Policy-Driven System Design

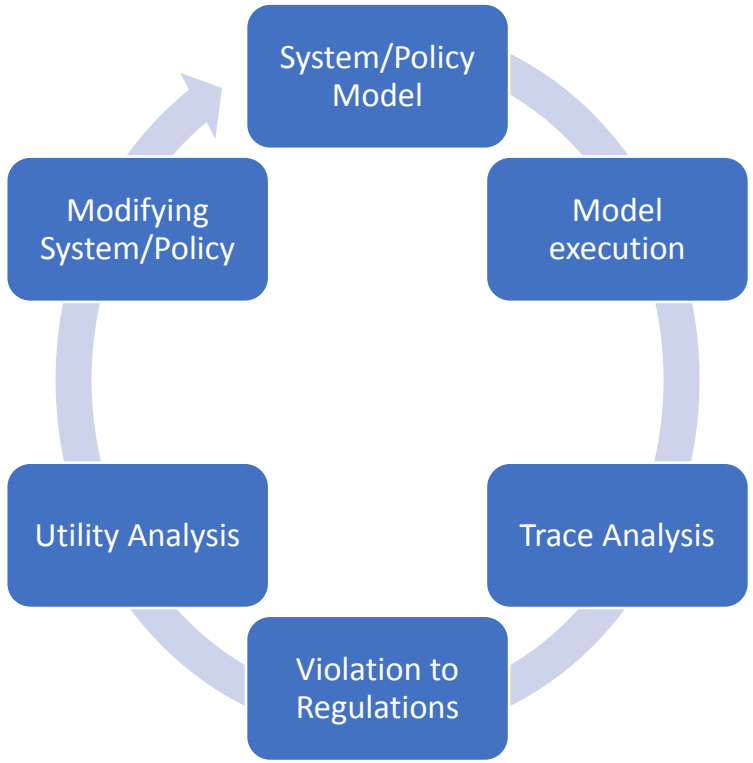
Mostafa Mohajeri
University of Amsterdam

CCI Meeting
Feb, 2022

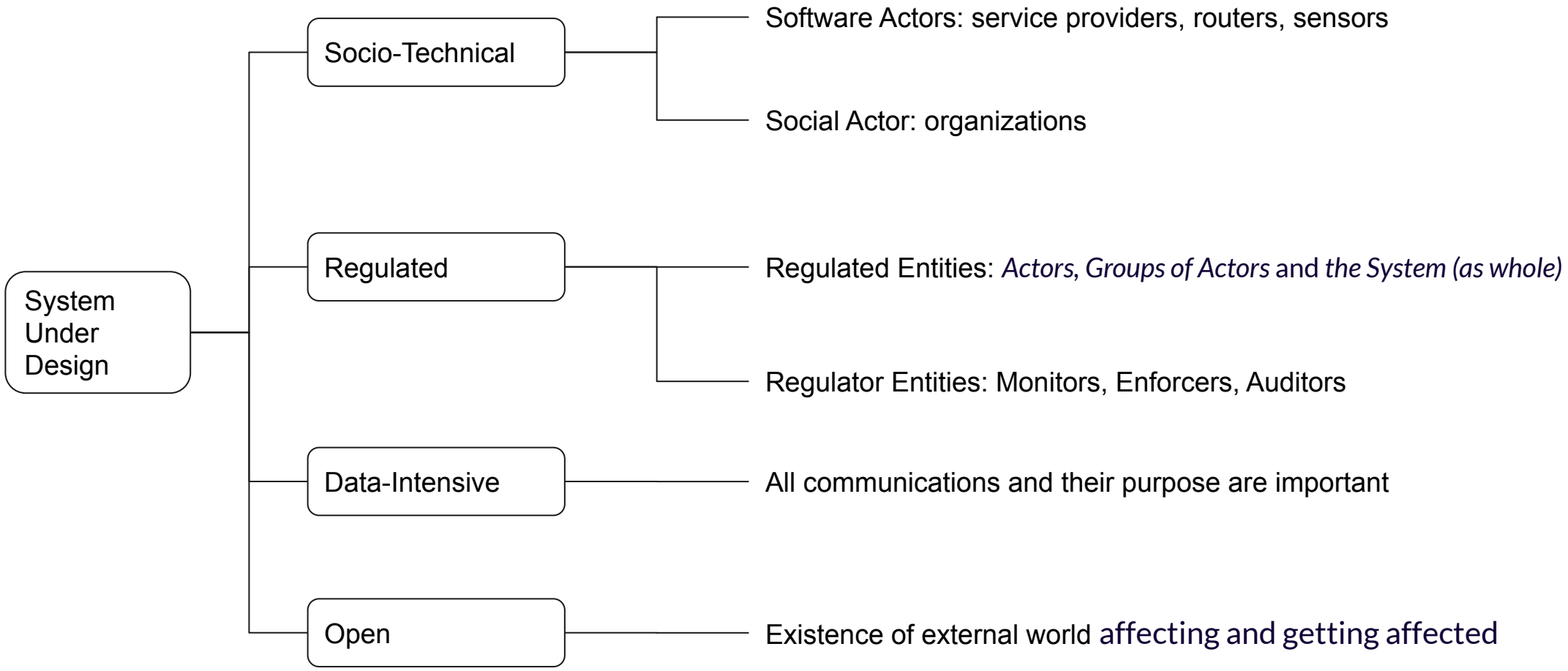
Project: LICCAM

- Legal Intervention in Connected Cooperative Automated Mobility
- Creating a demo implementation of the system
  - Monitoring roads and autonomous vehicles
  - Able to reason about possible future high risk states
  - Able to reason about possibility of intervention
    - Through 3rd party controllers (OEMs)
  - The legal process is part of the technical process

