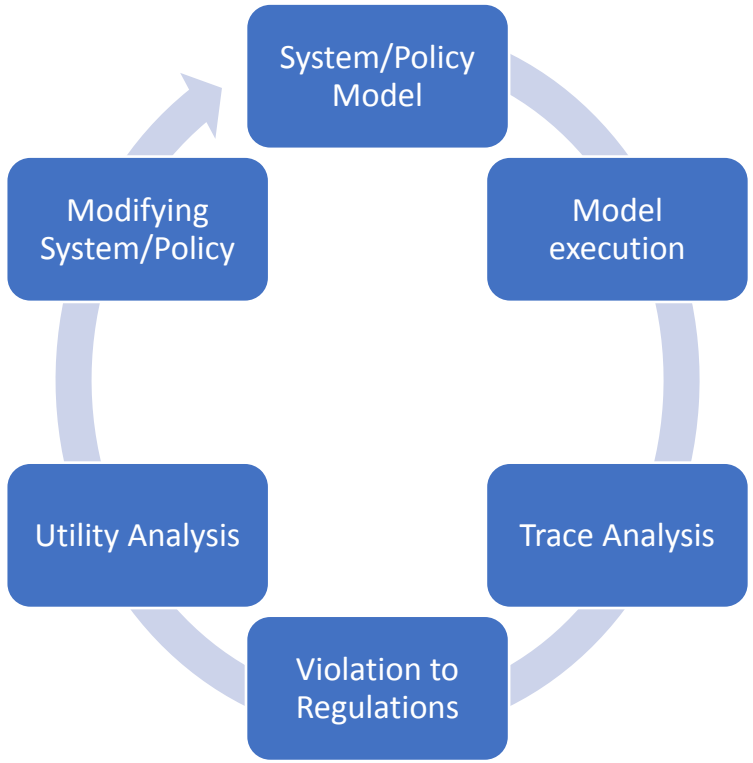
Policy-Driven System Design

Mostafa Mohajeri
University of Amsterdam

DL4LD consortium meeting
Nov 25, 2021

My PhD Research

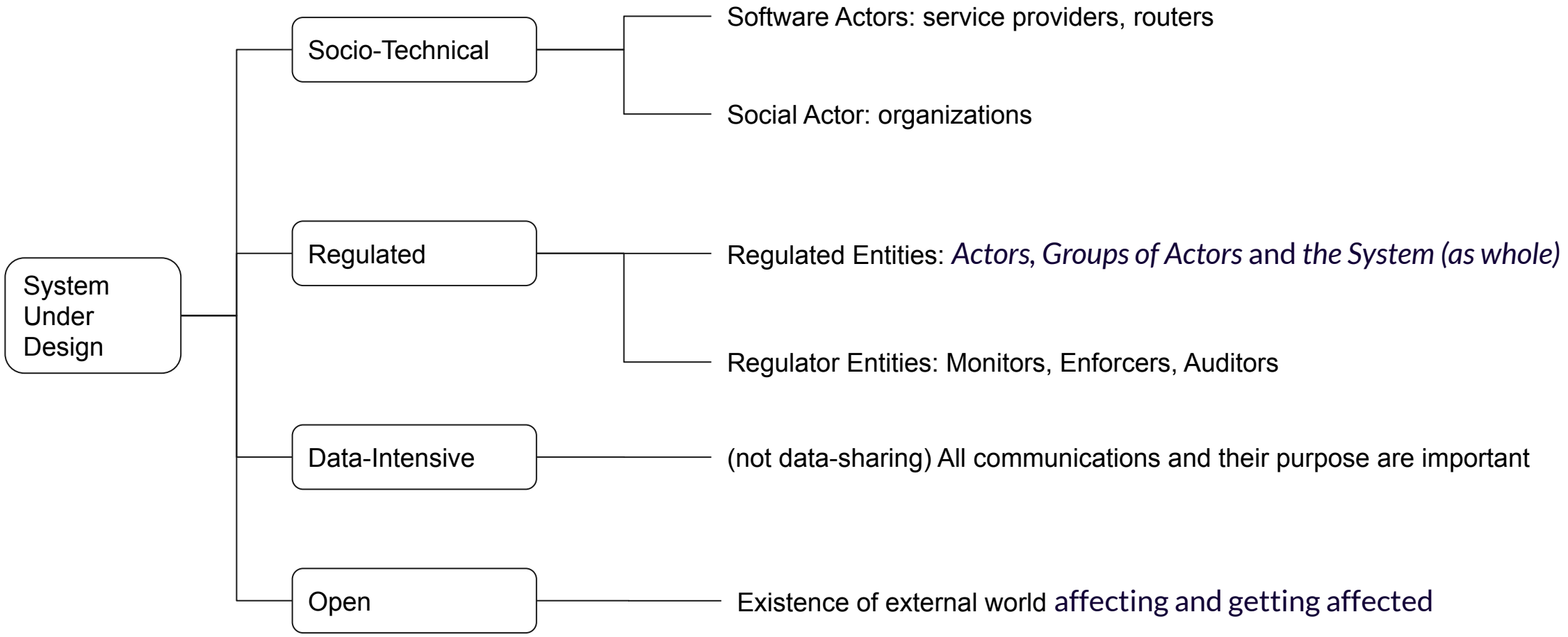
- Design and develop a computational environment for **policy-making**
- Exercising policy design as part of system design
- Focus: Utilizing agent-based models of actors



