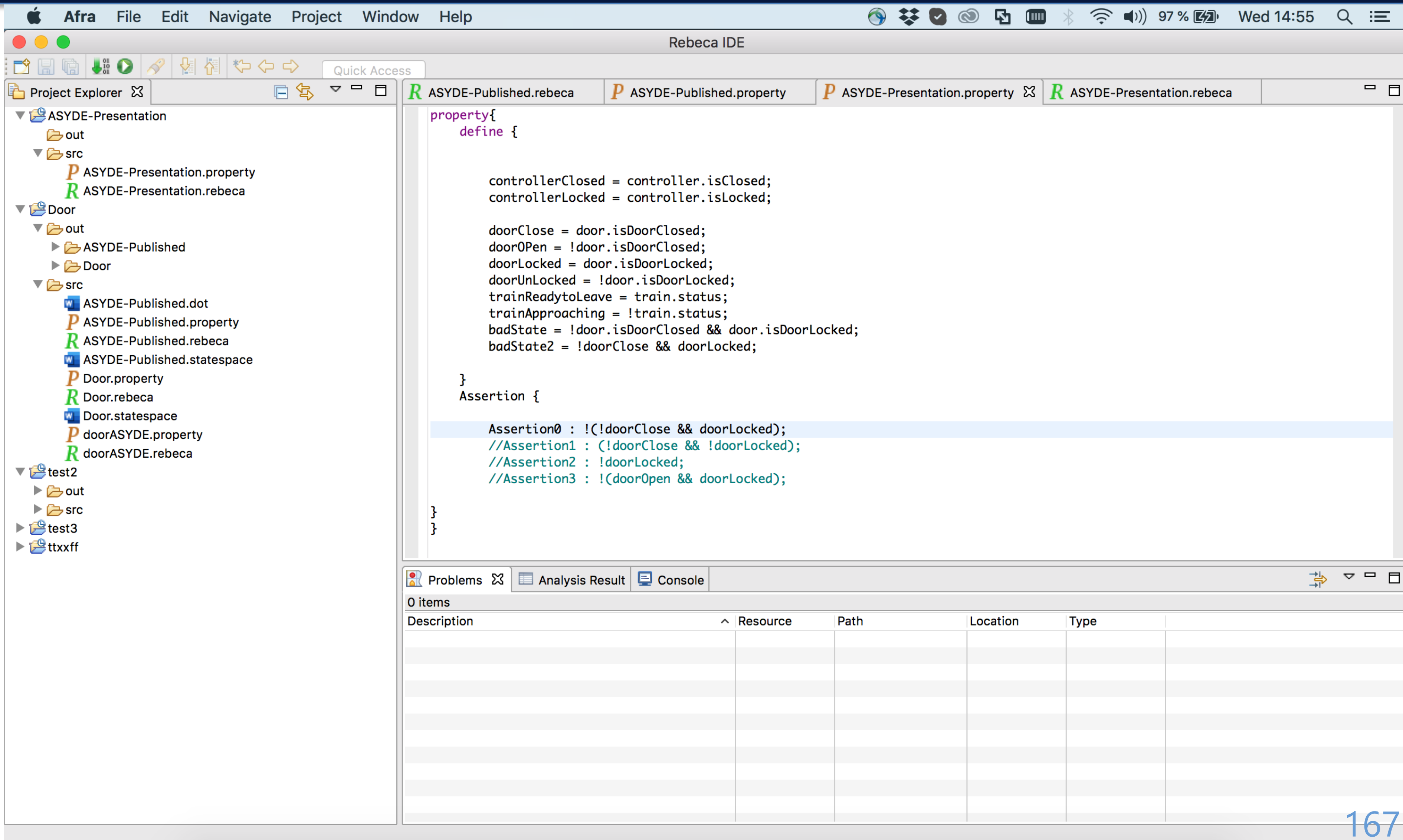
Assertion1:
!doorIsOpen && doorIsLocked

We want to verify that it is not possible to open a locked door or lock an open door.

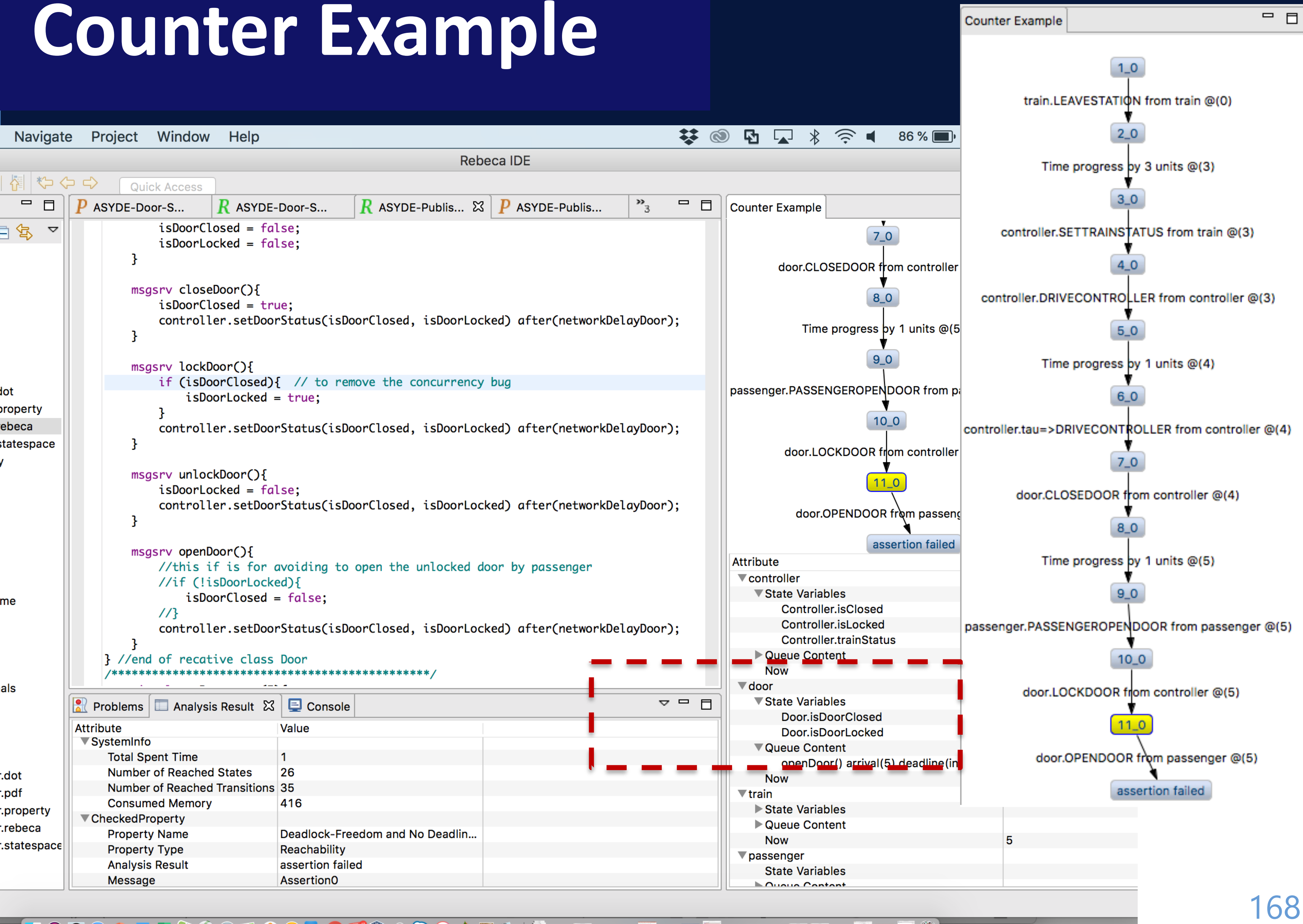
Model Checking Using Afra



Property File



Counter Example



Progress Property - Timing

REQ ID	REQ DESCRIPTION	Elicited REQ ID
SSysSpecReq1	GIVEN the train is ready to run WHEN the driver requests to lock the external doors THEN all the external doors in the train shall be closed and locked	SSysReq1

Property:

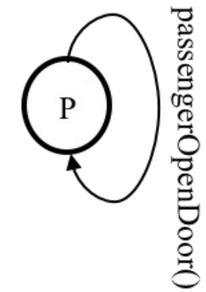
F train.running

Assertion:

!(trainRunning)

Leave at time 0,
Cannot lock the door and move until time 21

```
env byte networkDelayDoor = 1;
env byte networkDelayTrain = 3;
env byte reactionDelay = 5;
env byte passengerPeriod = 5;
env int runningTime = 15;
env byte atStationTime = 10;

reactiveclass passenger(10){
    knownrebecs{
        Door door; }
    Passenger(){
        self.passengerOpenDoor() after(passP);
    }
     msgsrv passengerOpenDoor(){
         door.openDoor();
         self.passengerOpenDoor() after(passP);
      }
}
```



Progress Property - Timing

REQ ID	REQ DESCRIPTION	Elicited REQ ID
SSysSpecReq1	GIVEN the train is ready to run WHEN the driver requests to lock the external doors THEN all the external doors in the train shall be closed and locked	SSysReq1

Property:

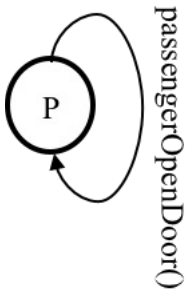
F train.running

Assertion:

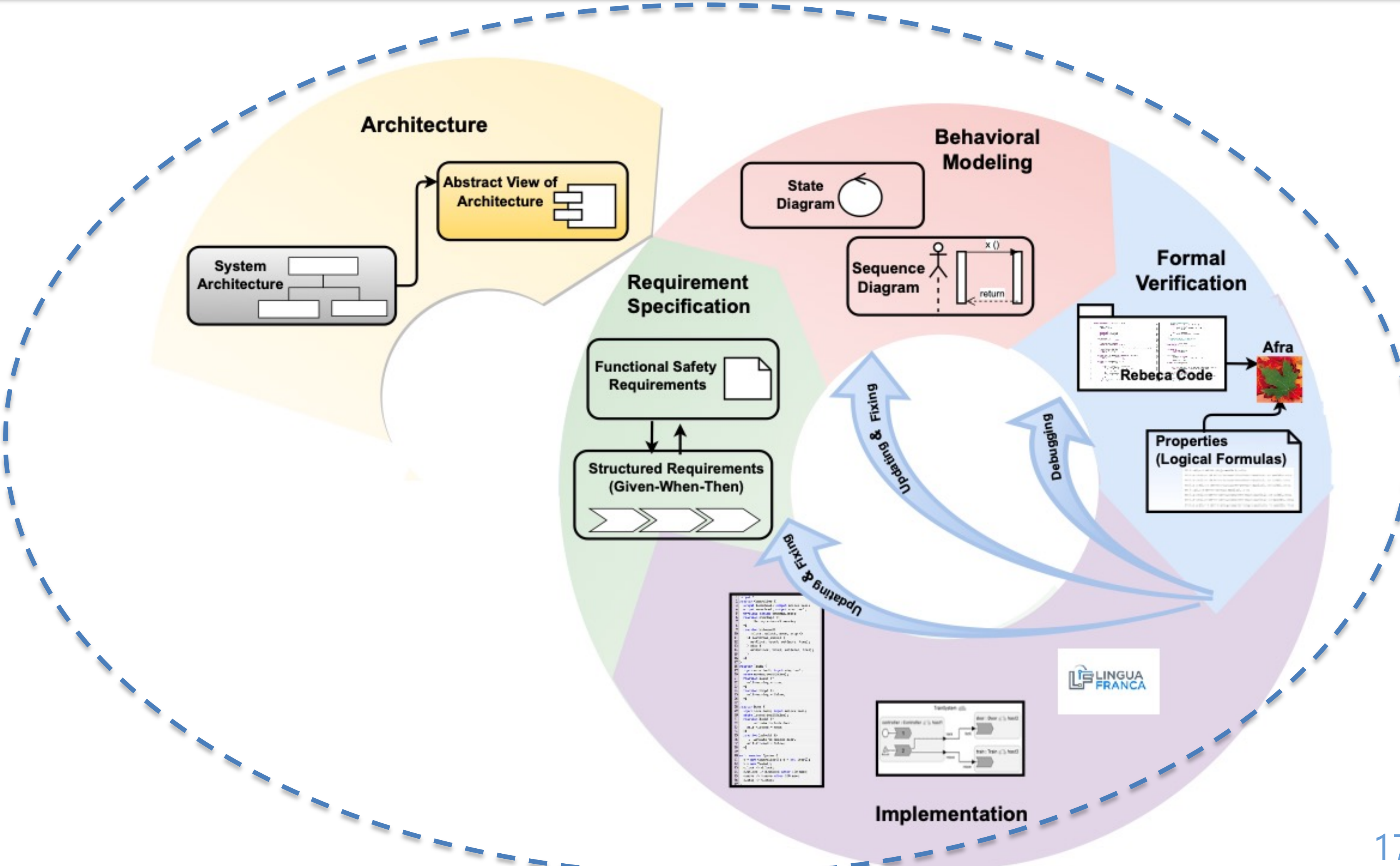
!(trainRunning)

Leave at time 0,
Cannot lock the door and move until time 21

```
env byte networkDelayDoor = 1;
env byte networkDelayTrain = 3;
env byte reactionDelay = 5;
env byte passengerPeriod = 5;
env int runningTime = 15;
env byte atStationTime = 10;

reactiveclass passenger(10){
    knownrebecs{
        Door door; }
    Passenger(){
        self.passengerOpenDoor() after(passP);
    }
     msgsrv passengerOpenDoor(){
         door.openDoor();
         self.passengerOpenDoor() after(passP);
      }
}
```



Process: Proceed to the Implementation



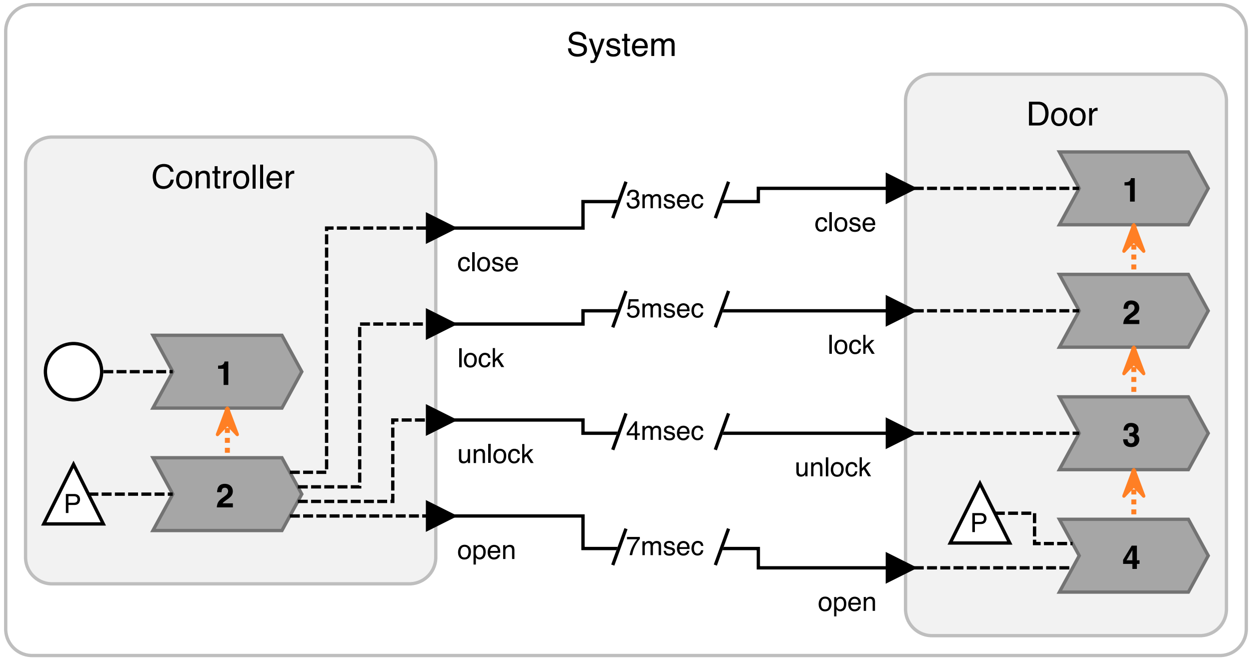
From Requirement to Code: **Lingua Franca**

- **Using Lingua Franca Language**

https://github.com/icyphy/lingua-franca/wiki

Led by Prof. Edward Lee
UC Berkeley

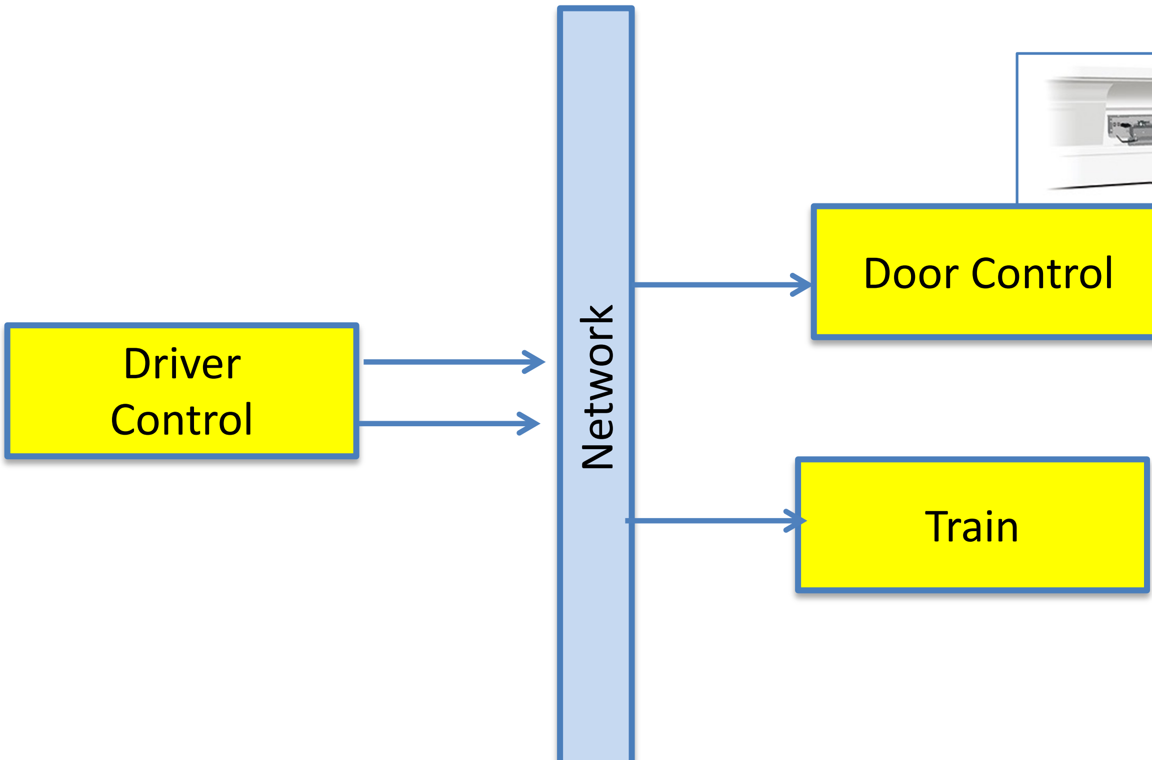
A twin for Rebeca to execute the verified code.



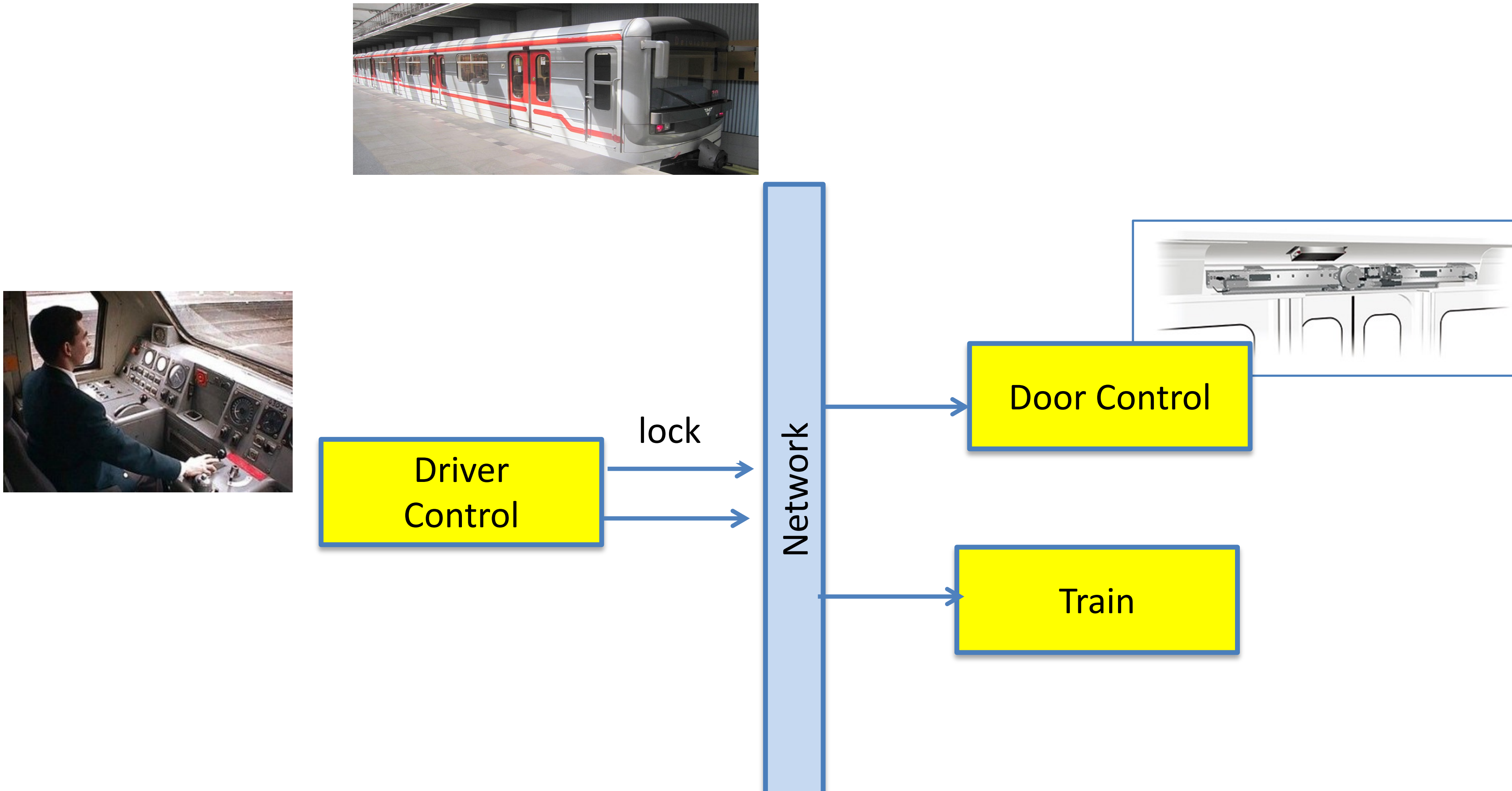
Lohstroh, M., Schoeberl, M., Goens, A., Wasicek, A., Gill, C., Sirjani, M., and Lee, E. A. Actors revisited for time-critical systems. In Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, ACM, pp. 152:1–152:4.

