Development of Smart Cyber-Physical Systems

(CBSE 2014 Tutorial)

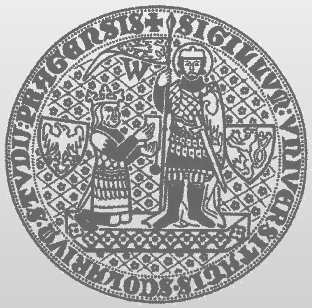
http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Tomáš Bureš

(on behalf of D3S)

bures@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

ascens

Outline

- Part 1
  - Overview of the Context: Smart Cyber Physical Systems (sCPS)
- Part 2
  - Challenges and existing approaches
- Part 3
  - Realization via intelligent component ensembles
  - DEECo component model and abstractions

(coffee break)

- Part 4
  - Goal-based design of sCPS
  - Invariant Refinement Method (IRM)
- Part 5
  - Showcase (DEECo + IRM) – intelligent parking
- Part 6
  - Going further: Analysis and simulation
  - Conclusion

Part 1

Overview of the Context: Smart Cyber Physical Systems (sCPS)

Context: Cyber-Physical Systems (CPS)

- Definition:
  - systems of collaborating entities, typically materialized as software components that closely interact and control the physical environment.

- Traditionally
  - Well known in embedded systems community
  - Focus on HW
  - Limited architecture, limited dynamicity
  - Sometimes not even perceived as distributed

Smart Cyber Physical Systems

- Term widespread by recent EU H2020 call.
  - **ICT 1 – 2014: Smart Cyber-Physical Systems**

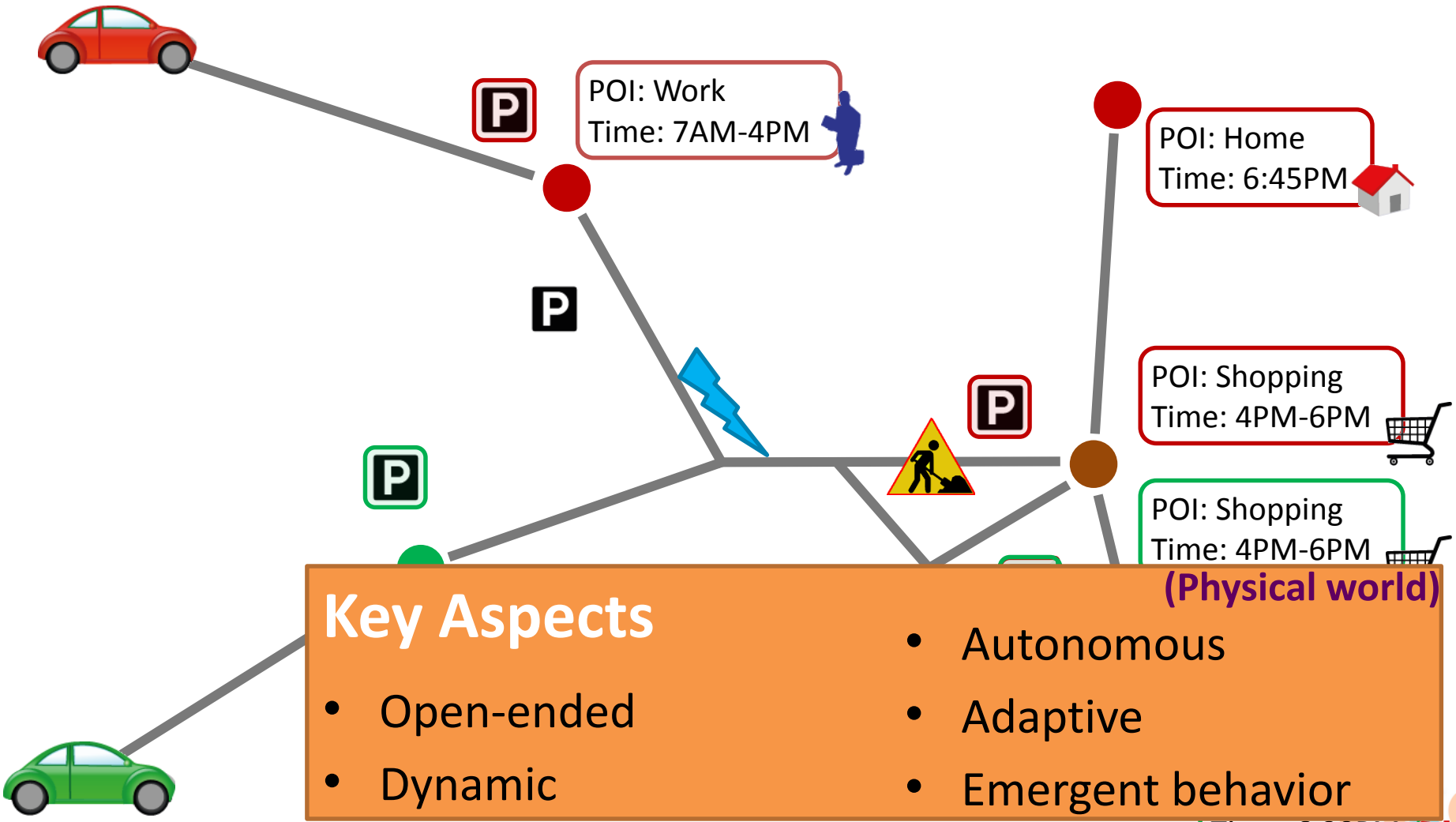
"Specific Challenge: Cyber-Physical Systems (CPS) refer to next generation embedded ICT systems that are interconnected and collaborating including through the Internet of things, and providing citizens and businesses with a wide range of innovative applications and services. These are the ICT systems increasingly embedded in all types of artefacts making 'smarter', more intelligent, more energy-efficient and more comfortable our transport systems, cars, factories, hospitals, offices, homes, cities and personal devices. Focus is on both reinforcing European industrial strengths as well as exploring new markets. Often endowed with control, monitoring and data gathering functions, CPS need to comply with essential requirements like safety, privacy, security and near-zero power consumption as well as size, usability and adaptability constraints."

[HORIZON 2020 – WORK PROGRAMME 2014-2015, LEIT – Information and Communication Technologies]

H2020-ICT Call 1 Statistics

Topic	RIA	IA	CSA	Grand Total
ICT-01-2014	140	10	8	158
ICT-02-2014	100	4	2	106
ICT-03-2014	30	7		37
ICT-05-2014	37			37
ICT-06-2014	87		3	90
ICT-07-2014	140	6	11	157
ICT-09-2014	75			75
ICT-11-2014	26		4	30
ICT-13-2014		44	64	108
ICT-15-2014		87	19	106
ICT-17-2014	4	23	3	30
ICT-18-2014		71	14	85
ICT-21-2014	50	41		91
ICT-22-2014	91	17		108
ICT-23-2014	125	29		154
ICT-26-2014	94	1	6	101
ICT-29-2014	9			9
ICT-31-2014	43		5	48
ICT-32-2014	55		4	59
ICT-33-2014			1	1
ICT-35-2014		35	20	55
Grand Total	**1106**	**375**	**164**	**1645**

