Scenarios@run.time – Distributed Execution of Specifications on IoT-Connected Robots

10th International Workshop on Models@run.time at MODELS 2015, Ottawa, Canada

Joel Greenyer, Daniel Gritzner, Timo Gutjahr, Tim Duente, Stefan Dulle, Falk-David Deppe, Nils Glade, Marius Hilbich, Florian Koenig, Jannis Luennemann, Nils Prenner, Kevin Raetz, Thilo Schnelle, Martin Singer, Nicolas Tempelmeier, Raphael Voges

SOFTWARE ENGINEERING

29 September 2015

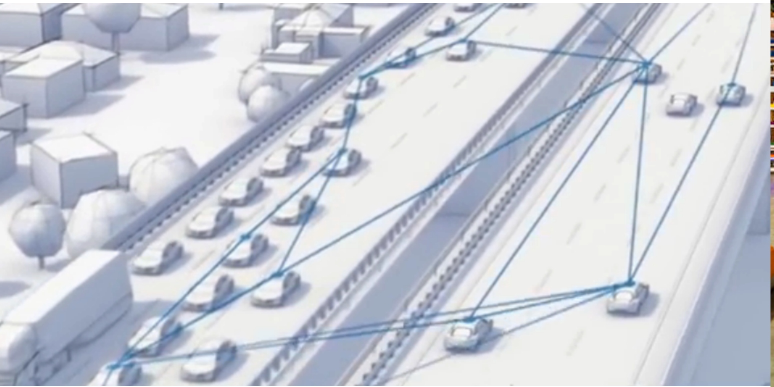
Leibniz Universität Hannover

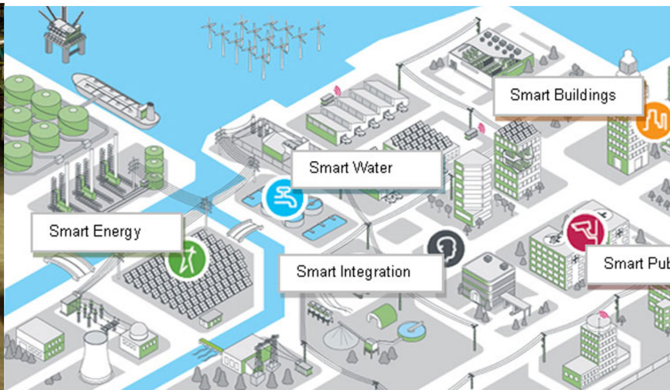
Student Project UbiBots 2015

Student Project Website: http://ubibots2015.scenariotools.org/
Youtube Video: http://youtu.be/g0hcGSYC2Wk
ScenarioTools Website: http://scenariotools.org

Motivation

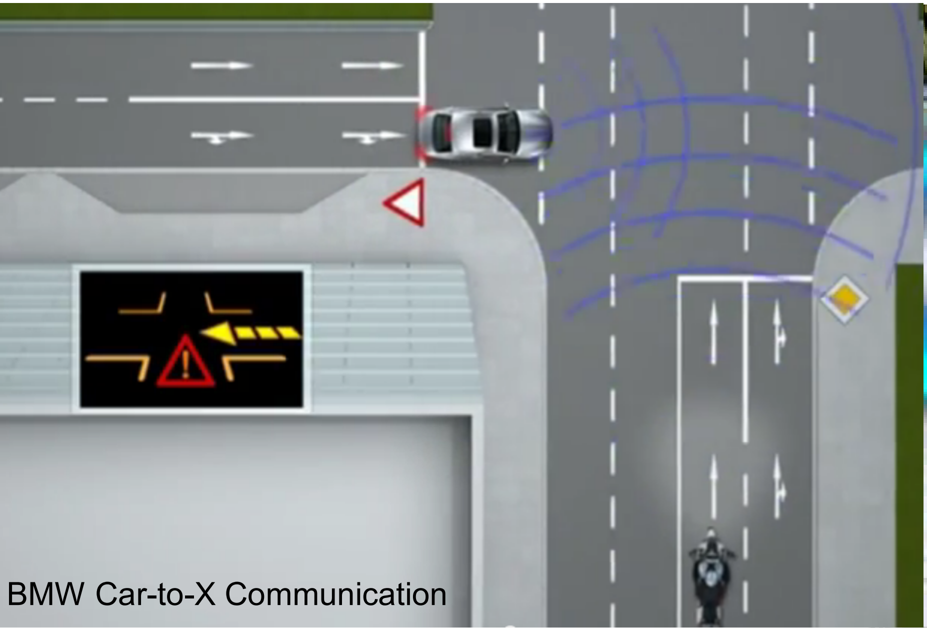
- **Examples**: CarToX, Intelligent Factories, Smart Cities, …
  - **reactive**: software continuously reacts to environment events
  - **cyber-physical**: multiple software components communicate to control processes in the physical world
  - **ubiquitous**: software interacts with users in diverse ways
  - **safety-critical**: failures can cause damage or cost lives
  - **dynamic structures**:
    - **relationships** between objects **change** (real and virtual)
    - **relationships affect** the communication behavior and **vice versa**





Example: An Advanced CarToX Driver-Assistance System

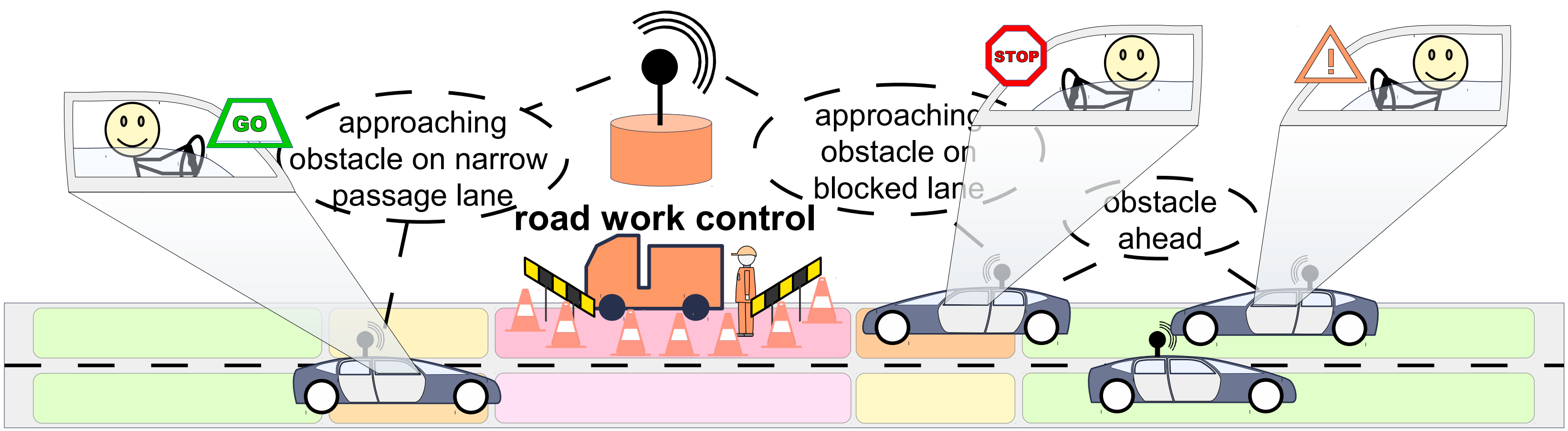
- Car-to-Car / Car-to-Infrastructure (**Car-to-X**) communication
  - provides advanced driver-assistance features
  - controls traffic more efficiently
- Examples:


BMW Car-to-X Communication


https://www.car-2-car.org/

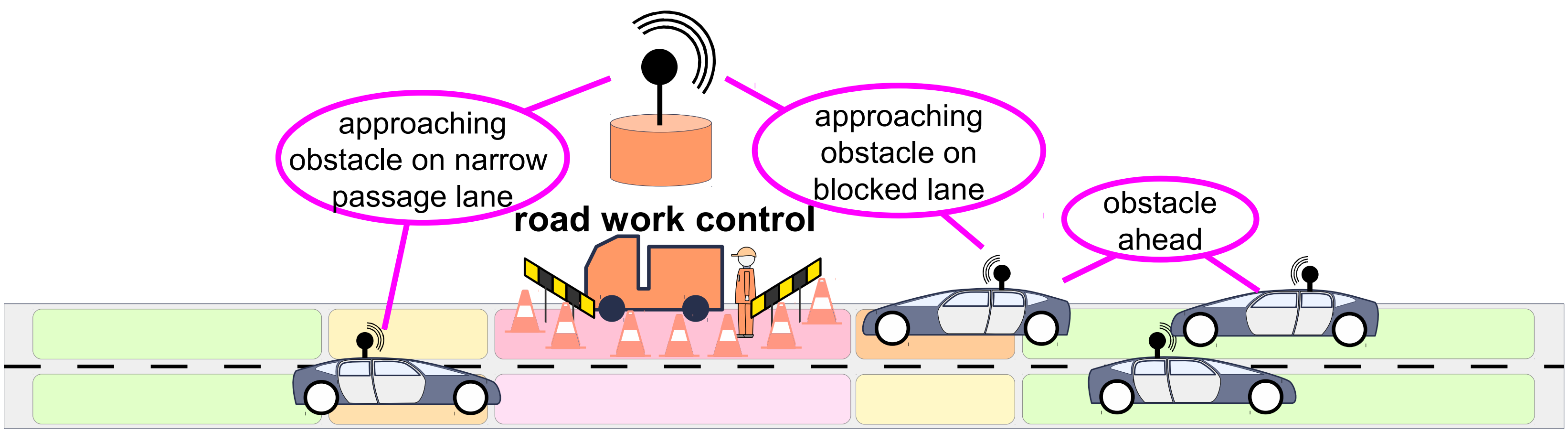
Example CarToX Use Case: coordinated passage of a road work site

- One lane of a two-lane street is blocked by road works
- cars communicate with a control station for a safe passage
  - instead of using traffic lights
  - an on-board display shows drivers whether they are allowed to enter the narrow passage or not



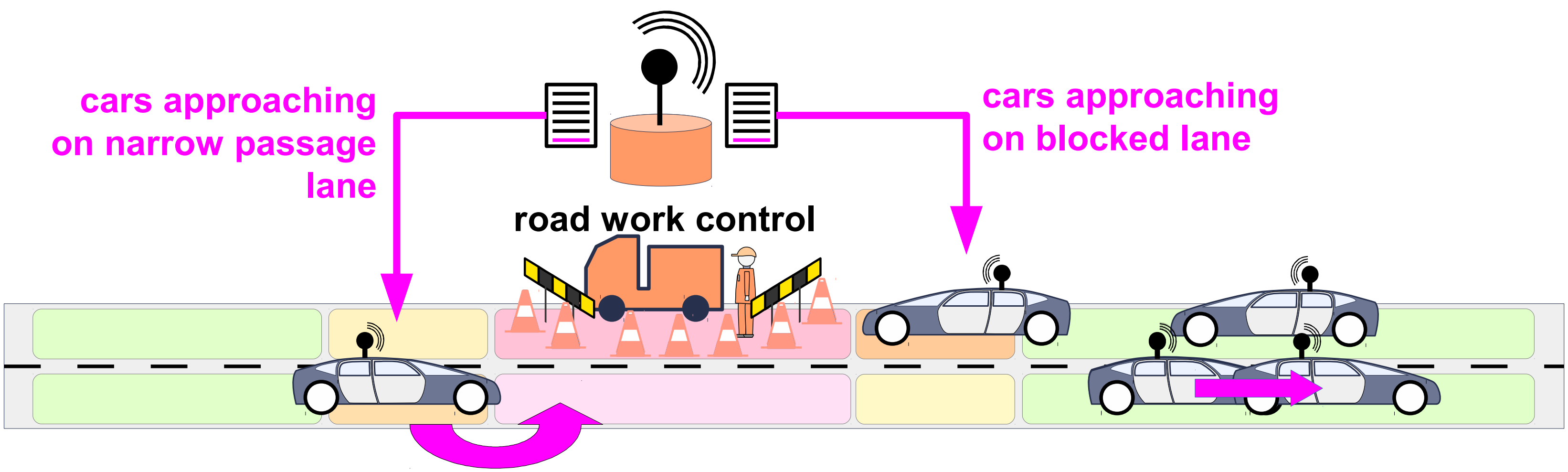
Example CarToX Use Case: coordinated passage of a road work site

- What kinds of dynamism do we see here?
  - **Message-based communication** of cars and control station