Marjan Sirjani, Edward A. Lee, Ehsan Khamespanah : Model Checking Cyberphysical Systems, Mathematics, 2020

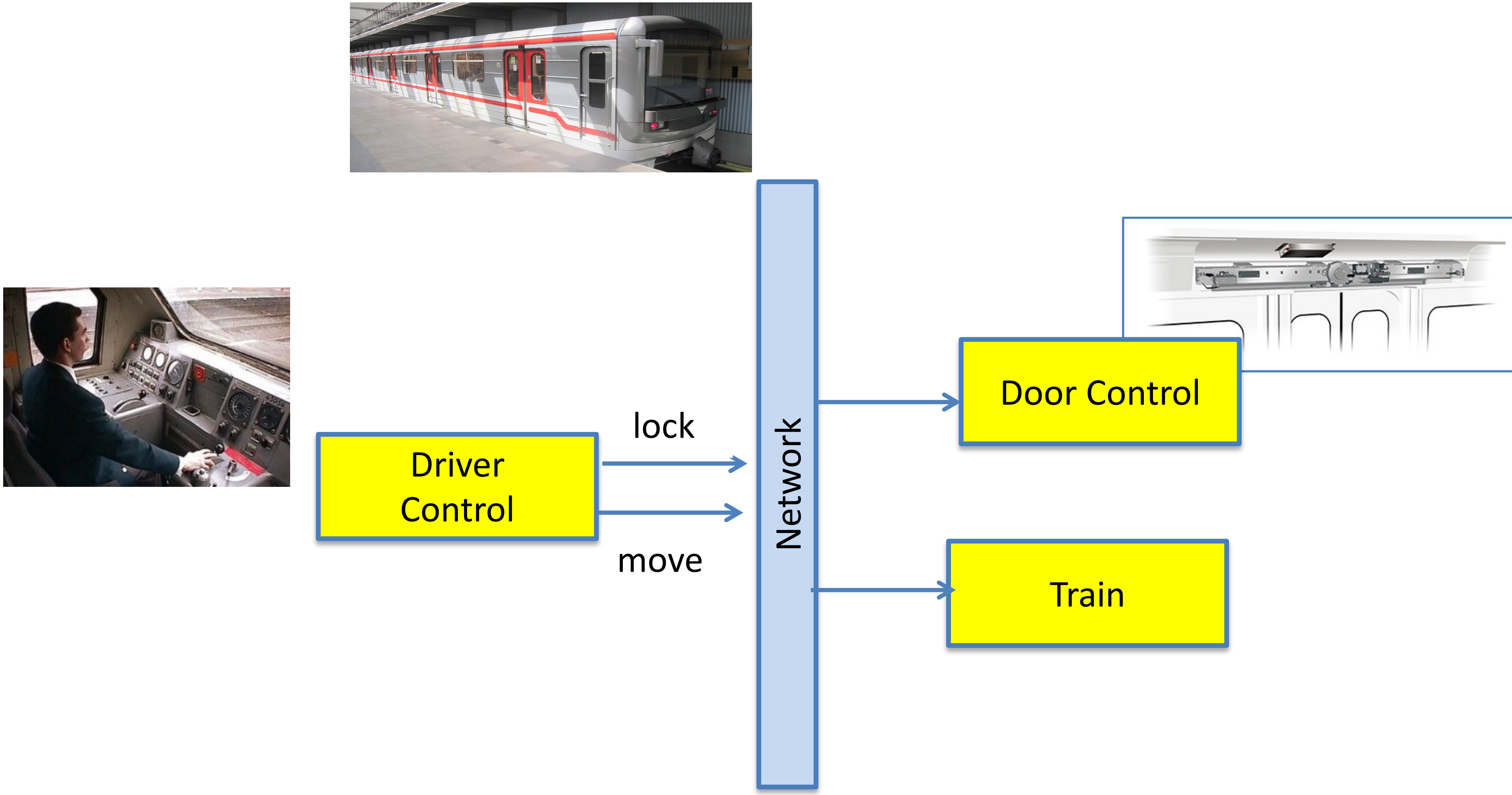
Train-Door Controller



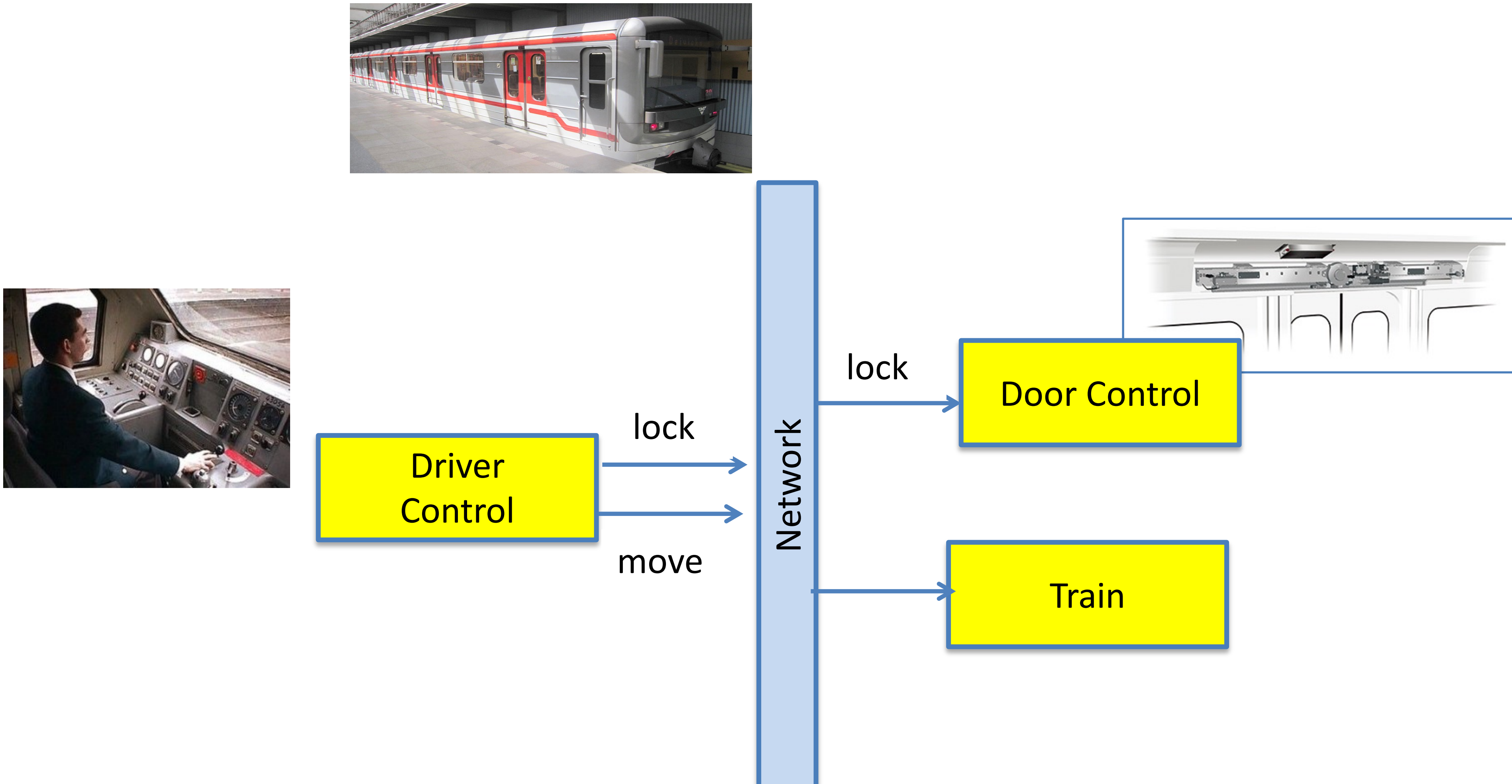
Train-Door Controller



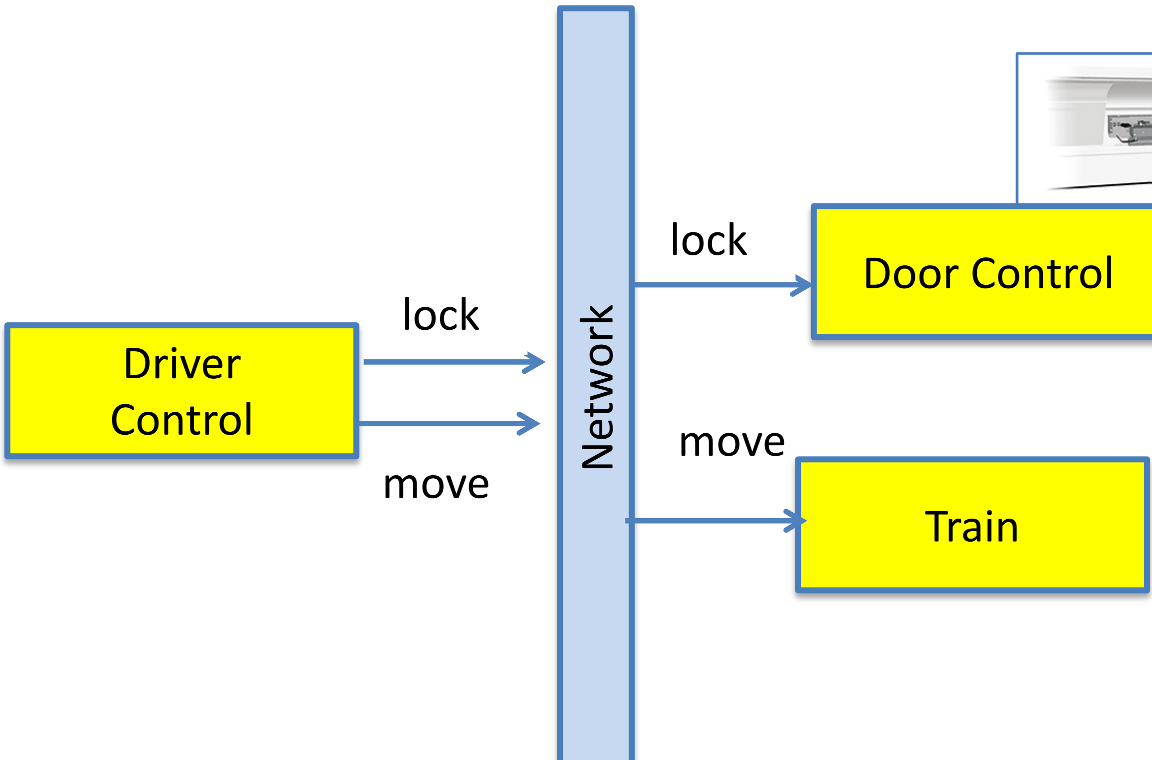
Train-Door Controller



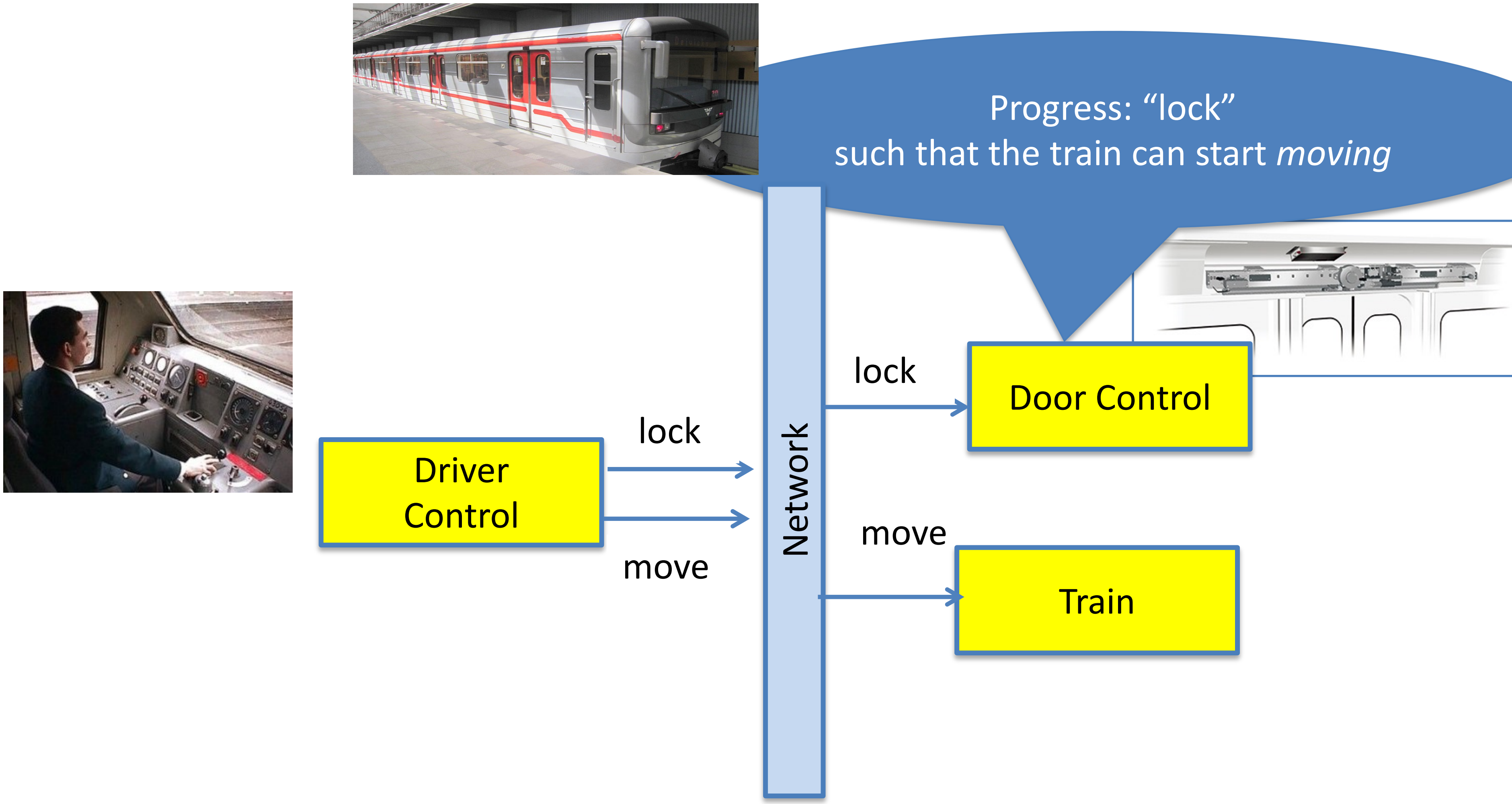
Train-Door Controller



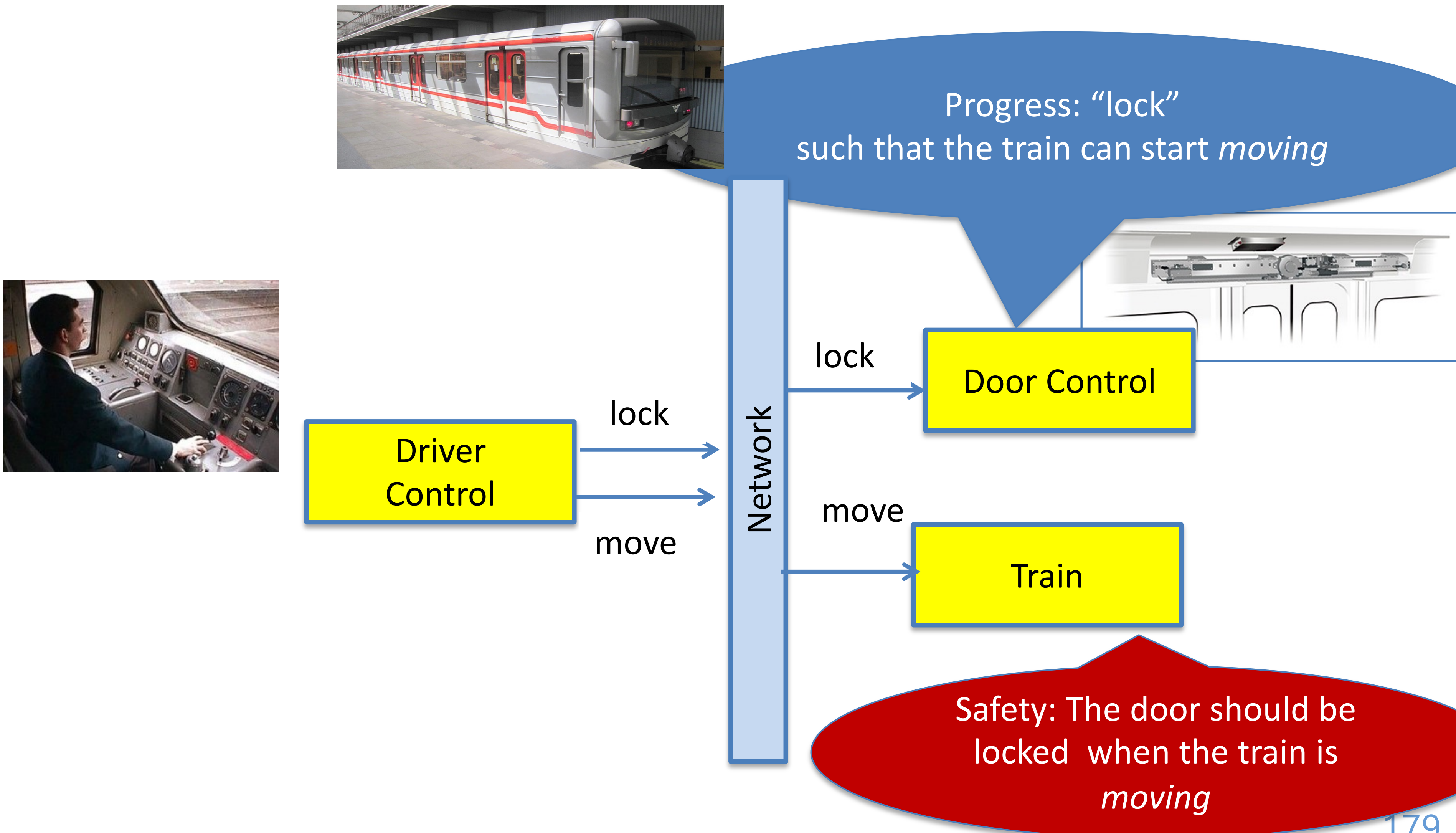
Train-Door Controller



Train-Door Controller



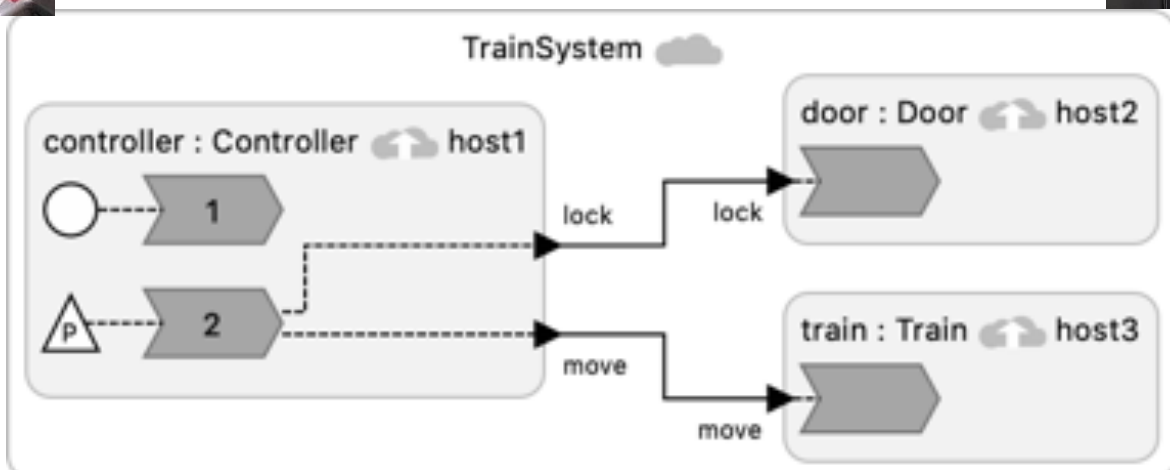
Train-Door Controller



From Requirement to Code: **Lingua Franca**

Led by Prof. Edward Lee
UC Berkeley

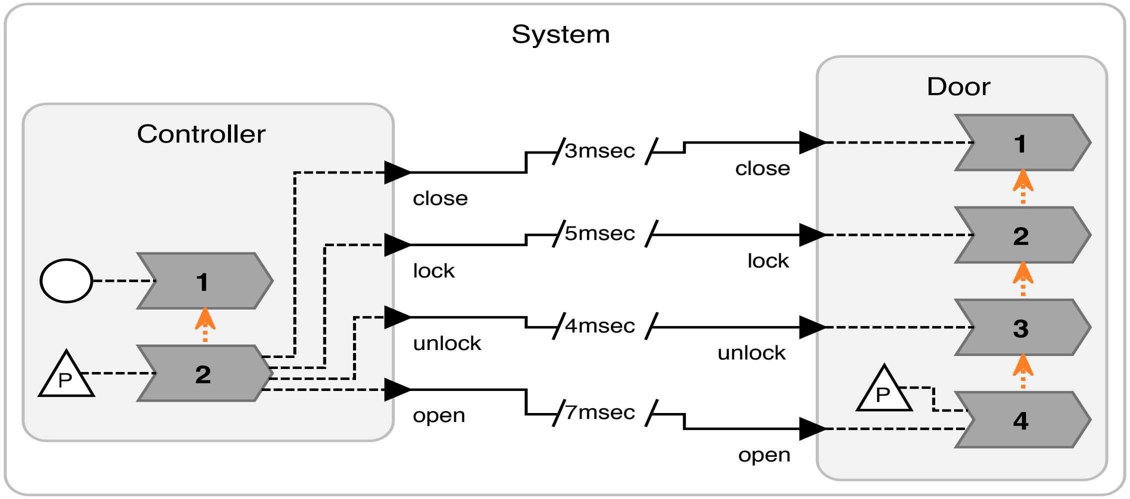
https://github.com/icyphy/lingua-franca/wiki

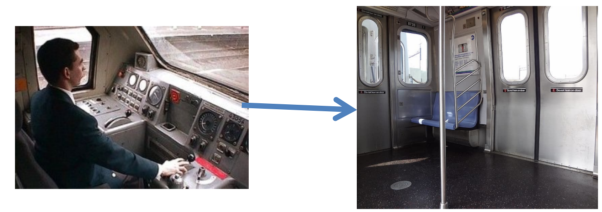


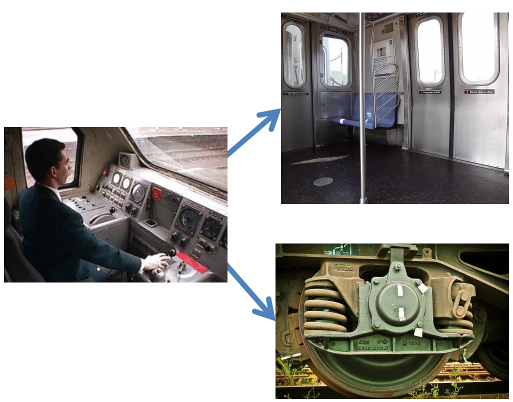
Lohstroh, M., Schoeberl, M., Goens, A., Wasicek, A., Gill, C., Sirjani, M., and Lee, E. A. Actors revisited for time-critical systems. In Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, ACM, pp. 152:1–152:4.