Example: E-mobility

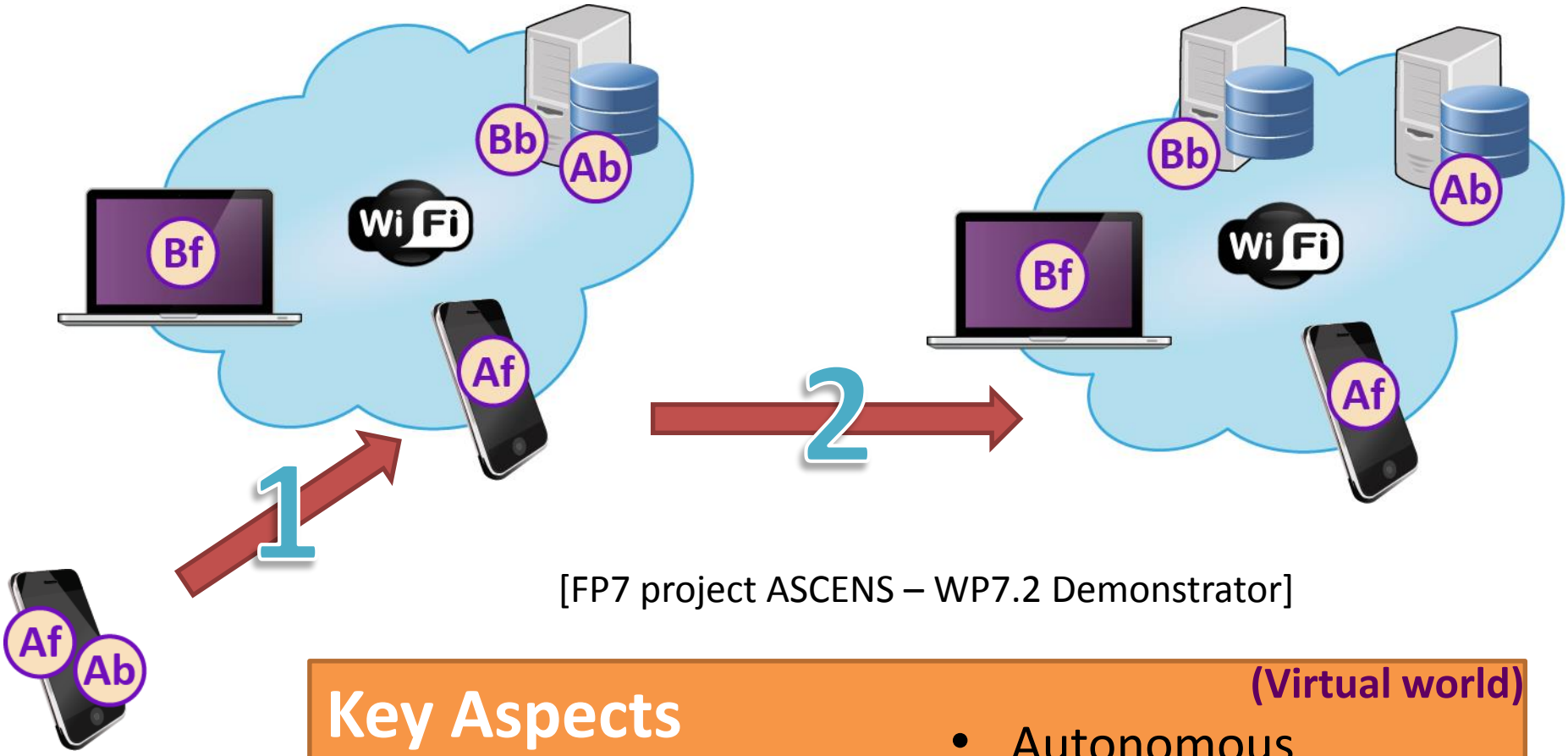


Key Aspects

- Open-ended
- Dynamic
- Autonomous
- Adaptive
- Emergent behavior

[FP7 project ASCENS – Deliverable D7.1 (VW Demonstrator)]

Example: Ad-Hoc Clouds



[FP7 project ASCENS – WP7.2 Demonstrator]

Key Aspects

(Virtual world)

- Open-ended
- Dynamic
- Autonomous
- Adaptive
- Emergent behavior

Smart Cyber-Physical Systems

- Characteristics:
  - Collaborating computational elements controlling physical entities
  - Designed as a network of interacting elements with physical input and output
  - Real-time, mobility, adaptation
  - **Software-intensive systems**

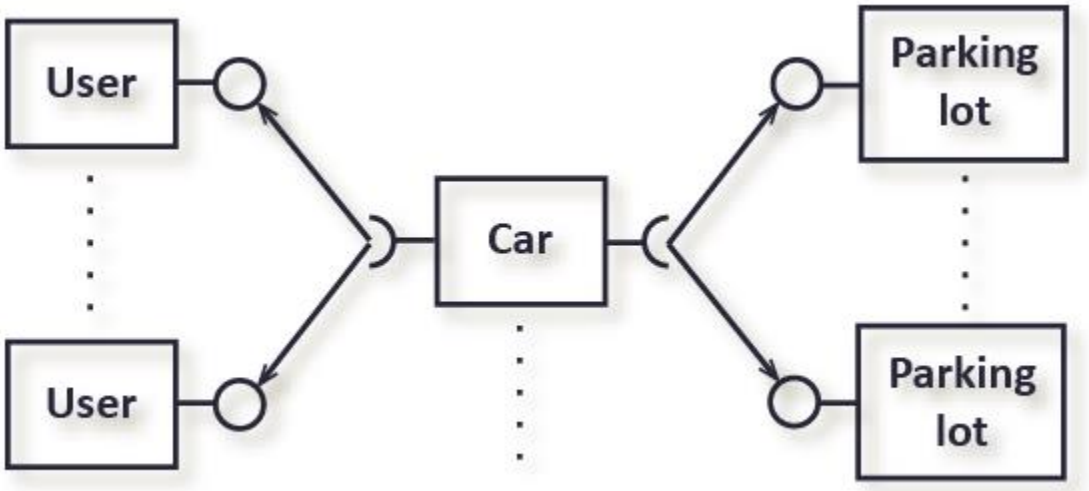
- CBSE goal: Component models, methodologies, and frameworks for systematic development of smart cyber-physical systems

Part 2

Challenges and existing approaches

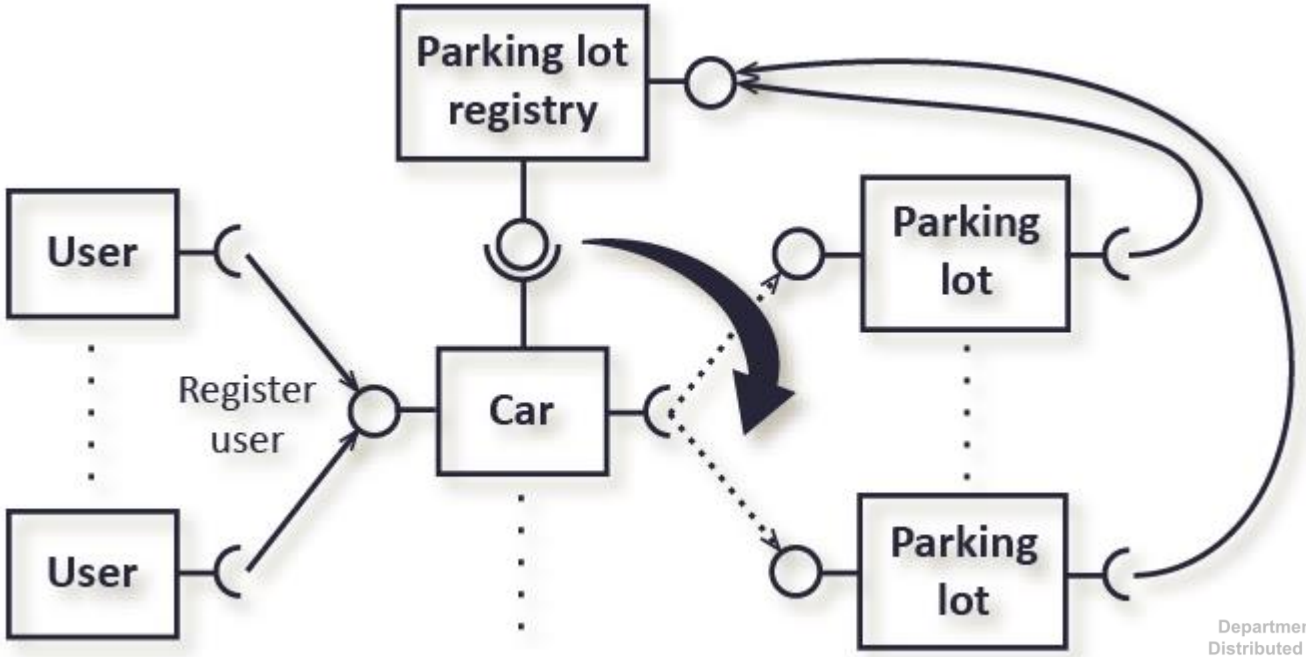
“Classical” Component-Based Approach

- Centralized ownership & deployment
- Cannot capture dynamic changes in architecture
- Guaranteed communication needed
- Strong reliance on other components



Service-Oriented Approach

- 3-rd party ownership & deployment
- Dynamic architecture (via service-driven discovery)
- Guaranteed communication needed
- Strong reliance on other services