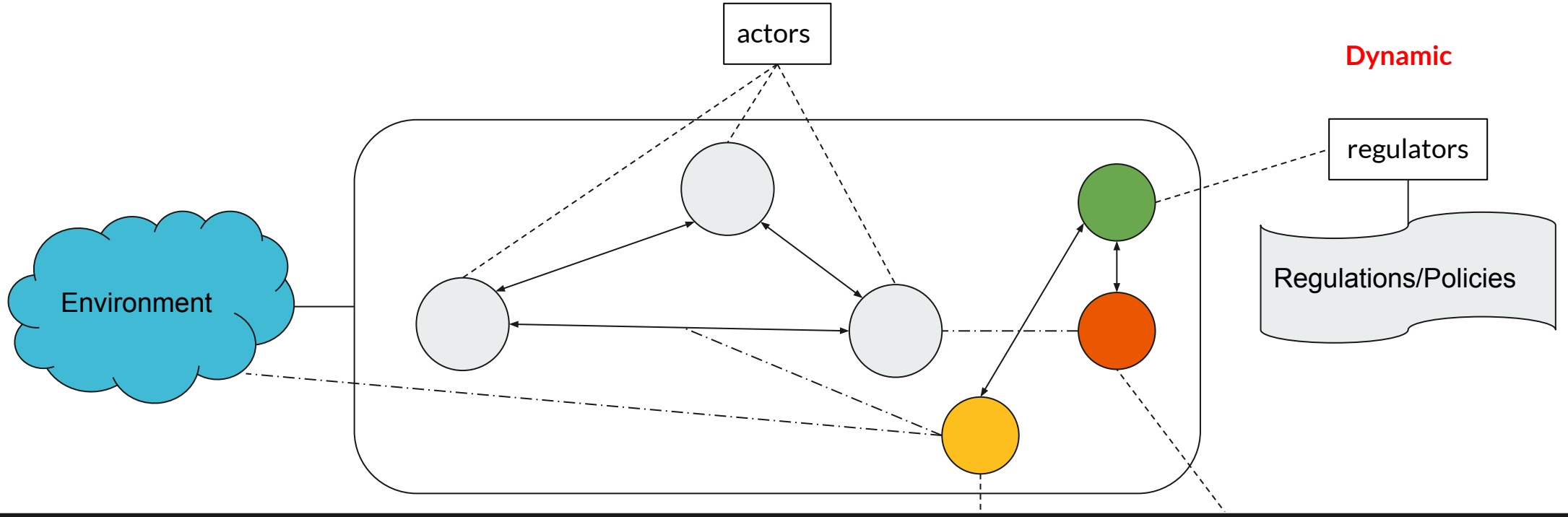
- My goals:
  - Exercising policy design as part of system design
  - Focus: Utilizing agent-based models of actors to reason about policies