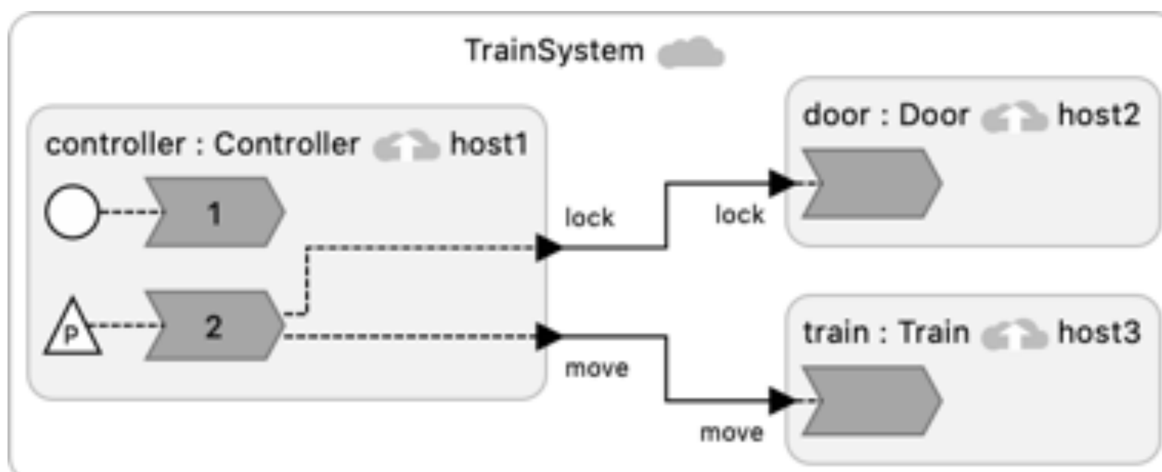
Marjan Sirjani, Edward A. Lee, Ehsan Khamespanah : Model Checking Cyberphysical Systems, Mathematics, 2020

Different Examples: Drivers in an actor
Actors in a network

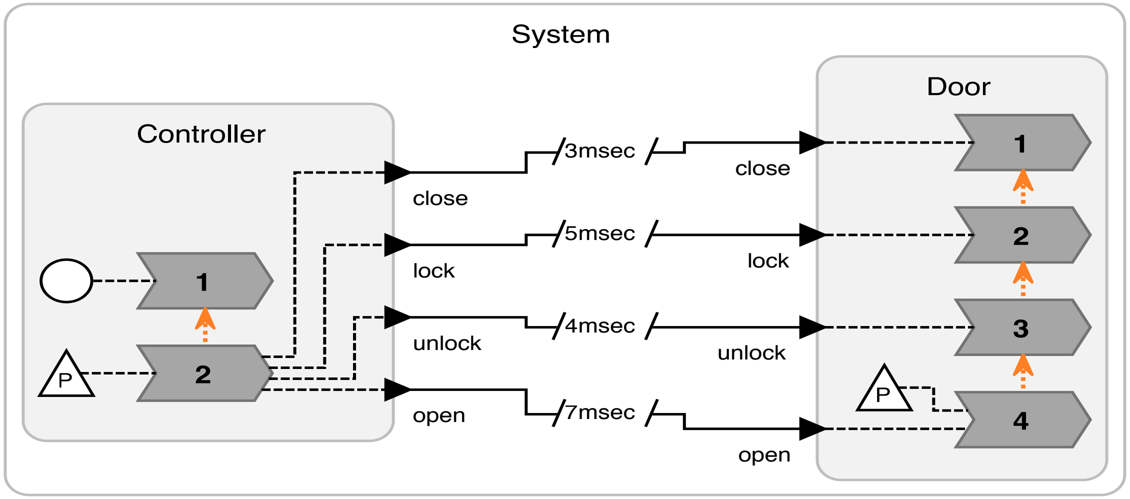




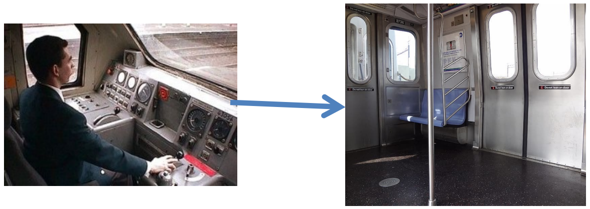


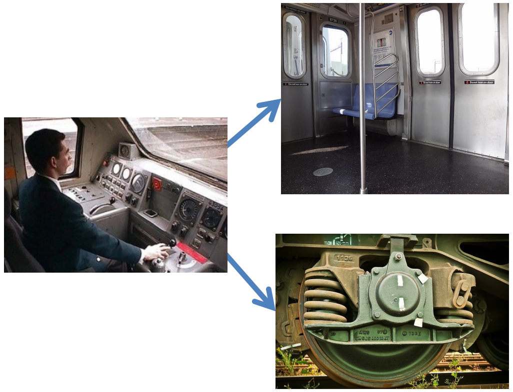


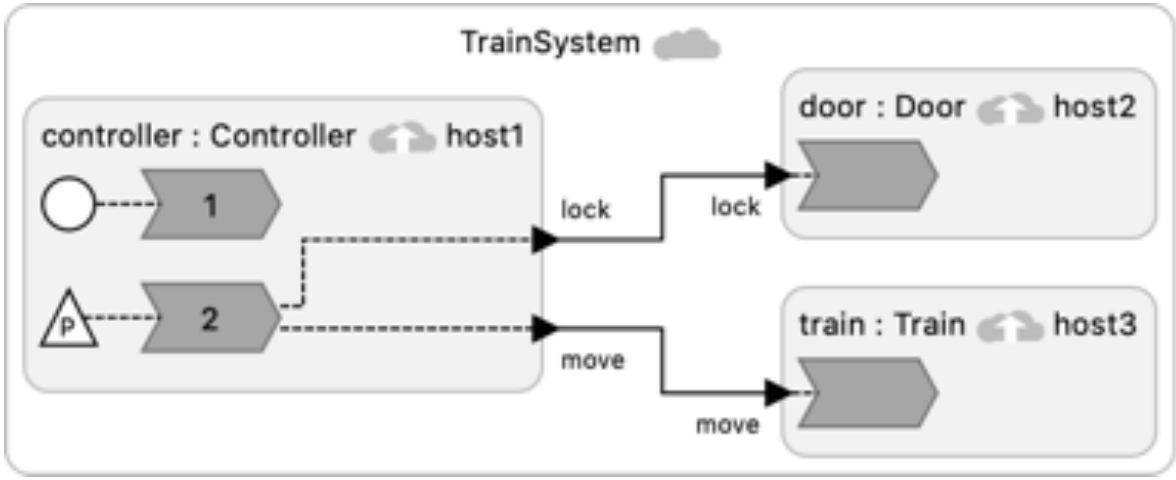
Different Examples: Drivers in an actor
Actors in a network



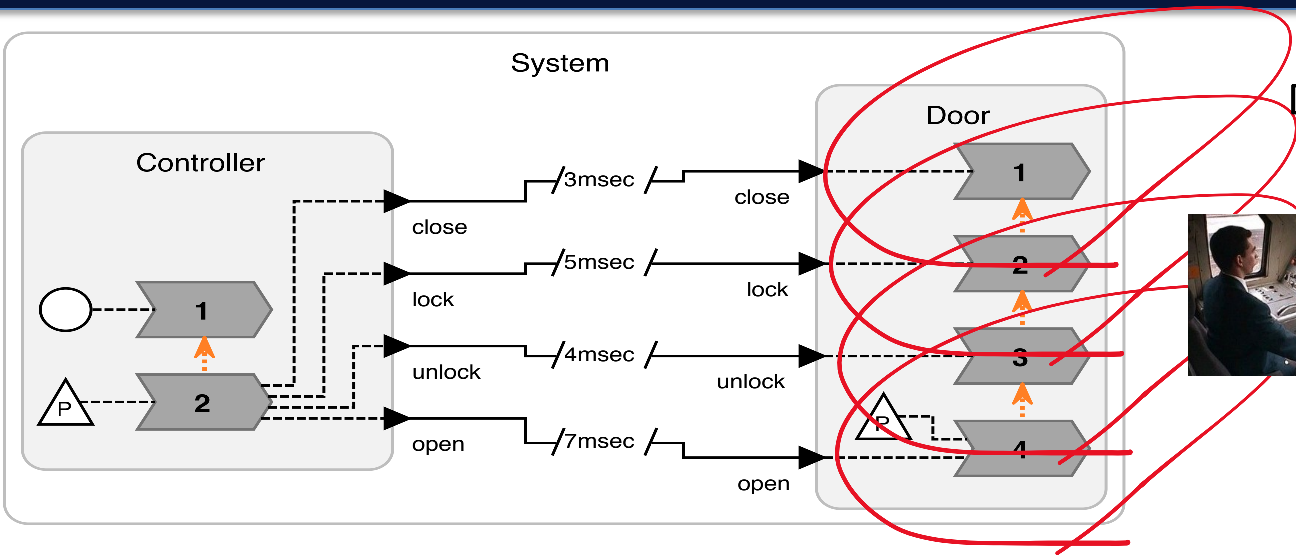
Drivers inside an actor



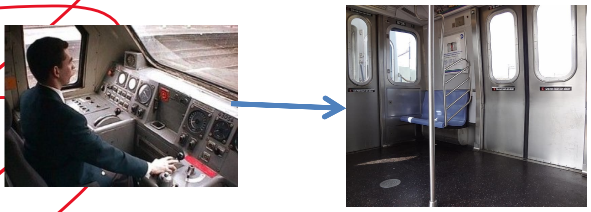


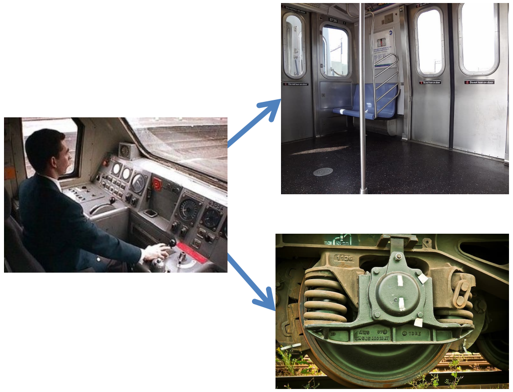


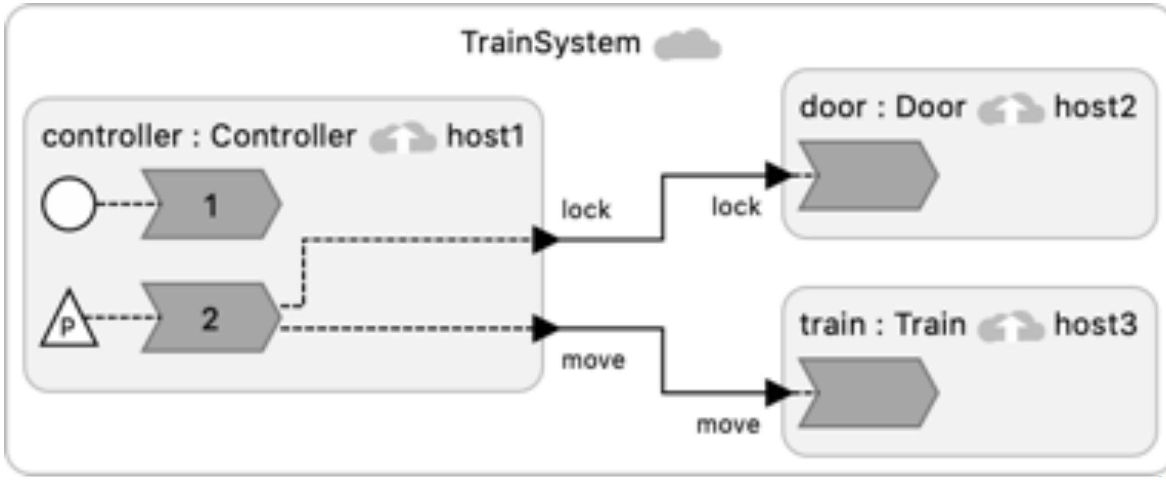
Different Examples: Drivers in an actor
Actors in a network



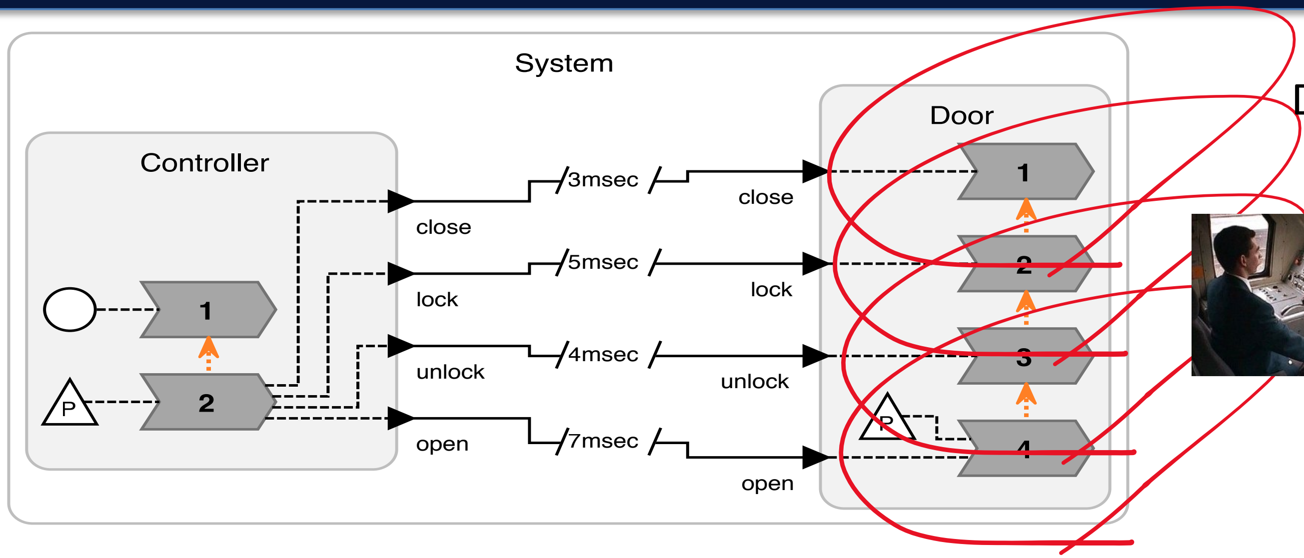
Drivers inside an actor



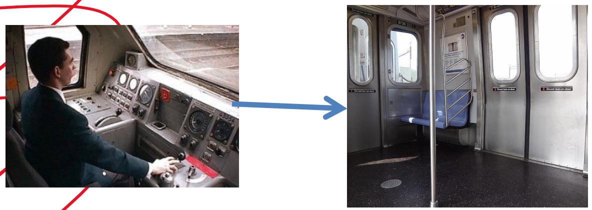




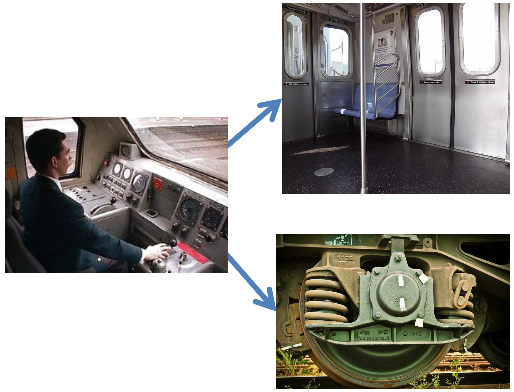
Different Examples: Drivers in an actor
Actors in a network

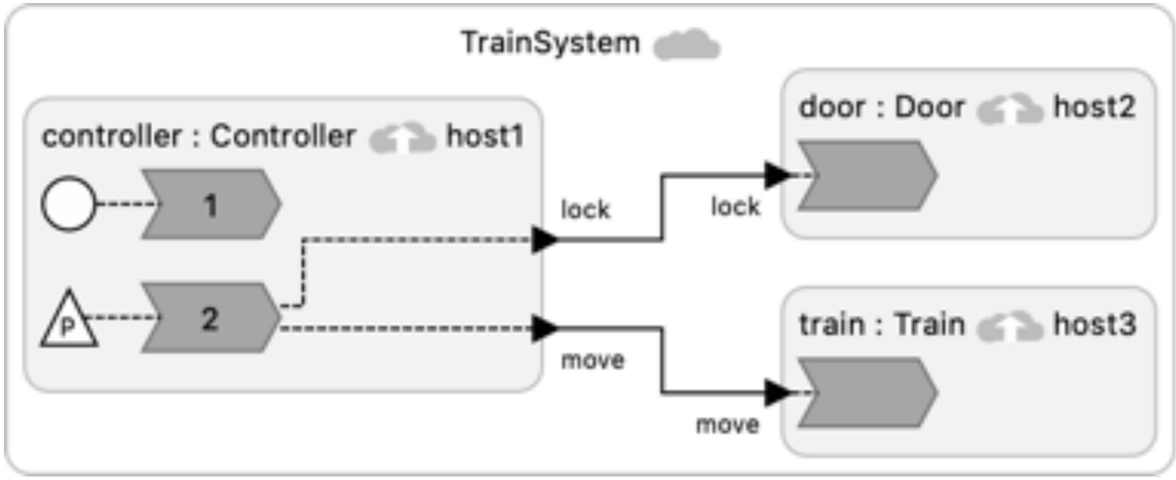


Drivers inside an actor

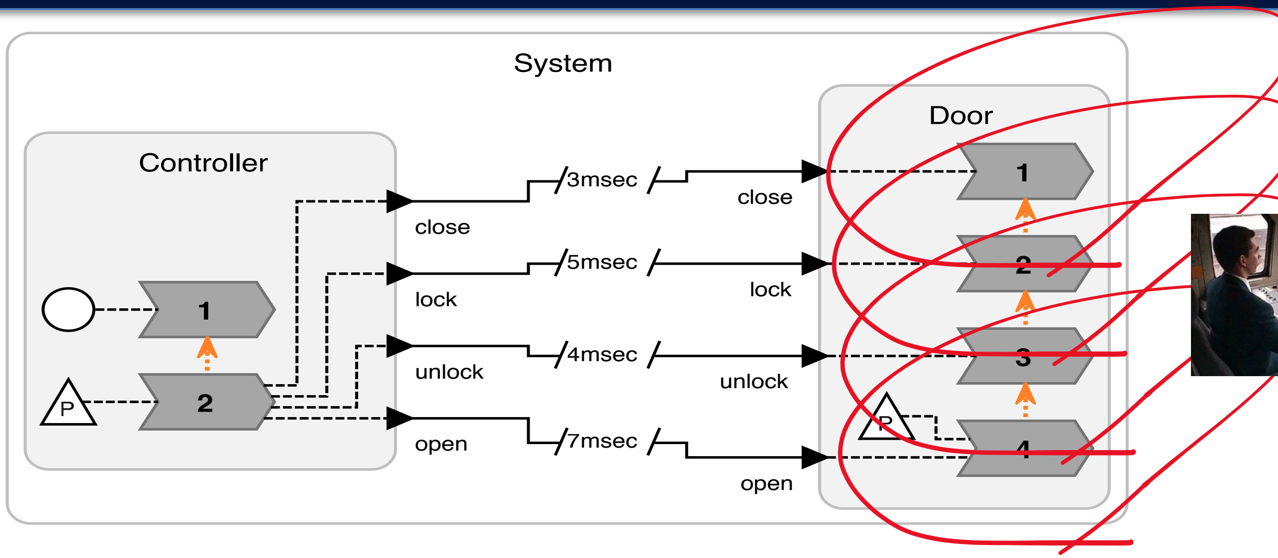


Different Actors in a network



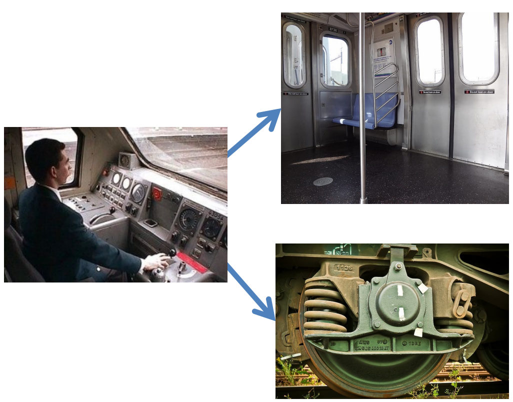


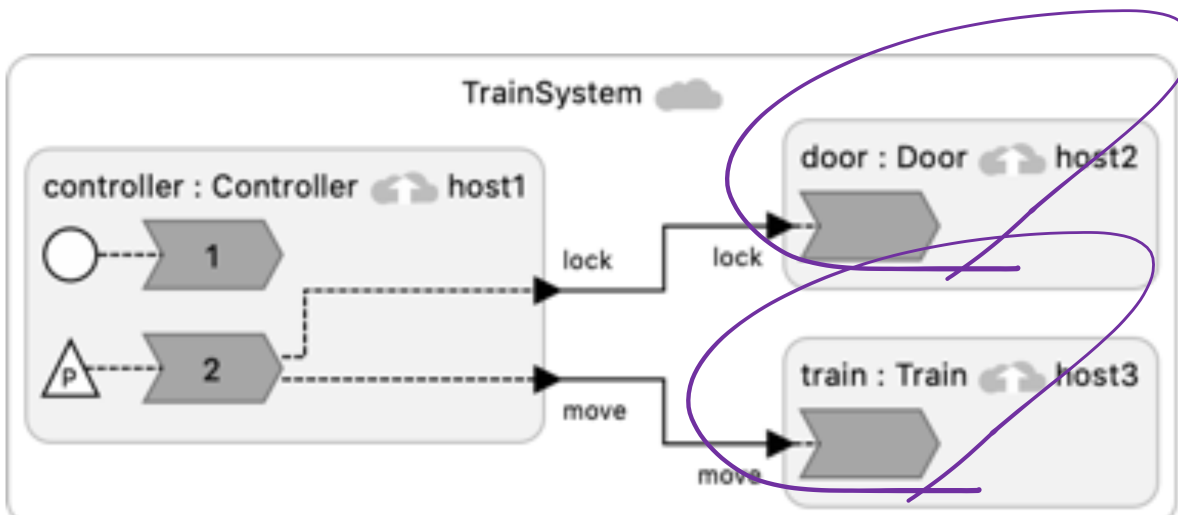
Different Examples: Drivers in an actor
Actors in a network



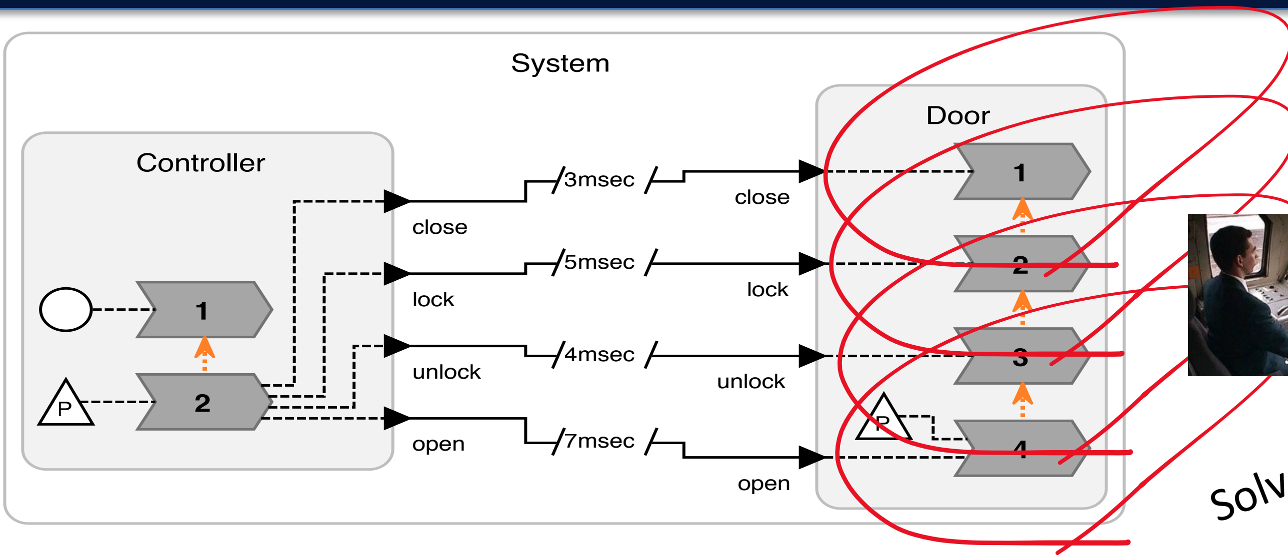
Drivers inside an actor

Different Actors in a network

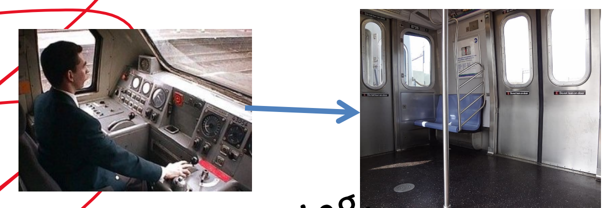




Different Examples: Drivers in an actor
Actors in a network

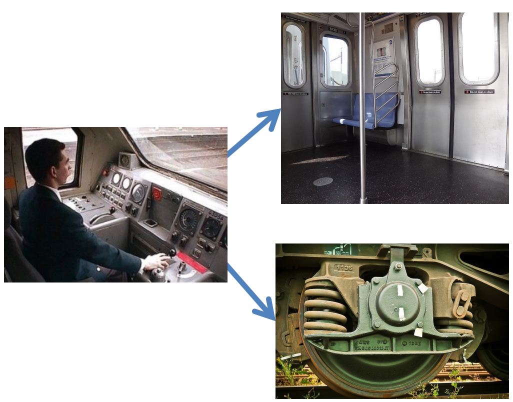


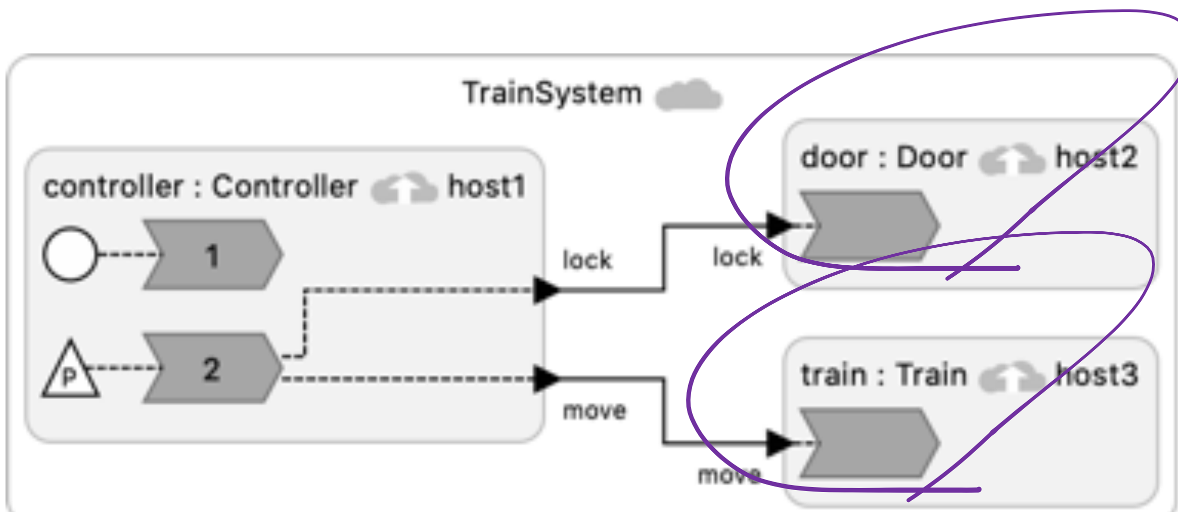
Drivers inside an actor



Solve by ordering.