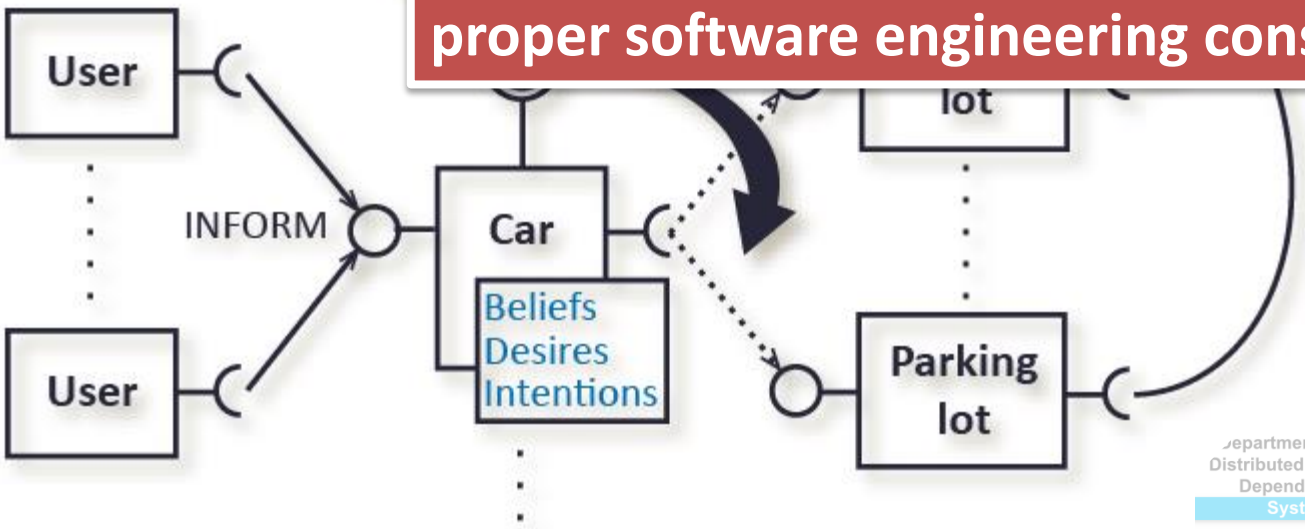


Agent-Based Approach

- 3-rd party ownership & deployment
- Dynamic architecture (via agent-driven discovery)
- Guaranteed communication needed
- Autonomous (beliefs – desires – intentions)

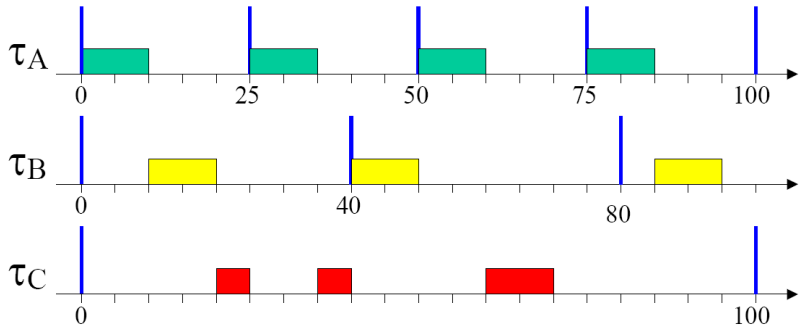
Agents bring conceptual autonomy

But do not sufficiently translate it to proper software engineering constructs



Distributed Control Systems

- Software as a set of real-time tasks
  - Sensing, actuating
  - Real-time scheduling
    - Period, deadline, WCET



- Distributed communication
  - Reliable, real-time guarantees
  - CAN, TTP, FlexRay, …

Part 3

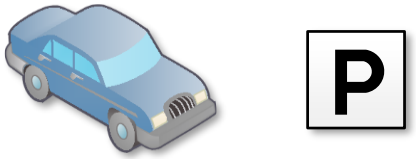
Realization via intelligent component ensembles
DEECo component model and abstractions

Component Ensembles

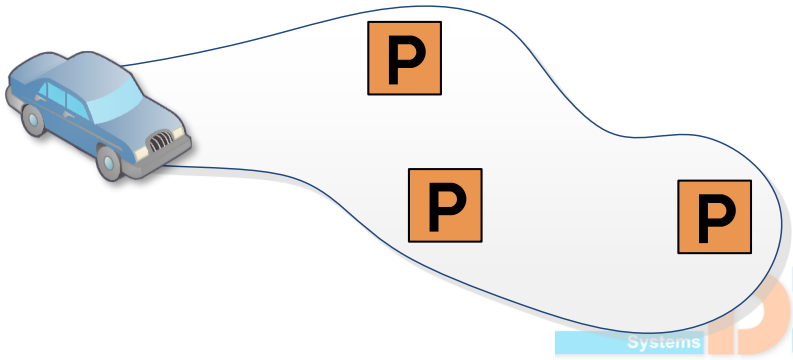
- Dynamic, goal-oriented groups of components
- Content-based addressing

- Featured by ASCENS project
- Stem from coordination languages KLAIM, SCEL

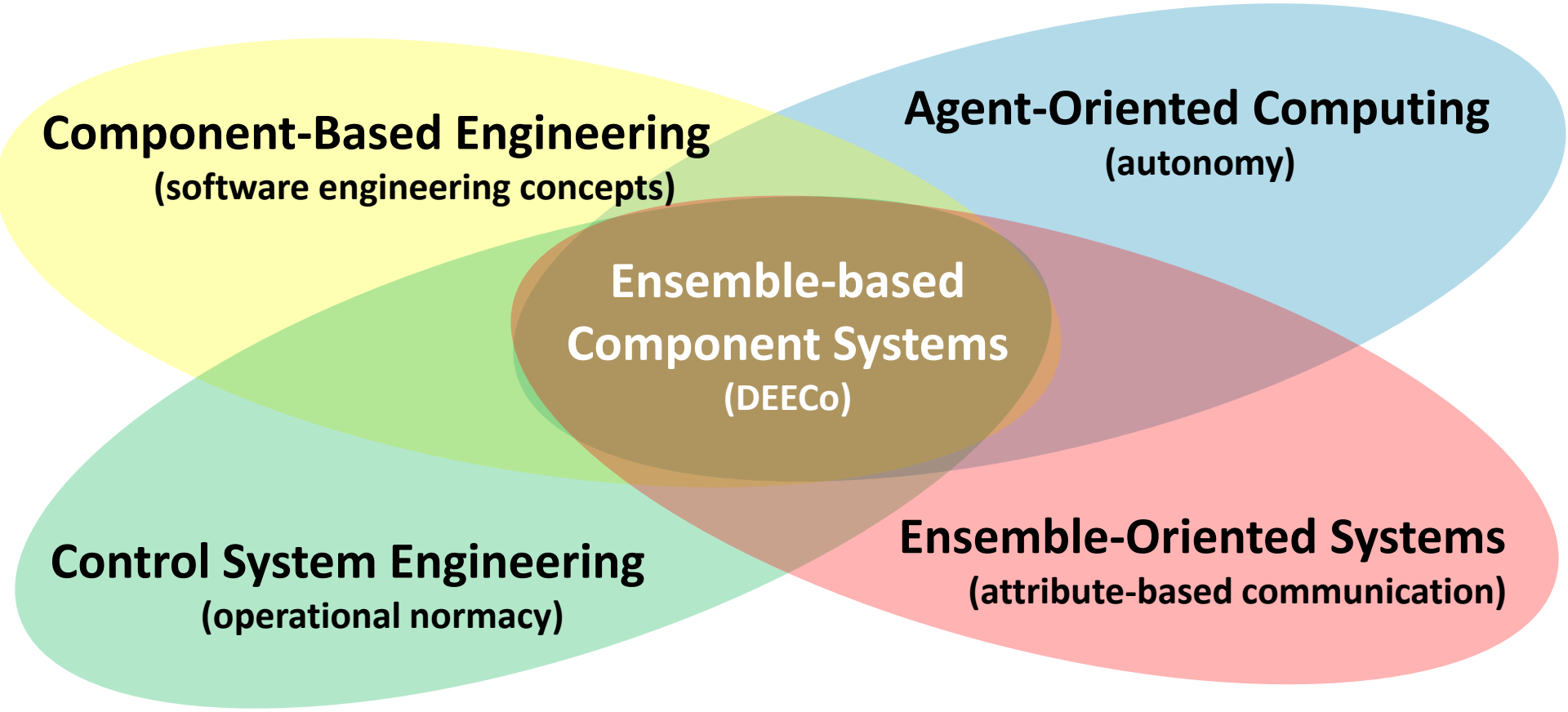
- Components
  - Autonomic
  - (Self-) adaptive