Target System Attributes



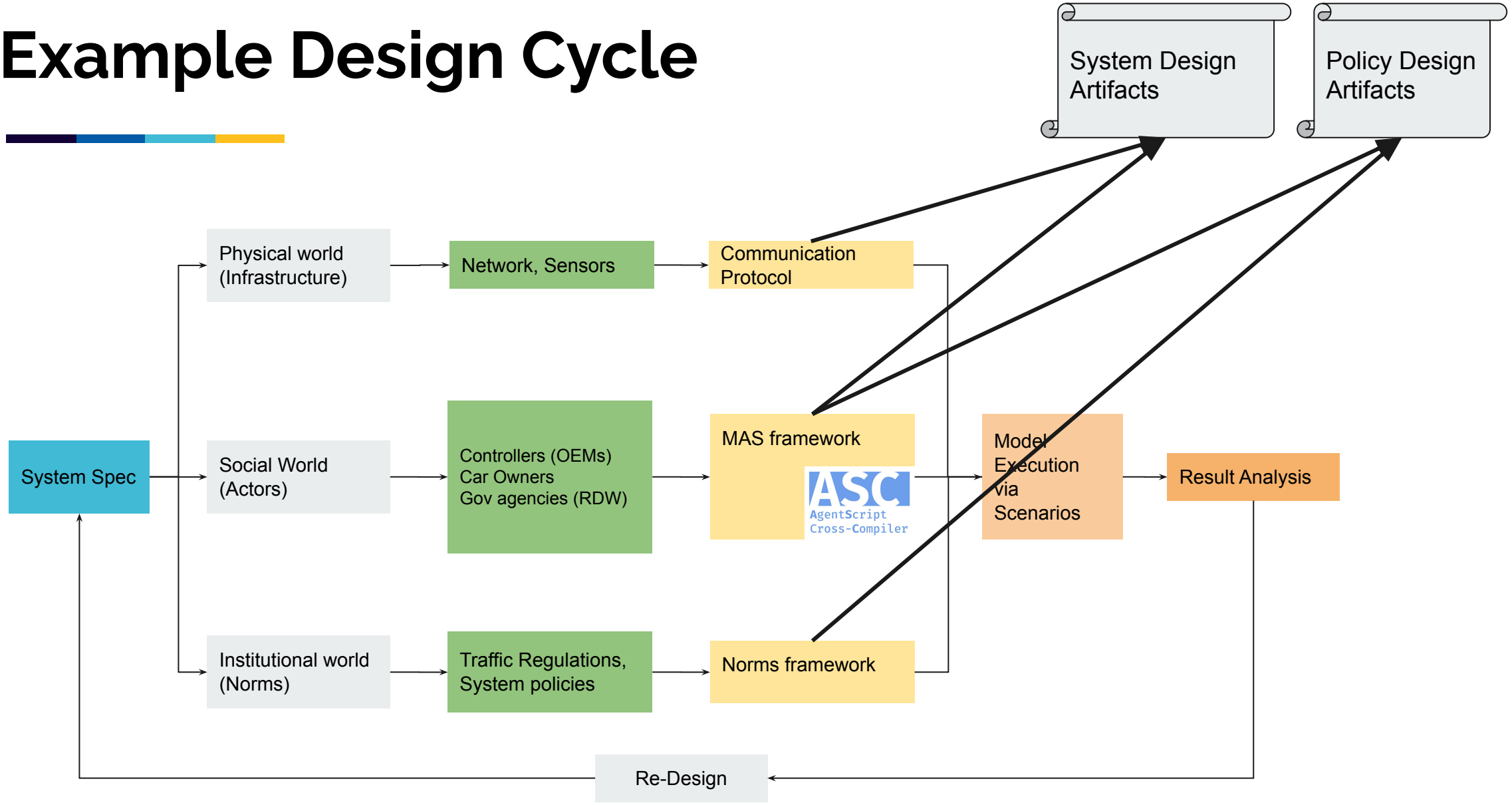
Regulated Data-intensive Socio-Technical Open System



Dynamic policy changes affect the behavior of the regulator actors which propagates to the system behavior

Regulations/Policies dictate the behavior of regulator actors which changes the behavior of the system