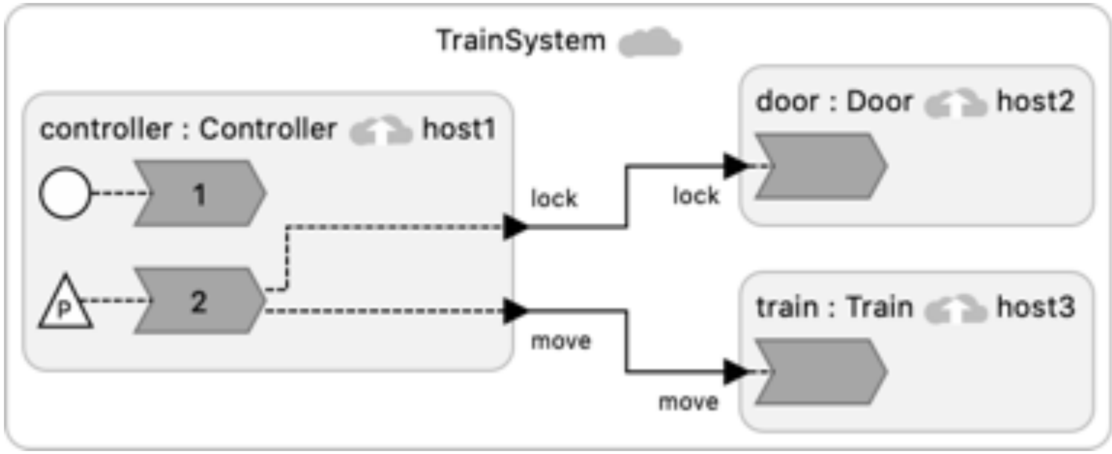
Different Actors in a network





Lingua Franca realization of the train-door example

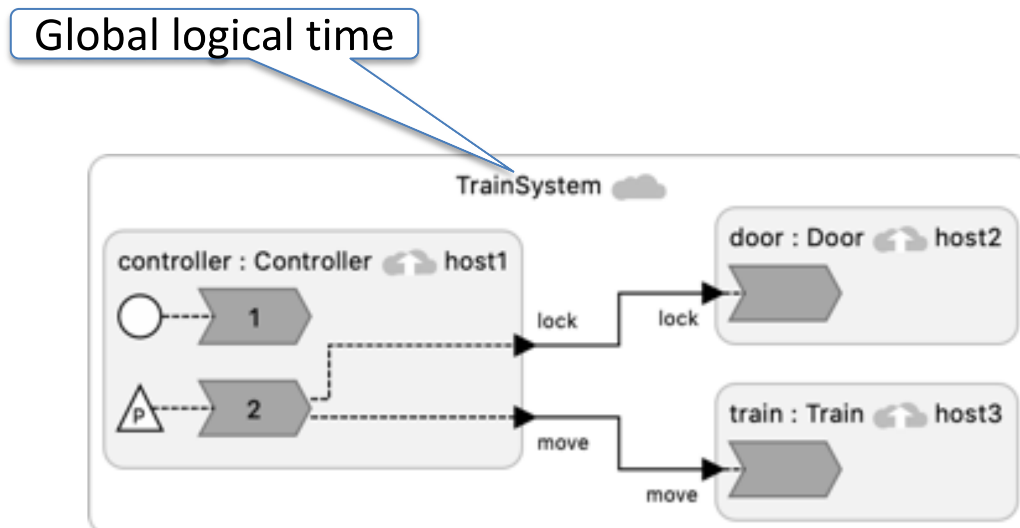
```
target C;
reactor Controller {
  output lock:bool;
  output move:bool;
  physical action external_move:bool;
  reaction(startup) {=
    ... Set up sensing.
  =}
  reaction(external_move)->lock, move {=
    set(lock, external_move_value);
    set(move, external_move_value);
  =}
}
reactor Train {
  input move:bool;
  state moving:bool(false);
  reaction(move) {=
    ... actuate to move or stop
    self->moving = move;
  =}
}
reactor Door {
  input lock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock or unlock door.
    self->locked = lock;
  =}
}
federated reactor TrainSystem {
  controller = new Controller() at host1;
  door = new Door() at host2;
  train = new Train() at host3;
  controller.lock -> door.lock;
  controller.move -> train.move;
}
```



[Sirjani, Lee, Khamespanah, "Verification of Cyberphysical Systems," *Mathematics*, July 2, 2020]

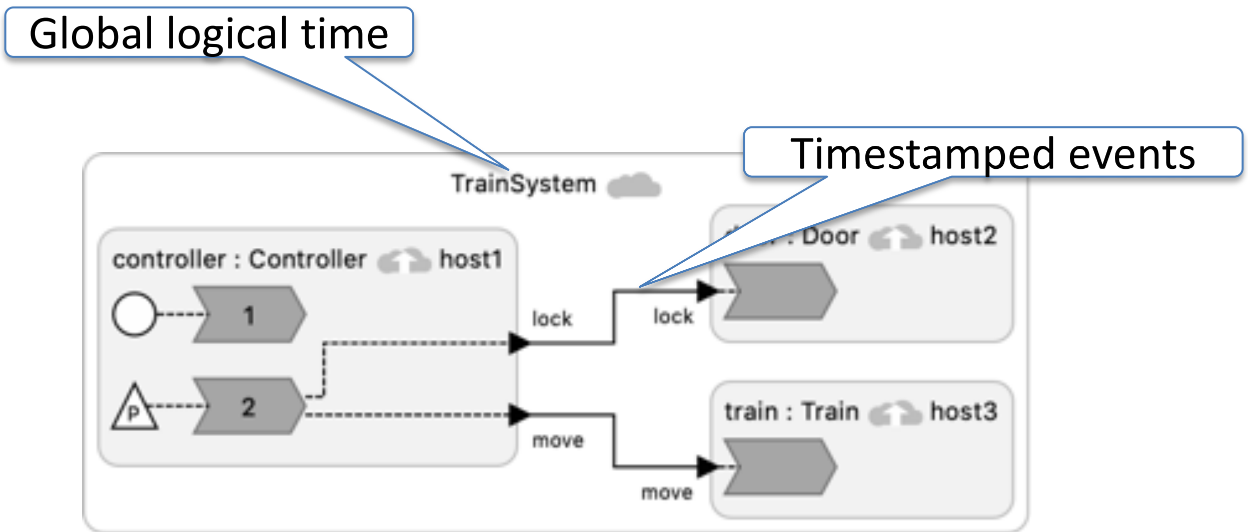
Lingua Franca

```
target C;
reactor Controller {
  output lock:bool;
  output move:bool;
  physical action external_move:bool;
  reaction(startup) {=
    ... Set up sensing.
  =}
  reaction(external_move)->lock, move {=
    set(lock, external_move_value);
    set(move, external_move_value);
  =}
}
reactor Train {
  input move:bool;
  state moving:bool(false);
  reaction(move) {=
    ... actuate to move or stop
    self->moving = move;
  =}
}
reactor Door {
  input lock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock or unlock door.
    self->locked = lock;
  =}
}
federated reactor TrainSystem {
  controller = new Controller() at host1;
  door = new Door() at host2;
  train = new Train() at host3;
  controller.lock -> door.lock;
  controller.move -> train.move;
}
```



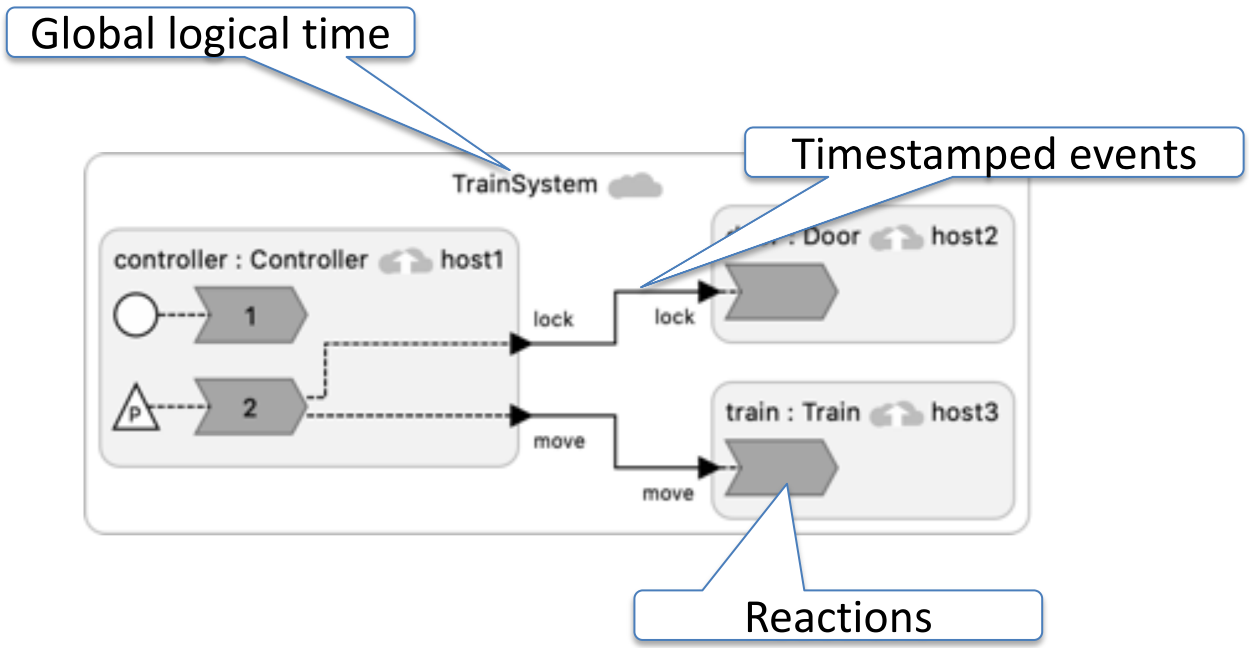
Lingua Franca

```
target C;
reactor Controller {
  output lock:bool;
  output move:bool;
  physical action external_move:bool;
  reaction(startup) {=
    ... Set up sensing.
  =}
  reaction(external_move)->lock, move {=
    set(lock, external_move_value);
    set(move, external_move_value);
  =}
}
reactor Train {
  input move:bool;
  state moving:bool(false);
  reaction(move) {=
    ... actuate to move or stop
    self->moving = move;
  =}
}
reactor Door {
  input lock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock or unlock door.
    self->locked = lock;
  =}
}
federated reactor TrainSystem {
  controller = new Controller() at host1;
  door = new Door() at host2;
  train = new Train() at host3;
  controller.lock -> door.lock;
  controller.move -> train.move;
}
```



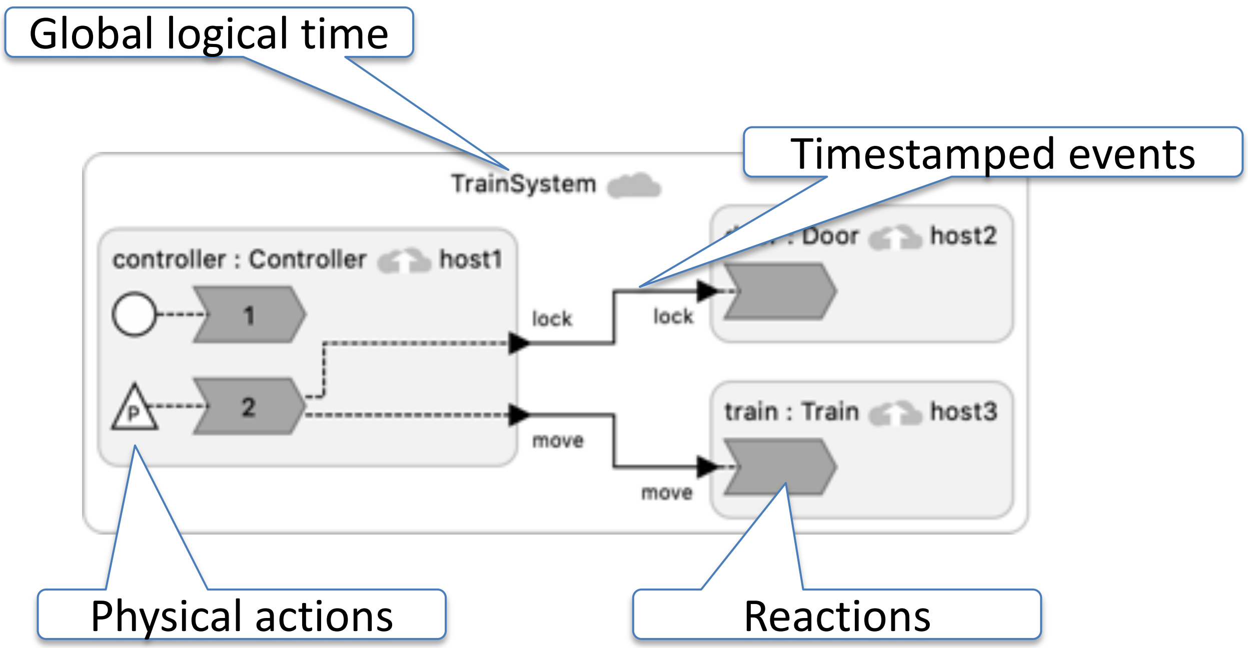
Lingua Franca

```
target C;
reactor Controller {
  output lock:bool;
  output move:bool;
  physical action external_move:bool;
  reaction(startup) {=
    ... Set up sensing.
  =}
  reaction(external_move)->lock, move {=
    set(lock, external_move_value);
    set(move, external_move_value);
  =}
}
reactor Train {
  input move:bool;
  state moving:bool(false);
  reaction(move) {=
    ... actuate to move or stop
    self->moving = move;
  =}
}
reactor Door {
  input lock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock or unlock door.
    self->locked = lock;
  =}
}
federated reactor TrainSystem {
  controller = new Controller() at host1;
  door = new Door() at host2;
  train = new Train() at host3;
  controller.lock -> door.lock;
  controller.move -> train.move;
}
```



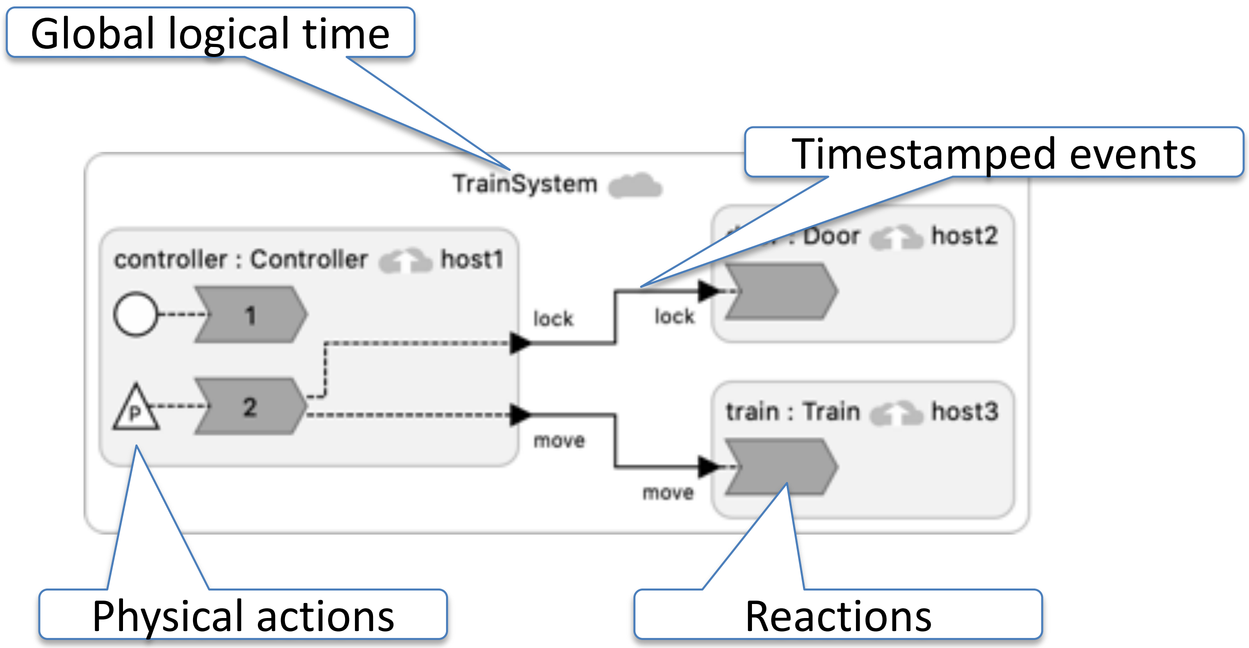
Lingua Franca

```
target C;
reactor Controller {
  output lock:bool;
  output move:bool;
  physical action external_move:bool;
  reaction(startup) {=
    ... Set up sensing.
  =}
  reaction(external_move)->lock, move {=
    set(lock, external_move_value);
    set(move, external_move_value);
  =}
}
reactor Train {
  input move:bool;
  state moving:bool(false);
  reaction(move) {=
    ... actuate to move or stop
    self->moving = move;
  =}
}
reactor Door {
  input lock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock or unlock door.
    self->locked = lock;
  =}
}
federated reactor TrainSystem {
  controller = new Controller() at host1;
  door = new Door() at host2;
  train = new Train() at host3;
  controller.lock -> door.lock;
  controller.move -> train.move;
}
```



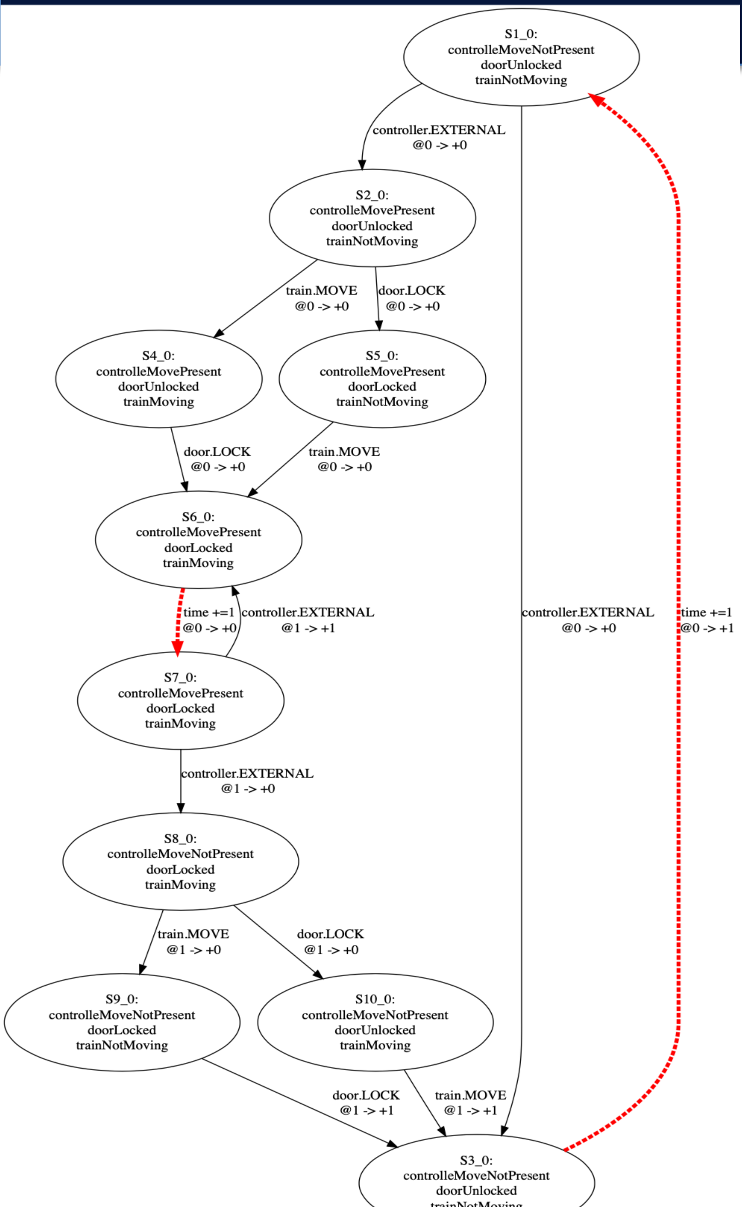
Lingua Franca

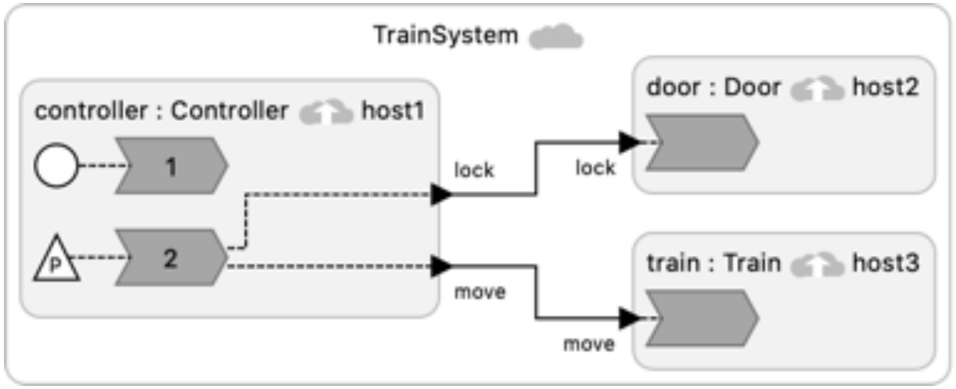
```
target C;
reactor Controller {
  output lock:bool;
  output move:bool;
  physical action external_move:bool;
  reaction(startup) {=
    ... Set up sensing.
  =}
  reaction(external_move)->lock, move {=
    set(lock, external_move_value);
    set(move, external_move_value);
  =}
}
reactor Train {
  input move:bool;
  state moving:bool(false);
  reaction(move) {=
    ... actuate to move or stop
    self->moving = move;
  =}
}
reactor Door {
  input lock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock or unlock door.
    self->locked = lock;
  =}
}
federated reactor TrainSystem {
  controller = new Controller() at host1;
  door = new Door() at host2;
  train = new Train() at host3;
  controller.lock -> door.lock;
  controller.move -> train.move;
}
```



- React to events in timestamp order.

The State Space

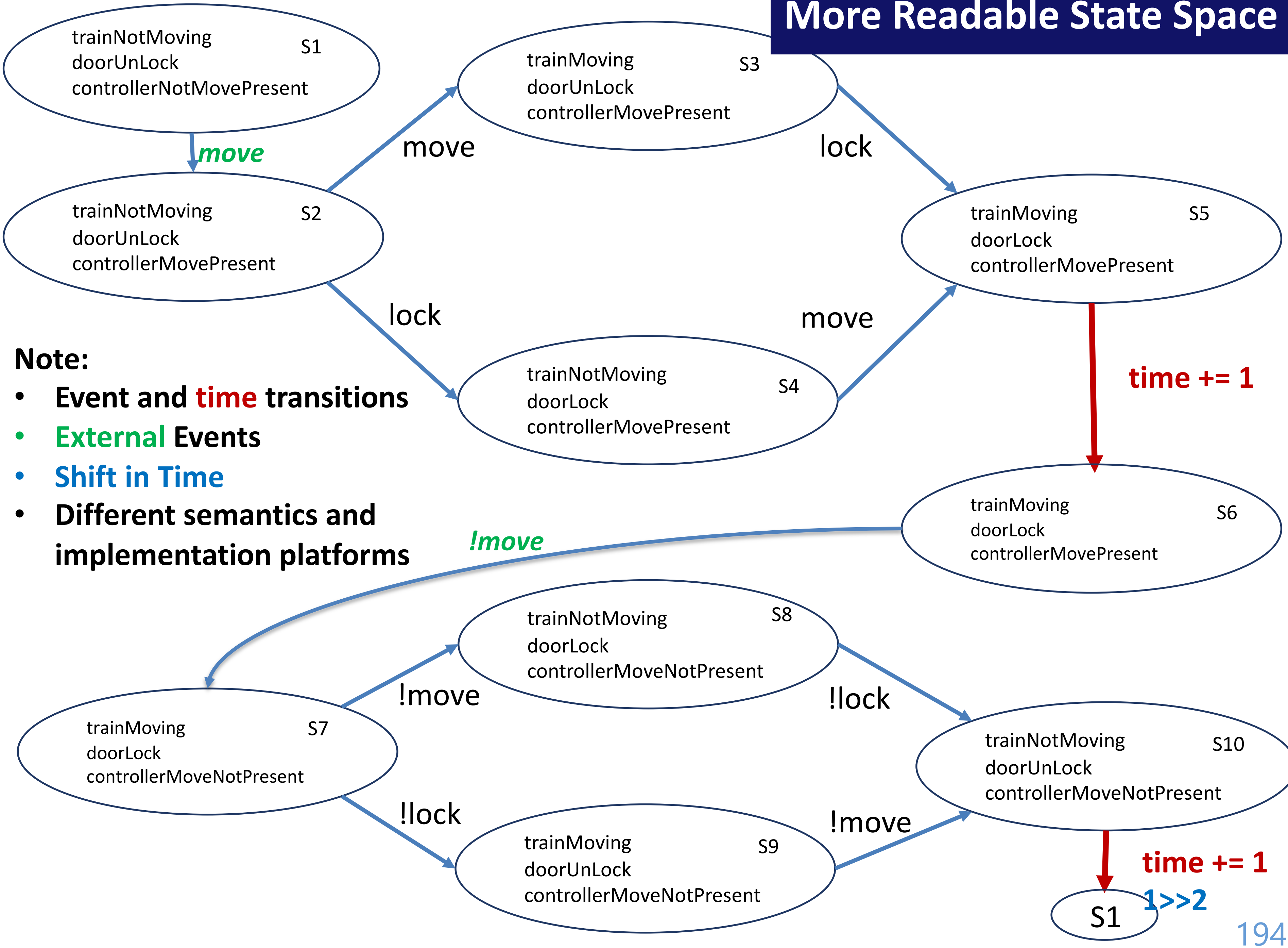




G ¬ (doorUnlocked ∧ trainMoving) ?