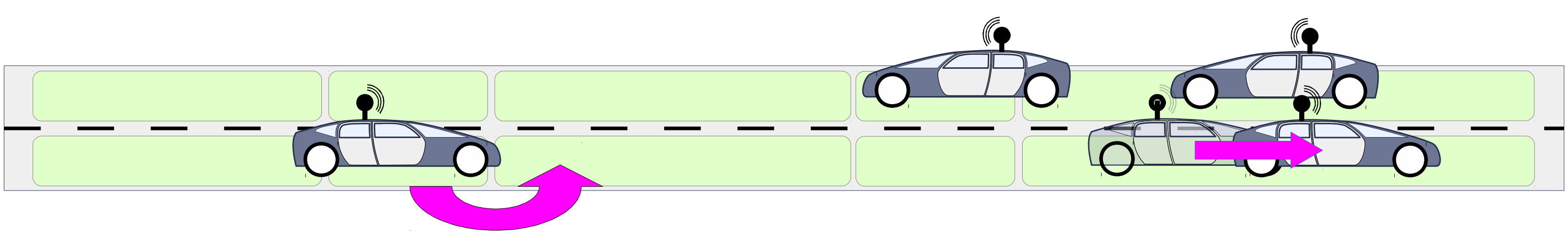
Example CarToX Use Case: coordinated passage of a road work site

- What kinds of dynamism do we see here?
  - **Message-based communication** of cars and control station
  - Structural dynamism:
    - **Physical**: cars move along different sections of the road
    - **Physical**: cars change their relative position relationships
    - **Virtual**: the control station registers approaching cars



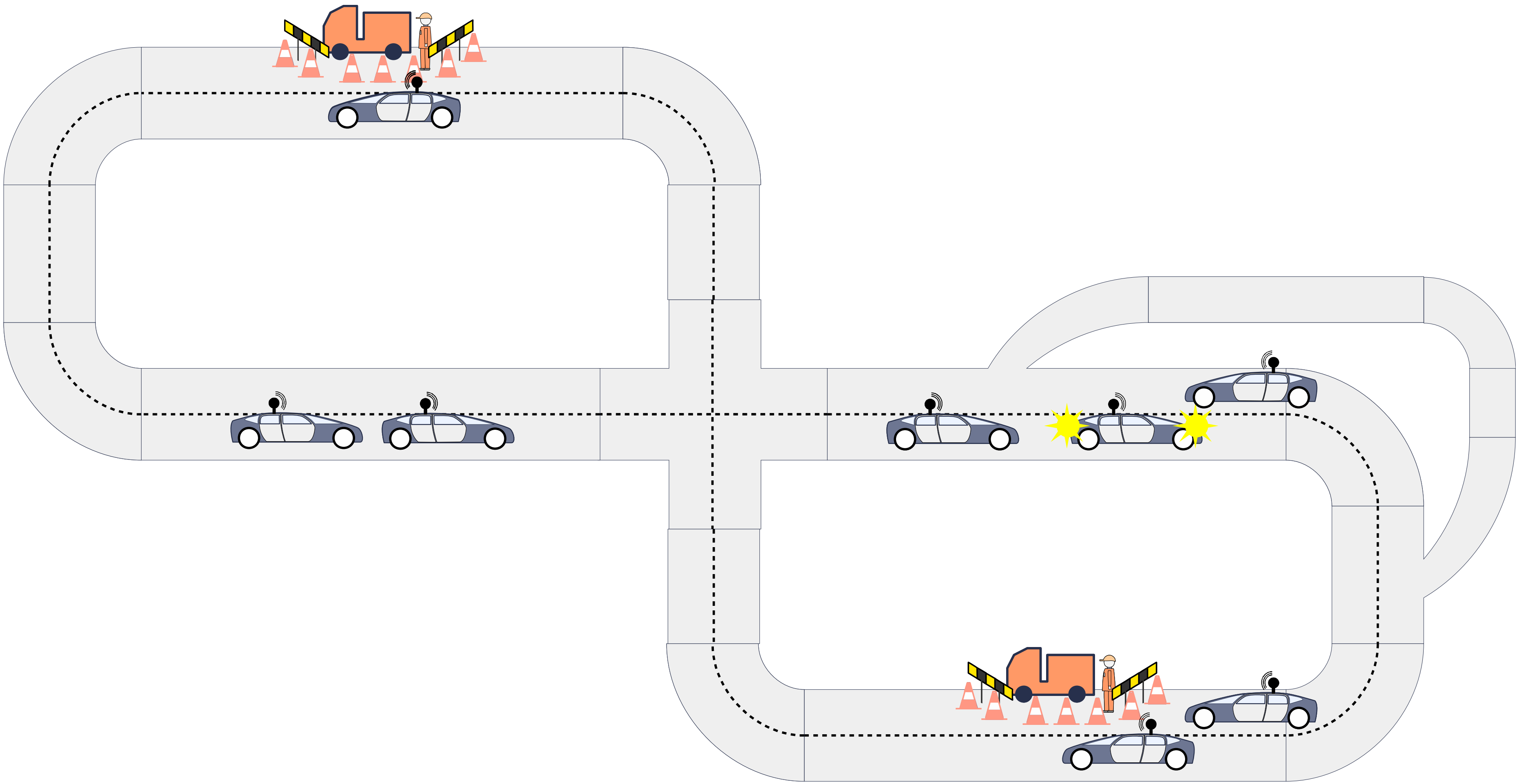
Example CarToX Use Case: coordinated passage of a road work site

- What kinds of dynamism do we see here?
  - **Message-based communication** of cars and control station
  - Structural dynamism:
    - **Physical**: cars move along different sections of the road
    - **Physical**: cars change their relative position relationships
    - **Virtual**: the control station registers approaching cars
    - **Physical**: even road works may appear and disappear



CarToX-System

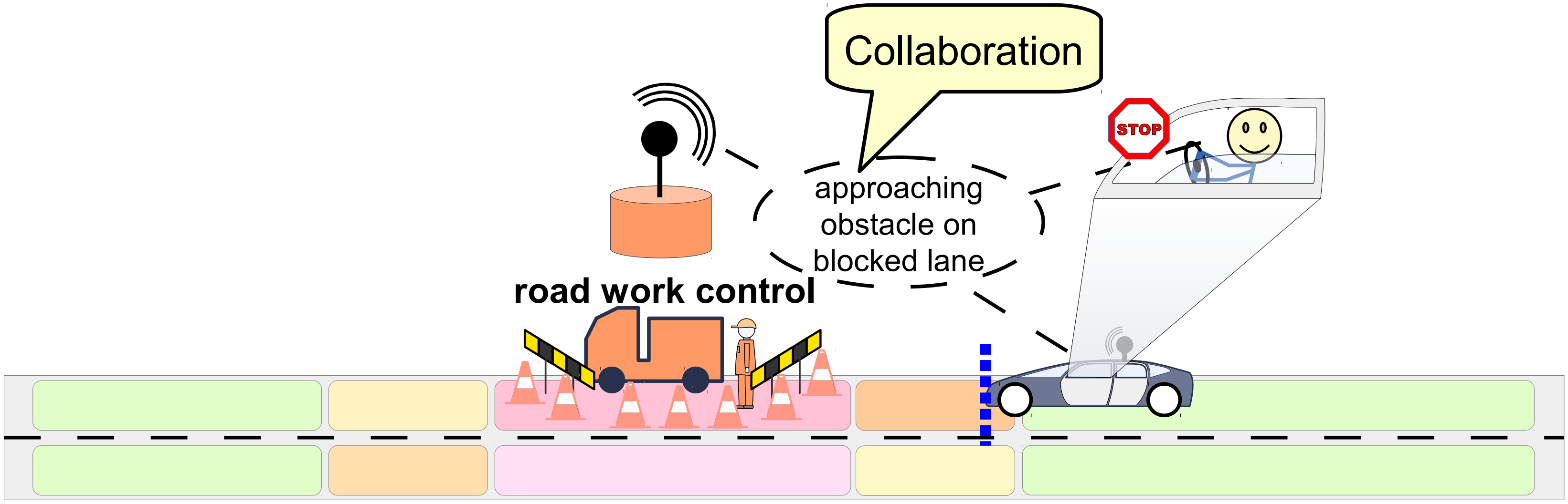
- **Question**: How would you approach the design of the software for such a system?



Collaboration: “approaching obstacle on blocked lane”

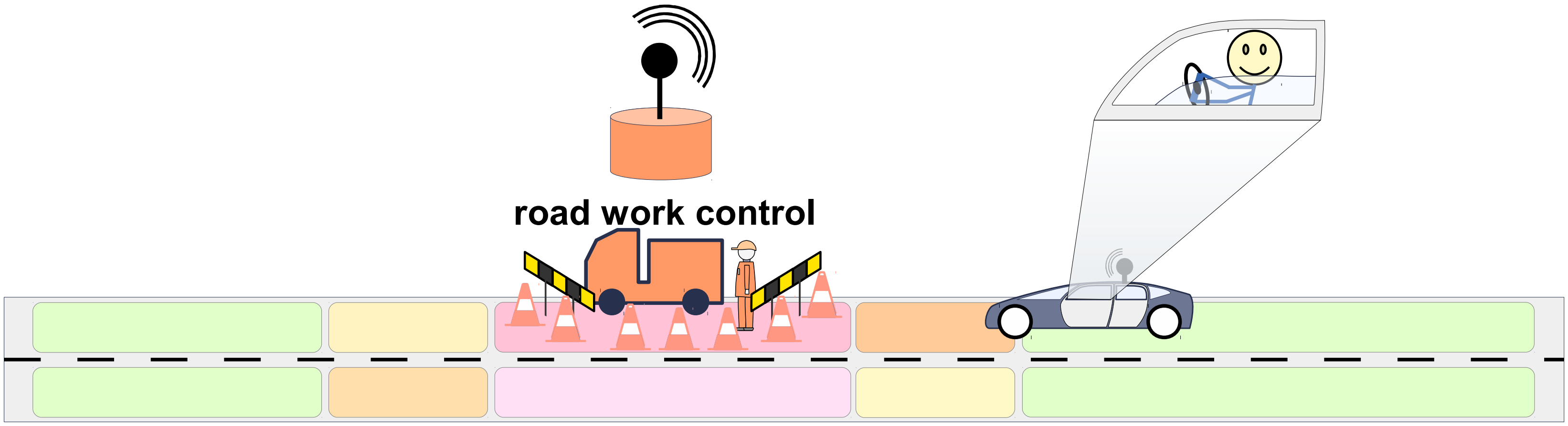
- Identify the **different situations** in which system and environment objects interact to fulfill a certain functionality
  - We call them **Use Cases** or ***Collaborations***

- Describe what the objects **may**, **must**, and **must not** do in the form of **scenarios**



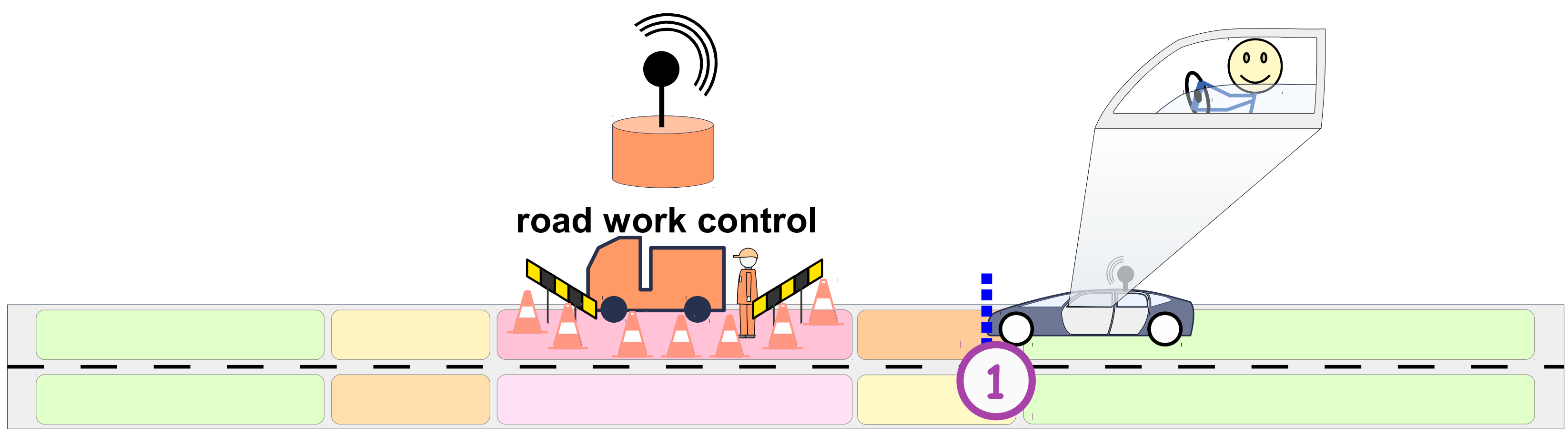
Collaboration: "approaching obstacle on blocked lane"