- Ensembles
  - Emergent, distributed
  - Mediate component cooperation to achieve global system goals

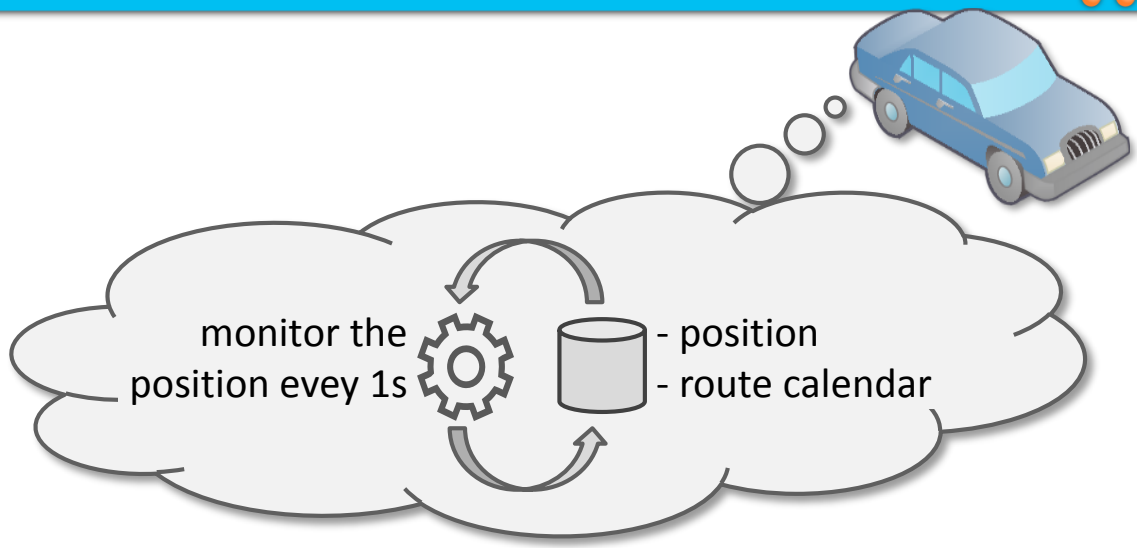


Solution Strategy: Hybrid approach

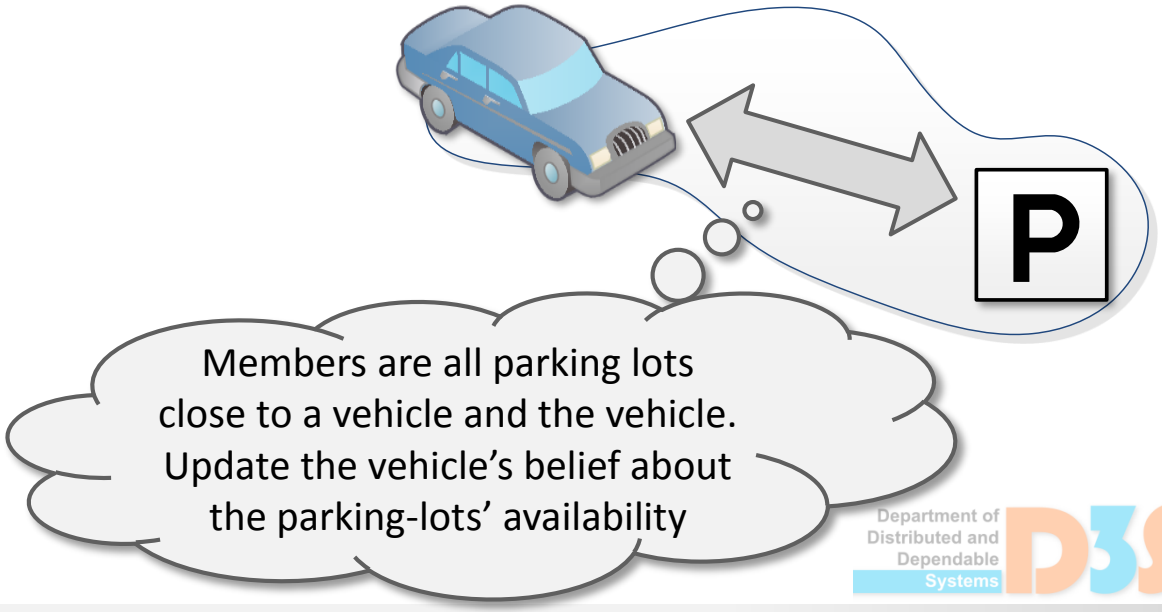


DEECo as a Refinement of EBCS

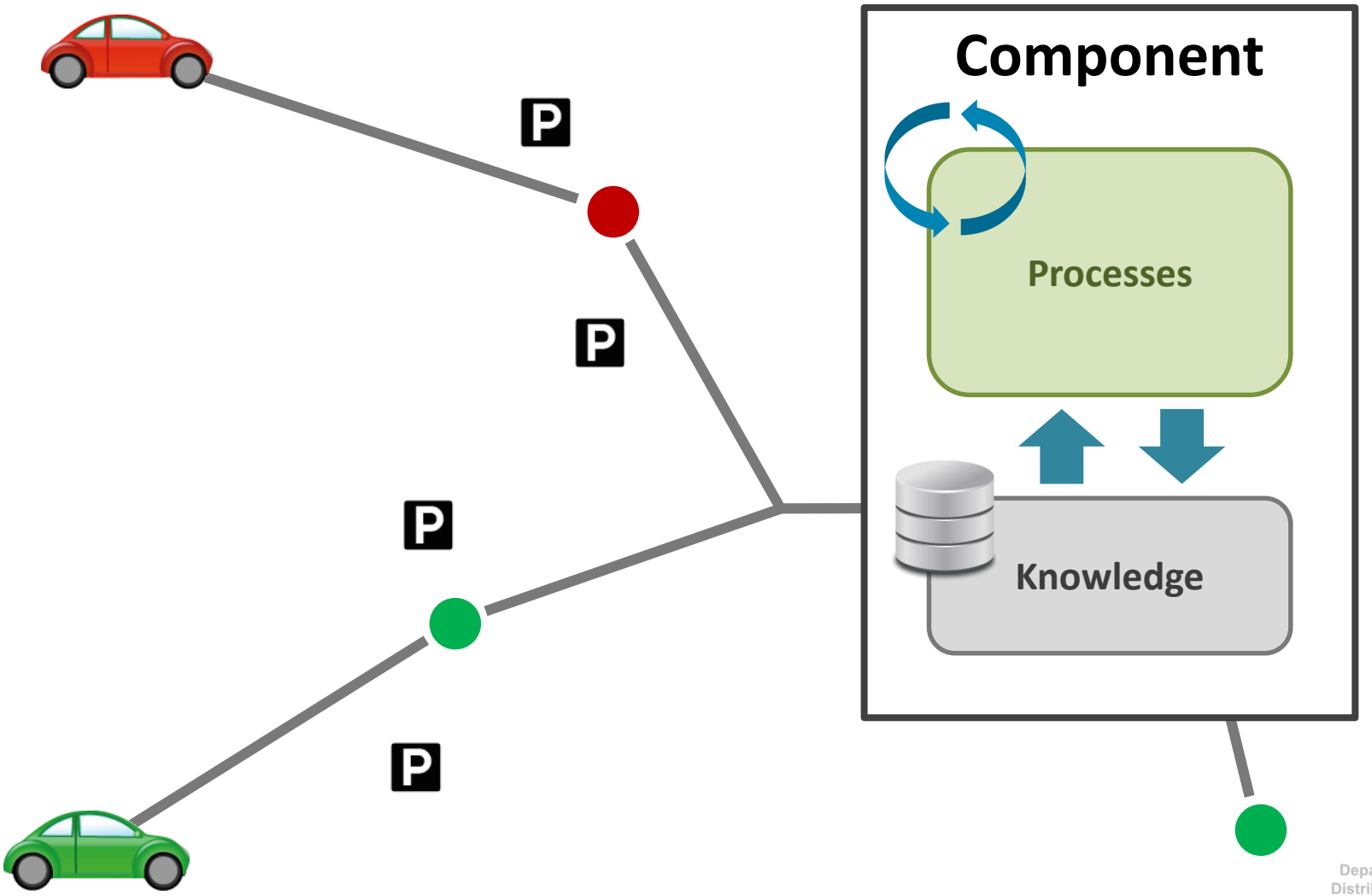
- Components
  - Knowledge
    - Local data + belief
  - Processes
    - Cyclic execution
    - Local knowledge only