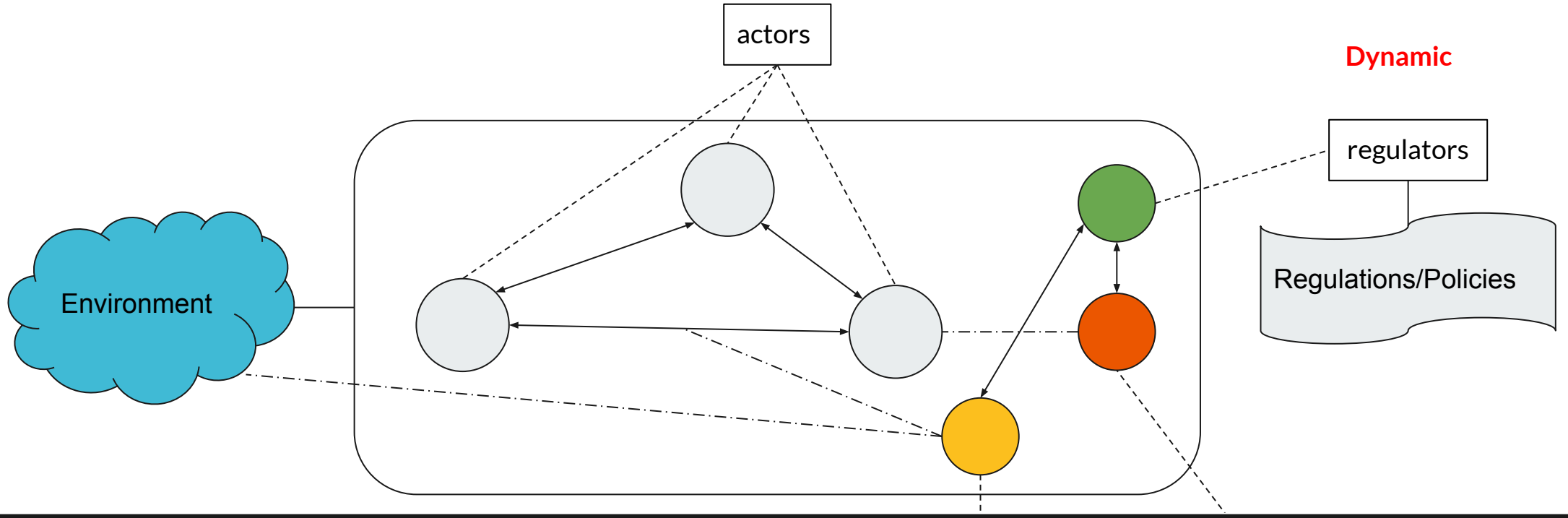
Executable model of Agents

- AgentScript Cross-Compiler framework (ASC2)
- A multi-agent system (MAS) framework to model the actors
  - An agent-oriented programming language
  - A compiler for the language
  - A run-time for the agents