- Scenario "Dashboard Of Car Approaching On Blocked Lane Shows Stop Or Go":



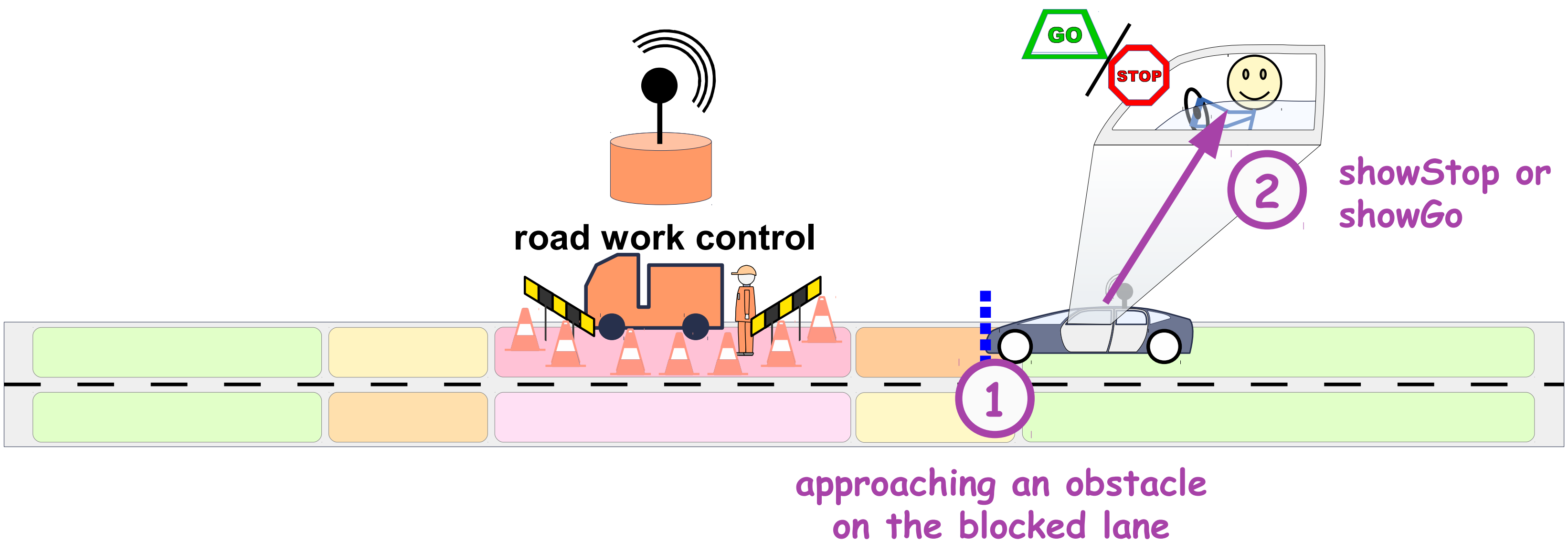
Collaboration: “approaching obstacle on blocked lane”

- Scenario “Dashboard Of Car Approaching On Blocked Lane Shows Stop Or Go”:
  1) When approaching an obstacle on the blocked lane



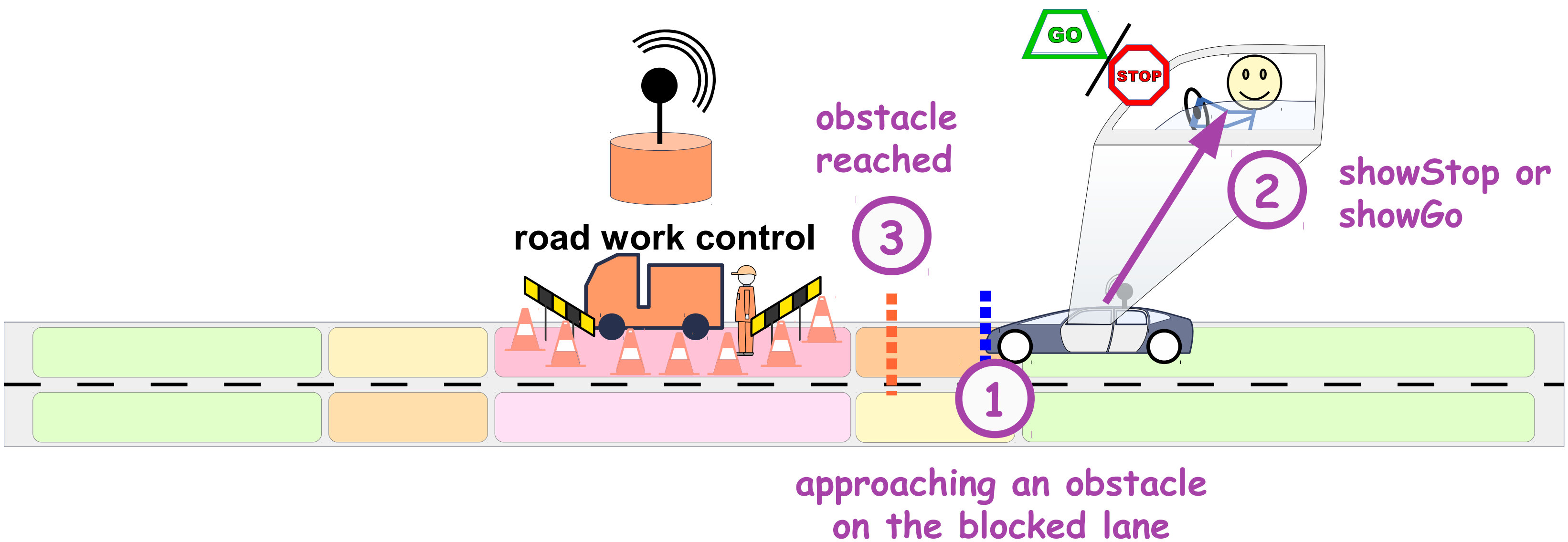
Collaboration: "approaching obstacle on blocked lane"

- Scenario "Dashboard Of Car Approaching On Blocked Lane Shows Stop Or Go":
  1) When approaching an obstacle on the blocked lane
  2) Then the dashboard must indicate to STOP or to GO



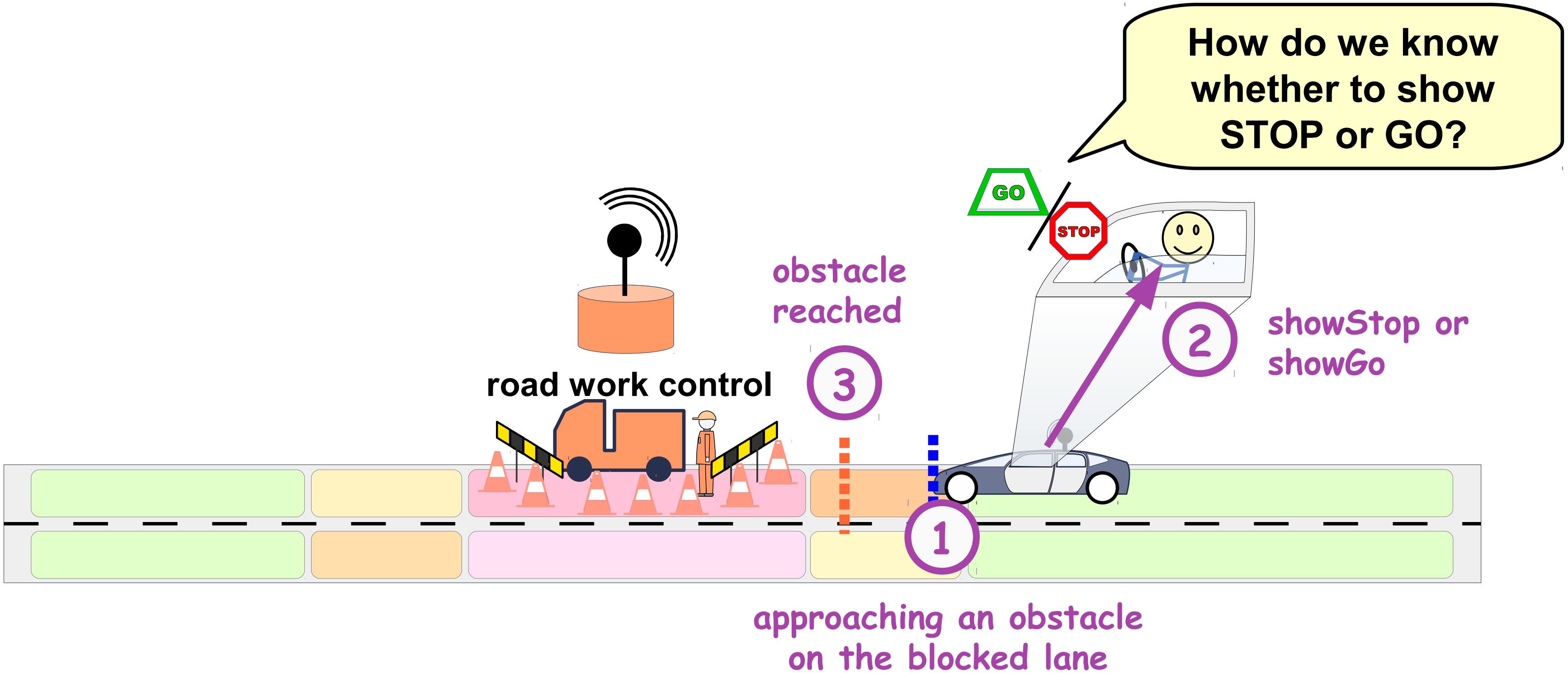
Collaboration: “approaching obstacle on blocked lane”

- Scenario “Dashboard Of Car Approaching On Blocked Lane Shows Stop Or Go”:
  1) When approaching an obstacle on the blocked lane
  2) Then the dashboard must indicate to STOP or to GO
  3) Before the car finally reaches the obstacle



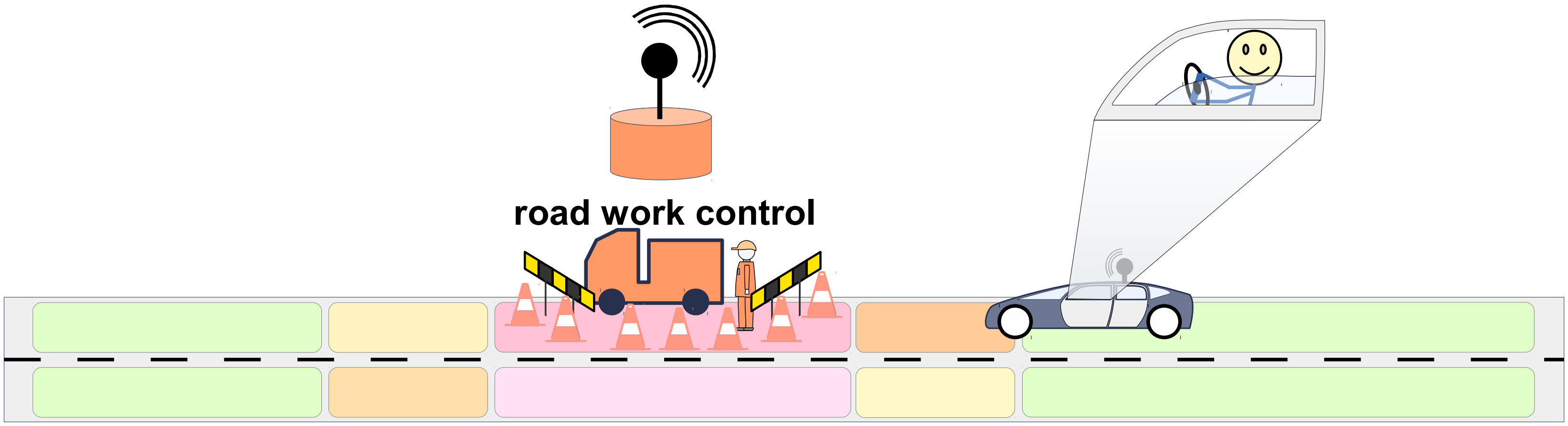
Collaboration: "approaching obstacle on blocked lane"

- Scenario "Dashboard Of Car Approaching On Blocked Lane Shows Stop Or Go":
  1) When approaching an obstacle on the blocked lane
  2) Then the dashboard must indicate to STOP or to GO
  3) Before the car finally reaches the obstacle