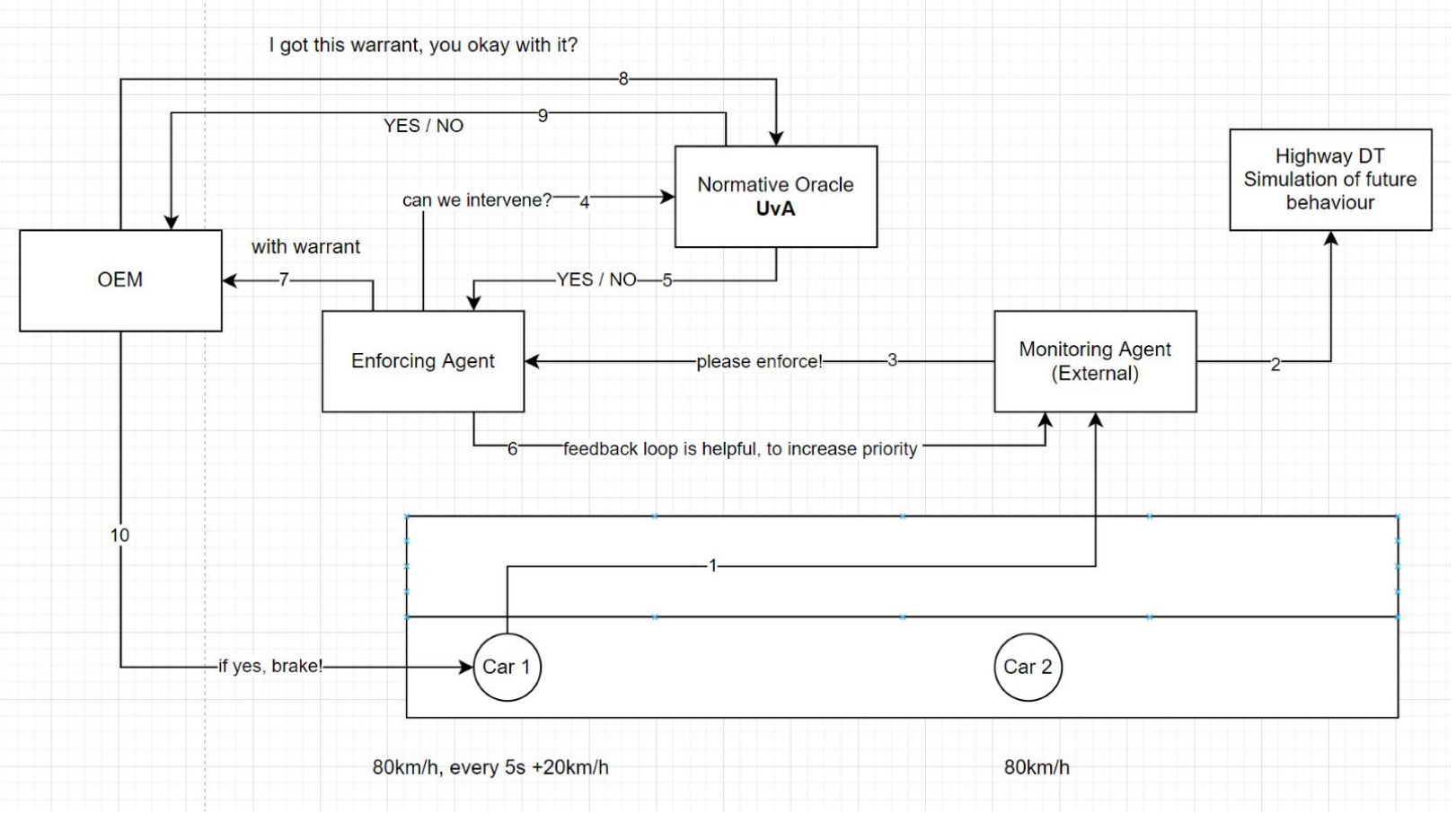
Example Design Cycle



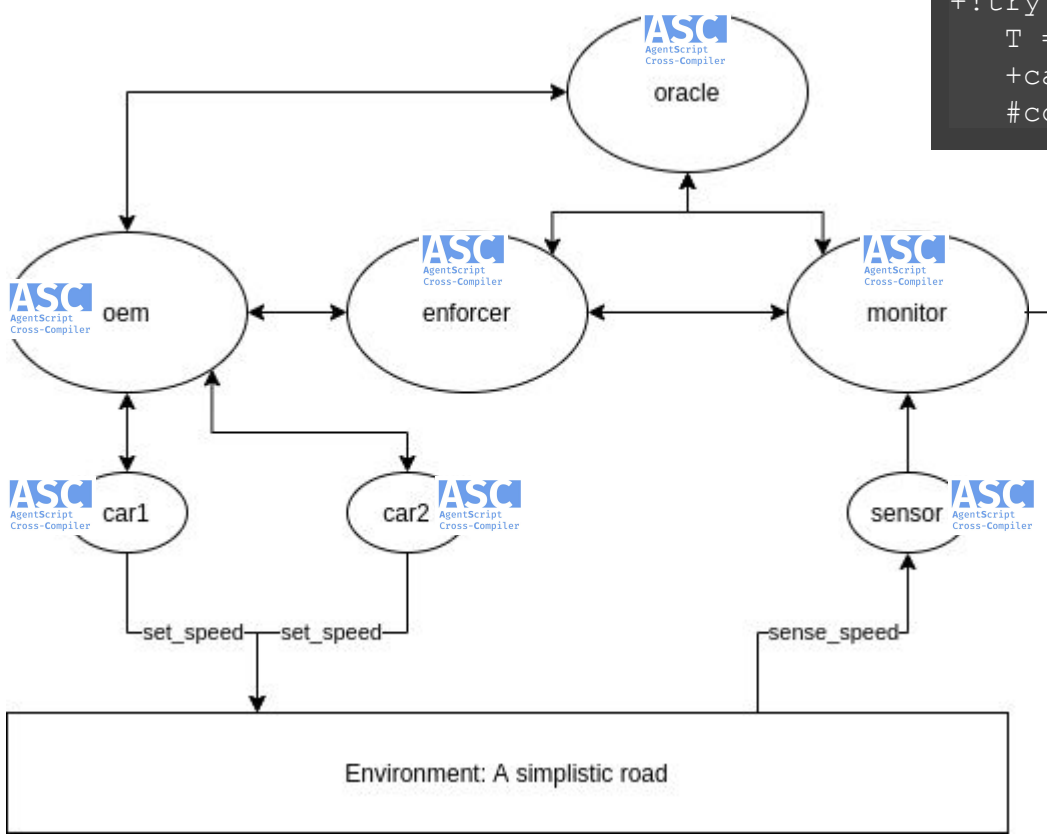
An Example Case: LICCAM

- Design: Applying the mentioned method in System/Policy design cycle
- Desired output: An executable model of the system containing:
  - Design artifacts
  - Policy artifacts
- The rest of the presentation is a recap of the experience