Target System Attributes



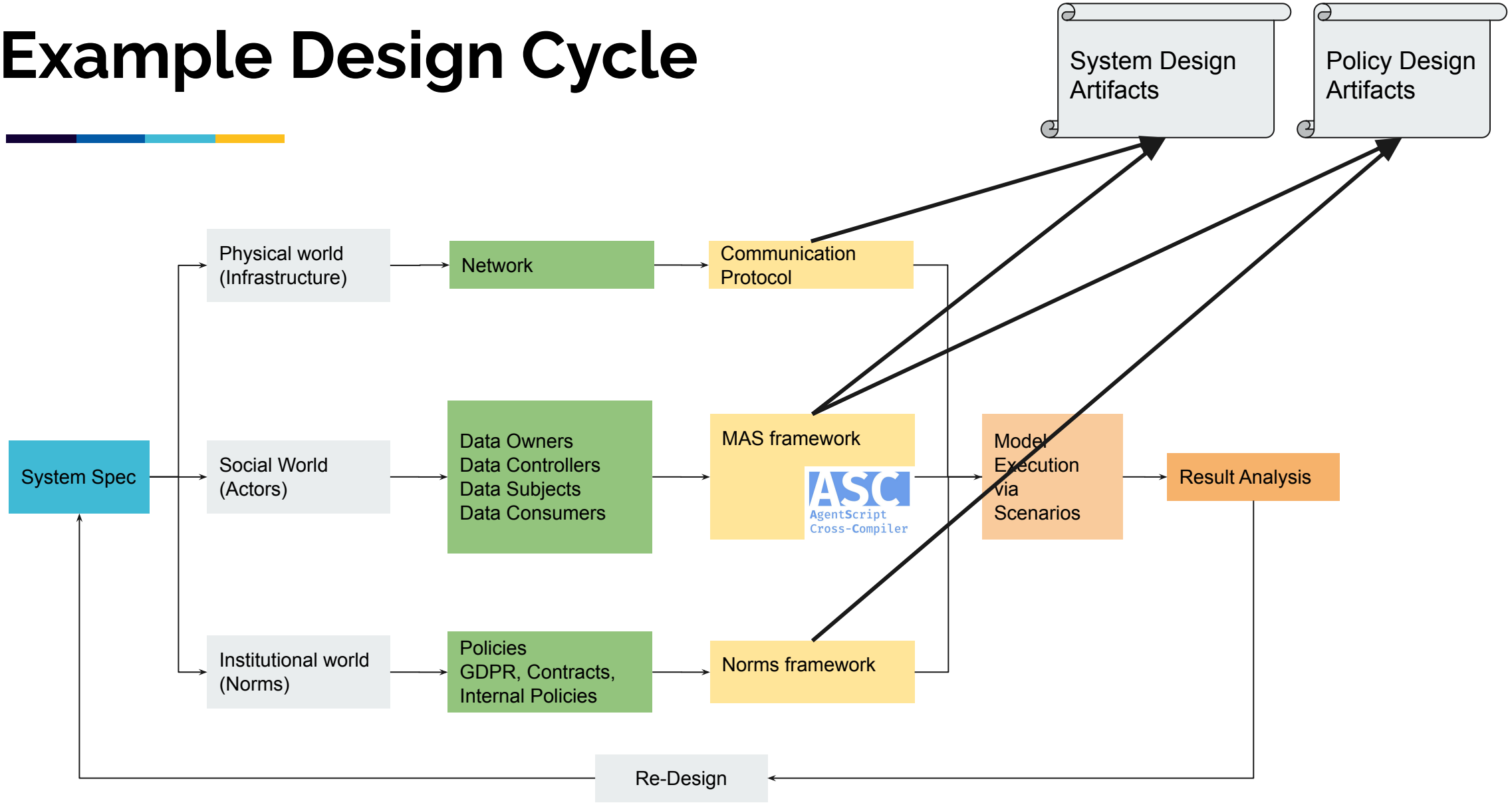
Regulated Data-intensive Socio-Technical Open System