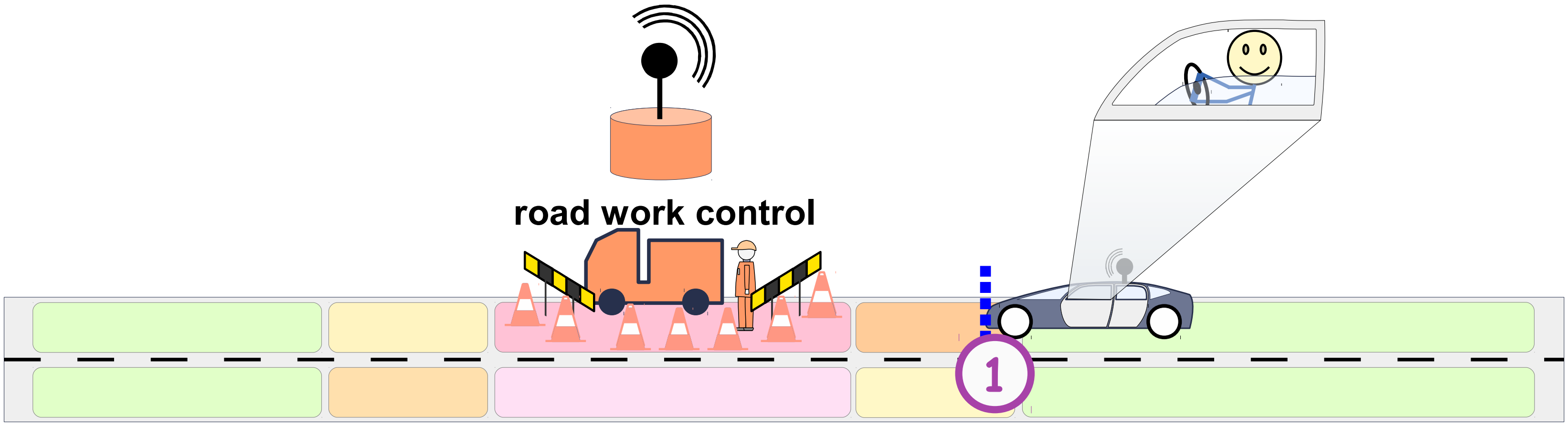
Collaboration: “approaching obstacle on blocked lane”

- Scenario “Control Station Checks for Car Approaching On Blocked Lane Entering Allowed Or Not”:



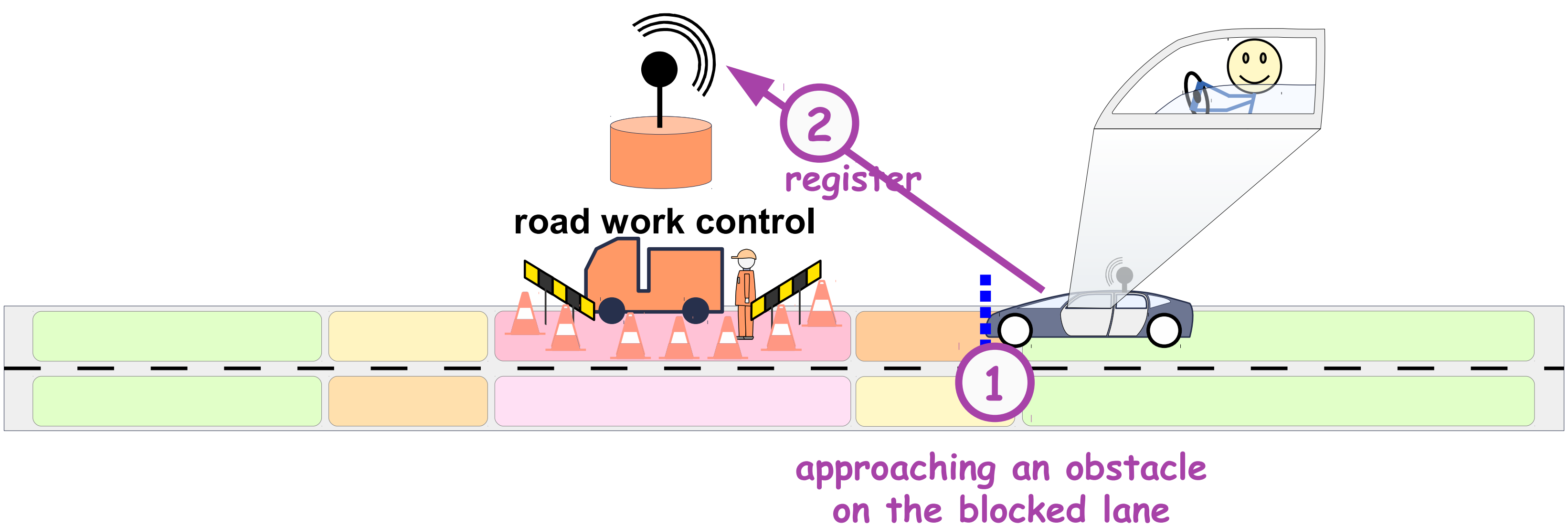
Collaboration: "approaching obstacle on blocked lane"

- Scenario "Control Station Checks for Car Approaching On Blocked Lane Entering Allowed Or Not":
  1) When approaching an obstacle on the blocked lane



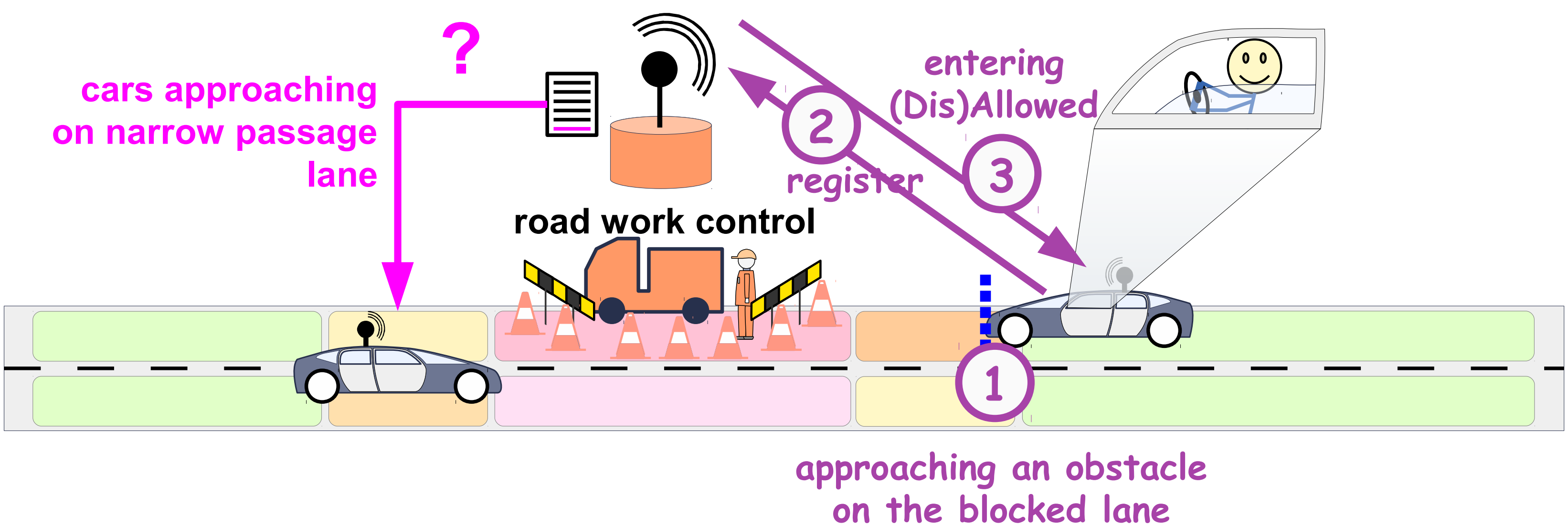
Collaboration: "approaching obstacle on blocked lane"

- Scenario "Control Station Checks for Car Approaching On Blocked Lane Entering Allowed Or Not":
  1) When approaching an obstacle on the blocked lane
  2) The car must register at the obstacle's control station



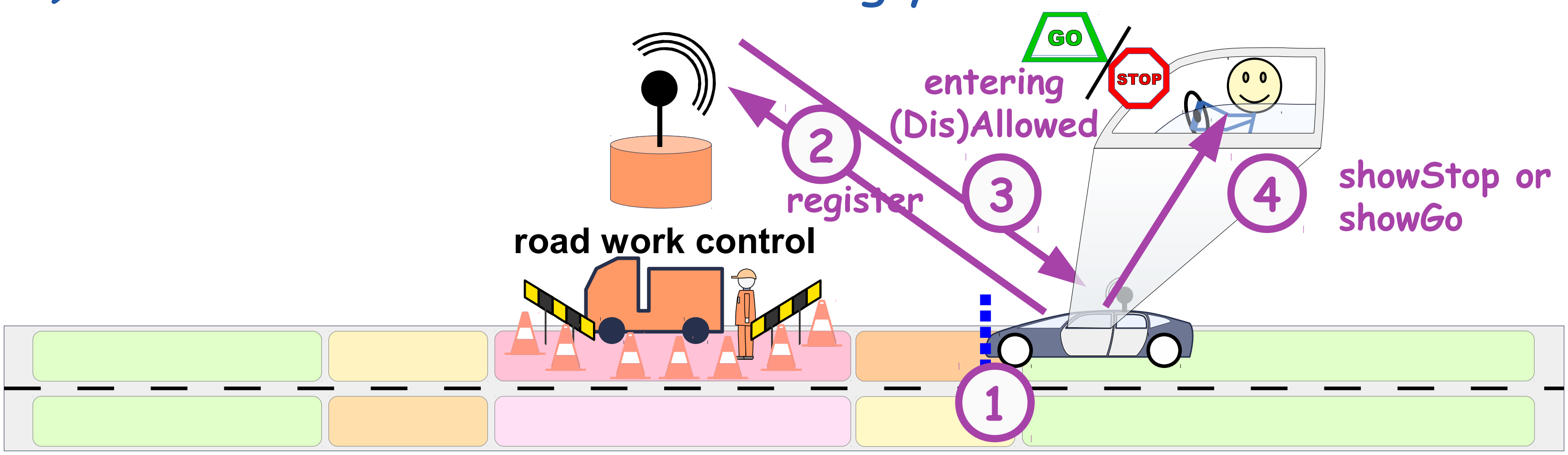
Collaboration: "approaching obstacle on blocked lane"

- Scenario "Control Station Checks for Car Approaching On Blocked Lane Entering Allowed Or Not":
  1) When approaching an obstacle on the blocked lane
  2) The car must register at the obstacle's control station
  3) If there is an approaching car in or before the narrow passage area: disallow the car entering the narrow passage
  - otherwise allow it