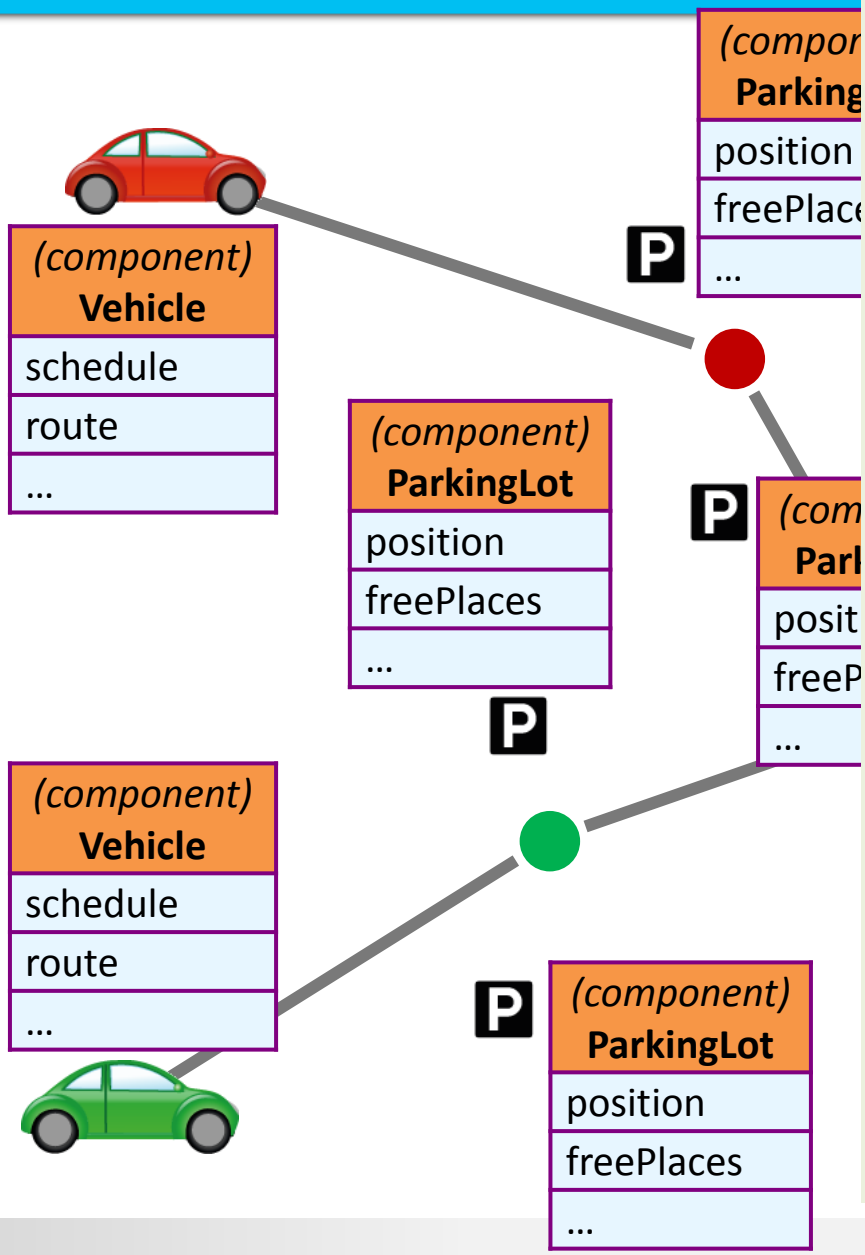
- Ensembles
  - Membership
    - Declarative
  - Knowledge exchange
    - Best-effort
    - Stateless
    - Belief



DEECo – Concepts



DEECo – Concepts



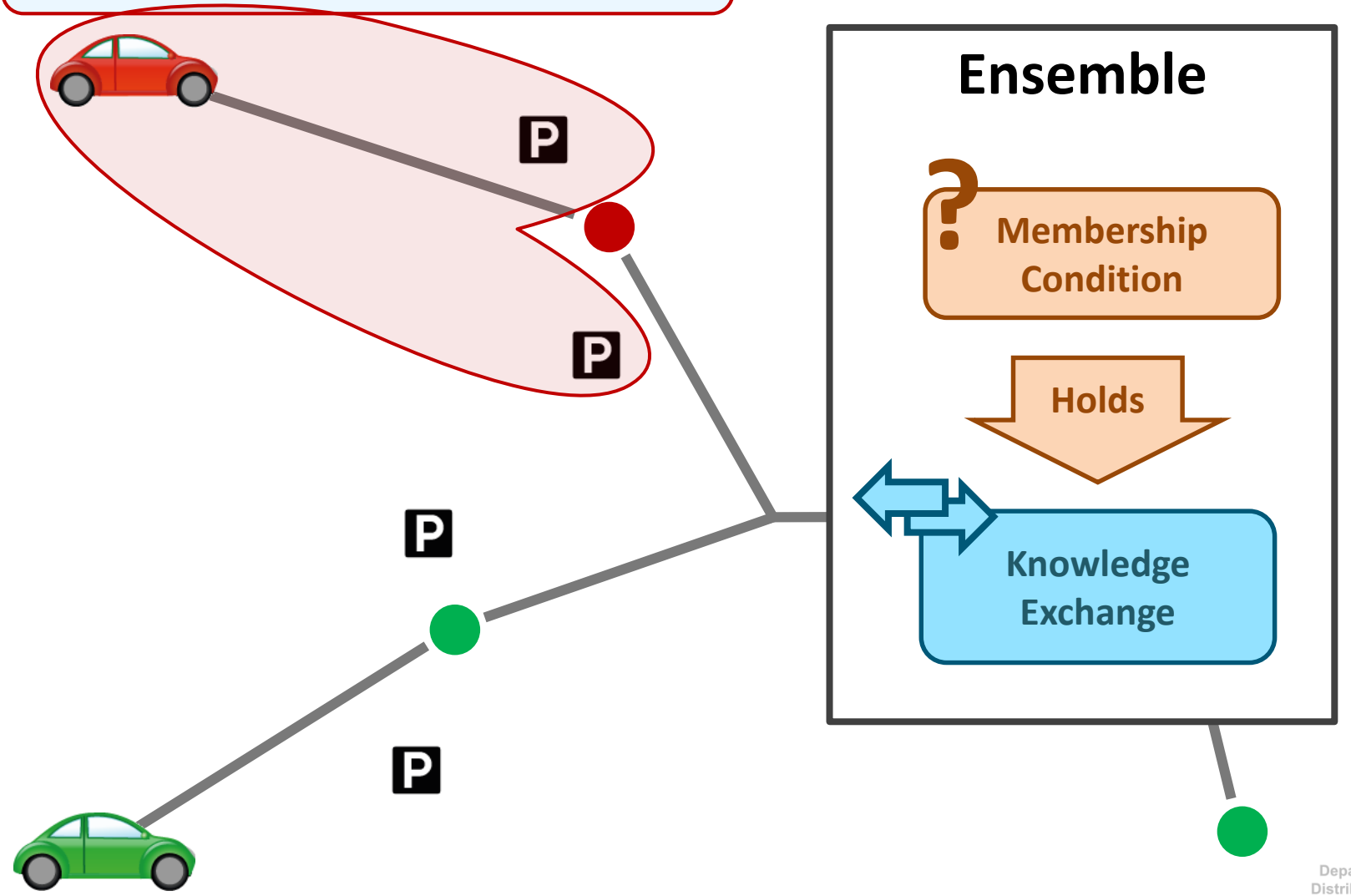
Component *Vehicle* = {
  position: *IPosition*
  availableParkingLots: *IParkingLot[]*
  route: *IRoute*
  schedule: ISchedule
  ...
  process updatePlan {
    function = updatePlan
    inputKnowledge = [position,
      availableParkingLots, ...]
    outputKnowledge = [route, ...]
    scheduling = periodic(1s)
   ...
} }