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

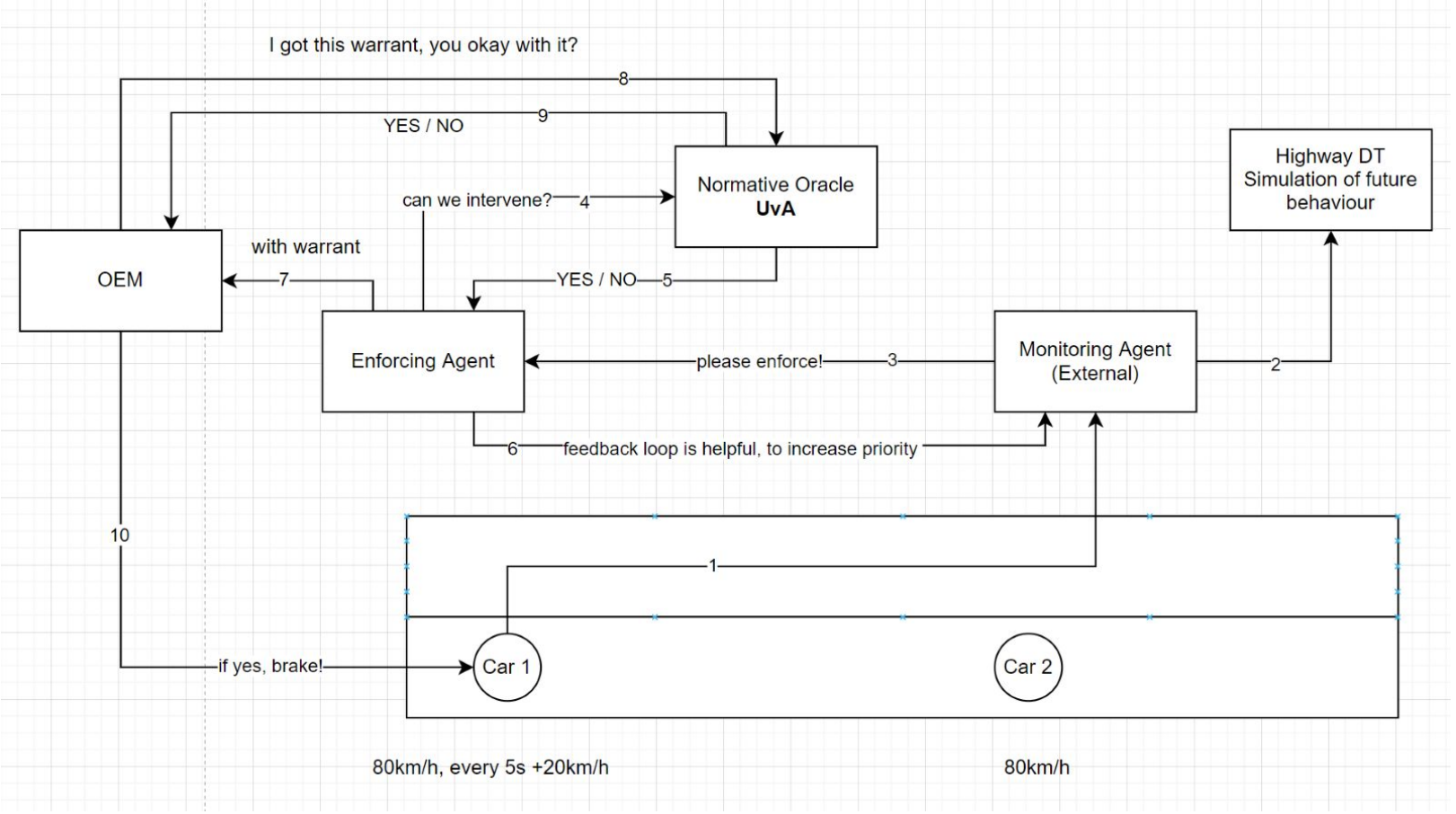
Example Design Cycle



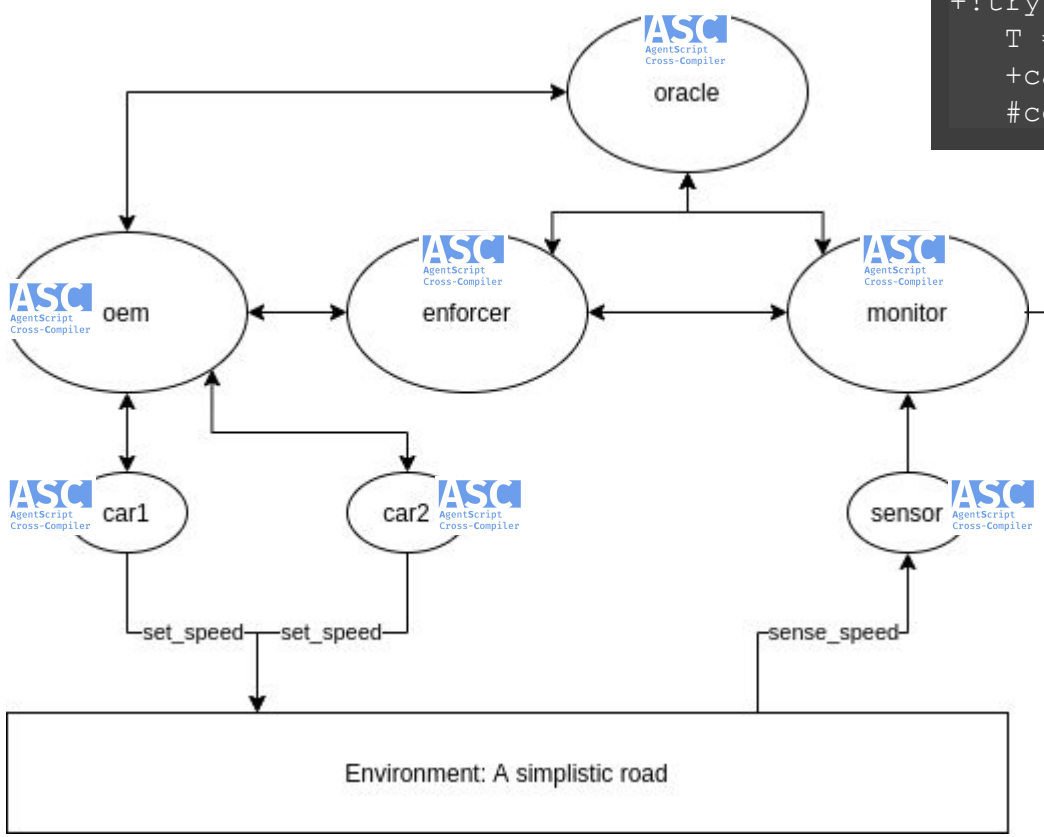
An Example Case: LICCAM

- **L**egal **I**nterventions for **C**onnected and **C**ooperative **A**utomated **M**obility
- Design: Applying the mentioned method in System/Policy design cycle
- The output:
  - Design artifacts
  - Policy artifacts
  - ***Do we want this system?***