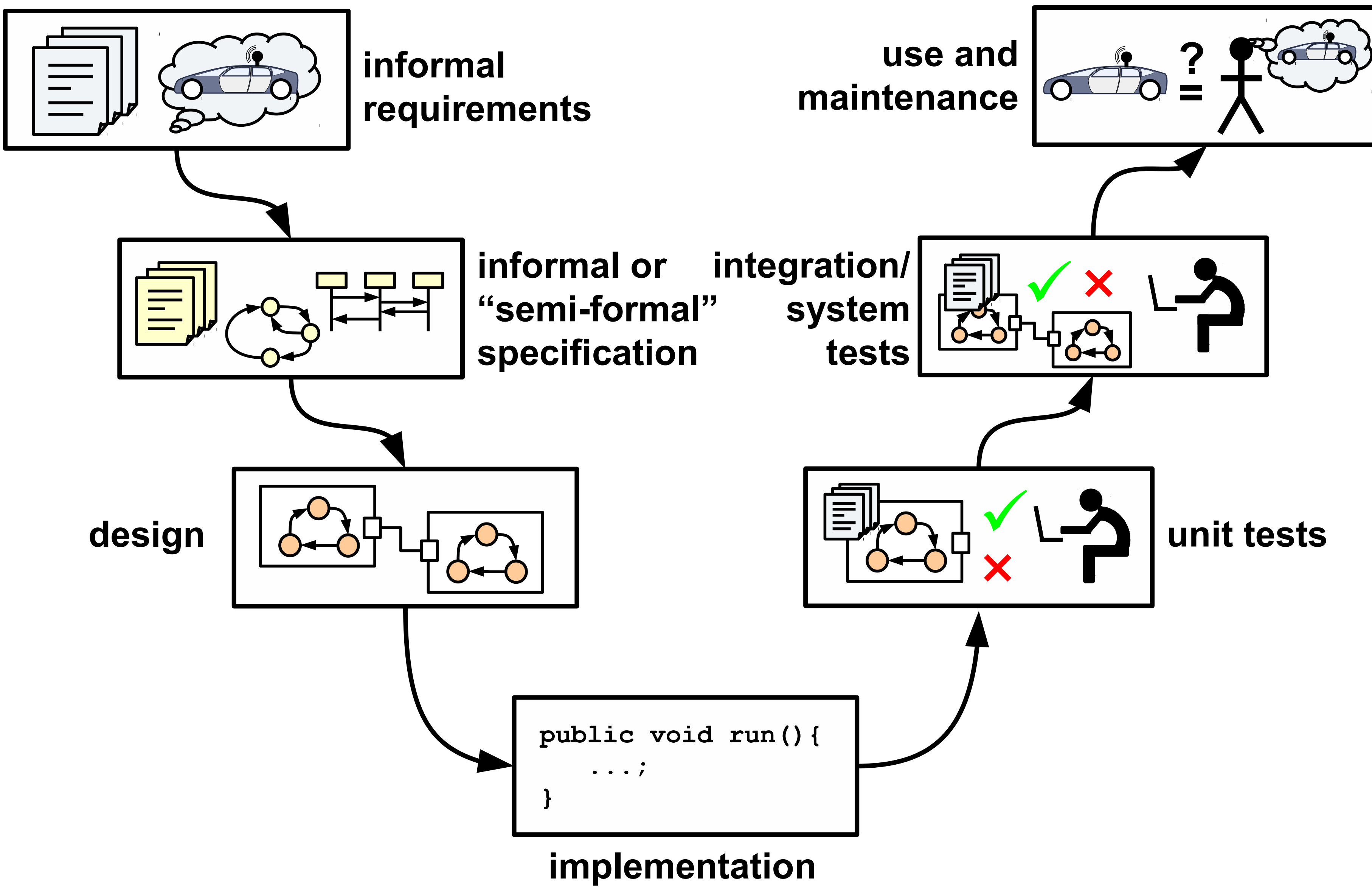


Collaboration: "approaching obstacle on blocked lane"

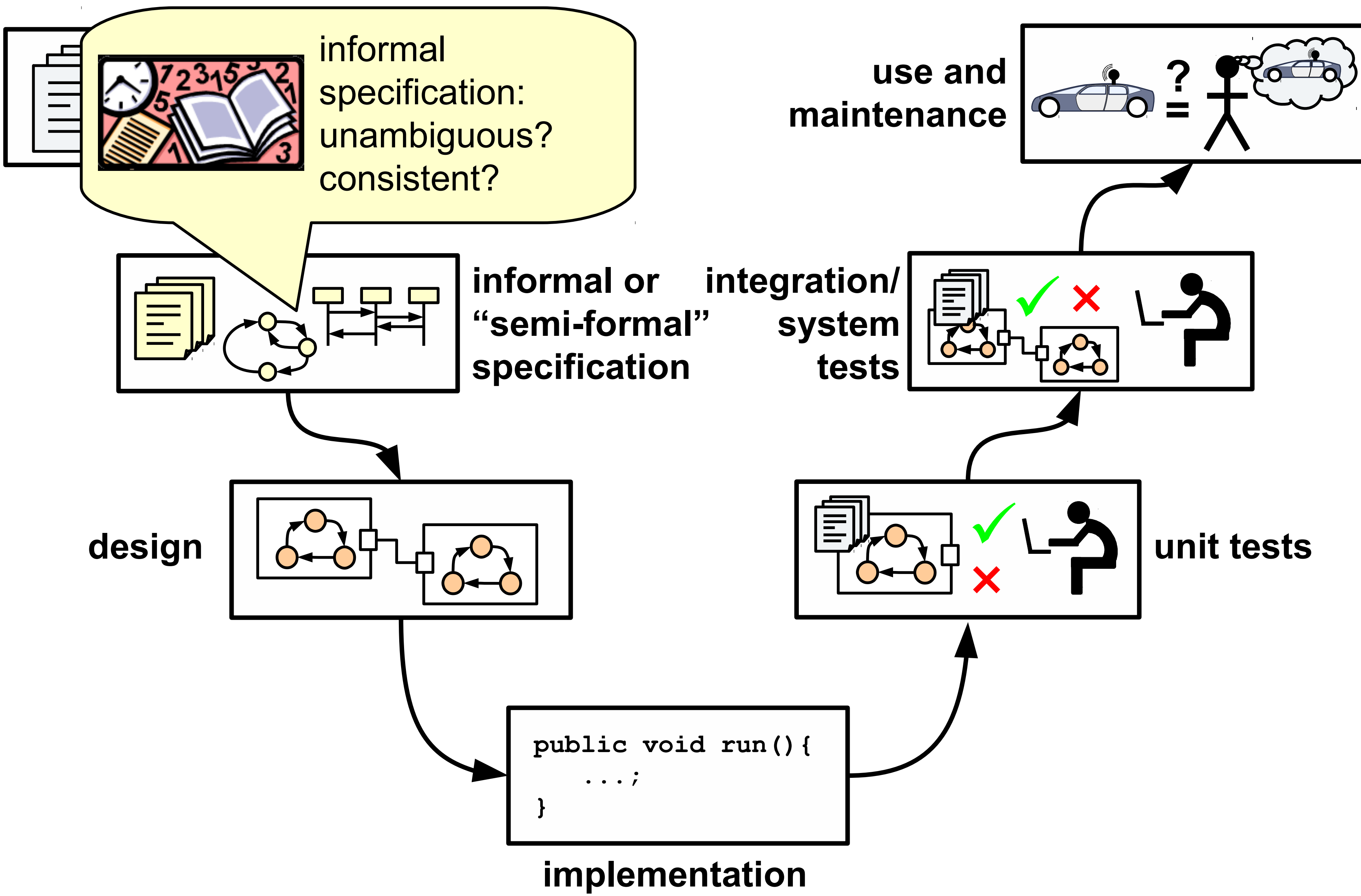
- Scenario "Control Station Checks for Car Approaching On Blocked Lane Entering Allowed Or Not":
  1) When approaching an obstacle on the blocked lane
  2) The car must register at the obstacle's control station
  3) If there is an approaching car in or before the narrow passage area: disallow the car entering the narrow passage
     - otherwise allow it
  4) Then show STOP/GO accordingly on the driver's dashboard



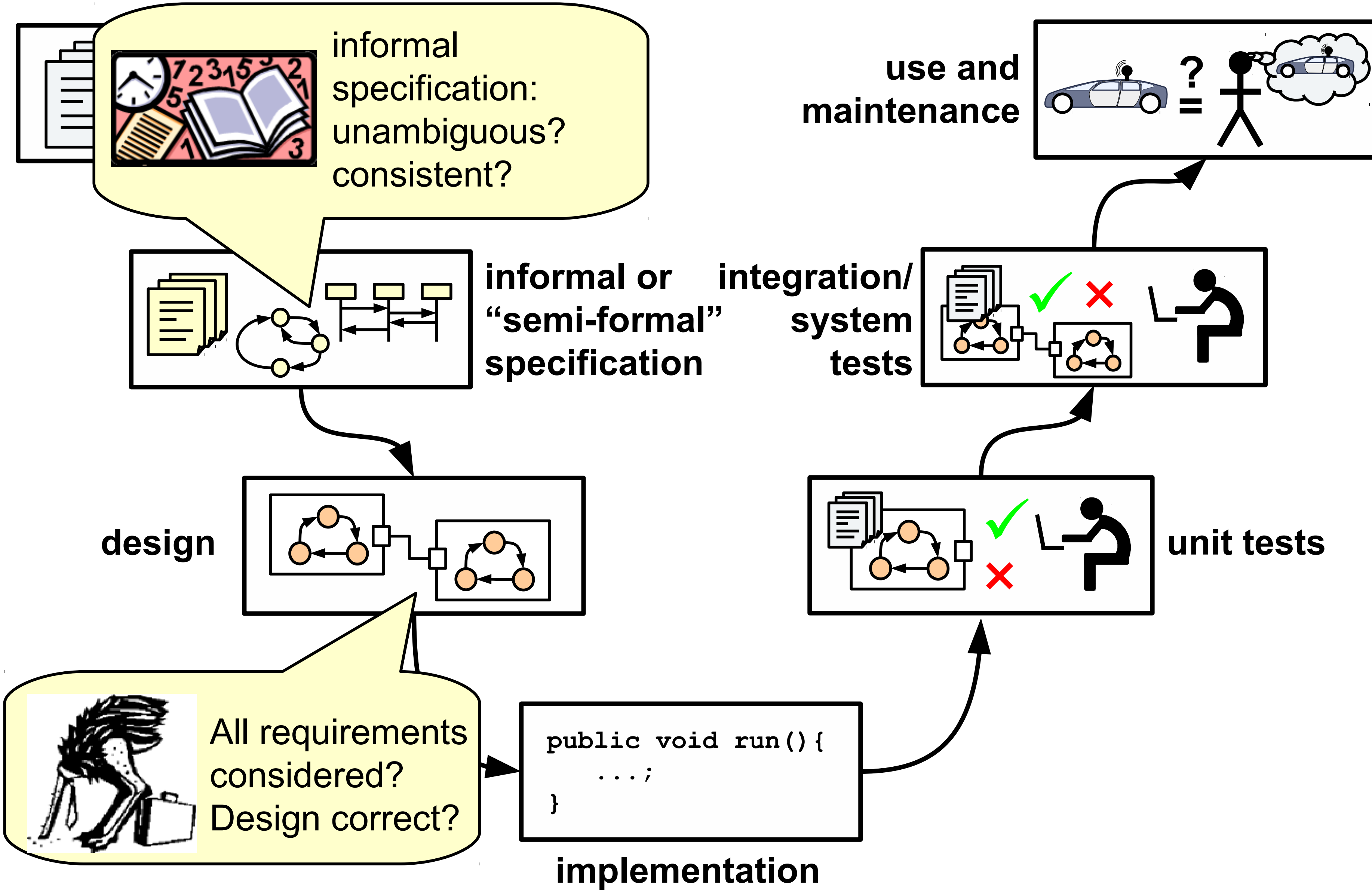
A typical Software/Systems Development Process...



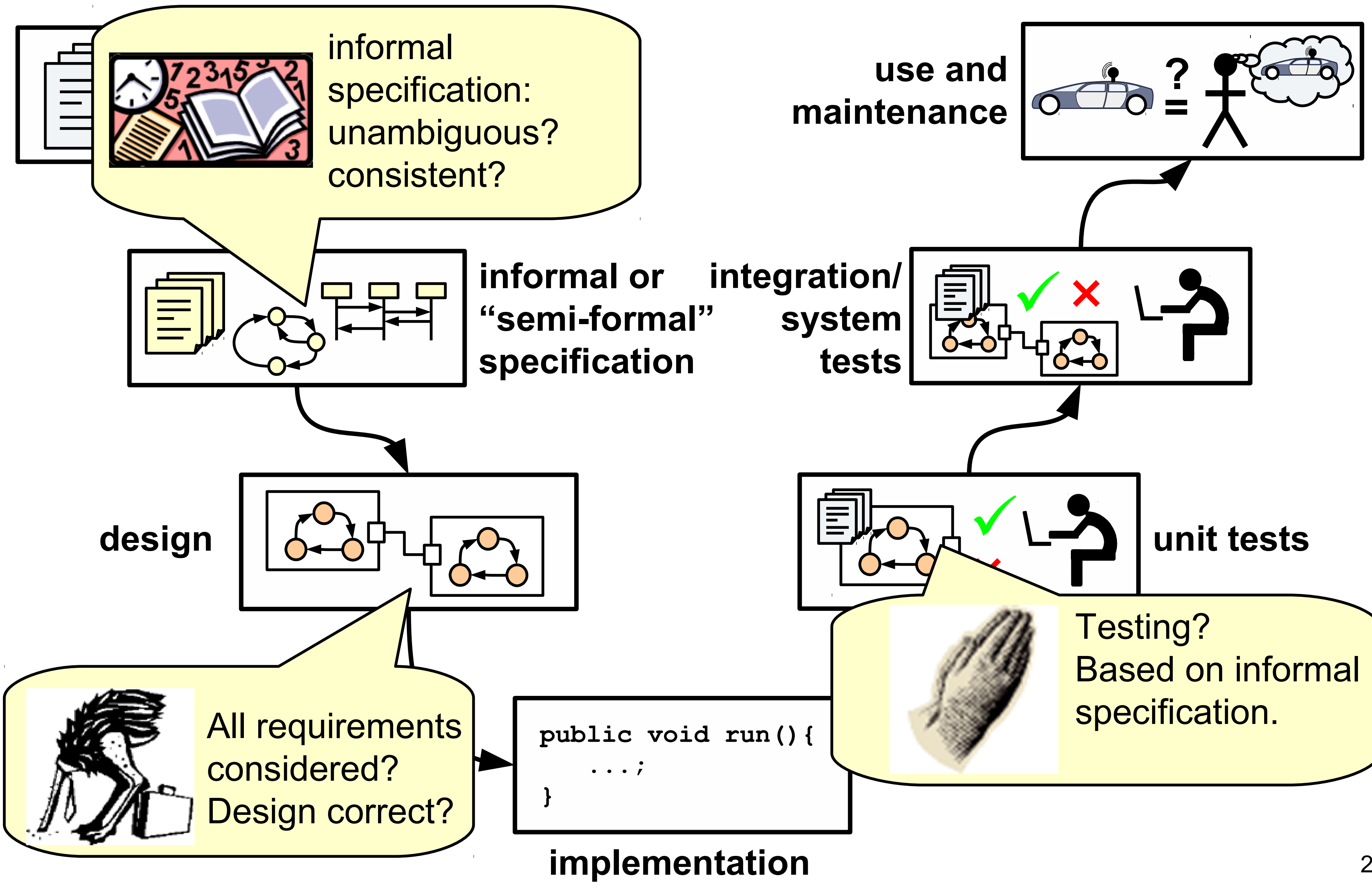
A typical Software/Systems Development Process...