Component *ParkingLot* = {
  freePlaces*: Int*
  position: *IPosition*
  ...
  process updateFreePlaces {
    ...
  }
}

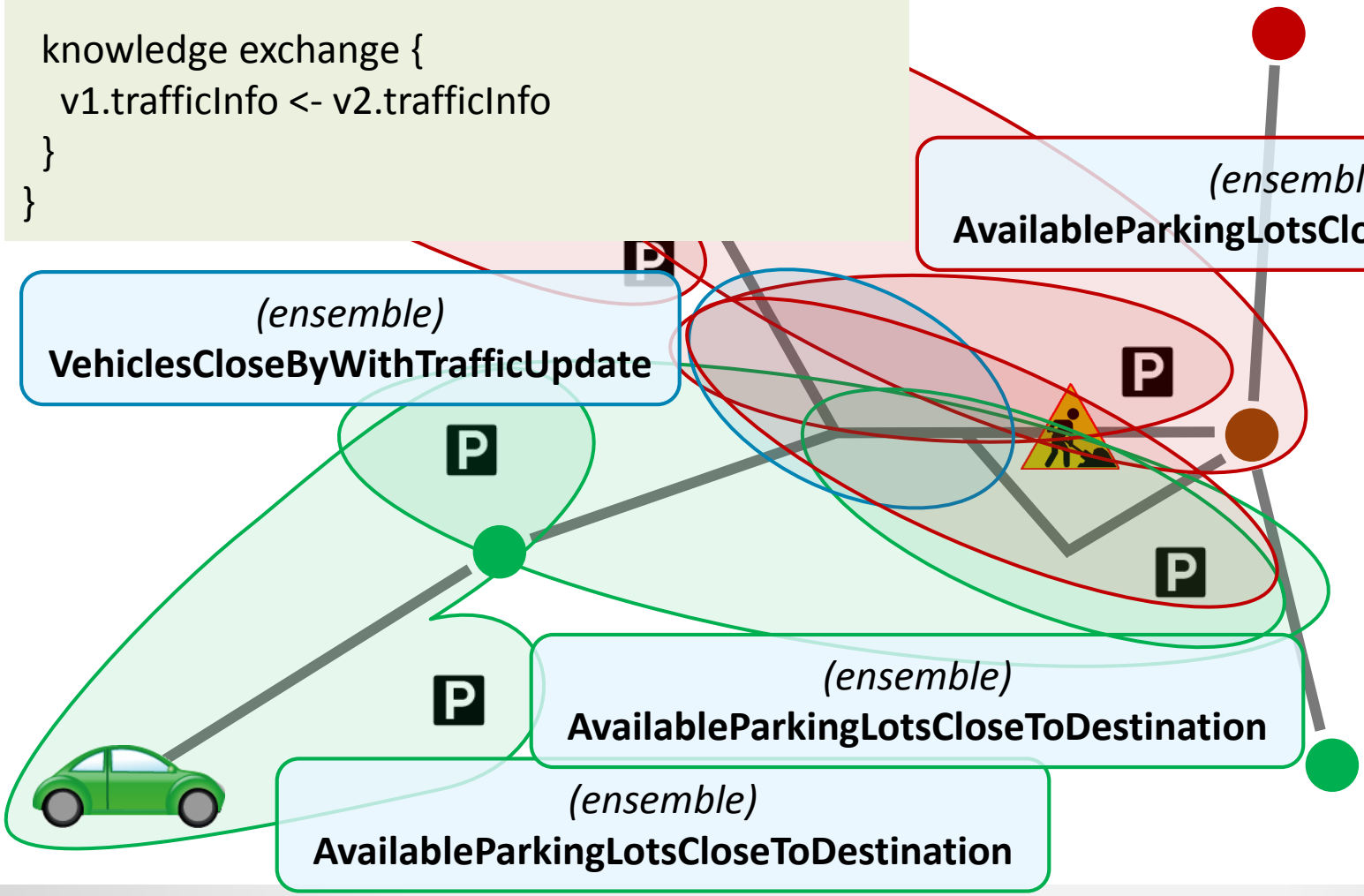
DEECo – Concepts

(ensemble)

AvailableParkingLotsCloseToDestination

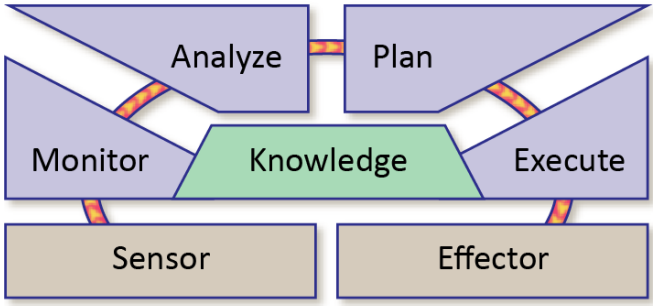


```
Ensemble VehiclesCloseByWithTrafficUpdate {
  v1: IVehicle
  v2: IVehicle

  membership :
    proximity(v1.position, v2.position) <= DIST_THR

  knowledge exchange {
    v1.trafficInfo <- v2.trafficInfo
  }
}
```



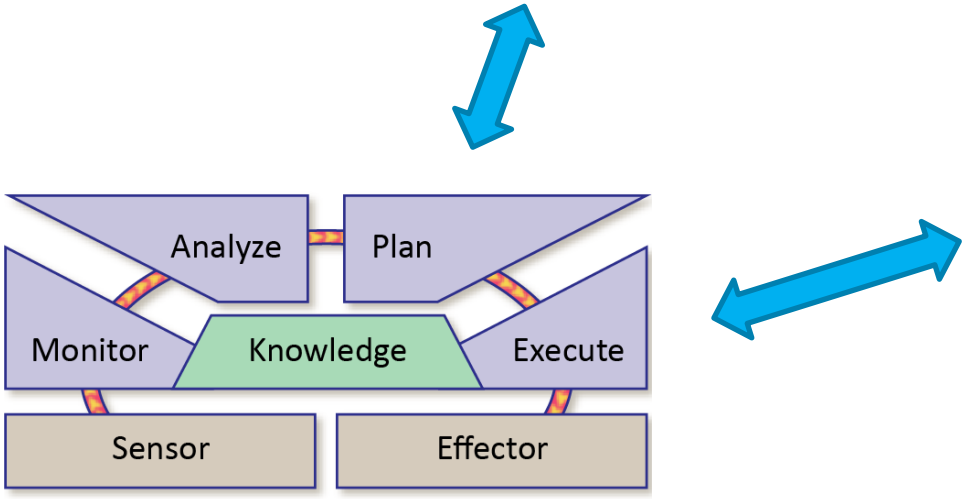
Component Ensembles

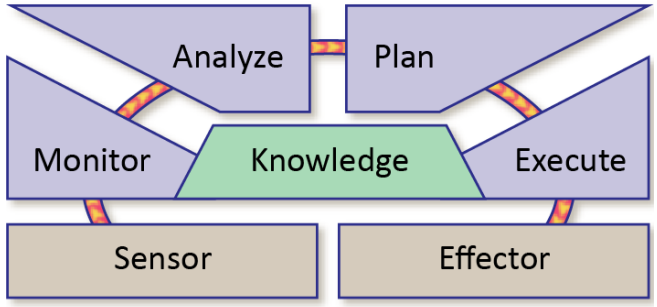
Can be seen as a system of conditionally interacting MAPE-K loops