Initial System Spec

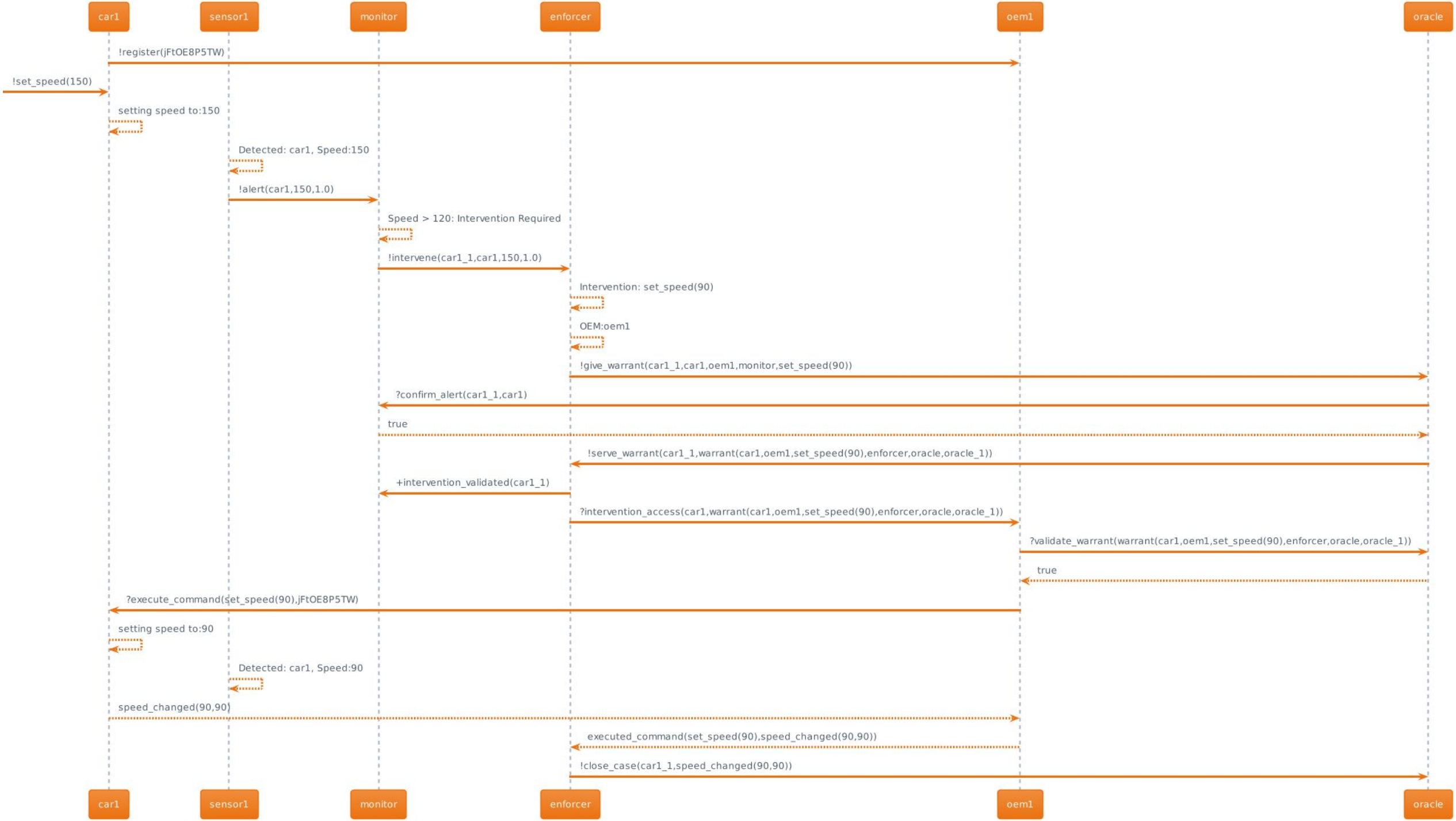


Initial ASC2 Model



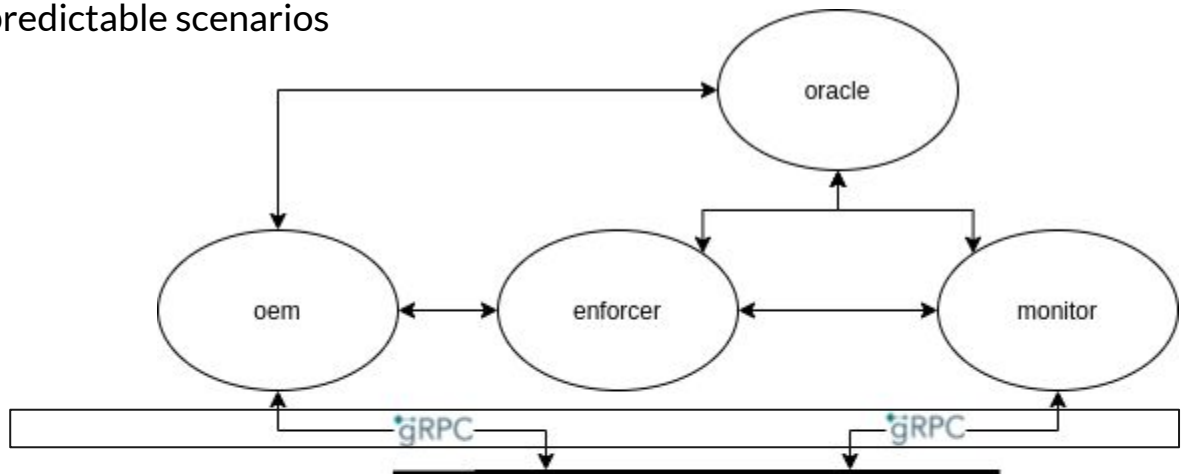

```
+!try intervention(Id,Car,Speed,Confidence) : Speed >= 120 =>
   T = #java.time.Instant.now().getEpochSecond;
   +case(Id,Car,Speed,Confidence,T);
   #coms.achieve("enforcer",intervene(Id,Car,Speed,Confidence)).
```

We can execute scenarios to generate design artifacts and 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 (WEBots, SUMO)