Initial System Spec

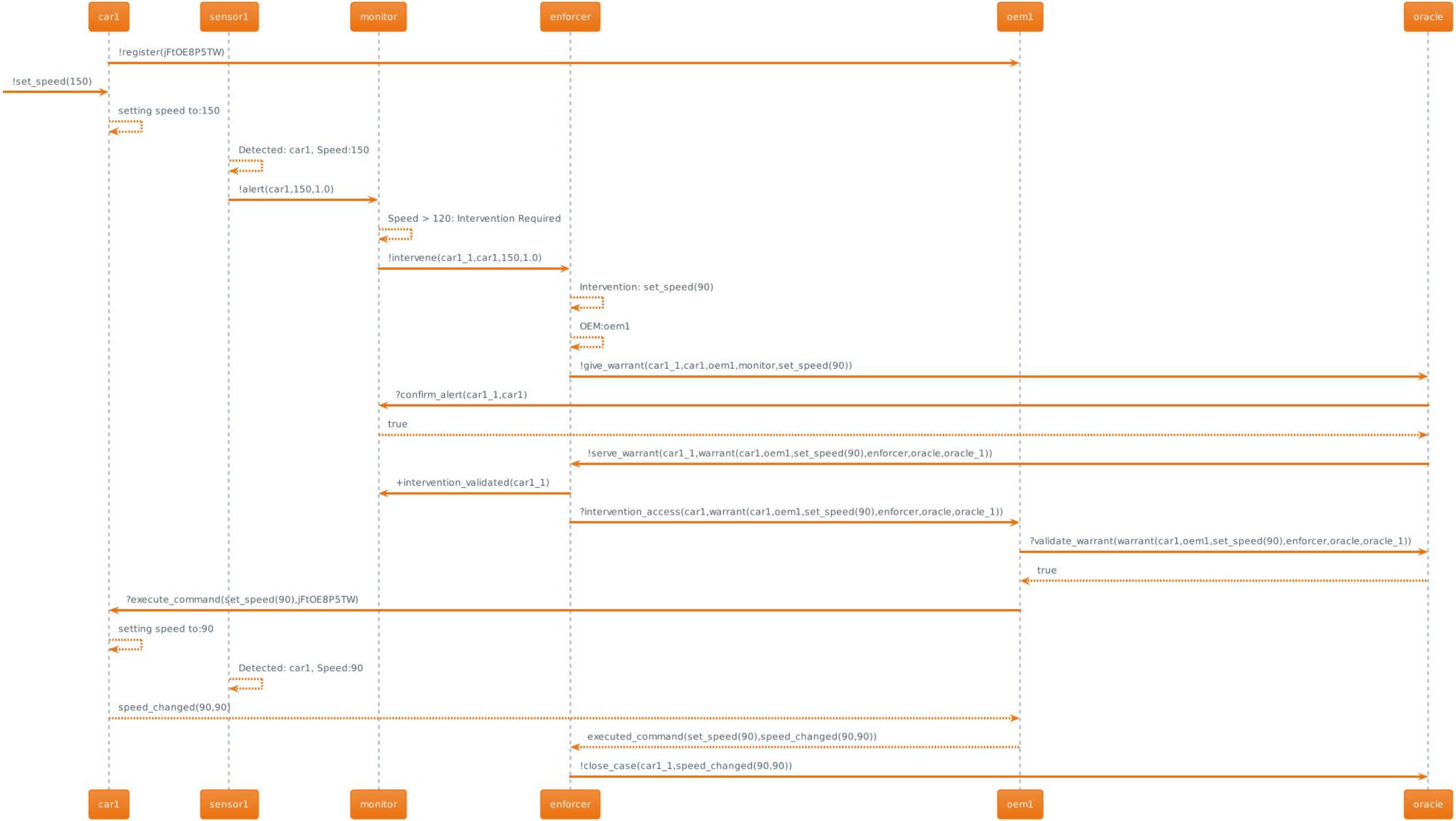


Initial ASC2 Model



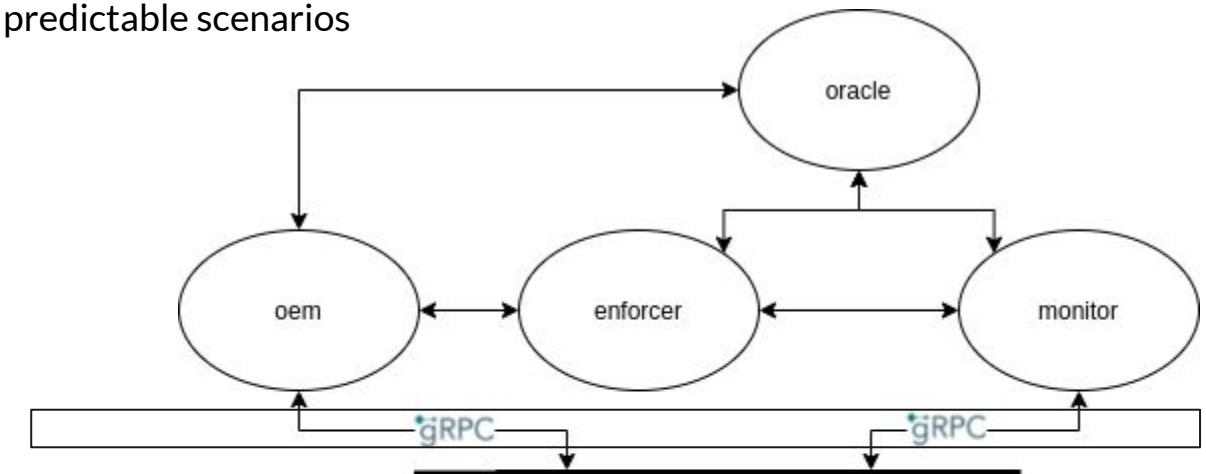

```
+!try intervention(Id,Car,Speed,Confidence) : Speed >= 120 =>
   T = #java.time.Instant.now().getEpochSecond;
   +case(Id,Car,Speed,Confidence,T);
   #coms.achieve("enforcer",intervene(Id,Car,Speed,Confidence)).
```