MAPE-K Loop

- Central concept of autonomic computing
- Introduced by IBM





Department of Distributed and Dependable Systems D3S

Formal Semantics

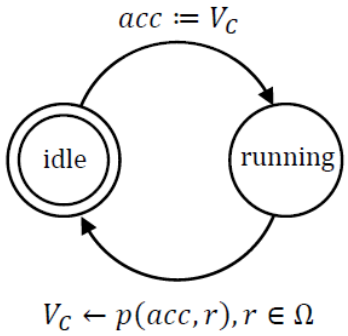
Communication matters

- Takes time, is unreliable

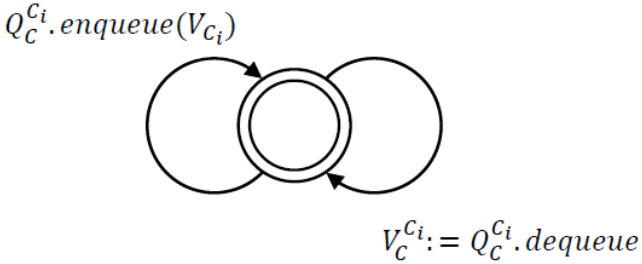
Components and ensembles have strict time-based semantics

- Cartesian product of automatons of three types:

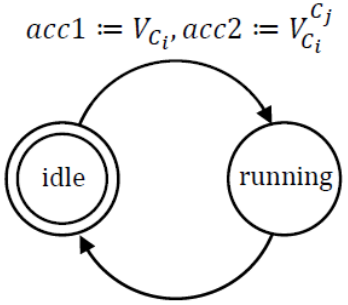
Component processes



Network communication

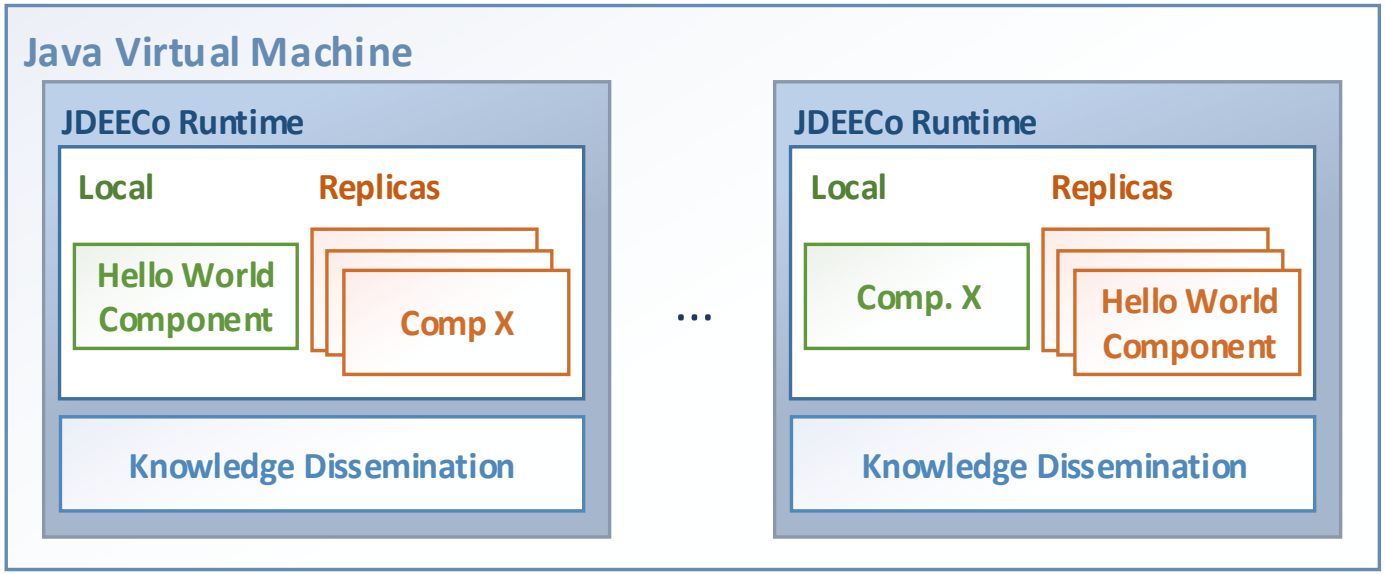


Ensembles



$$A(S) = \prod_{\forall C \in \mathbb{C}} \prod_{\forall p \in P_C} A(p) \times \prod_{\substack{\forall C_i, C_j \in \mathbb{C}, \\ C_i \neq C_j}} A\left(Q_{C_i}^{C_j}\right) \times \prod_{\forall E \in \mathbb{E}} A(E)$$

JDEECo – Framework



Example 0

- Hello world component
  - Periodically prints out “Hello world”
  - In parallel increments a counter at a double speed

All examples along with instructions how to launch them can be found at:
https://github.com/d3scomp/cbse-tutorial

Hello World Component

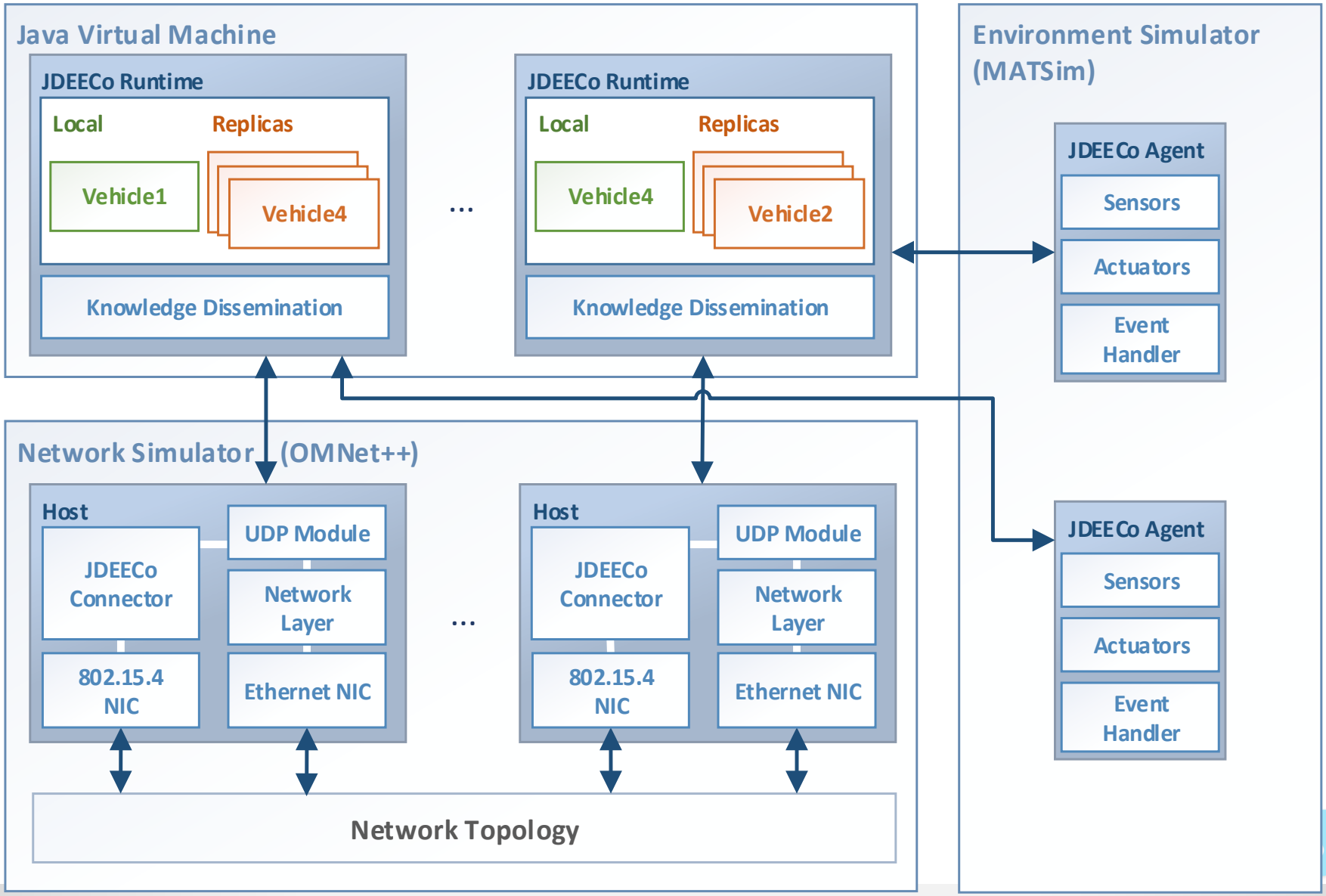
```
@Component
public class HelloWorld {

  /**
   * Id of the vehicle component.
   */
  public String id;

  /**
   * Output of count process
   */
  public int counter;

  public HelloWorld(String id) {
    this.id = id;
    this.counter = 1;
  }
```

Knowledge

```
  /**
   * Periodically prints "Hello world!"
   */
  @Process
  @PeriodicScheduling(period=1000)
  public static void sayHello(@In("id") String id) {
    System.out.format("Hello world!\n");
  }

  /**
   * Periodically increments the counter.
   */
  @Process
  @PeriodicScheduling(period=500)
  public static void updateCounter(
    @InOut("counter") ParamHolder<Integer> counter
  ) {
    counter.value ++;
  }

}
```

Processes

JDEECo – Framework (Simulation)



Example 1

- Vehicle component
  - Randomly generates place to go
  - When it gets there, prints out a message and generates another random destination
  - Keeps doing this ad infinitum
  - In parallel, periodically prints out the status of the component

All examples along with instructions how to launch them can be found at:
https://github.com/d3scomp/cbse-tutorial

Example 2

- Components as in Example 1
- Components form ensembles
  - When more components come close, they all follow the destination of the component with the lowest ID.