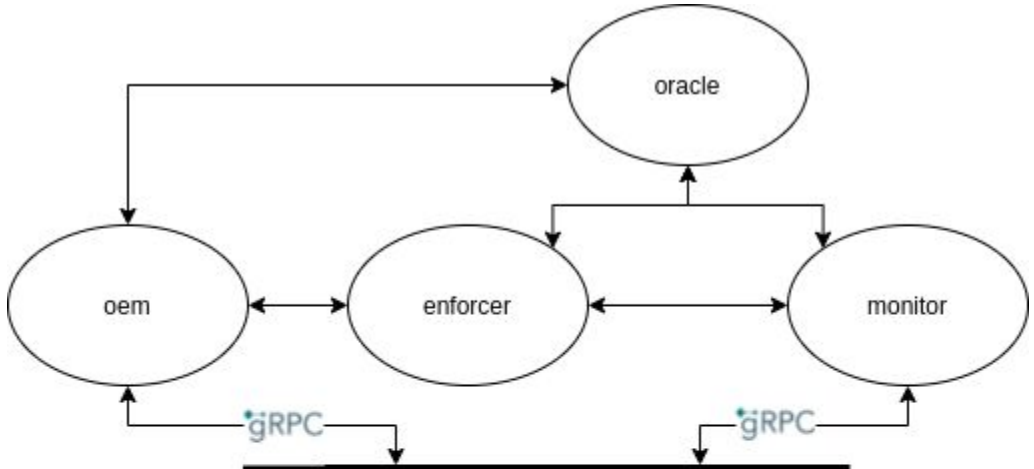
Execute less predictable scenarios

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



SUMO

System Design Artifacts

straightforward non-functional requirement

Policy Design Artifacts

not so straightforward regulation

What is the incentive?

What are the punishments?

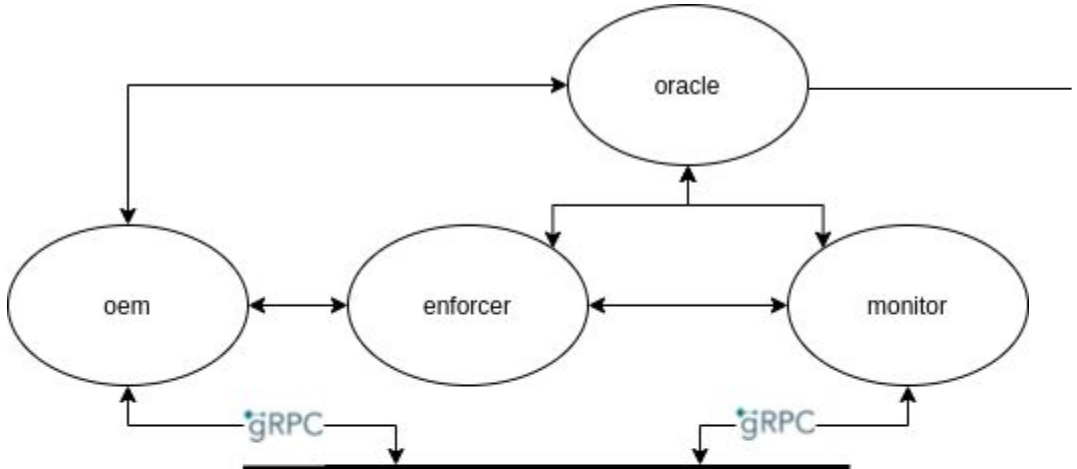
Are violations observable?
Need extra monitoring?

Policies become part of the system design

Explicit (dynamic) Policies

Example 1: In normal situations, a warrant for intervention should be generated only if there is an impending ***HIGH RISK*** state