We can execute scenarios to verify the system


car1
sensor1
monitor
enforcer
oem1
oracle
!register(jFtOE8P5TW)
!set_speed(150)
setting speed to:150
Detected: car1, Speed:150
!alert(car1,150,1.0)
Speed > 120: Intervention Required
!intervene(car1_1,car1,150,1.0)
Intervention: set_speed(90)
OEM:oem1
!give_warrant(car1_1,car1,oem1,monitor,set_speed(90))
?confirm_alert(car1_1,car1)
true
!serve_warrant(car1_1,warrant(car1,oem1,set_speed(90),enforcer,oracle,oracle_1))
+intervention_validated(car1_1)
?intervention_access(car1,warrant(car1,oem1,set_speed(90),enforcer,oracle,oracle_1))
?validate_warrant(warrant(car1,oem1,set_speed(90),enforcer,oracle,oracle_1))
true
?execute_command(set_speed(90),jFtOE8P5TW)
setting speed to:90
Detected: car1, Speed:90
speed_changed(90,90)
executed_command(set_speed(90),speed_changed(90,90))
!close_case(car1_1,speed_changed(90,90))
car1
sensor1
monitor
enforcer
oem1
oracle

Decoupling the Environment

We had very limited and predictable scenarios



ASC2 is protocol agnostic

Traffic Simulation

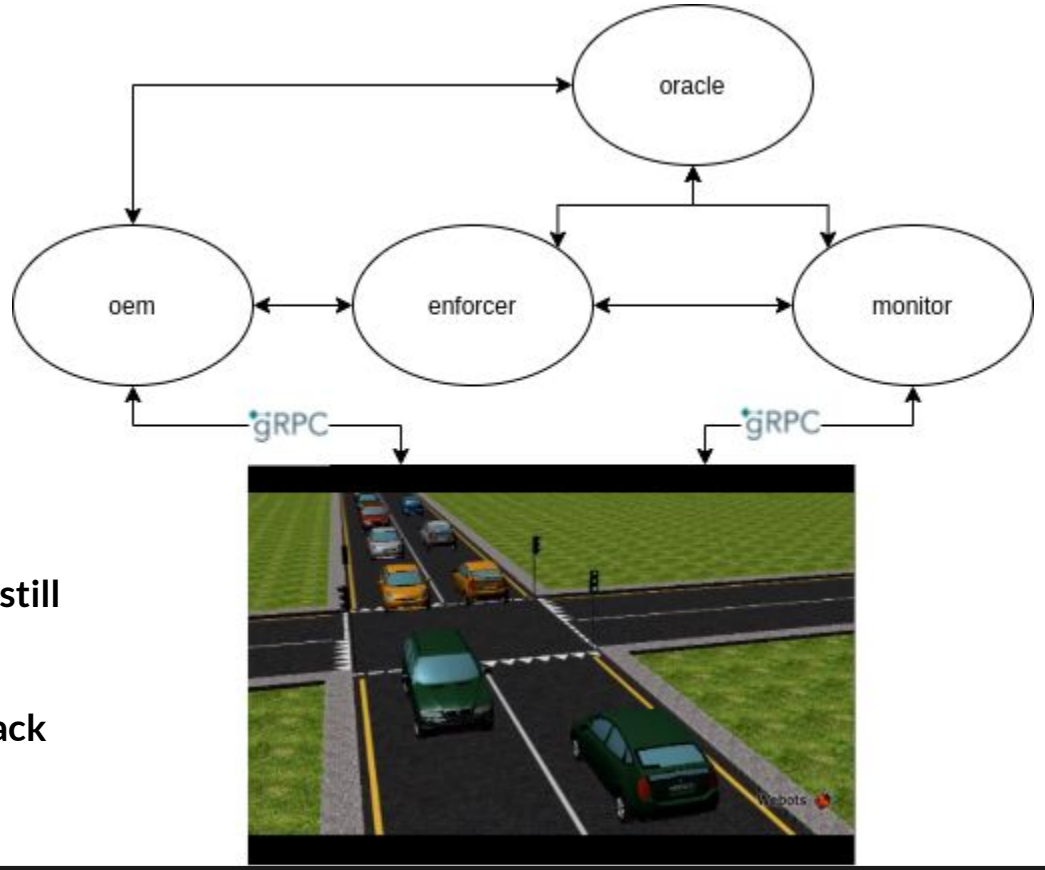
Execute less predictable scenarios

Policies vs. Control

Example: The OEM should execute an intervention within a timeframe if there is a warrant from oracle

The system as a whole should be verifiable against regulations by using execution traces

eflint



The verification happens on the model at design time where it is still feasible

Policy and System design feedback to each other

System Design Artifacts

straightforward non-functional requirement

Policy Design Artifacts

not so straightforward regulation

What is the incentive?

What are the punishments?

What is the evidence?