Model checking using Rebeca
Implementation using Lingua Franca

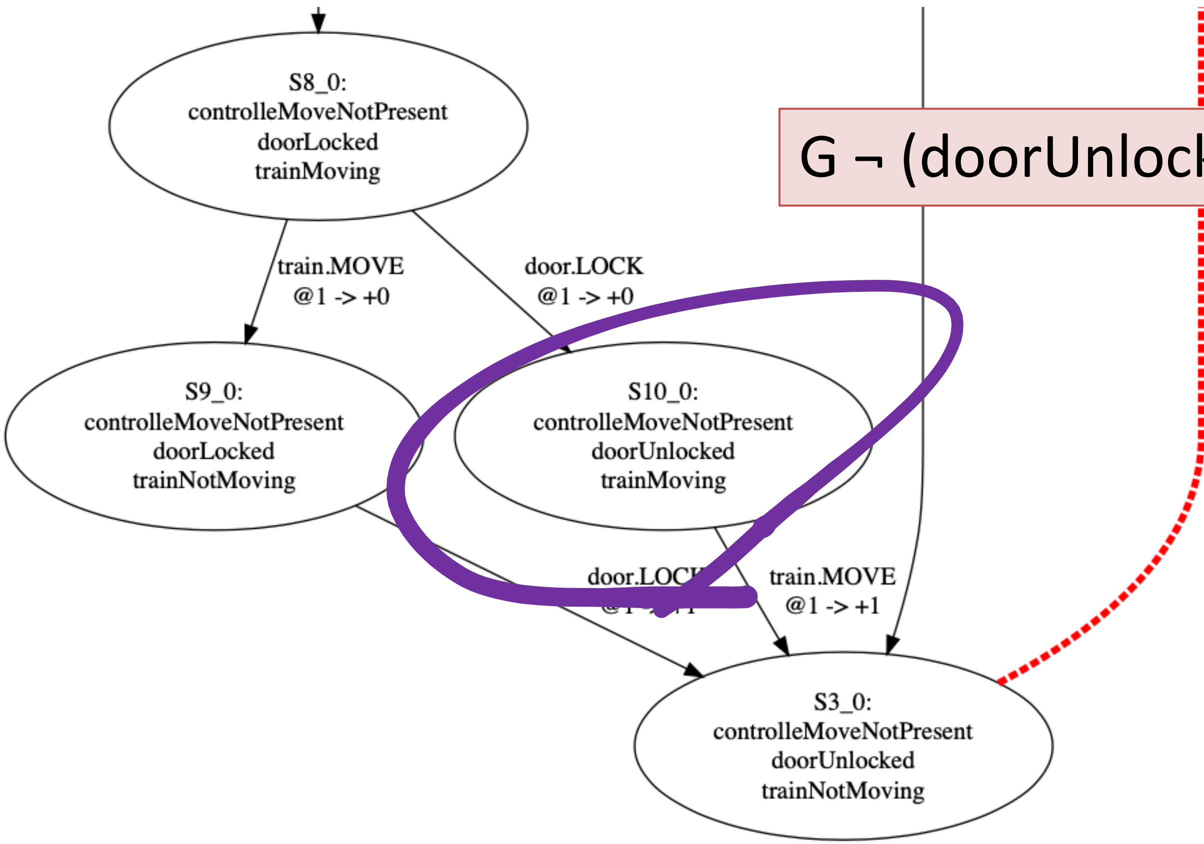
More Readable State Space



Note:

- **Event and time transitions**
- **External Events**
- **Shift in Time**
- **Different semantics and implementation platforms**

Counterexample!



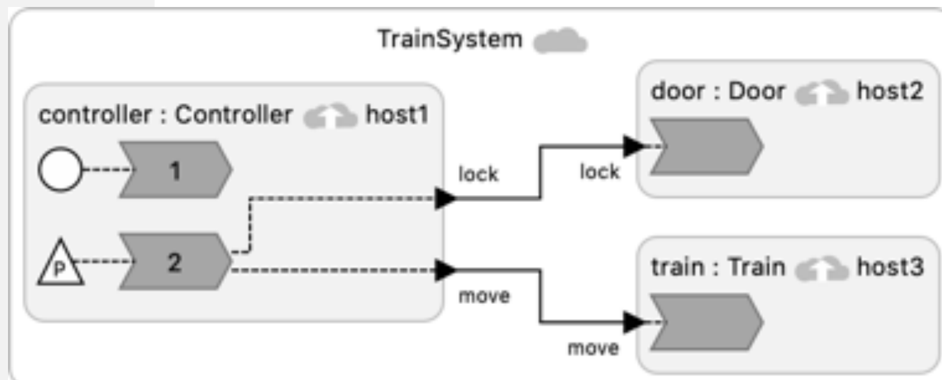
G ¬ (doorUnlocked ∧ trainMoving) ?

Transition diagram using Timed Rebeca and Afra

From Timed Rebeca to Lingua Franca

```
reactiveclass Controller(5) {
knownrebecs {
Door door;
Train train;
}
statevars { boolean moveP; }
Controller() {
self.external();
}
msgsrv external() {
boolean oldMoveP = moveP;
moveP = ?(true,false);
if(moveP != oldMoveP) {
door.lock(moveP);
train.move(moveP);
}
self.external() after(1);
}
}
reactiveclass Train(5) {
statevars { boolean moving; }
Train() {
moving = false;
}
msgsrv move(boolean tmove) {
if (tmove) {
moving = true;
} else {
moving = false;
}
}
}
reactiveclass Door(5) {
statevars { boolean is_locked; }
Door() {
is_locked = false;
}
msgsrv lock (boolean lockPar) {
is_locked = lockPar;
}
}
main {
@priority(1) Controller controller(do
train):();
@priority(2) Train train():();
@priority(2) Door door():();
}
```

```
target C;
reactor Controller {
output lock:bool;
output move:bool;
physical action external:bool;
reaction(startup) {=
... Set up sensing.
=}
reaction(external)->lock, move {=
set(lock, external_value);
set(move, external_value);
=}
}
reactor Train {
input move:bool;
state moving:bool(false);
reaction(move) {=
... actuate to move or stop
self->moving = move;
=}
}
reactor Door {
input lock:bool;
state locked:bool(false);
reaction(lock) {=
... Actuate to lock or unlock door.
self->locked = lock;
=}
}
main reactor System {
controller = new Controller();
door = new Door();
train = new Train();
controller.lock -> door.lock;
controller.move -> train.move;
}
```



Verification of Cyberphysical Systems, Marjan Sirjani, Edward A. Lee and Ehsan Khamespanah, Mathematics journal, Mathematics, July 2020.

From Timed Rebeca to Lingua Franca

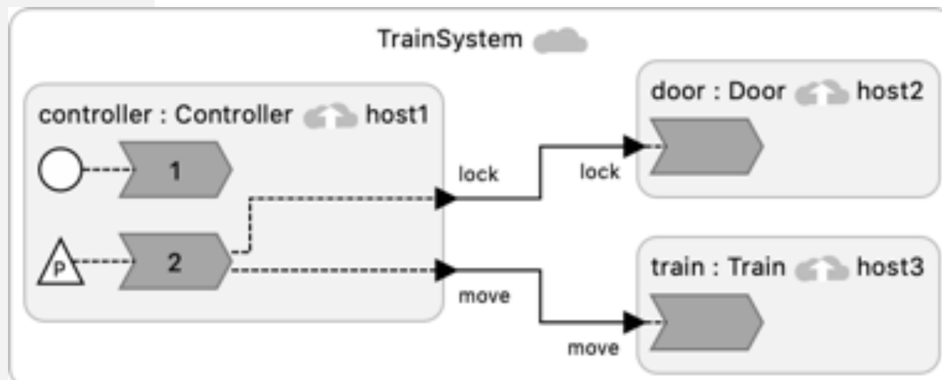
Afra 3.0

```
reactiveclass Controller(5) {
knownrebecs {
Door door;
Train train;
}
statevars { boolean moveP; }
Controller() {
self.external();
}
msgsrv external() {
boolean oldMoveP = moveP;
moveP = ?(true,false);
if(moveP != oldMoveP) {
door.lock(moveP);
train.move(moveP);
}
self.external() after(1);
}
}
reactiveclass Train(5) {
statevars { boolean moving; }
Train() {
moving = false;
}
msgsrv move(boolean tmove) {
if (tmove) {
moving = true;
} else {
moving = false;
}
}
}
reactiveclass Door(5) {
statevars { boolean is_locked; }
Door() {
is_locked = false;
}
msgsrv lock (boolean lockPar) {
is_locked = lockPar;
}
}
main {
@priority(1) Controller controller(do
train):();
@priority(2) Train train():();
@priority(2) Door door():();
}
```

LINGUA FRANCA

```
target C;
reactor Controller {
output lock:bool;
output move:bool;
physical action external:bool
reaction(startup) {=
... Set up sensing.
=}
reaction(external)->lock, mov
set(lock, external_value);
set(move, external_value);
=}
}
reactor Train {
input move:bool;
state moving:bool(false);
reaction(move) {=
... actuate to move or stop
self->moving = move;
=}
}
reactor Door {
input lock:bool;
state locked:bool(false);
reaction(lock) {=
... Actuate to lock or unlock door.
self->locked = lock;
=}
}
main reactor System {
controller = new Controller();
door = new Door();
train = new Train();
controller.lock -> door.lock;
controller.move -> train.move;
}
```

Lingua Franca Construct/Features	Timed Rebeca Construct/Features
reactor	*reactiveclass*
reaction	*msgsrv*
trigger	*msgsrv name*
state	*statevars*
input	*msgsrv*
output	*known rebecs*
physical action	*msgsrv*
implicit in the topology	*Priority*
main	*main*
instantiation ***(new)***	*instantiation of rebecs*
connection	*implicit in calling message servers*
after	*after*
–	*delay*



Verification of Cyberphysical Systems, Marjan Sirjani, Edward A. Lee and Ehsan Khamespanah, Mathematics journal, Mathematics, July 2020.

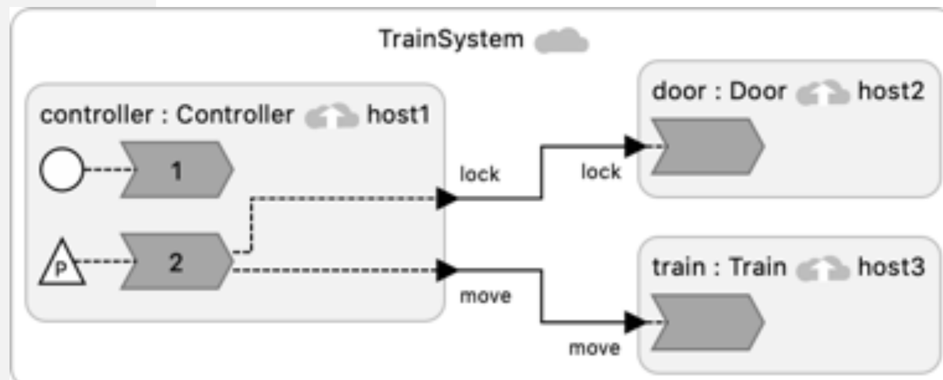
From Timed Rebeca to Lingua Franca


```
reactiveclass Controller(5) {
knownrebecs {
Door door;
Train train;
}
statevars { boolean moveP; }
Controller() {
self.external();
}
msgsrv external() {
boolean oldMoveP = moveP;
moveP = ?(true,false);
if(moveP != oldMoveP) {
door.lock(moveP);
train.move(moveP);
}
self.external() after(1);
}
}
reactiveclass Train(5) {
statevars { boolean moving; }
Train() {
moving = false;
}
msgsrv move(boolean tmove) {
if (tmove) {
moving = true;
} else {
moving = false;
}
}
}
reactiveclass Door(5) {
statevars { boolean is_locked; }
Door() {
is_locked = false;
}
msgsrv lock (boolean lockPar) {
is_locked = lockPar;
}
}
main {
@priority(1) Controller controller(do
train):();
@priority(2) Train train():();
@priority(2) Door door():();
}
```

LINGUA FRANCA

```
target C;
reactor Controller {
output lock:bool;
output move:bool;
physical action external:bool
reaction(startup) {=
... Set up sensing.
=}
reaction(external)->lock, mov
set(lock, external_value);
set(move, external_value);
=}
}
reactor Train {
input move:bool;
state moving:bool(false);
reaction(move) {=
... actuate to move or stop
self->moving = move;
=}
}
reactor Door {
input lock:bool;
state locked:bool(false);
reaction(lock) {=
... Actuate to lock or unlock door.
self->locked = lock;
=}
}
main reactor System {
controller = new Controller();
door = new Door();
train = new Train();
controller.lock -> door.lock;
controller.move -> train.move;
}
```

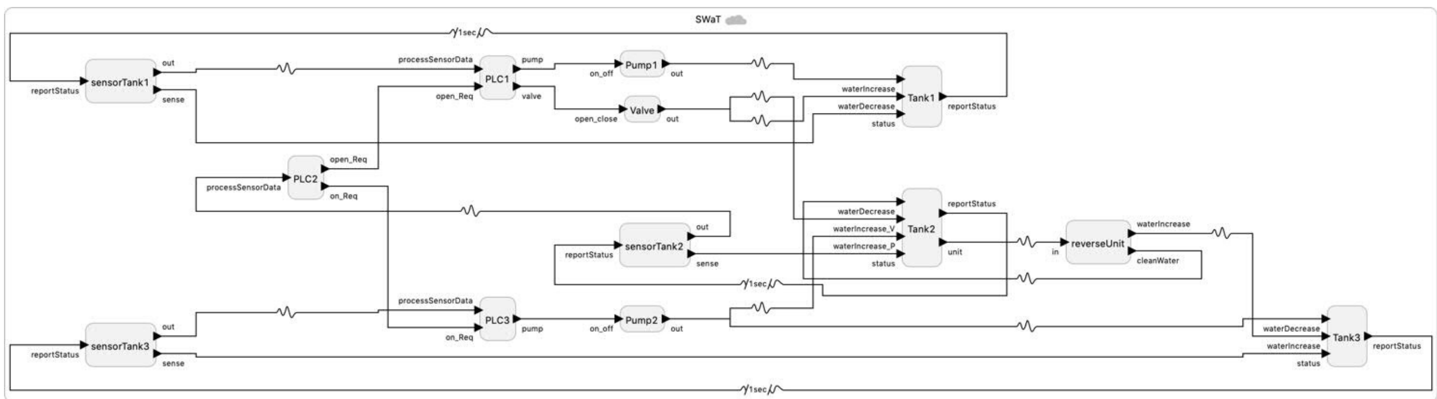
Lingua Franca Construct/Features	Timed Rebeca Construct/Features
reactor	*reactiveclass*
reaction	*msgsrv*
trigger	*msgsrv name*
state	*statevars*
input	*msgsrv*
output	*known rebecs*
physical action	*msgsrv*
implicit in the topology	*Priority*
main	*main*
*instantiation (**new**)*	*instantiation of rebecs*
connection	*implicit in calling message servers*
after	*after*
–	*delay*

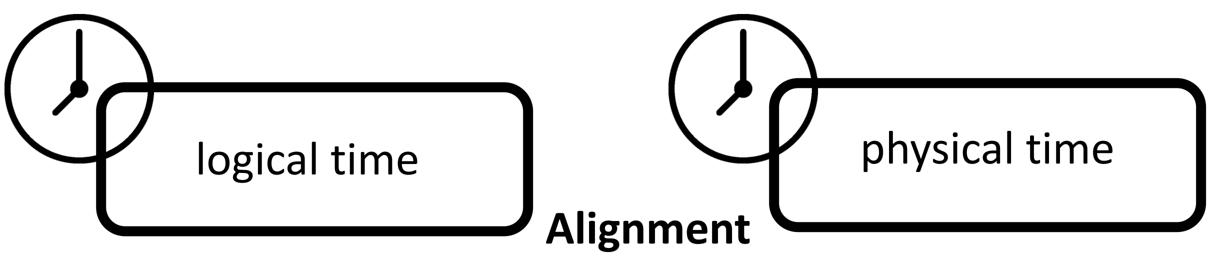


Verification of Cyberphysical Systems, Marjan Sirjani, Edward A. Lee and Ehsan Khamespanah, Mathematics journal, Mathematics, July 2020.

We may have to tweak Afra for different domains.

Alignment of Time by Lingua Franca



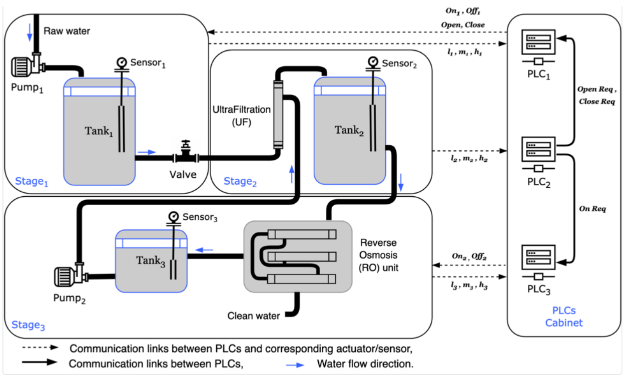


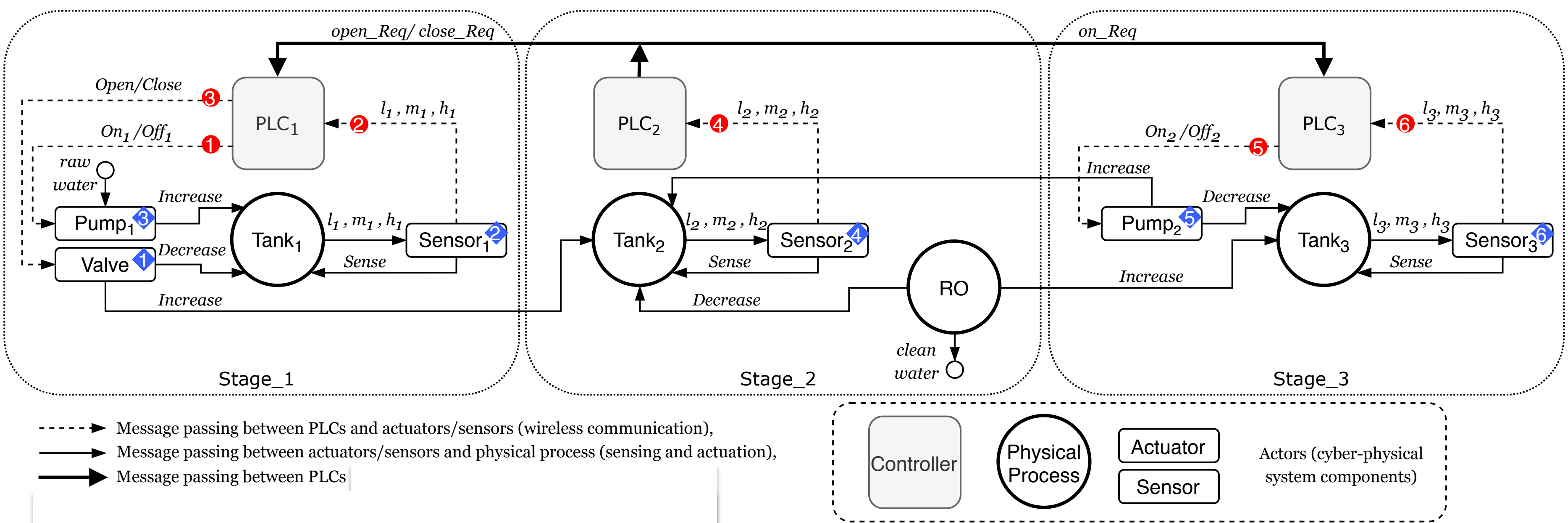
Toward a Lingua Franca for Deterministic Concurrent Systems, Marten Lohstroh, Christian Menard, Soroush Bateni, and Edward A. Lee, ACM Transactions on Embedded Computing Systems (TECS), 20(4), May 2021.

Lingua Franca suggests a Paradigm Shift

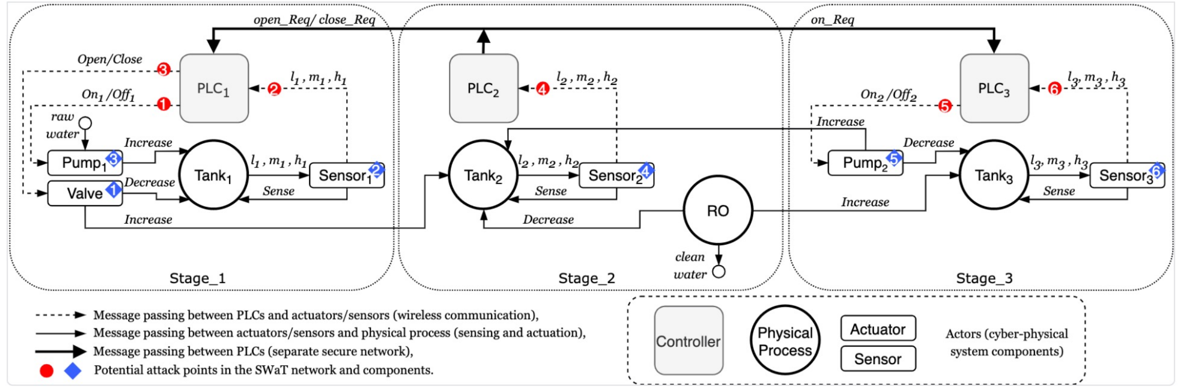
- Write a deterministic program
- Reduce the risk of bugs
- Have a more predictable system

Secure Water Treatment System





SWaT



Water Treatment System
Rebeca Model

```
reactiveclass PLC1(5){...}
reactiveclass PLC2(5){...} reactiveclass PLC3(5){...}
reactiveclass Tank1(10){...}
reactiveclass Tank2(10){...} reactiveclass Tank3(10){...}
reactiveclass Pump1(10){...}
reactiveclass Pump2(10){...} reactiveclass Valve(10){...}
reactiveclass SensorTank1(10){...} reactiveclass SensorTank2(10){...}
reactiveclass SensorTank3(10){...} reactiveclass reverseOsmosisUnit(5){...}
reactiveclass Attacker(3){...}
main{
    PLC1 plc1(pump1,valve,sensor1):();
    PLC2 plc2(plc1,plc3,sensor2):();
    PLC3 plc3(pump2,tank3,sensor3):();
    Tank1 tank1(sensor1):();
    Tank2 tank2(sensor2,unit):();
    ...
    Attacker attacker(plc1,plc2,plc3,pump1,pump2,valve):(chl,malMsg,attackTime);
}
```

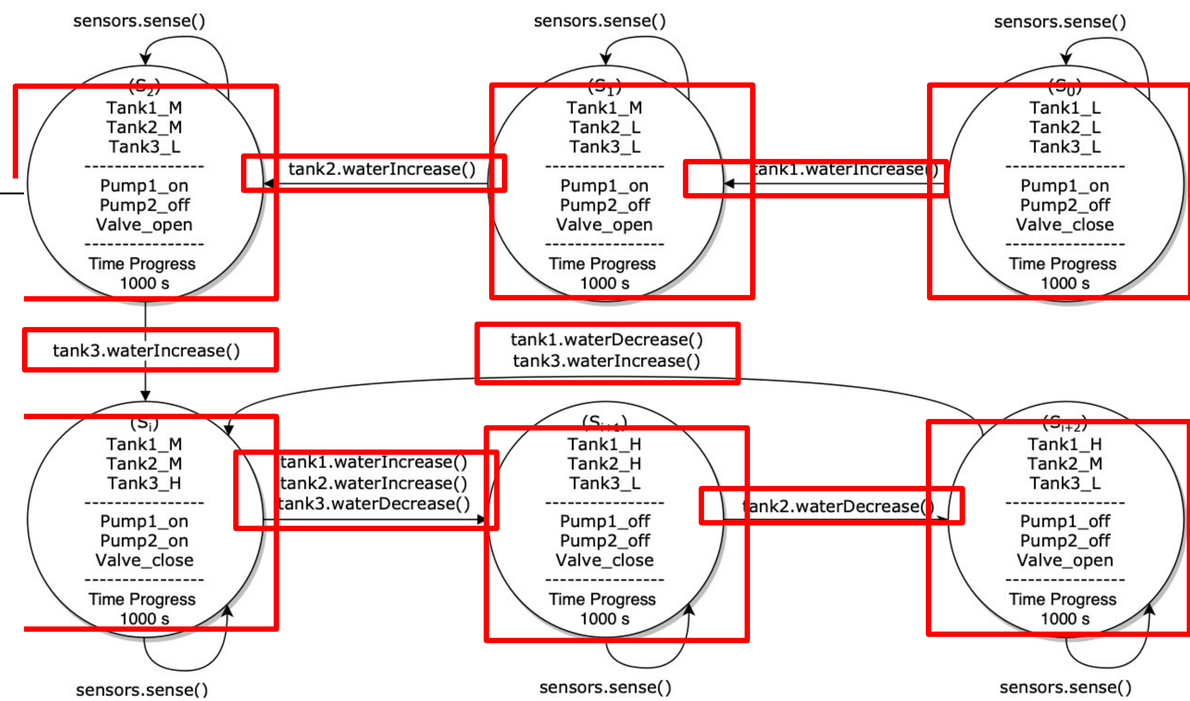
Model Checking

- **Tanks Overflow and Underflow**

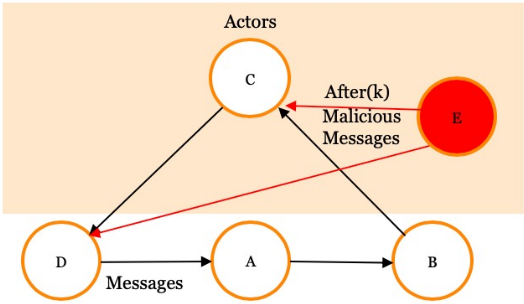
```
property {
    define {
        t1_overFlow = tank1.overFlow;
        t1_underFlow = tank1.underFlow;
        t2_overFlow = tank2.overFlow;
        t2_underFlow = tank2.underFlow;
        t3_overFlow = tank3.overFlow;
        t3_underFlow = tank3.underFlow;}
    Assertion{
        safety_tank1_over: !(t1_overFlow);
        safety_tank1_under: !(t1_underFlow);
        safety_tank2_over: !(t2_overFlow);
        safety_tank2_under: !(t2_underFlow);
        safety_tank3_over: !(t3_overFlow);
        safety_tank3_under: !(t3_underFlow);}
}
```