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



eflint

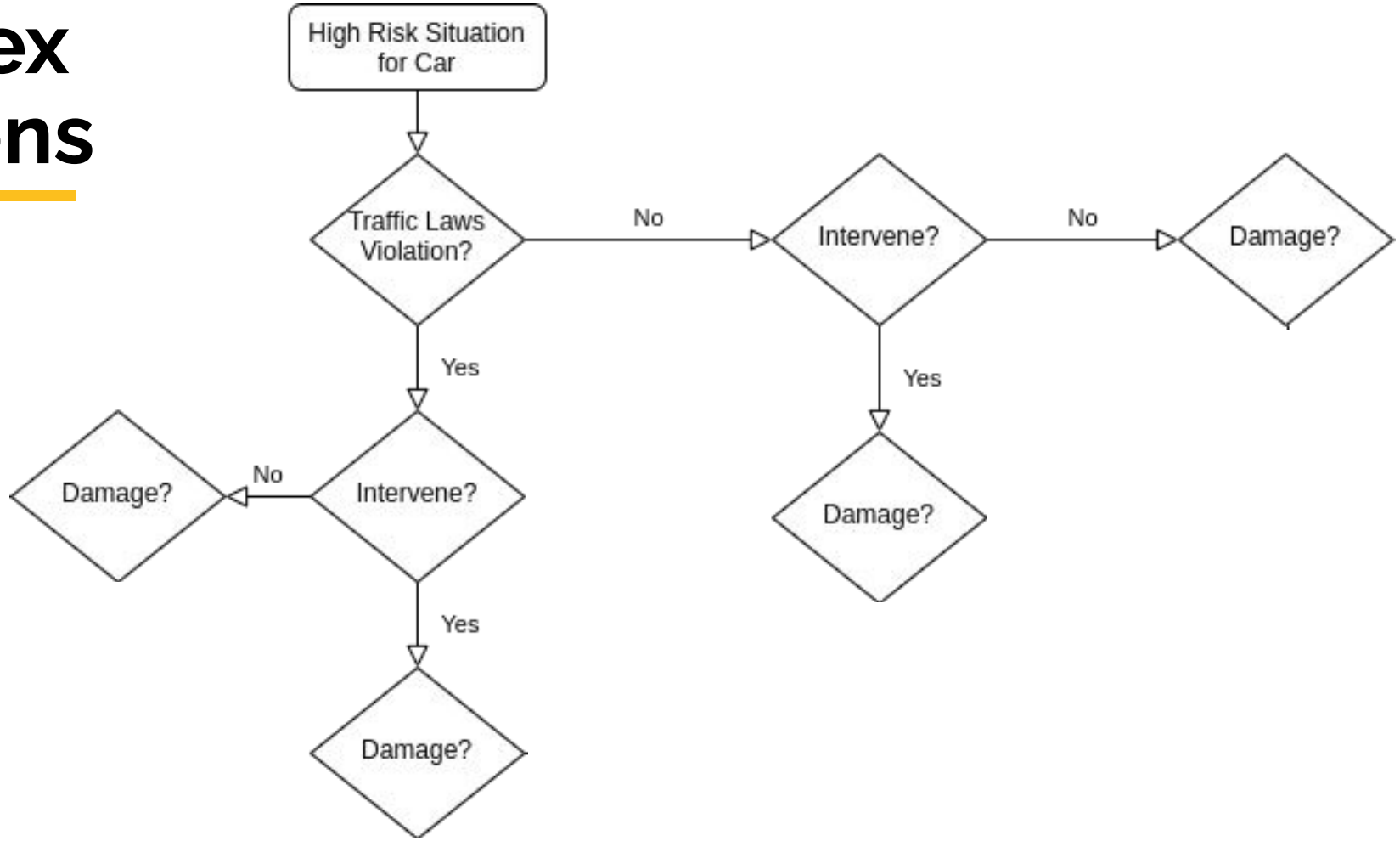
Some actors act based on explicit norms, specially the ones with dynamic policies

System Design Artifacts

Policy Design Artifacts

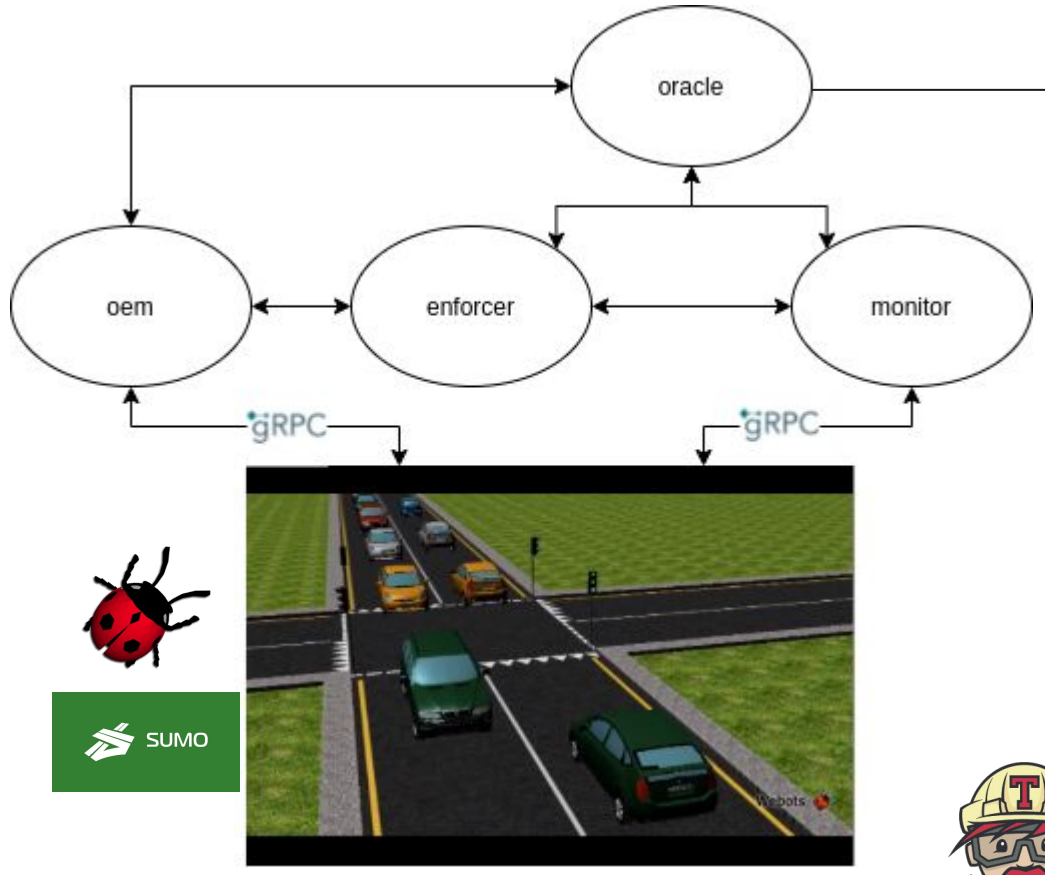
To have a formal description of policies

Complex Decisions



Do we want this system?