Policies become part of the system design

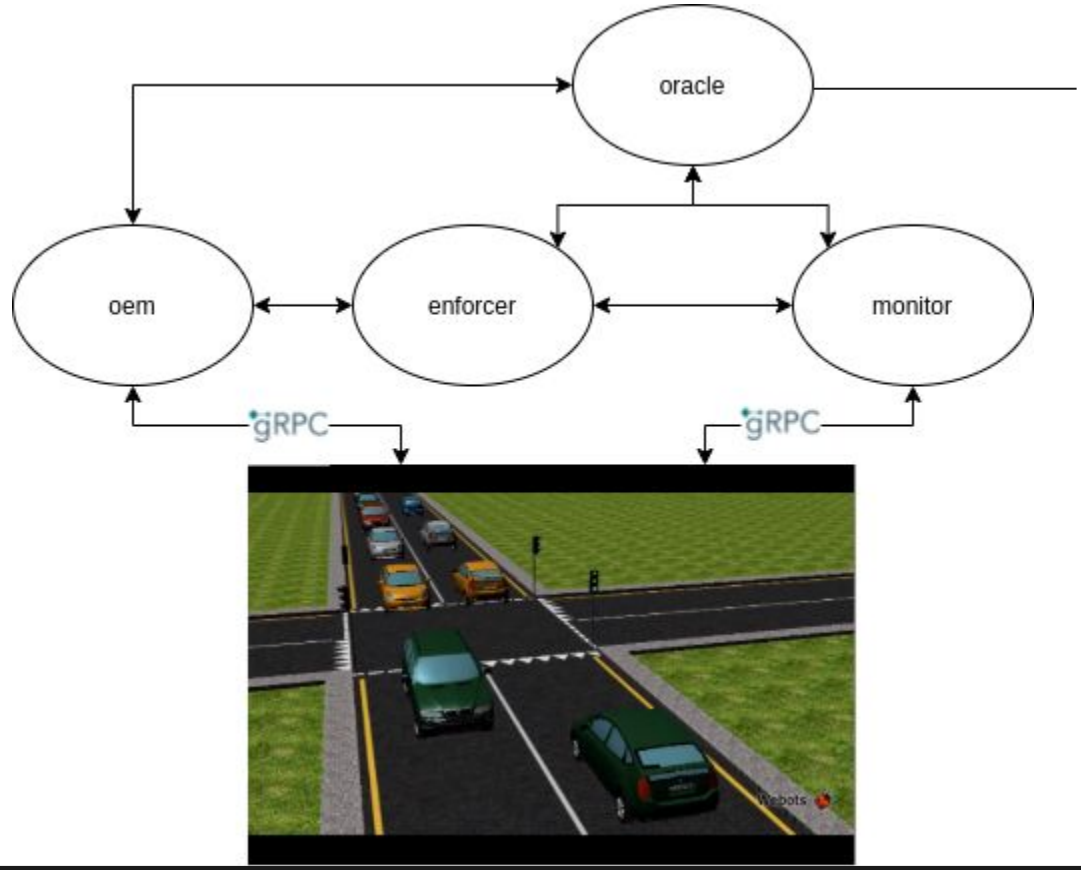
Explicit (dynamic) Policies

Example 1: In normal situations, a warrant for intervention should be issued only with intention to stop a ***HIGH RISK*** state

Example 2: In extreme situations (terrorist attack), a warrant can be issued in any intention