Properties

State Transition Diagram



Security Analysis

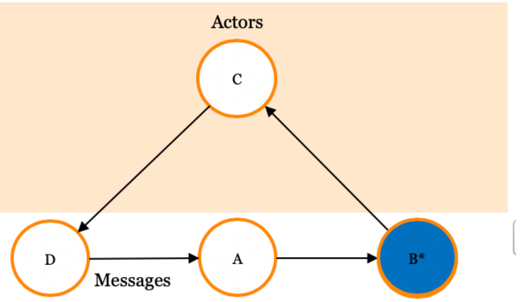


Successful Attack Scenarios

Attack on Communications

#	*Tank*	*Property*	*Injected Message*	*Communication Channel*	*System State*
1	$Tank_1$	Overflow	Water level in $Tank_1$ is low	$Sensor_1$ to PLC_1	S_{i+1}
2	$Tank_1$	Overflow	Turn on $Pump_1$	PLC_1 to $Pump_1$	S_{i+1}
3	$Tank_1$	Overflow	Water level in $Tank_1$ is low	$Sensor_1$ to PLC_1	S_{i+2}
4	$Tank_1$	Overflow	Turn on $Pump_1$	PLC_1 to $Pump_1$	S_{i+2}
5	$Tank_1$	Underflow	Water level in $Tank_1$ is high	$Sensor_1$ to PLC_1	S_0
6	$Tank_2$	Overflow	Water level in $Tank_2$ is medium	$Sensor_2$ to PLC_2	S_{i+1}
7	$Tank_2$	Overflow	Open Valve	PLC_1 to Valve	S_{i+1}
8	$Tank_3$	Overflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	S_i
9	$Tank_3$	Overflow	Open Valve	PLC_1 to Valve	S_i
10	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_0
11	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_1
12	$Tank_3$	Underflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	S_2
13	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_2
14	$Tank_3$	Underflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	S_{i+2}
15	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_{i+2}

Security Analysis



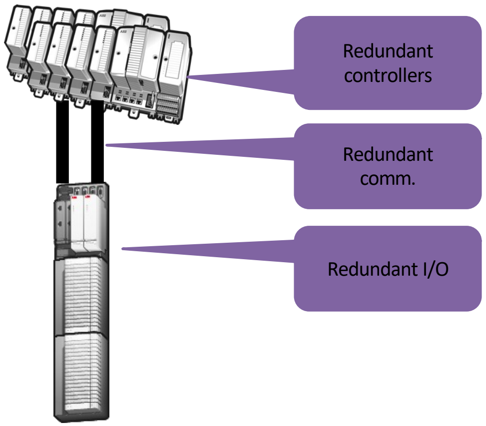
Successful Attack Scenarios

Attack on Components

#	*Tank*	*Property*	*Compromised Component*	*Malicious Behaviour*	*System State*
1	$Tank_1$	Overflow	$Sensor_1$	Water level in $Tank_1$ is low	S_{i+1}
2	$Tank_1$	Overflow	$Pump_1$	Turn on	S_{i+1}
3	$Tank_1$	Overflow	$Sensor_1$	Water level in $Tank_1$ is low	S_{i+2}
4	$Tank_1$	Underflow	$Sensor_1$	Water level in $Tank_1$ is high	S_0
5	$Tank_2$	Overflow	$Sensor_2$	Water level in $Tank_2$ is medium	S_{i+1}
6	$Tank_3$	Overflow	$Sensor_2$	Water level in $Tank_2$ is low	S_i
7	$Tank_3$	Overflow	Valve	Open	S_i
8	$Tank_3$	Underflow	$Pump_2$	Turn on	S_1
9	$Tank_3$	Underflow	$Sensor_3$	Water level in $Tank_3$ is high	S_2
10	$Tank_3$	Underflow	$Pump_2$	Turn on	S_{i+1}
11	$Tank_3$	Underflow	$Sensor_3$	Water level in $Tank_3$ is high	S_{i+2}

Industrial Controller Redundancy

- Controller redundancy!



Redundancy motivation:

Critical applications/domains → downtime costly

- Redundancy – hardware multiplication.
- Standby units (backup) ready to resume incase of primary failure

From Bjarne Johansson, ABB

Network oriented controllers

- Controller redundancy impact

The trend:

Controller redundancy today:

Controller redundancy tomorrow:

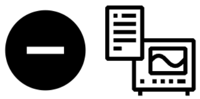
Controller Redundancy

Controller redundancy synchronization over dedicated link.

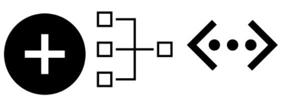
Controller Redundancy

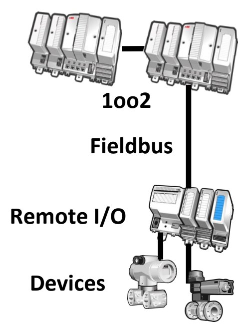
Controller redundancy synchronization over network..

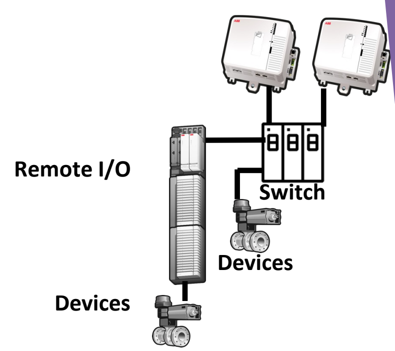
Less specialized HW



More Ethernet and networking

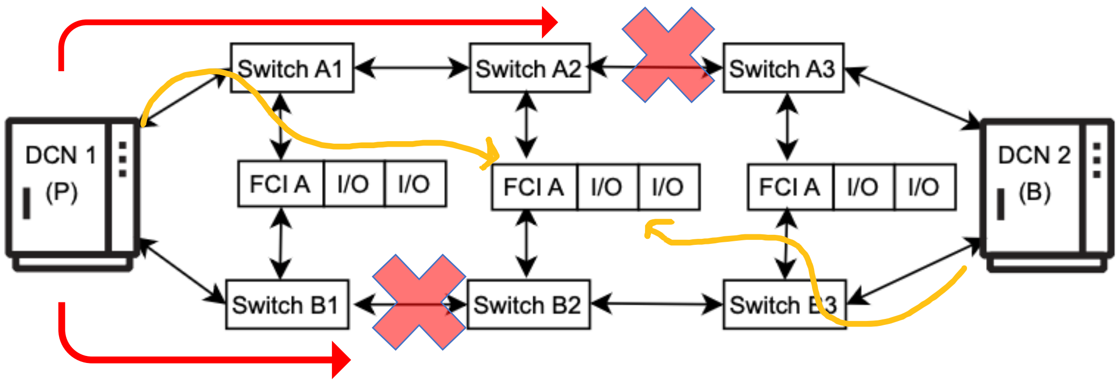






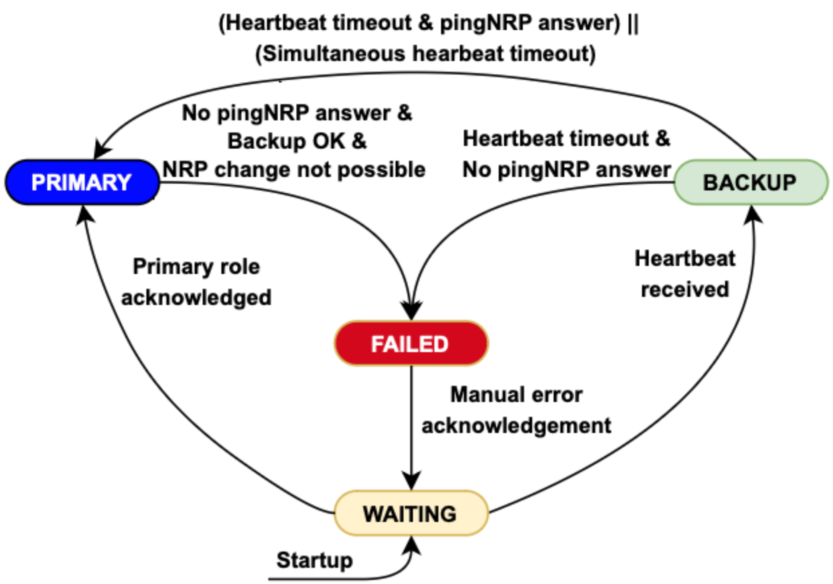
From Bjarne Johansson, ABB Industrial Automation

Distributed control systems



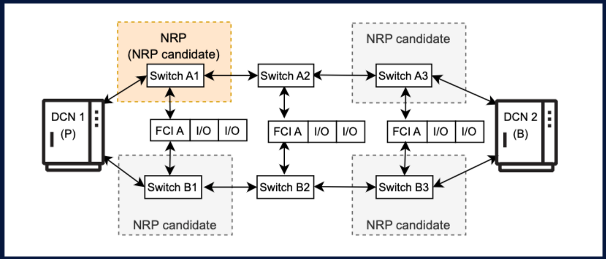
Network Reference Point Failure Detection (NRP FD) algorithm

Johansson et al. (2023)



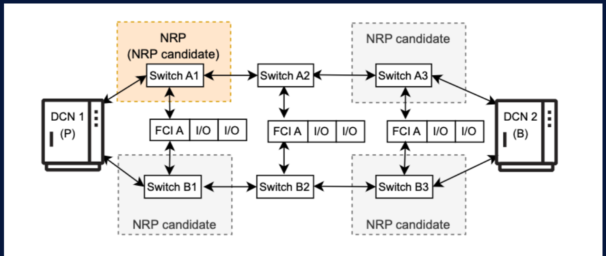
Inconsistency:
existence of more than one primary controller

Modeling NRP FD using Timed Rebeca




```
env int heartbeat_period = 1000;
env int max_missed_heartbeats = 2;
env int ping_timeout =500;
env int nrp_timeout = 500;
env byte NumberOfNetworks = 2;
env int switchA1failtime = 2500;
...
env int networkDelay = 1;
env int networkDelayForNRPPing = 1;
reactiveclass Node (4){ //'\label{line:ls1_line9}'
    knownrebecs {Switch out1, out2;}
    statevars {...}
    Node (int Myid, int Myprimary, int NRPCan1_id, int NRPCan2_id, int myFailTime) {
        id = Myid;
        NRPCandidates[0] =NRPCan1_id;
        NRPCandidates[1] =NRPCan2_id;
        NRP_network = -1;
        NRP_switch_id = -1;
        primary = Myprimary;
        init=true;
        mode = WAITING;
        ...
        if(myFailTime!=0) nodeFail() after(myFailTime);
        runMe();
    }
    msgsrv new_NRP_request_timed_out(){...}
    msgsrv ping_timed_out() {...}
    msgsrv pingNRP_response(int mid){...}
    msgsrv new_NRP(int mid, int mNRP_network, int mNRP_switch_id) {...}
    msgsrv runMe(){
        if(?(true,false)) nodeFail();
        switch(mode){
            case 0: //WAITING : ...
            case 1: //PRIMARY : ...
            case 2: //BACKUP : ...
            case 3: //FAILED : ...
        self.runMe() after(heartbeat_period);
     }
     msgsrv heartBeat(byte networkId, int senderid) {...}
     msgsrv nodeFail(){...}
}
```

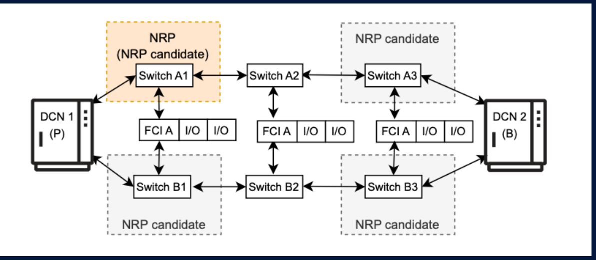
Modeling NRP FD using Timed Rebeca




```
env int heartbeat_period = 1000;
env int max_missed_heartbeats = 2;
env int ping_timeout =500;
env int nrp_timeout = 500;
env byte NumberOfNetworks = 2;
env int switchA1failtime = 2500;
...
env int networkDelay = 1;
env int networkDelayForNRPPing = 1;
reactiveclass Node (4){ //'\label{line:ls1_line9}'
    knownrebecs {Switch out1, out2;}
    statevars {...}
    Node (int Myid, int Myprimary, int NRPCan1_id,
        id = Myid;
        NRPCandidates[0] =NRPCan1_id;
        NRPCandidates[1] =NRPCan2_id;
        NRP_network = -1;
        NRP_switch_id = -1;
        primary = Myprimary;
        init=true;
        mode = WAITING;
        ...
        if(myFailTime!=0) nodeFail() after(myFailT
        runMe();
    }
    msgsrv new_NRP_request_timed_out(){...}
    msgsrv ping_timed_out() {...}
    msgsrv pingNRP_response(int mid){...}
    msgsrv new_NRP(int mid, int mNRP_network, int
    msgsrv runMe(){
        if(?(true,false)) nodeFail();
        switch(mode){
            case 0: //WAITING : ...
            case 1: //PRIMARY : ...
            case 2: //BACKUP : ...
            case 3: //FAILED : ...
        self.runMe() after(heartbeat_period);
     }
     msgsrv heartBeat(byte networkId, int senderid
     msgsrv nodeFail(){...}
}
```

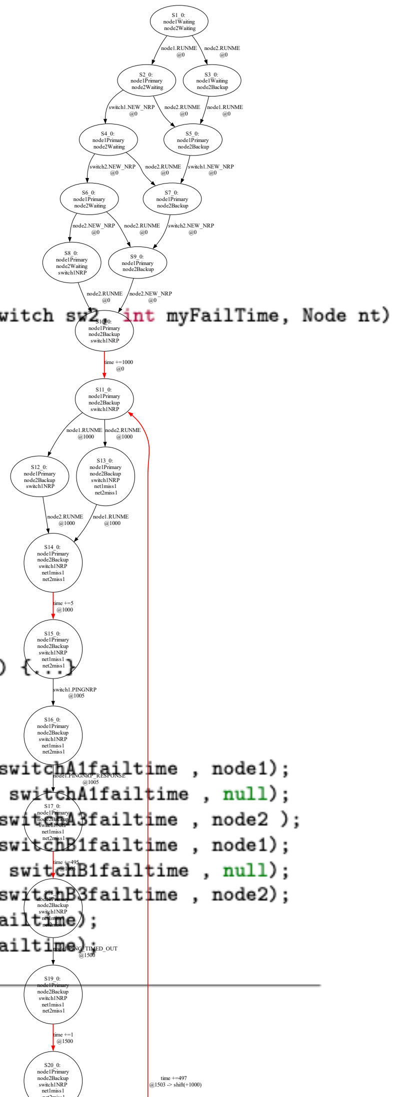
```
reactiveclass Switch(10){
    knownrebecs {...}
    statevars {...}
    Switch (int myid, byte networkId, boolean endSwitch , Switch sw1, Switch sw2, int myFailTime, Node nt)
       mynetworkId = networkId;
       id = myid;
       terminal=endSwitch;
       amINRP = false;
       failed = false;
       switchTarget1 = sw1;
       switchTarget2 = sw2;
       nodeTarget1 = nt;
    }
    msgsrv switchFail(){ failed = true; amINRP=false;}
    msgsrv request_new_NRP(int senderNode) {...}
    msgsrv pingNRP_response(int senderNode){...}
    msgsrv pingNRP( int senderNode, int NRP) {...}
    msgsrv new_NRP(int senderNode, int mNRP_network, int mNRP_switch_id) {...}
    msgsrv heartBeat(byte networkId, int senderNode) {...}
}
main {
    @Priority(1) Switch switchA1():(1, 0, true , switchA2 , switchA2 , switchA1failtime , node1);
    @Priority(1) Switch switchA2():(2, 0, false , switchA1 , switchA3 , switchA1failtime , null);
    @Priority(1) Switch switchA3():(3, 0, true , switchA2 , switchA2 , switchA3failtime , node2 );
    @Priority(1) Switch switchB1():(4, 1, true , switchB2 , switchB2 , switchB1failtime , node1);
    @Priority(1) Switch switchB2():(5, 1, false , switchB1 , switchB3 , switchB1failtime , null);
    @Priority(1) Switch switchB3():(6, 1, true , switchB2 , switchB2 , switchB3failtime , node2);
    @Priority(2) Node node1(switchA1, switchB1):(100, 100, 1, 4, node1failtime);
    @Priority(2) Node node2(switchA3, switchB3):(101, 100, 3, 6, node2failtime);
}
```

Modeling NRP FD using Timed Rebeca




```
env int heartbeat_period = 1000;
env int max_missed_heartbeats = 2;
env int ping_timeout =500;
env int nrp_timeout = 500;
env byte NumberOfNetworks = 2;
env int switchA1failtime = 2500;
...
env int networkDelay = 1;
env int networkDelayForNRPPing = 1;
reactiveclass Node (4){ //'\label{line:ls1_line9}'
    knownrebecs {Switch out1, out2;}
    statevars {...}
    Node (int Myid, int Myprimary, int NRPCan1_id,
        id = Myid;
        NRPCandidates[0] =NRPCan1_id;
        NRPCandidates[1] =NRPCan2_id;
        NRP_network = -1;
        NRP_switch_id = -1;
        primary = Myprimary;
        init=true;
        mode = WAITING;
        ...
        if(myFailTime!=0) nodeFail() after(myFailT
        runMe();
    }
    msgsrv new_NRP_request_timed_out(){...}
    msgsrv ping_timed_out() {...}
    msgsrv pingNRP_response(int mid){...}
    msgsrv new_NRP(int mid, int mNRP_network, int
    msgsrv runMe(){
        if(?(true,false)) nodeFail();
        switch(mode){
            case 0: //WAITING : ...
            case 1: //PRIMARY : ...
            case 2: //BACKUP : ...
            case 3: //FAILED : ...
        self.runMe() after(heartbeat_period);
     }
     msgsrv heartBeat(byte networkId, int senderid
     msgsrv nodeFail(){...}
}
```

```
reactiveclass Switch(10){
    knownrebecs {...}
    statevars {...}
    Switch (int myid, byte networkId, boolean endSwitch , Switch sw1, Switch sw2, int myFailTime, Node nt)
       mynetworkId = networkId;
       id = myid;
       terminal=endSwitch;
       amINRP = false;
       failed = false;
       switchTarget1 = sw1;
       switchTarget2 = sw2;
       nodeTarget1 = nt;
    }
    msgsrv switchFail(){ failed = true; amINRP=false;}
    msgsrv request_new_NRP(int senderNode) {...}
    msgsrv pingNRP_response(int senderNode){...}
    msgsrv pingNRP( int senderNode, int NRP) {...}
    msgsrv new_NRP(int senderNode, int mNRP_network, int mNRP_switch_id) {...}
    msgsrv heartBeat(byte networkId, int senderNode) {...}
}
main {
    @Priority(1) Switch switchA1():(1, 0, true , switchA2 , switchA2 , switchA1failtime , node1);
    @Priority(1) Switch switchA2():(2, 0, false , switchA1 , switchA3 , switchA1failtime , null);
    @Priority(1) Switch switchA3():(3, 0, true , switchA2 , switchA2 , switchA3failtime , node2 );
    @Priority(1) Switch switchB1():(4, 1, true , switchB2 , switchB2 , switchB1failtime , node1);
    @Priority(1) Switch switchB2():(5, 1, false , switchB1 , switchB3 , switchB1failtime , null);
    @Priority(1) Switch switchB3():(6, 1, true , switchB2 , switchB2 , switchB3failtime , node2);
    @Priority(2) Node node1(switchA1, switchB1):(100, 100, 1, 4, node1failtime);
    @Priority(2) Node node2(switchA3, switchB3):(101, 100, 3, 6, node2failtime);
}
```



Schedulability Analysis of
Distributed Real-Time Sensor Network Applications

(collaboration with OSL, UIUC, Gul Agha, and Ehsan Khamespanah, UT)

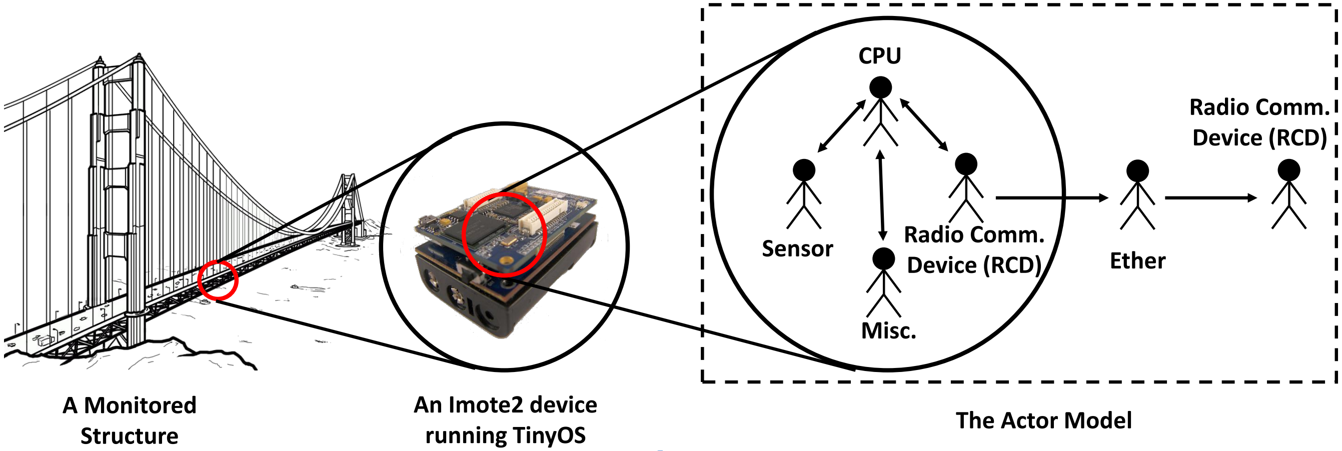
Smart Structures

"... one highly intelligent bridge knows what to do when trouble arises: send [the engineers] an e-mail."

The New York Times

Finding the best configuration

- Modeling the interactions between
  - the CPU, sensor and radio within each node
  - interactions among the nodes
  - tasks belonging to other applications, middleware services, and operating system components.