Usability: Automated Tests



Seamless Integration and Testing for MAS Engineering

Mostafa Mohajeri Parizi[1], Giovanni Sileno[1], and Tom van Engers[1]

nformatics Institute, University of Amsterdam, Amsterdam, the Netherlands
{m.mohajeriparizi,g.sileno,t.m.vanengers}@uva.nl

Abstract. Testing undeniably plays a central role in the daily practice of software engineering, and this explains why better and more efficient libraries and services are continuously made available to developers and designers. Could the MAS developers community similarly benefit from utilizing state-of-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

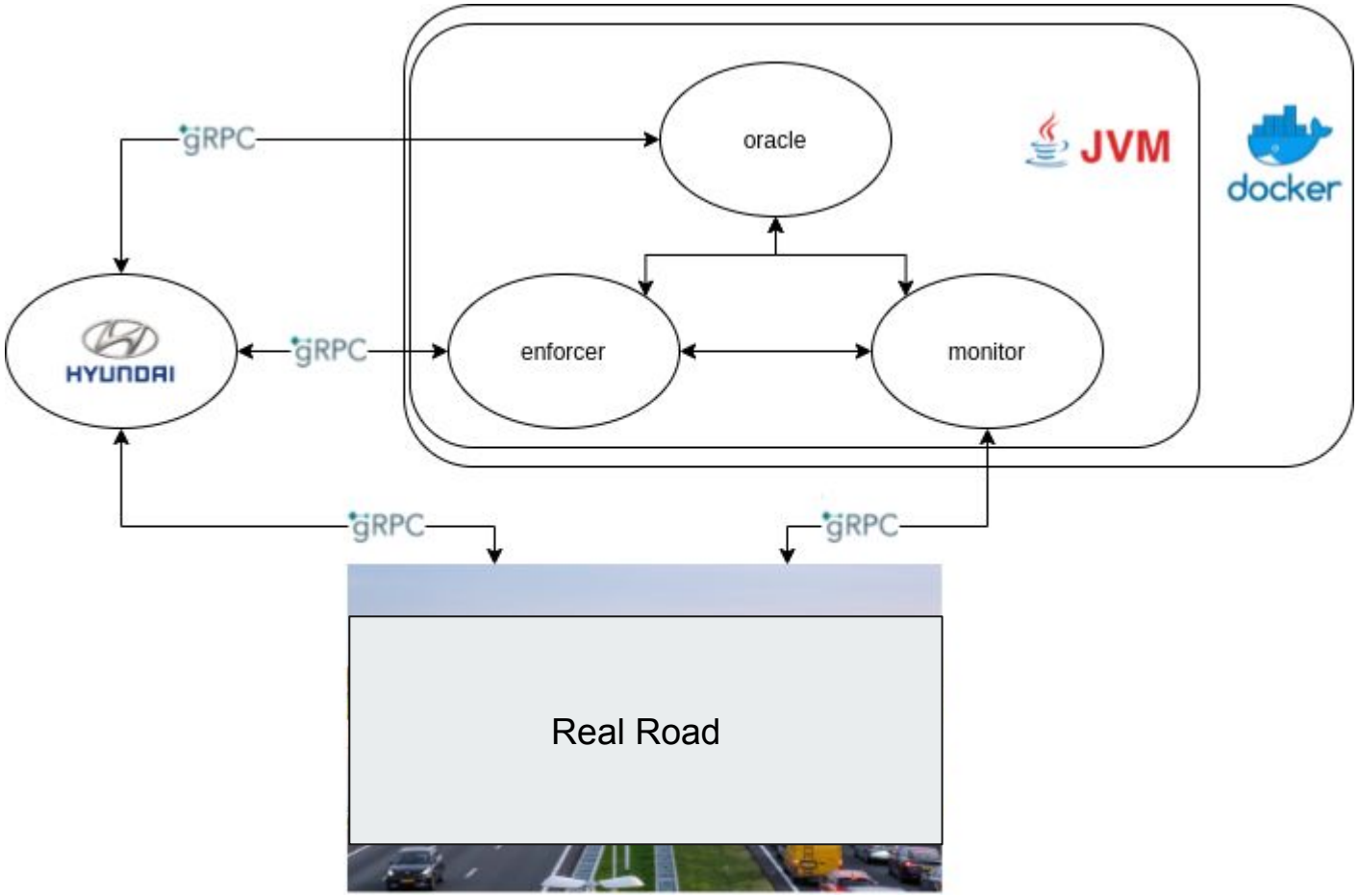
Abstract

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-

Prototyping in real system



Policy-Driven System Design

Mostafa Mohajeri
University of Amsterdam

DL4LD consortium meeting
Nov 25, 2021

Thank You! :)

- Questions?