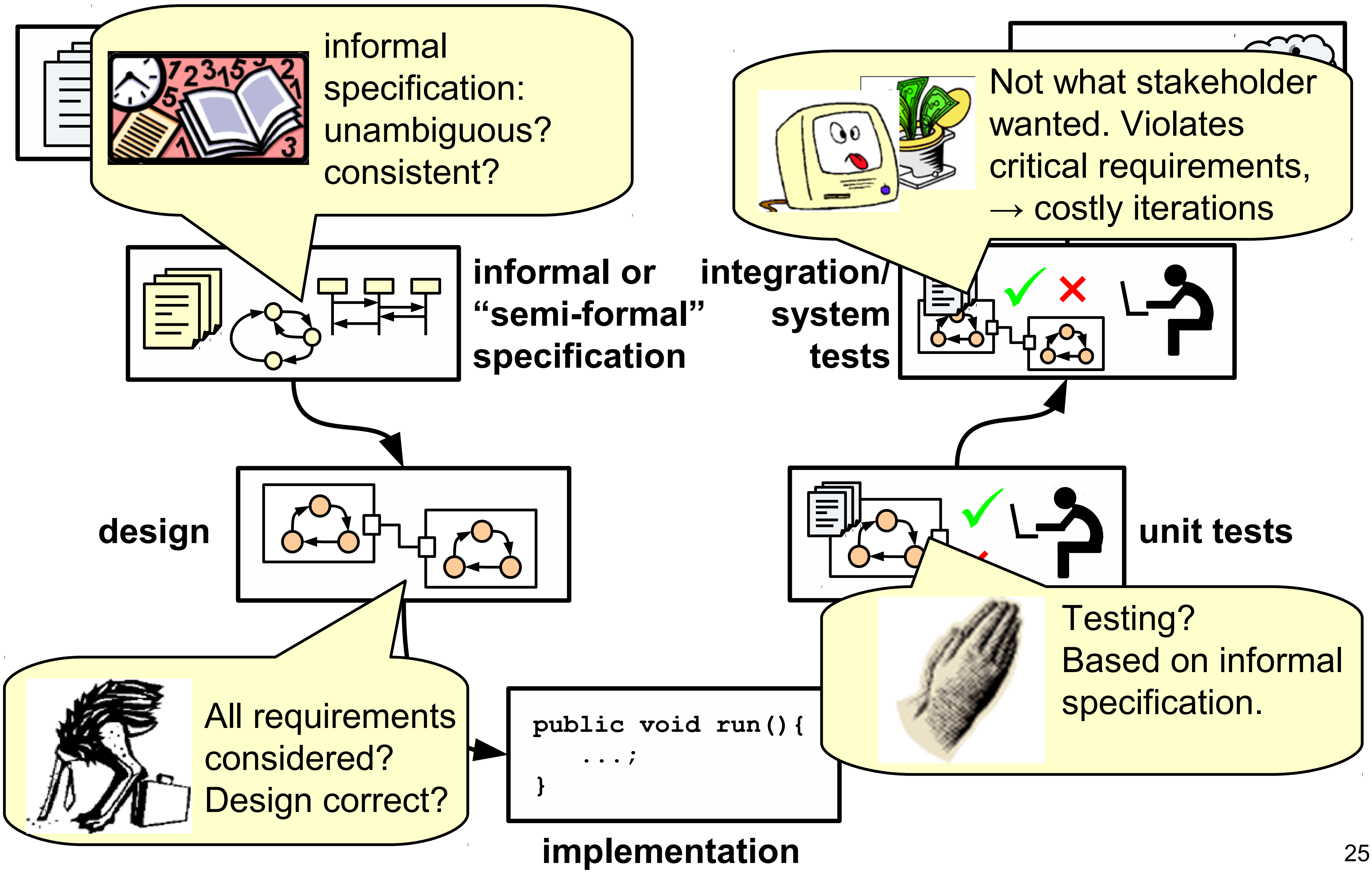
A typical Software/Systems Development Process...



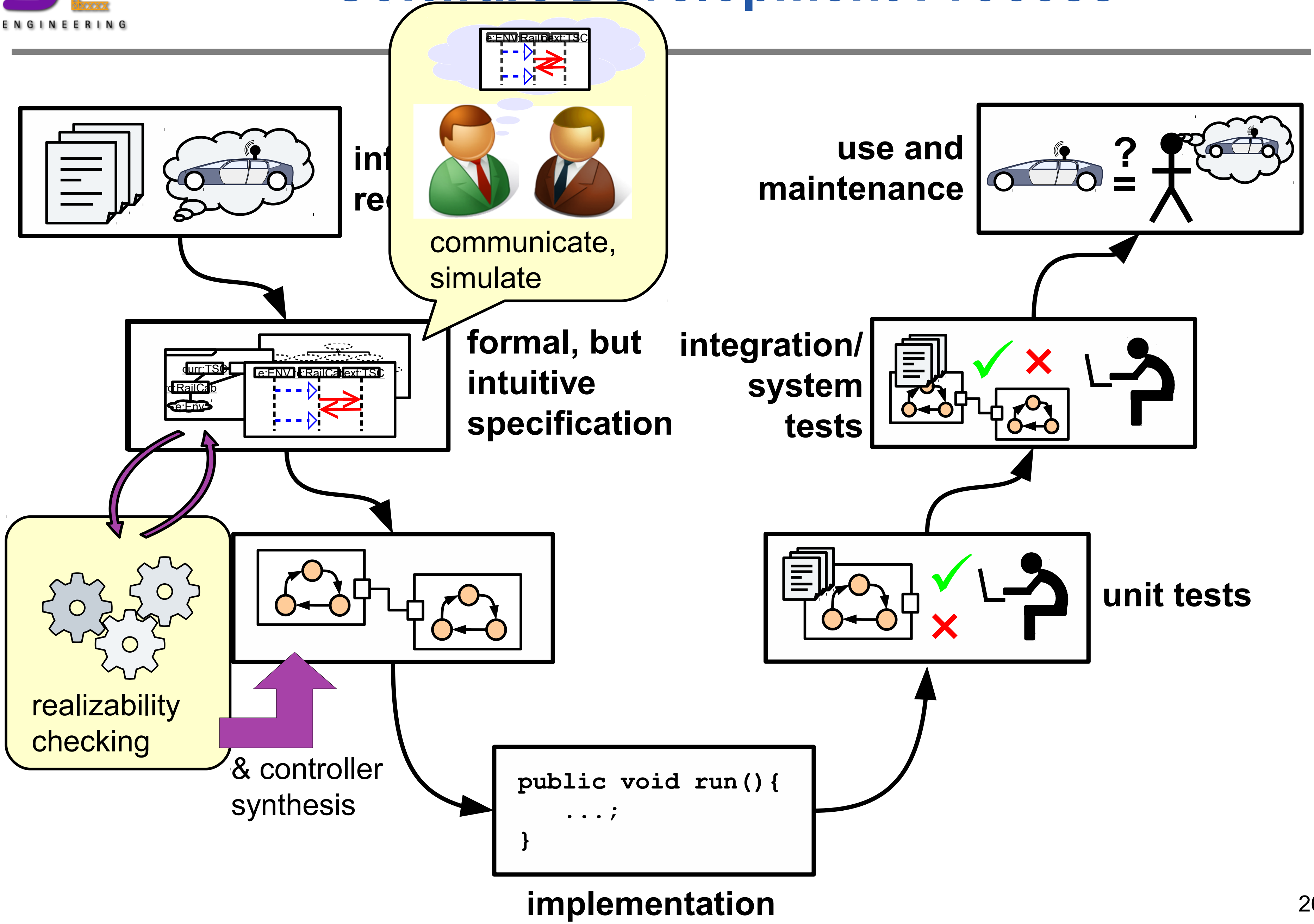
A typical Software/Systems Development Process...



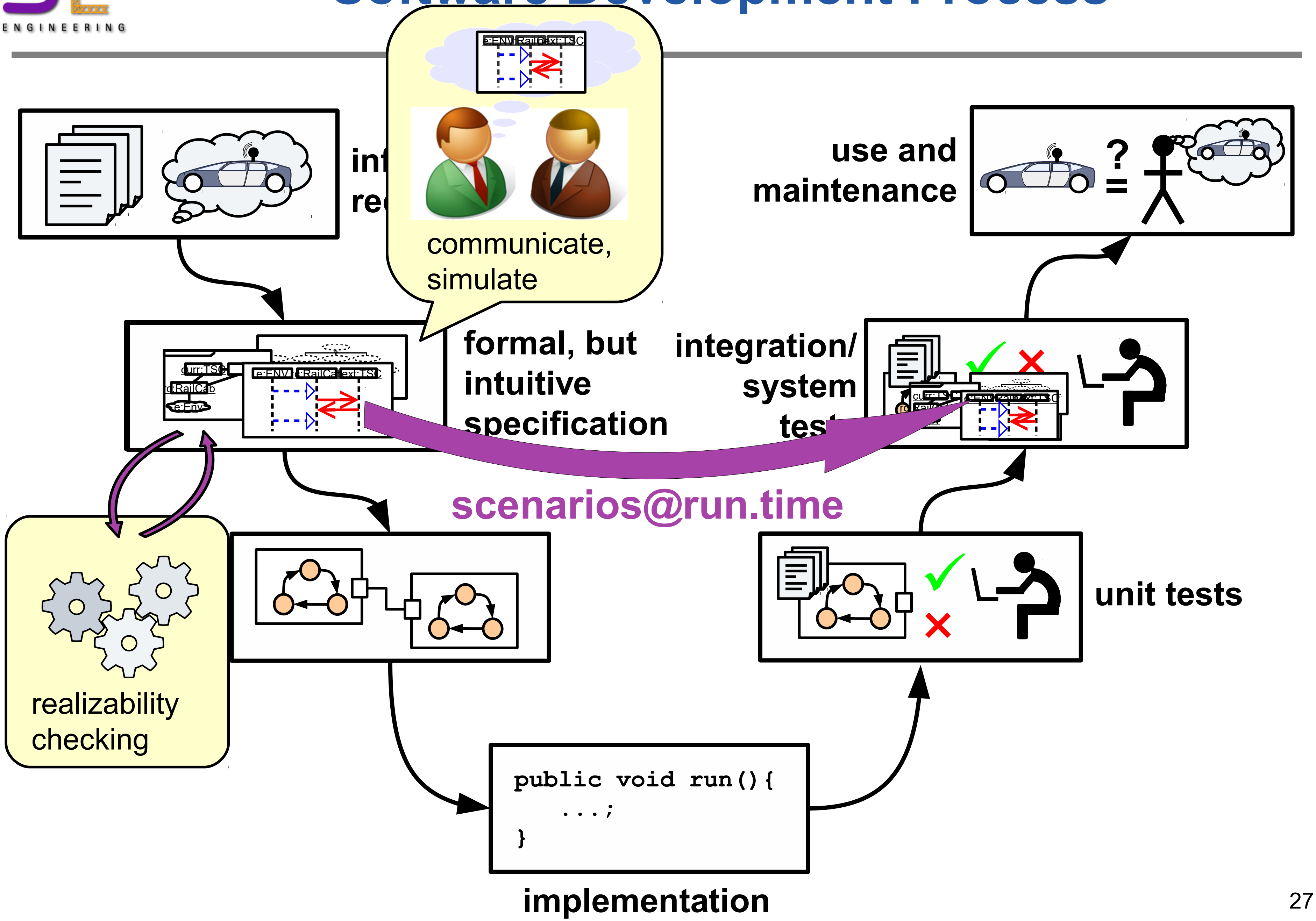
A typical Software/Systems Development Process...