Rebeca Home Projects Tools Documents Examples Publications About

http://www.rebeca-lang.org/

Rebeca Modeling Language

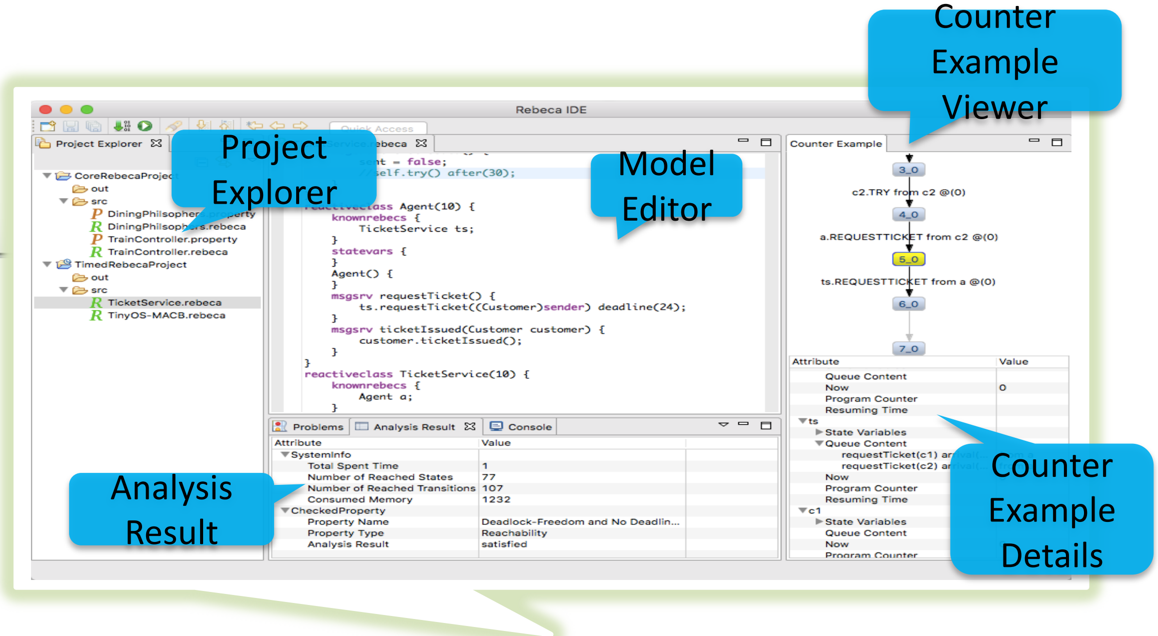
Actor-based Language with Formal Foundation

Rebeca (Reactive Objects Language) is an actor-based language with a formal foundation, designed in an effort to bridge the gap between formal verification approaches and real applications. It can be considered as a reference model for concurrent computation, based on an operational interpretation of the actor model. It is also a platform for developing object-based concurrent systems in practice. Learn More

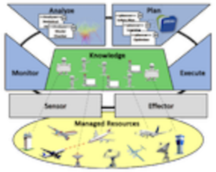
Actors and Components

Formal Semantics

Rebeca provides a formal semantics



Projects



SEADA

In SEADA (Self-Adaptive Actors) we will use Ptolemy to represent the architecture, and extensions of Rebeca for modeling and verification. Our models@runtime will be coded in an extension of Probabilistic Timed Rebeca, and supporting tools for customized run-time formal verification

RoboRebeca

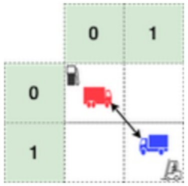
RoboRebeca is a framework which provides facilities for developing safe/correct source codes for robotic applications. In RoboRebeca, models are developed using Rebeca family language and automatically transformed into ROS compatible source codes. This framework is

HybridRebeca

Hybrid Rebeca, is an extension of actor-based language Rebeca, to support modeling of cyber-physical systems. In this extension, physical actors are introduced as new computational entities to encapsulate the physical behaviors. Learn more

Tangramob

Tangramob offers an Agent-Based



AdaptiveFlow

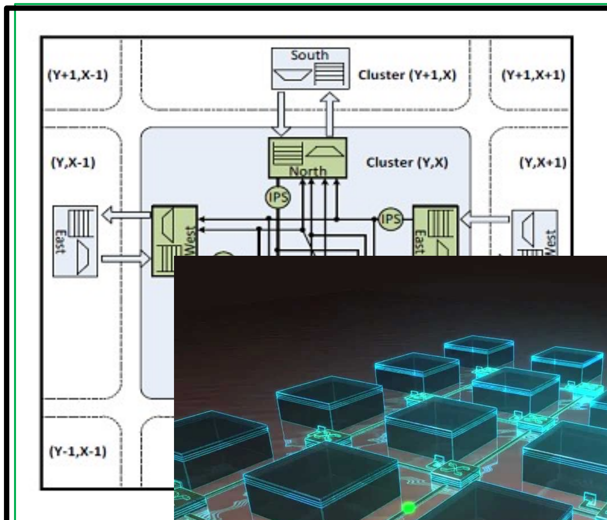
AdaptiveFlow is an actor-based eulerian

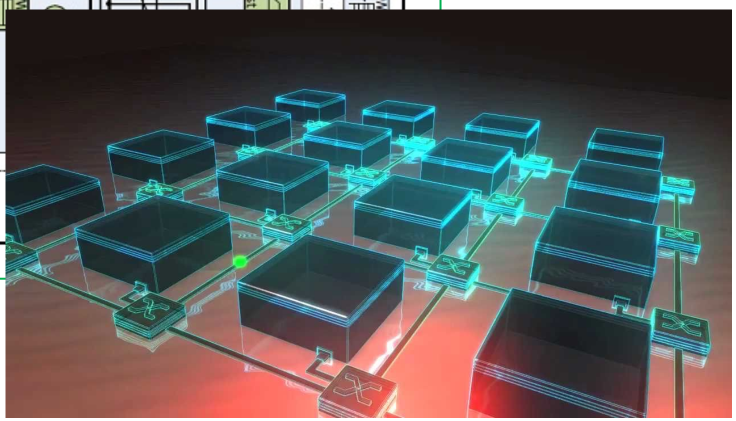
wRebeca

wRebeca is an actor-based modeling

Design Decisions
Network on Chip

Siamak Mohammadi, Zeinab Sharifi, UT





Design Decisions:

routing algorithms

Buffer length

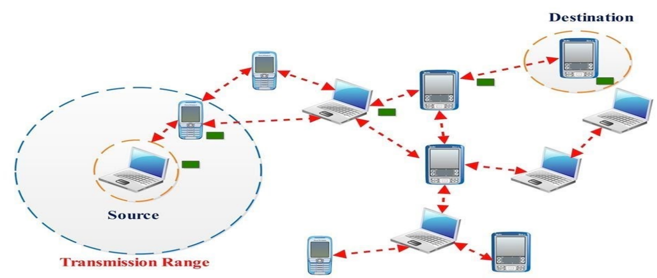
Memory Allocation

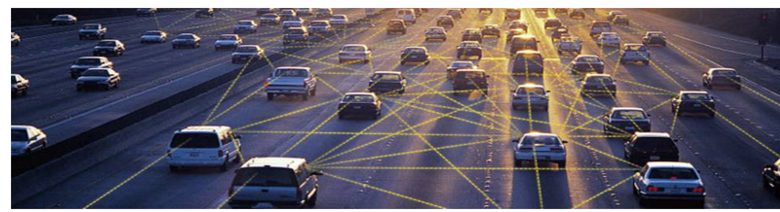
Zeinab Sharifi, Mahdi Mosaffa, Siamak Mohammadi, and Marjan Sirjani: Functional and Performance Analysis of Network-on-Chips Using Actor-based Modeling and Formal Verification, AVoCS, 2013.
https://rebeca-lang.org/assets/papers/2013/Performance-Analysis-of-NoC.pdf

Bug Check
Network Protocols

Fatemeh Ghassemi, Ramtin Khosravi, UT

MANET (*Mobile Ad Hoc Network*)





Deadlock and loop-freedom of Mobile Adhoc Networks

Behnaz Yousefi, Fatemeh Ghassemi, and Ramtin Khosravi: Modeling and Efficient Verification of Wireless Ad hoc Networks, volume 29, Issue 6, pp 1051–1086, Formal Aspects of Computing, 2017.
https://link.springer.com/article/10.1007/s00165-017-0429-z

Performance Optimization

Smart Structures

Gul Agha, OSI, UIUC and Ehsan Khamespanah, UT

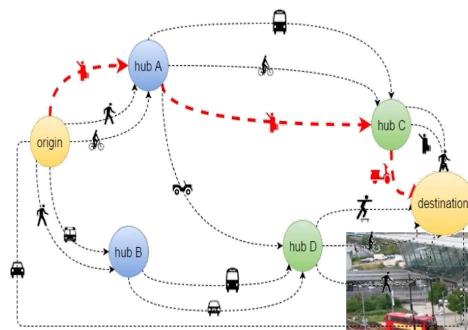
Schedulability Analysis of Distributed Real-Time Sensor Network: Finding the best configuration

Ehsan Khamespanah, Kirill Mechitov, Marjan Sirjani, Gul Agha: Modeling and Analyzing Real-Time Wireless Sensor and Actuator Networks Using Actors and Model Checking, Software Tools for Technology Transfer, 2017.
https://rebeca-lang.org/assets/papers/2017/Modeling-and-Analyzing-Real-Time-Wireless-Sensor-and-Actuator-Networks-Using-Actors-and-Model-Checking.pdf

Resource Management

Smart Transport Hubs

Andrea Polini, Francesco De Angelis, Unicam Smart Mobility Lab.



Minimize:

Number of service disruptions

Number of mobility resources in smart hubs

Cost of mobility for commuters

Travel time for commuters

Travel distance for commuters

Jacopo de Berardinis, Giorgio Forcina, Ali Jafari, Marjan Sirjani:
Actor-based macroscopic modeling and simulation for smart urban planning. Sci. Comput. Program. 168: 142-164 (2018)
https://www.sciencedirect.com/science/article/pii/S0167642318303459?via%3Dihub

Performance Optimization

Smart Structures

Gul Agha, OSI, UIUC and Ehsan Khamespanah, UT

Resource Management

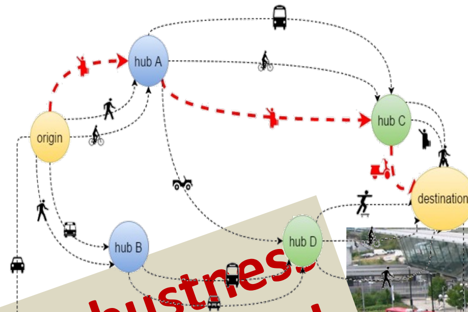
Smart Transport Hubs

Andrea Polini, Francesco De Angelis, Unicam Smart Mobility Lab.

Not only Safety and Robustness but also Performance, Cost and User Satisfaction

Schedulability Ana[...] of Distributed Real-Time Sensor Network: Finding the best configuration

Ehsan Khamespanah, Kirill Mechitov, Marjan Sirjani, Gul Agha: Modeling and Analyzing Real-Time Wireless Sensor and Actuator Networks Using Actors and Model Checking, Software Tools for Technology Transfer, 2017.
https://rebeca-lang.org/assets/papers/2017/Modeling-and-Analyzing-Real-Time-Wireless-Sensor-and-Actuator-Networks-Using-Actors-and-Model-Checking.pdf



Minimize:

Number of service disruptions

Number of mobility resources in smart hubs

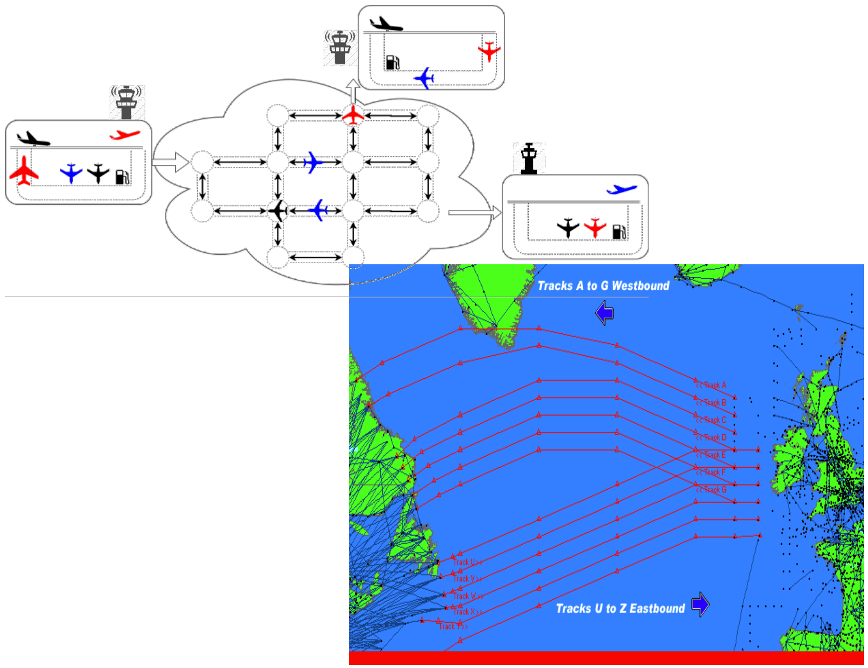
Cost of mobility for commuters

Travel time for commuters

Travel distance for commuters

Jacopo de Berardinis, Giorgio Forcina, Ali Jafari, Marjan Sirjani:
Actor-based macroscopic modeling and simulation for smart urban planning. Sci. Comput. Program. 168: 142-164 (2018)
https://www.sciencedirect.com/science/article/pii/S0167642318303459?via%3Dihub

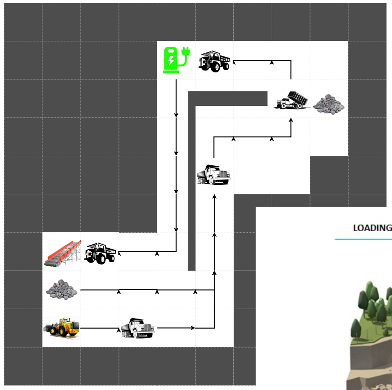
Adaptive Flow Management
Air Traffic Control
UC Berkeley, Edward Lee and Sharif, Ali Movaghar

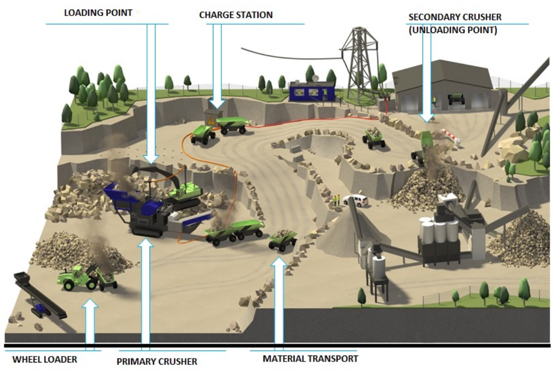


Adaptive Air Traffic Control: Safe rerouting of airplanes using Magnifier

Maryam Bagheri, Marjan Sirjani, Ehsan Khamespanah, Christel Baier, Ali Movaghar, Magnifier: A Compositional Analysis Approach for Autonomous Traffic Control, IEEE Transactions on Software Engineering, 2021
https://rebeca-lang.org/assets/papers/2021/Magnifier-A-Compositional-Analysis-Approach-for-Autonomous-Traffic-Control.pdf

Adaptive Flow Management
Volvo CE Quarry Site
Volvo-CE, Stephan Baumgart and Torbjörn Martinsson





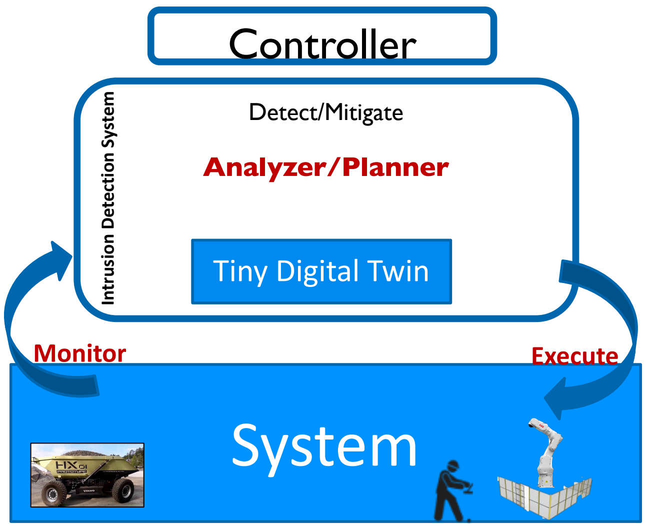
Safe and optimized fleet control

Marjan Sirjani, Giorgio Forcina, Ali Jafari, Stephan Baumgart, Ehsan Khamespanah, Ali Sedaghatbaf: An Actor-based Design Platform for System of Systems, IEEE 43th Annual Computers, Software, and Applications Conference (COMPSAC), 2019
https://rebeca-lang.org/assets/papers/2019/An-Actor-based-Design-Platform-for-System-of-Systems.pdf

Anomaly Detection

Model-Based Cyber-Security

UC Berkeley, Edward Lee and Sharif, Ali Movaghar



MAPE-K architecture
(Monitor- Analysis – Plan – Execute)- Knowledge

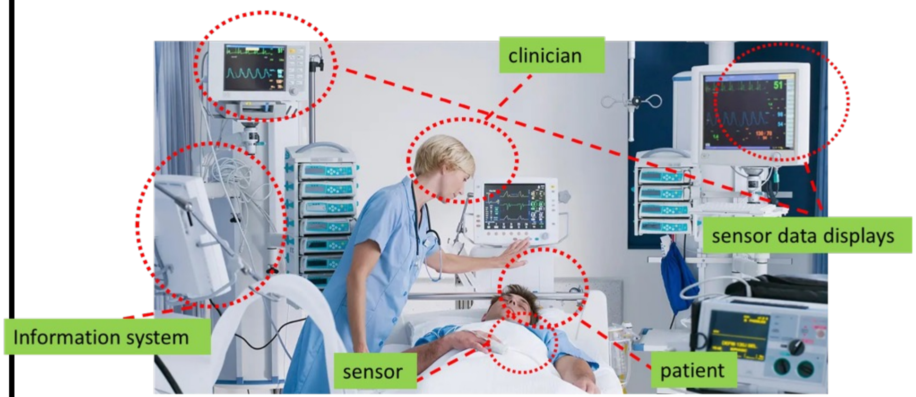
- Runtime **monitor** to check the system behavior using a **Tiny Digital Twin**

Fereidoun Moradi, Maryam Bagheri, Hanieh Rahmati, Hamed Yazdi, Sara Abbaspour Asadollah, Marjan Sirjani, Monitoring Cyber-Physical Systems using a Tiny Twin to Prevent Cyber-Attacks, 28th International Symposium on Model Checking of Software (SPIN), 2022
https://rebeca-lang.org/assets/papers/2022/Monitoring-Cyber-Physical-Systems-Using-a-Tiny-Twin-to-Prevent-Cyber-Attacks.pdf

Time Analysis

Connected Medical Systems

John Hatcliff, U. of Kansas, and Fatemeh Ghassemi, UT



Local properties of devices are assured by the vendors at the development time.

Verify the satisfaction of timing communication requirements.

Helpful for dynamic network configuration or capacity planning.

Mahsa Zarneshan, Fatemeh Ghassemi, Ehsan Khamespanah, Marjan Sirjani, John Hatcliff: Specification and Verification of Timing Properties in Interoperable Medical Systems. Log. Methods Comput. Sci. 18(2) (2022)
https://lmcs.episciences.org/9639

Final Message

We need both

Robustness

and

Friendliness!!