System Design Artifacts

Policy Design Artifacts



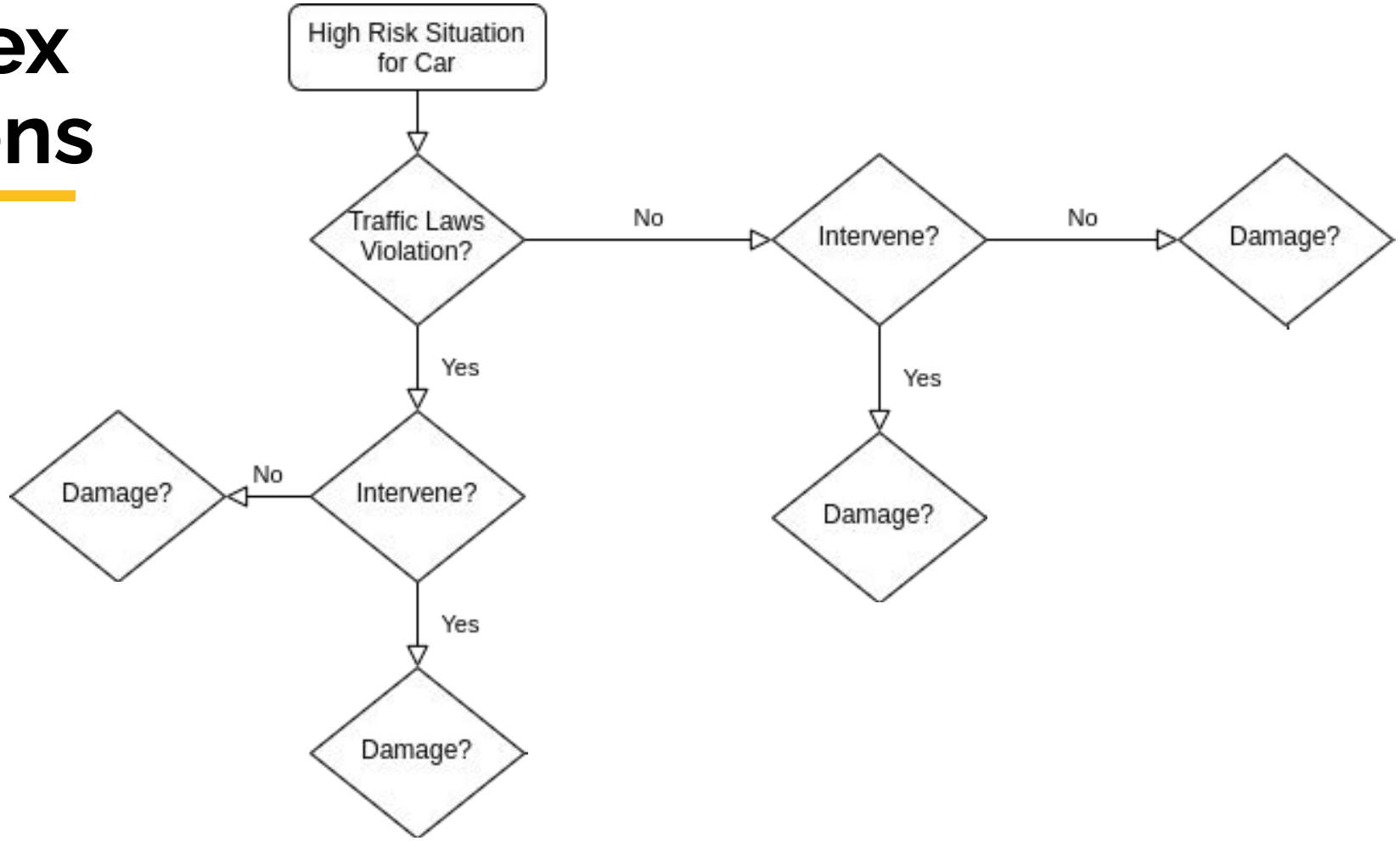
eflint

Some actors act based on explicit norms, specially actors with dynamic policies

They change the system behaviour by changing policies

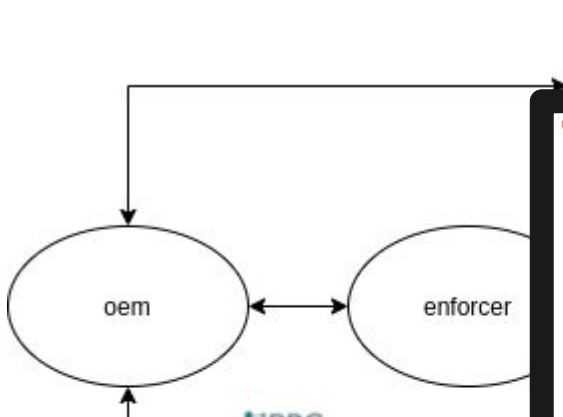
To have a formal specification of policies

Complex Decisions



Even monitoring the environment can create liability! even more in affecting it
Do we want this system?

Usability: Automated Tests




```
"An oracle" should {
  "always issue a warrant if there are proper evidence" in {
    val oem = agents("oem1")
    // Send the request to oracle
    agents("oracle").send(warrant_request("case_1","set_speed(90)","oem1")).
    // Intercept response
    val message = oem.receiveMessage()
    // Assert
    assert(message isInstanceOf[GoalMessage])
    assert(message toString startsWith "warrant(execute(set_speed(90),car1")
  }
}
```

circleci

Travis CI

Seamless Integration and Testing for MAS Engineering

jeri Parizi[1], Giovanni Sileno[1], and Tom van Engers[1]

te, University of Amsterdam, Amsterdam, the Netherlands
ajeriparizi,g.sileno,t.m.vanengers}@uva.nl

ting undeniably plays a central role in the daily practice
neering, and this explains why better and more efficient
rvices are continuously made available to developers and
d the MAS developers community similarly benefit from
f-the-art testing approaches? The paper investigates the

Run, Agent, Run! Architecture and Benchmarking of Actor-Based Agents

Mostafa Mohajeri Parizi
m.mohajeriparizi@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Giovanni Sileno
g.sileno@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Tom van Engers
vanengers@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Sander Klous
s.klous@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Abstract

The paper introduces an Agent-Oriented Programming (AOP) framework based on the Belief-Desire-Intention (BDI) model of agency. The novelty of this framework is in relying on the Actor model, instantiating each intentional agent as an

1 Introduction

Agent-based models have an intuitive mapping to behavioural descriptions, and for this reason are extensively used for modeling and simulations of social systems. However, *agent-based programming* is not only relevant for simulation. Data-

Conclusion

- Applied an ABM approach to System/Policy design cycle
- Policy and System design should be done together
  - They are affected by each other
  - They feedback to each other
    - e.g, need for evidence requires adding monitoring
- Just like software tests, compliance verification can not be an afterthought
  - More challenging to test and verify
  - Much more challenging to fix

Policy-Driven System Design

Mostafa Mohajeri
University of Amsterdam

CCI Meeting
Feb, 2022

Thank You! :)

- Questions?