Software Development Process



Software Development Process



Scenario Design Language (SDL)

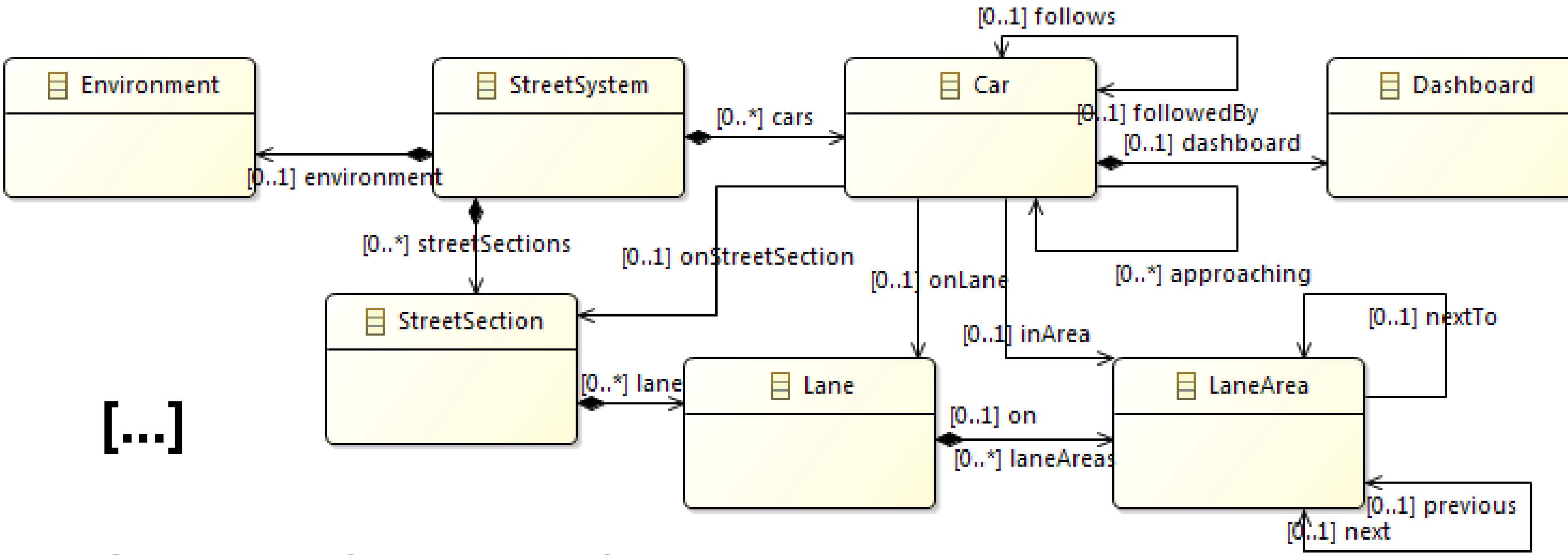
- Textual language based on **Live Sequence Charts (LSCs)**
- **Collaborations** describe, by a set of roles, a structure of objects that collaborate to fulfill a certain functionality

```
collaboration ApproachingObstacleOnBlockedLane{
        dynamic role Environment env
        dynamic role Car car
        dynamic role Dashboard dashboard
        ...
```

- **Scenarios** describe properties that must be satisfied by all message-based interactions of objects

```
specification scenario DashboardOfCarApproachingOn
                                    -BlockedLaneShowsStopOrGo{
        message env->car.approachingObstacleOnBlockedLane()
        ...
}
```

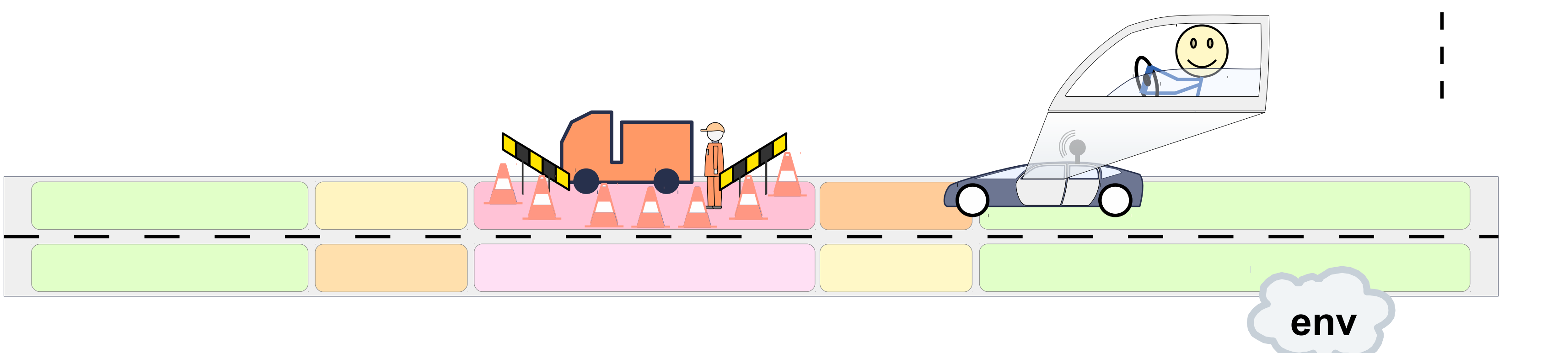
The Object System



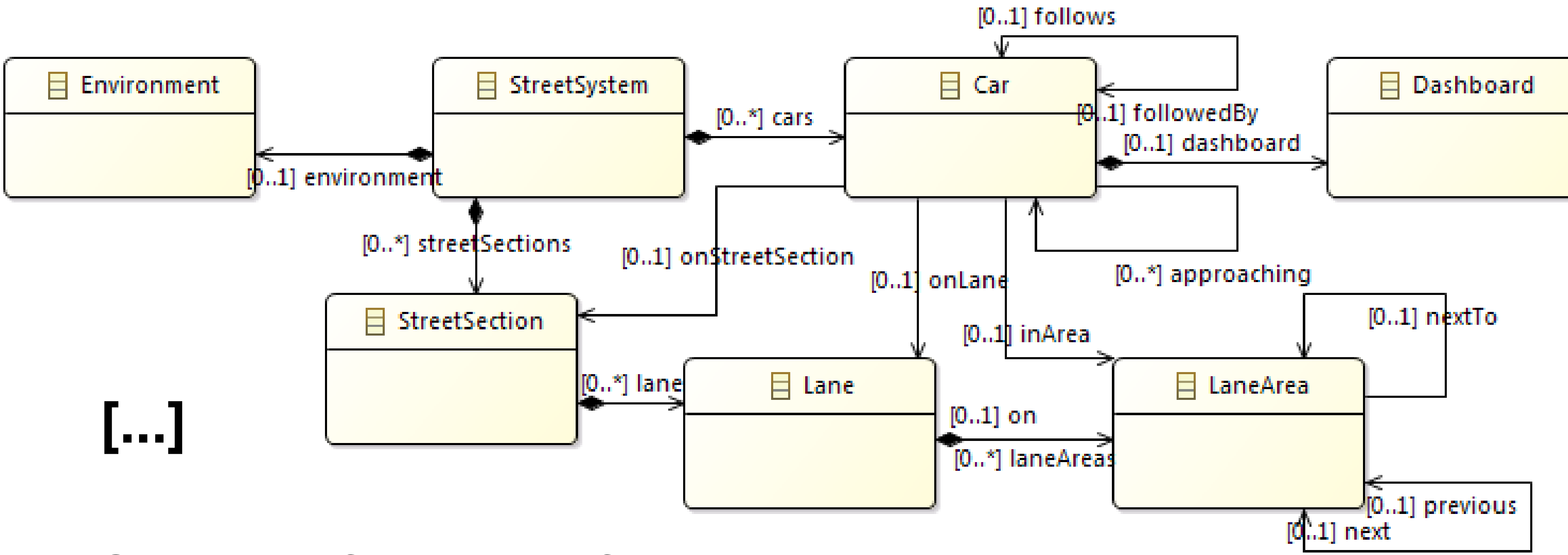
[...]

- SDL specifications refer to systems of objects (instances of a class model)

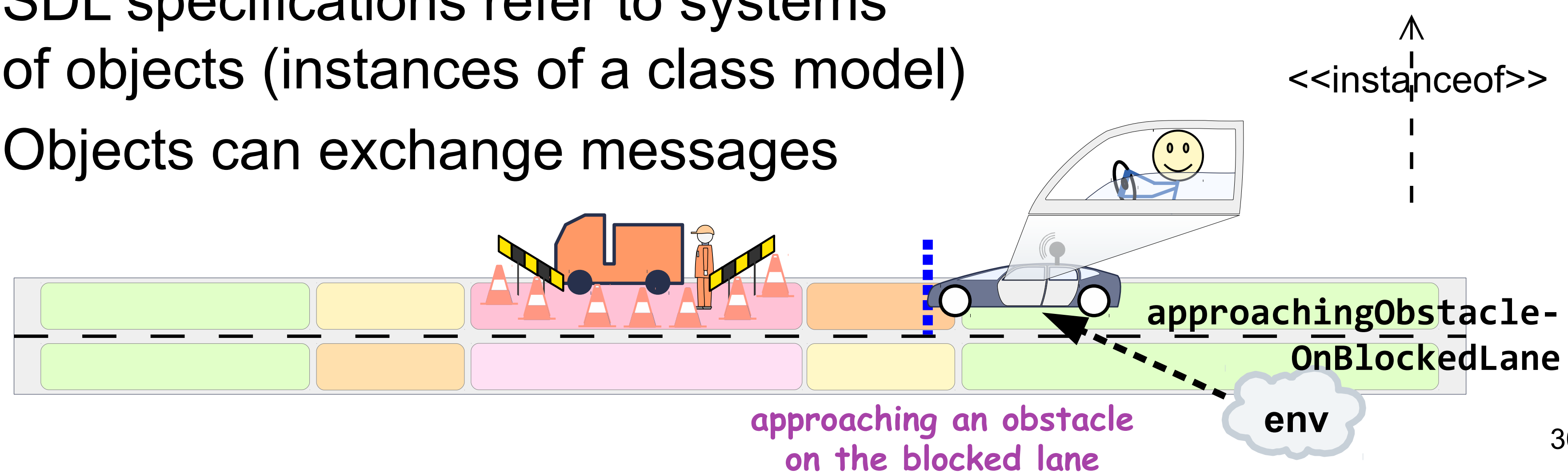
<<instanceof>>



The Object System



- SDL specifications refer to systems of objects (instances of a class model)
- Objects can exchange messages



Scenario Design Language

- Formalizing our first scenario:

```
specification scenario DashboardOfCarApproachingOn
                             -BlockedLaneShowsStopOrGo
with dynamic bindings [
   bind dashboard to car.dashboard
]{
   message env->car.approachingObstacleOnBlockedLane()
   alternative{
      message strict requested car->dashboard.showGo()
   } or {
      message strict requested car->dashboard.showStop()
   }
   message env->car.obstacleReached()
}
```

- With the modalities **strict** and **requested**, we can express what may, must, and must not happen

Scenario Design Language

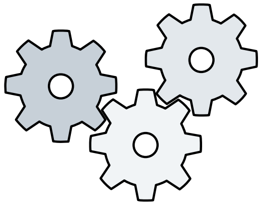
- Formalizing our second scenario:

```
specification scenario ControlStationChecksForCarOnBlockedLane
with dynamic bindings [...]{
   message env->car.approachingObstacleOnBlockedLane()
   message strict requested car->obstacleControl.register()
   alternative if [
        obstacleControl.carsOnNarrowPassageLaneApproaching.isEmpty()
   ] {
      message strict requested obstacleControl->car.enteringAllowed()
      message strict requested car->dashboard.showGo()
   } or if[
      !obstacleControl.carsOnNarrowPassageLaneApproaching.isEmpty()
   ] {
      message strict requested obstacleControl->car.enteringDisallowed()
      message strict requested car->dashboard.showStop()
   }
}
```