Examples from Industrial Partners

- ABB
- Volvo Construction Equipment
- Volvo Trucks

ABB Robotics Example

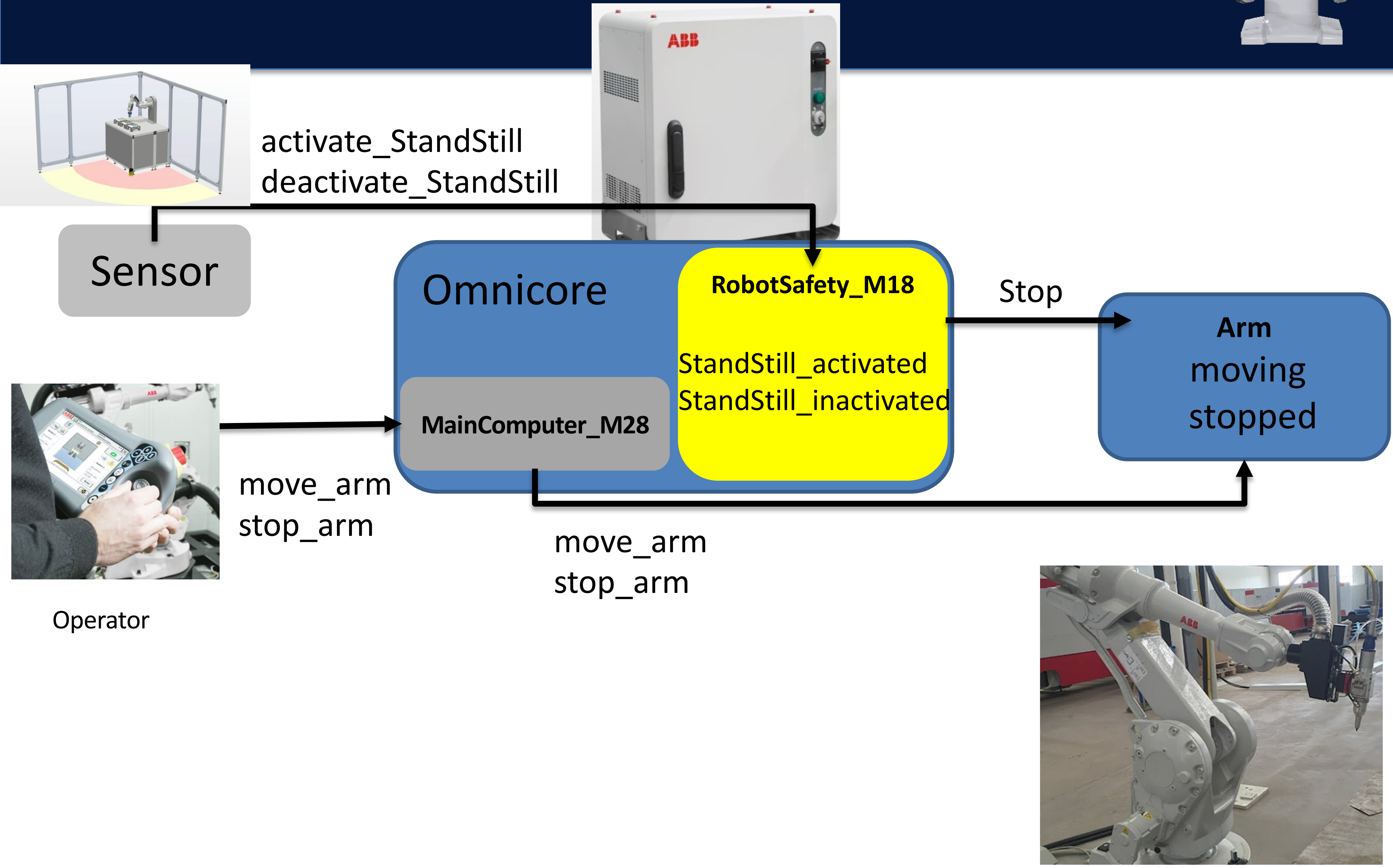


ABB Robotics Example

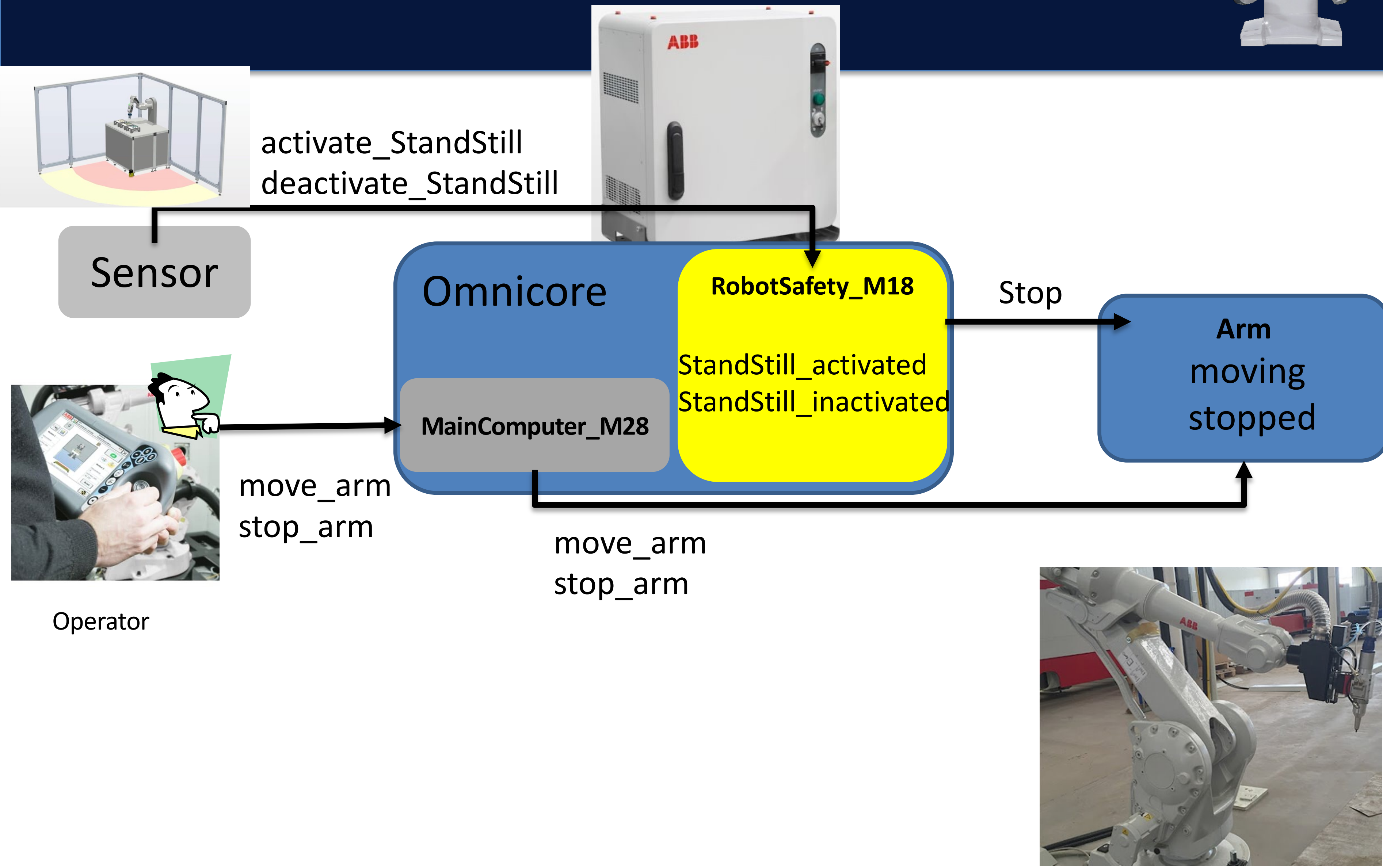


ABB Robotics Example

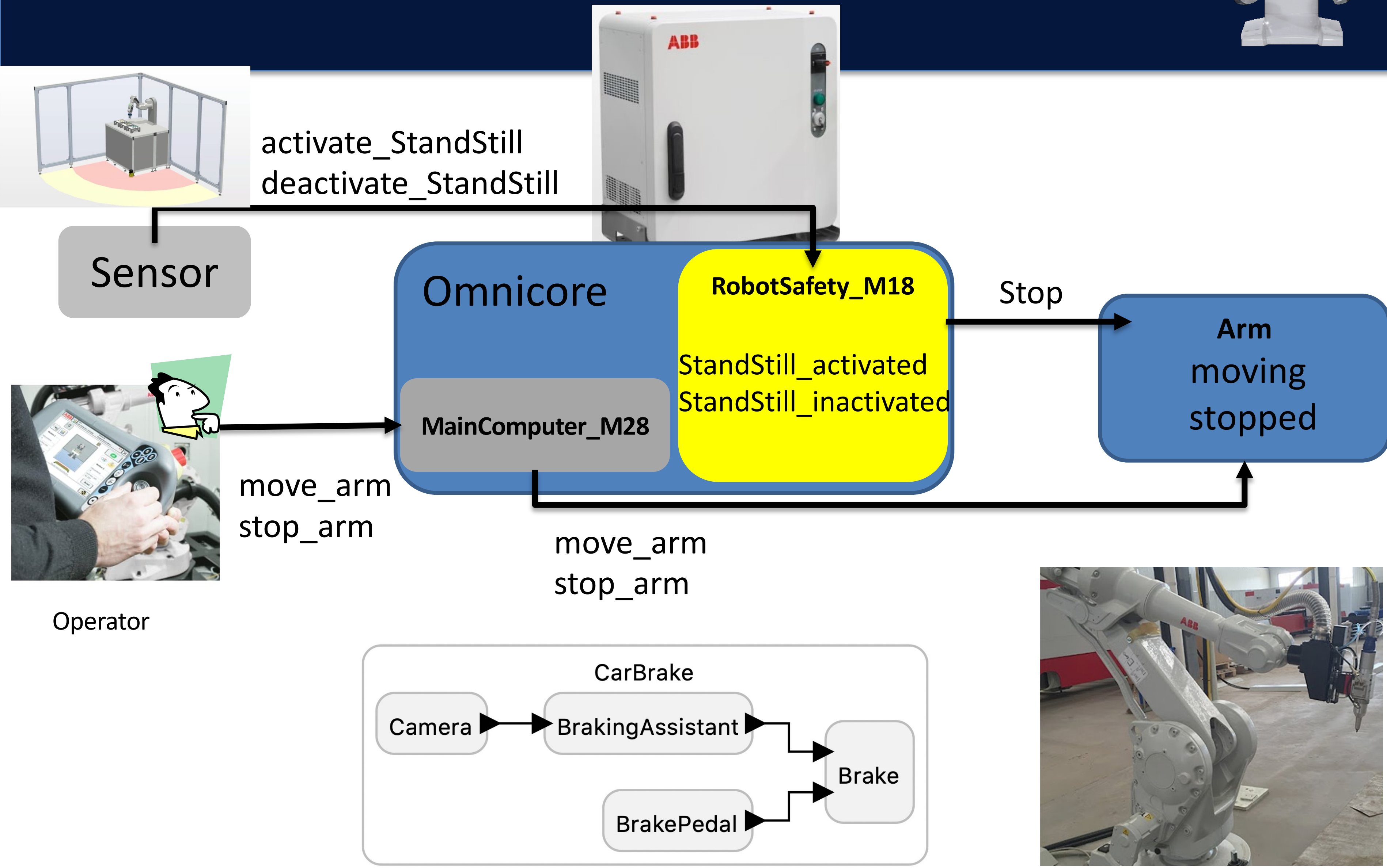
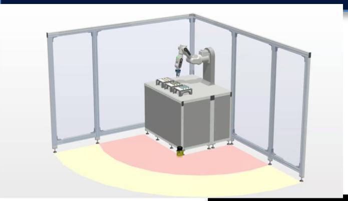


ABB Robotics Example



activate_StandStill
deactivate_StandStill

Sensor

Omnicore

RobotSafety_M18

StandStill_activated
StandStill_inactivated

MainComputer_M28

Stop

Arm
moving
stopped

move_arm
stop_arm

Operator

move_arm
stop_arm

CarBrake

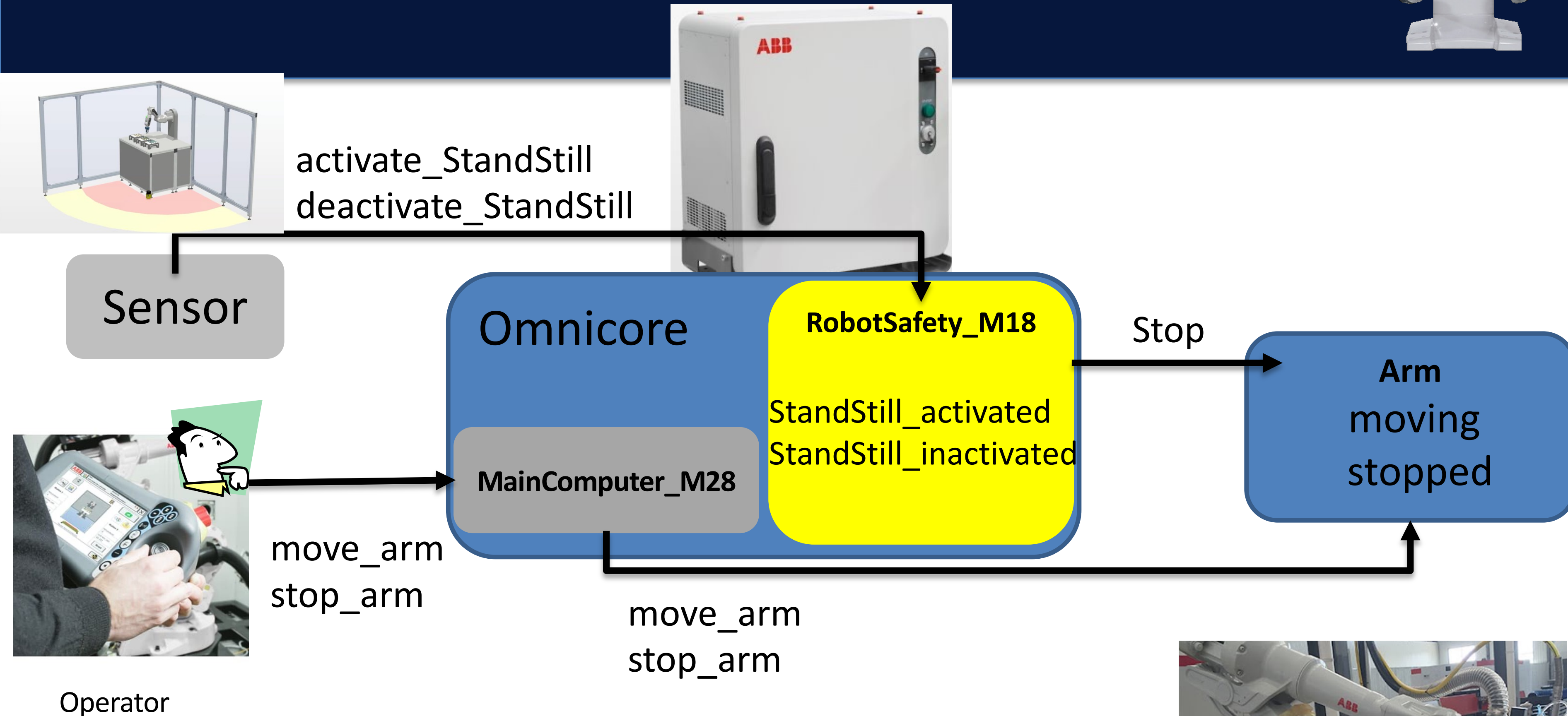
Camera → BrakingAssistant → Brake

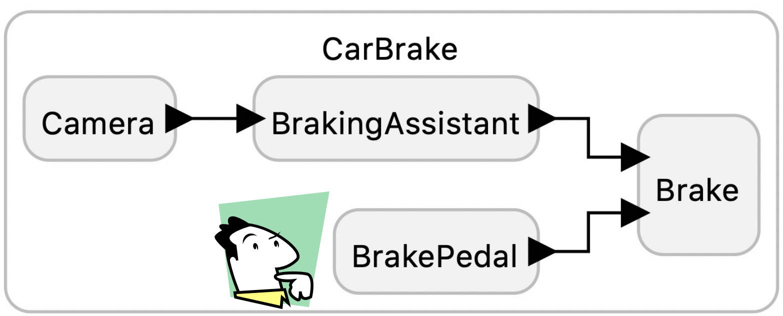
BrakePedal → Brake

Denso autonomous braking demonstrating Advanced Driver-Assistance System (ADAS) in Oct. 2018 [Reported in The Daily Times]

Thanks to Christian Menard (TU Dresden) for this example.

ABB Robotics Example

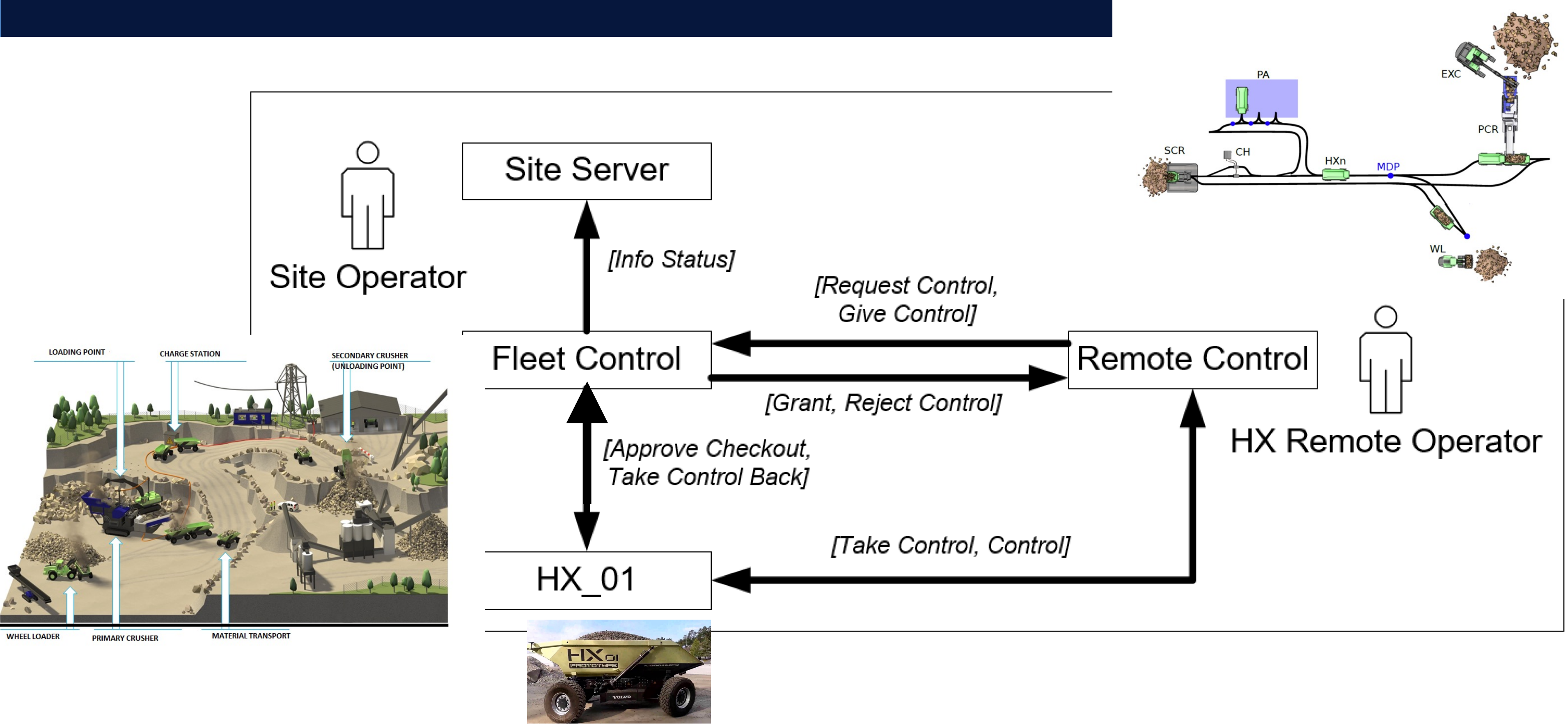




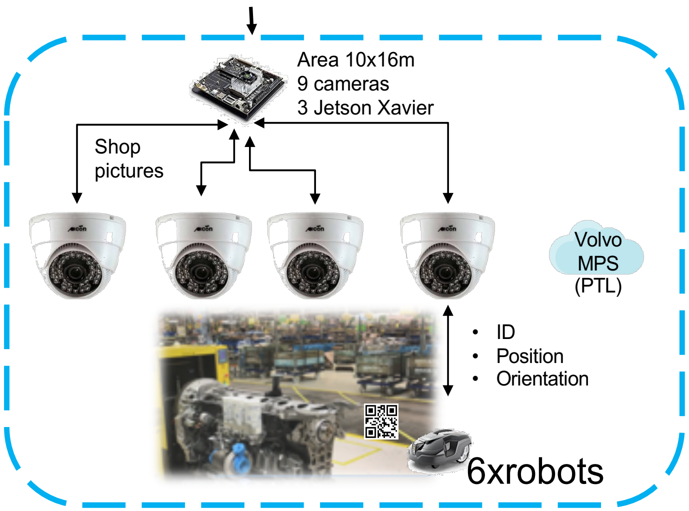
Denso autonomous braking demonstrating Advanced Driver-Assistance System (ADAS) in Oct. 2018 [Reported in The Daily Times]

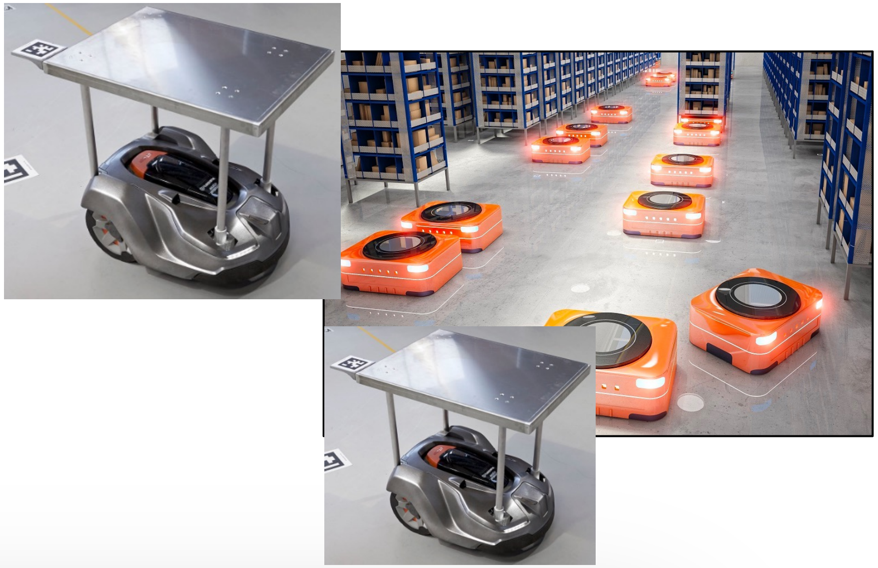
Thanks to Christian Menard (TU Dresden) for this example.

Volvo CE Example



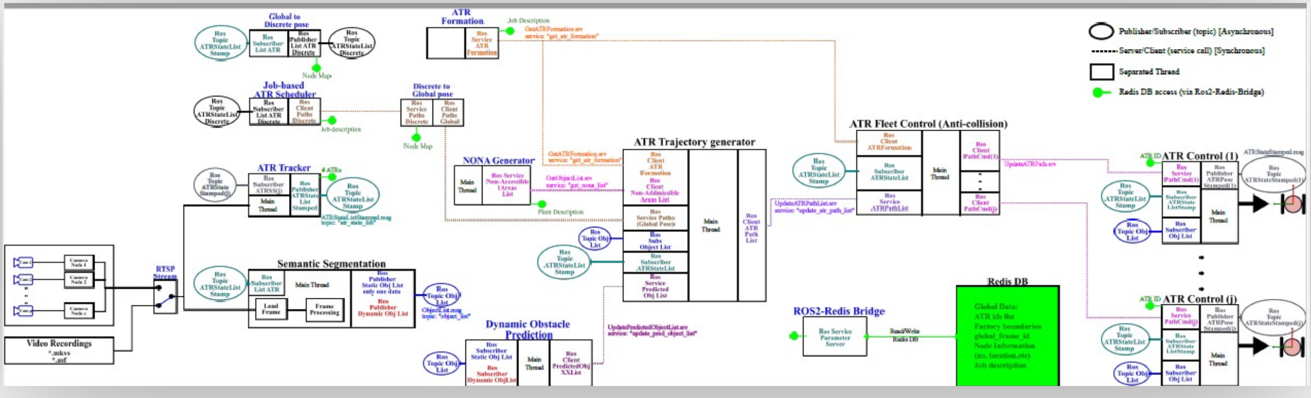
Volvo Trucks Example



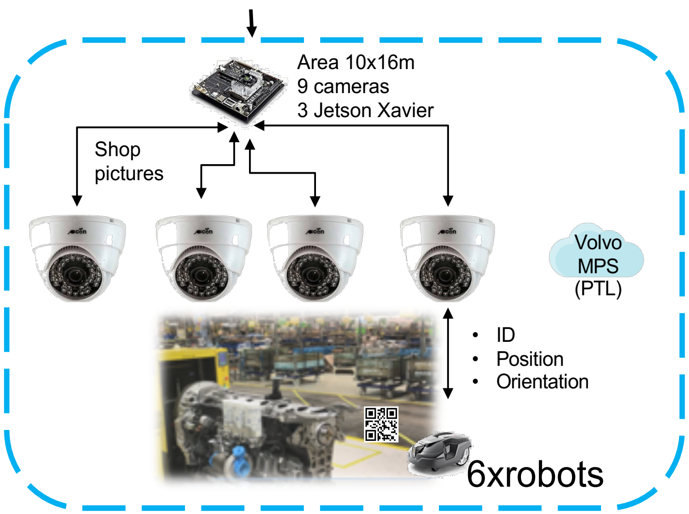


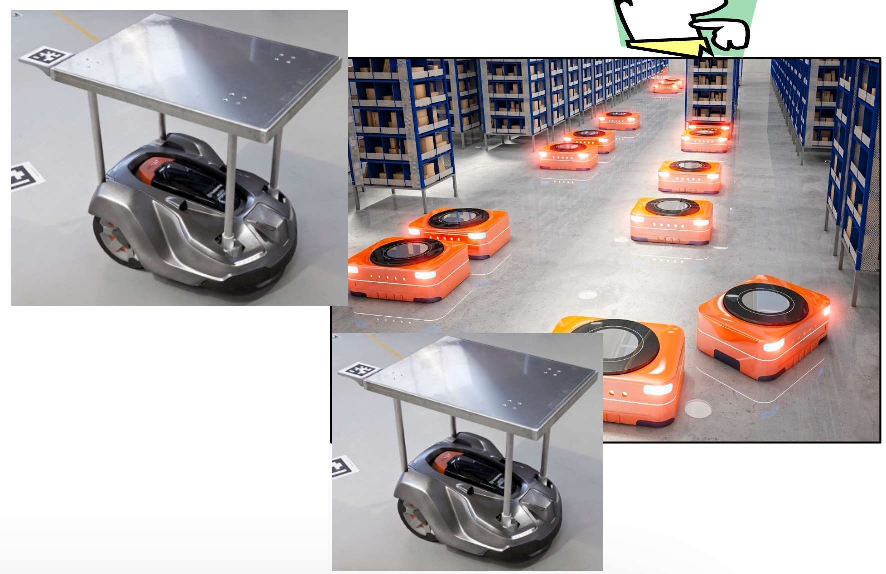
Volvo GPSS

A Generic Photogrametry based Sensor System



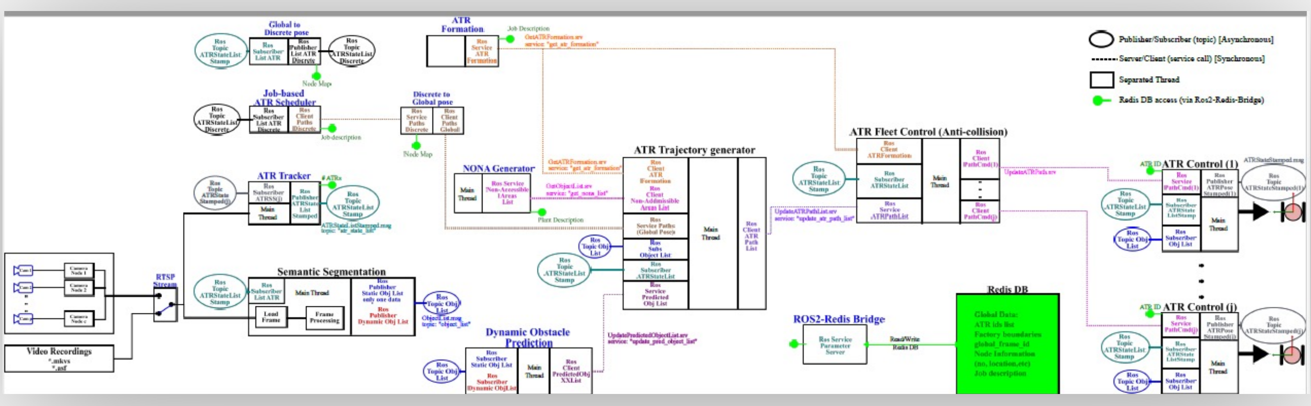
Volvo Trucks Example



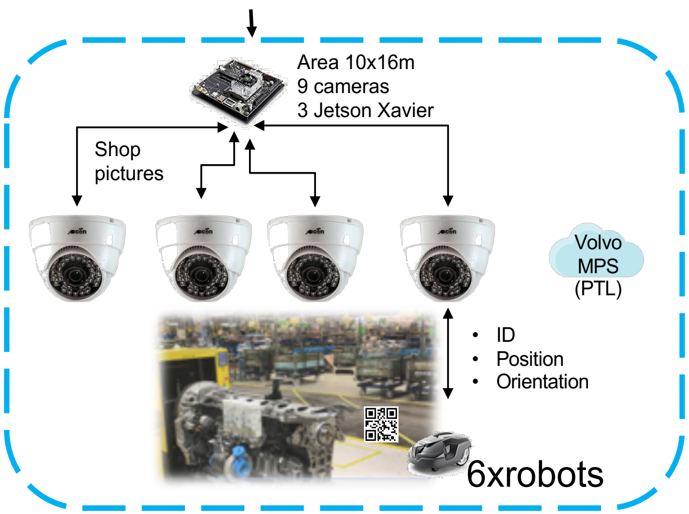


Volvo GPSS

A Generic Photogrametry based Sensor System

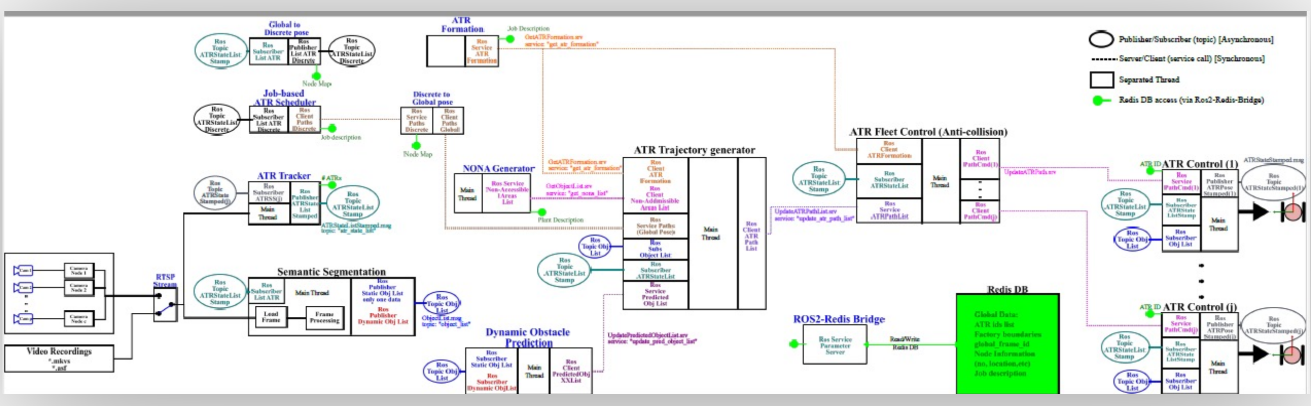


Volvo Trucks Example



Volvo GPSS

A Generic Photogrametry based Sensor System



Thank you!!