Simulation via Play-Out

- An SDL specification can be executed via p**lay-out**
  – an executable interpretation of the scenarios

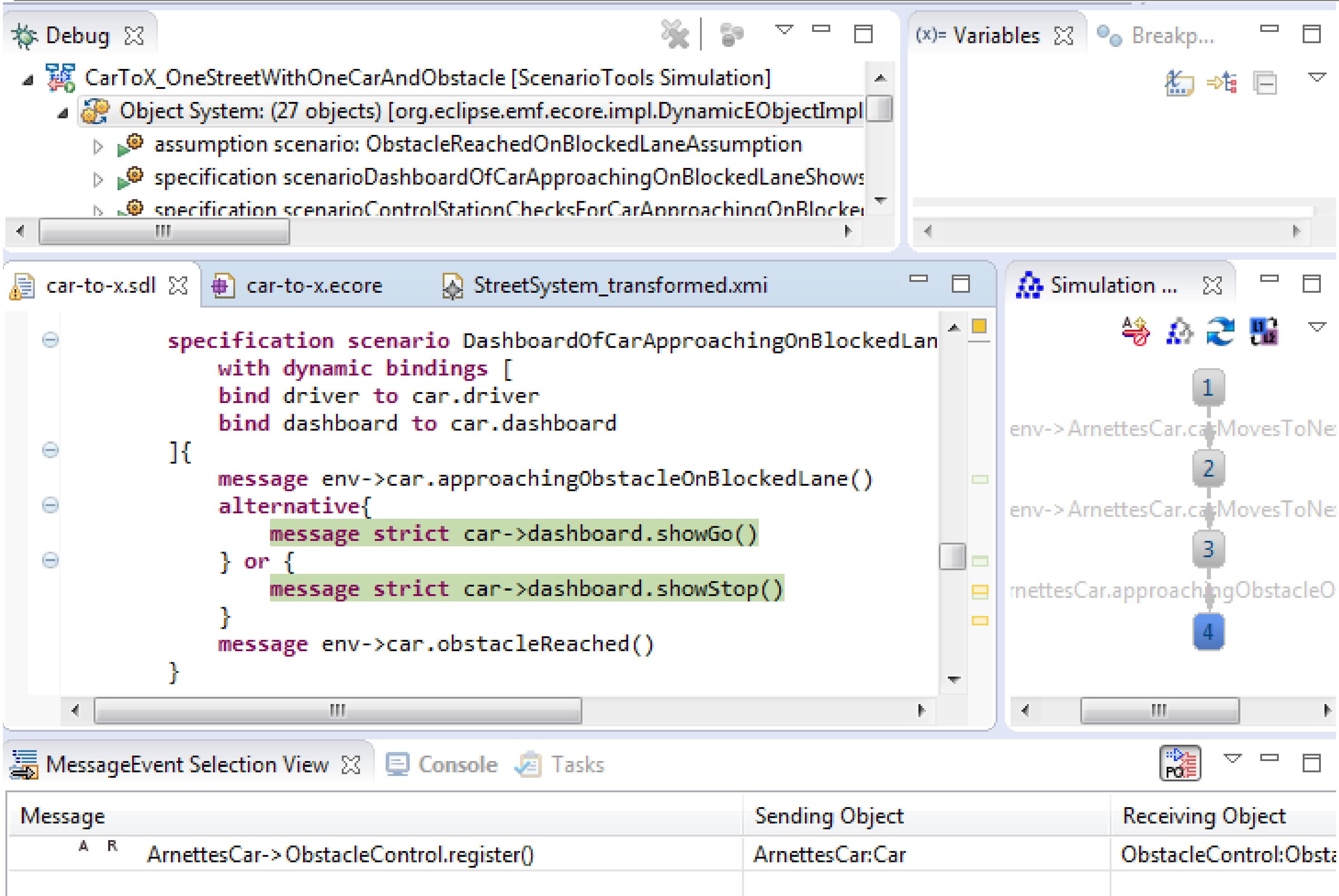
- This can be used for **simulation**
  – to **analyze** and **understand** the **interplay of the scenarios**

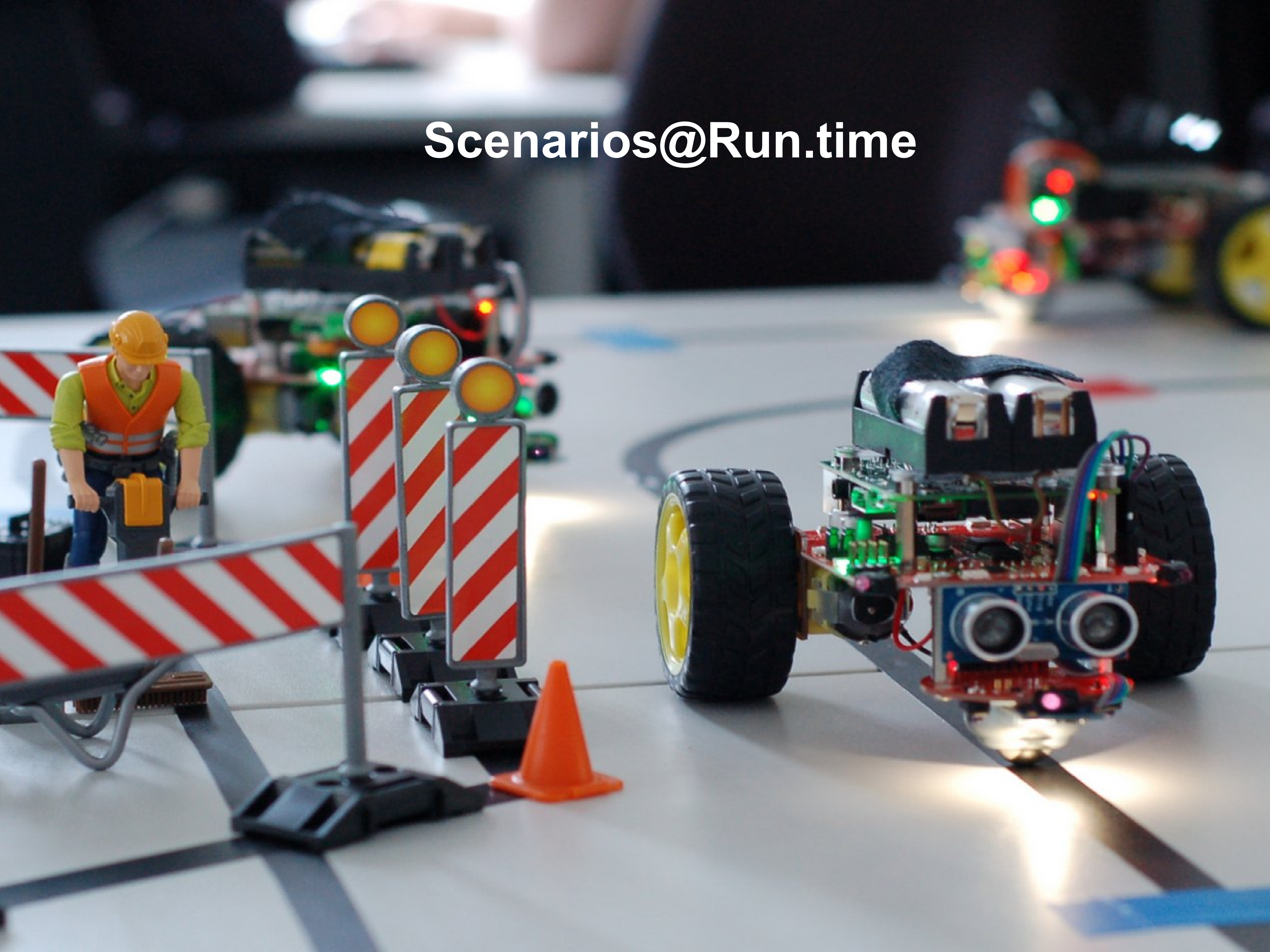


- The **play-out** algorithm In a nutshell:
  1) environment events occur and activate scenarios
  2) the scenarios prescribe events that the system must execute
  3) play-out executes these events while trying to avoid violations
  4) when all system reactions are executed, wait for the next environment event (goto Step 1)

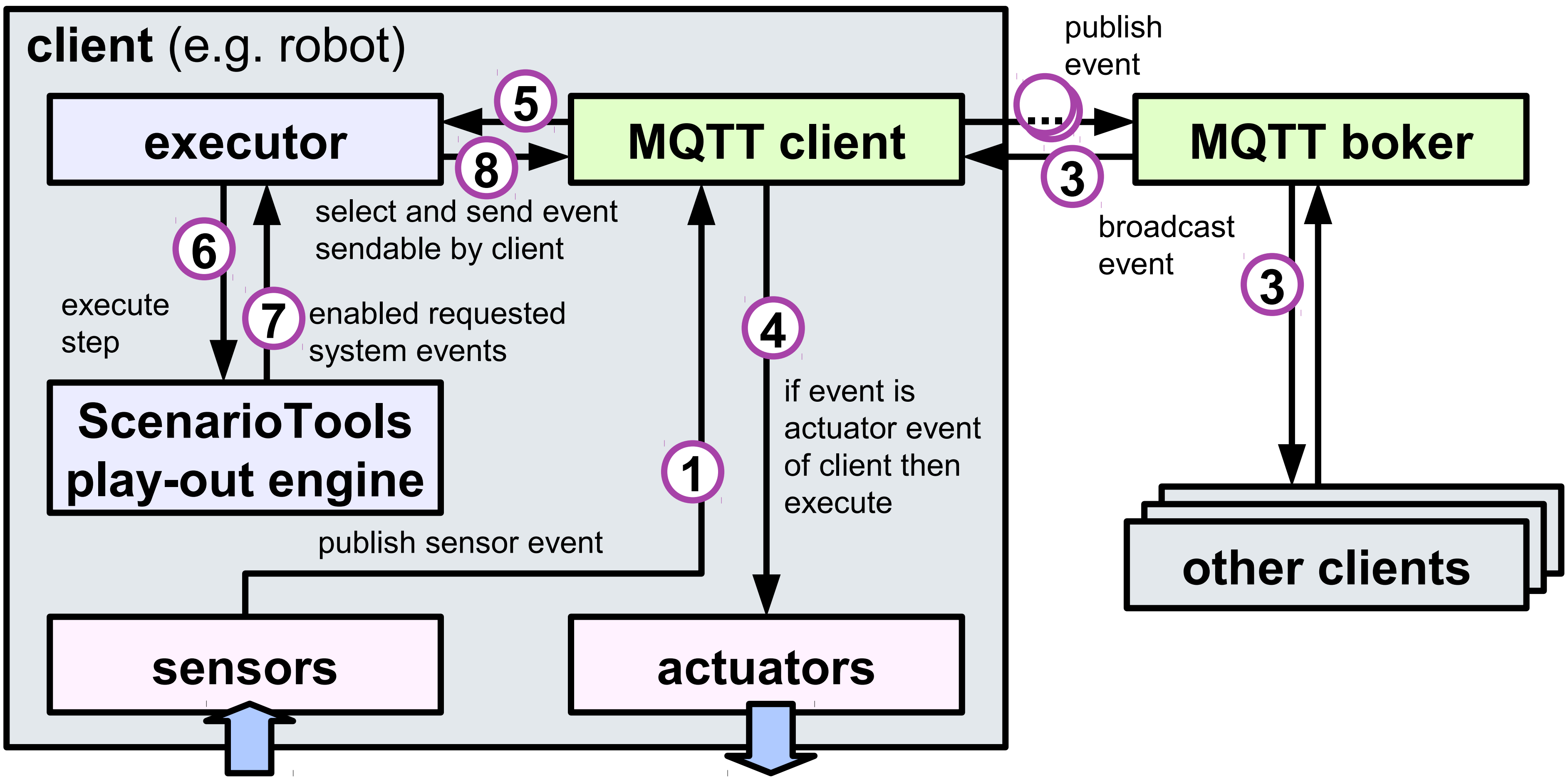
Simulation via Play-Out in ScenarioTools

ScenarioTools.org




Scenarios@Run.time

Steps in the Scenarios@Run.time Framework



Summary & Perspective

- Execute scenario specifications on **distributed** systems
- **New**:
  - **dynamic structures**: interpretation of dynamic role bindings
  - also execute **environment assumptions**
    - **run-time monitoring** if environment behaves as assumed

- Relies on full synchronization of all components on all events
  - this overhead must be reduced:
    - only synchronize objects in certain parts of the system
    - analyze **at run-time** minimal set of components to synchronize

- Future work:
  - safe run-time updates of specification changes
  - dependability: how to recover from run-time failures