All examples along with instructions how to launch them can be found at:
https://github.com/d3scomp/cbse-tutorial

Ensemble

```
@Ensemble
@PeriodicScheduling(period = 1000)
public class FollowerLeaderEnsemble {

public static final double ENSEMBLE_RADIUS = 2000.0; // in meters
```

Membership condition

```
@Membership
public static boolean membership(
  @In("member.id") String mId,
  @In("coord.id") String cId,
  @In("member.position") Coord mPos,
  @In("coord.position") Coord cPos) {

  return getEuclidDistance(cPos, mPos) <= ENSEMBLE_RADIUS && cId.compareTo(mId) == -1;
}
```

Knowledge exchange

```
@KnowledgeExchange
public static void exchange(
  @In("coord.destinationLink") Id cDestinationLink,
  @Out("member.leaderDestinationLink") ParamHolder<Id> mLeaderDestinationLink) {

    mLeaderDestinationLink.value = cDestinationLink;
}

...
}
```

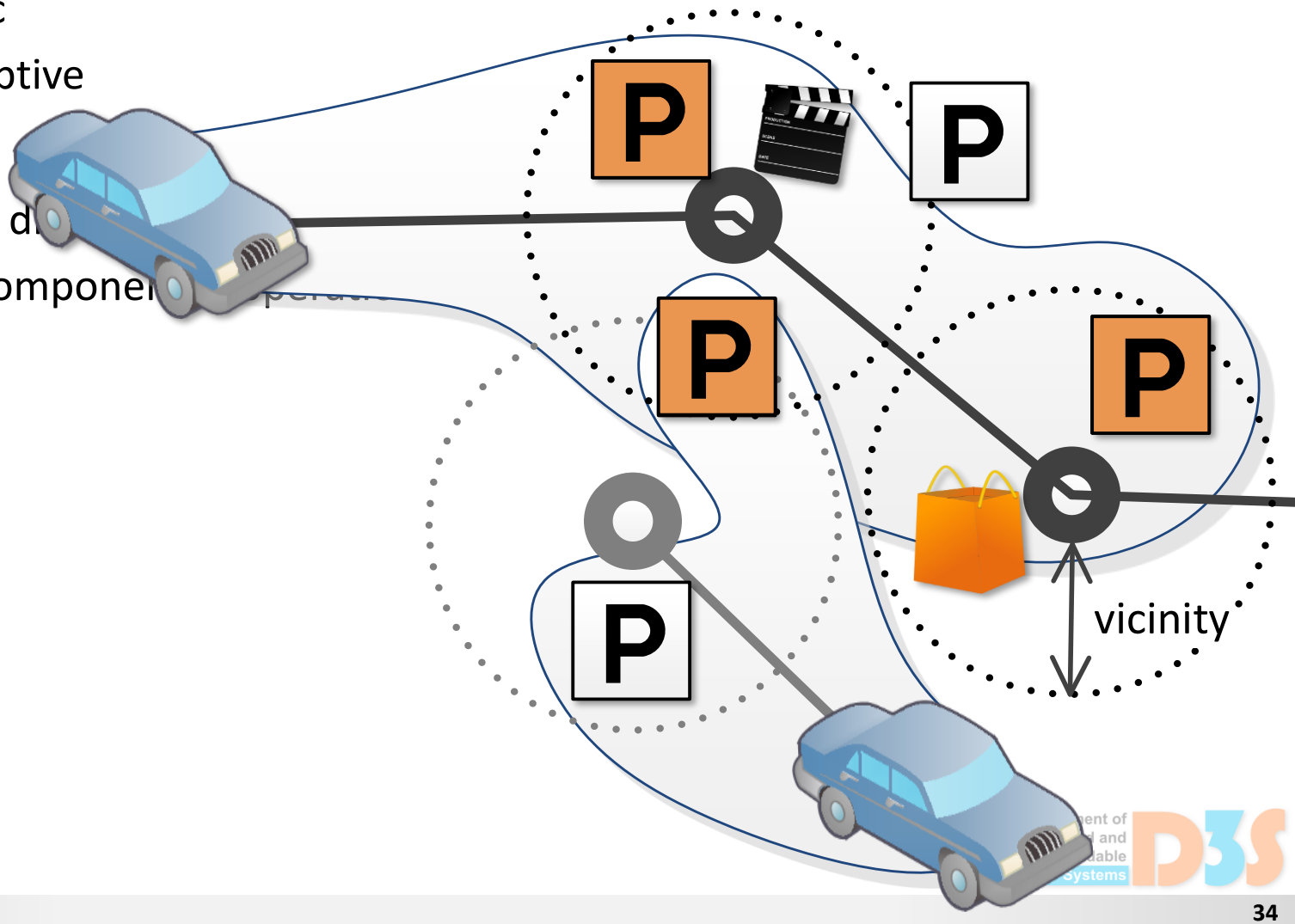
Department of Distributed and Dependable Systems

Part 4

Goal-based design of sCPS
Invariant Refinement Method (IRM)

Recall: Ensemble-Based Component Systems

- Components
  - autonomic
  - (self-) adaptive
- Ensembles
  - emergent, d
  - mediate compone



Design Process

- Problem:
  - Component ensembles have relatively exotic computational model
    - Very far from classical procedure call-based sequential programming
    - Much closer to design of real-time systems – but also adaptivity and open-endedness
  - Method for high-level design are necessary
    - To help developers “think” about such systems
    - Requirements ⟹ … ⟹ Components + Ensembles

Challenge

High-level
System Requirements

All vehicles meet their route/parking calendars

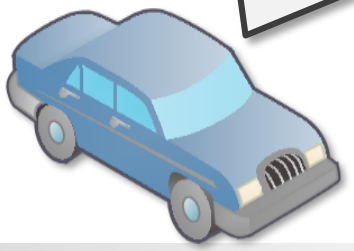
?

Formally grounded, rigorous refinement

Low-level Design
DEECo Processes/Ensembles

- Compute a route plan
- Every 5s check the plan feasibility
- Re-compute the plan if infeasible or once every 60s.

- Every 15s monitor the availability



Every 10s update the vehicle's belief about availability of nearby parking lots

P

Classical Approaches

Use-cases, User stories, …

Use-case Example

Schedule meeting

1. User enters the possible dates of the meeting
2. Use enters e-mails of the participants
3. System validates the e-mail addresses
4. System sends an e-mail with an invitation to each participant
5. System confirms e-mails being sent

…

Describes “how” instead of “what”.

Inherently less adaptable/evolvable.

Goal-oriented model-building at RE time

Goal-orientation enables:

- early, incremental analysis
- completeness and pertinence of the model
- reasoning about alternative options
- validation by stakeholders
- backward traceability

Thinking about goals in the early phases of software development is a natural thing; in GORE it is just made explicit

Approaches in GORE

- **KAOS**

"a GORE approach with a rich set of formal analysis techniques"

→ Axel van Lamsweerde et al.

KATHOLIEKE UNIVERSITEIT LEUVEN

- **¡***

"an agent-oriented modeling framework that can be used for requirements engineering"

→ Eric Yu

UNIVERSITY OF TORONTO

UNIVERSITÀ DEGLI STUDI DI TRENTO

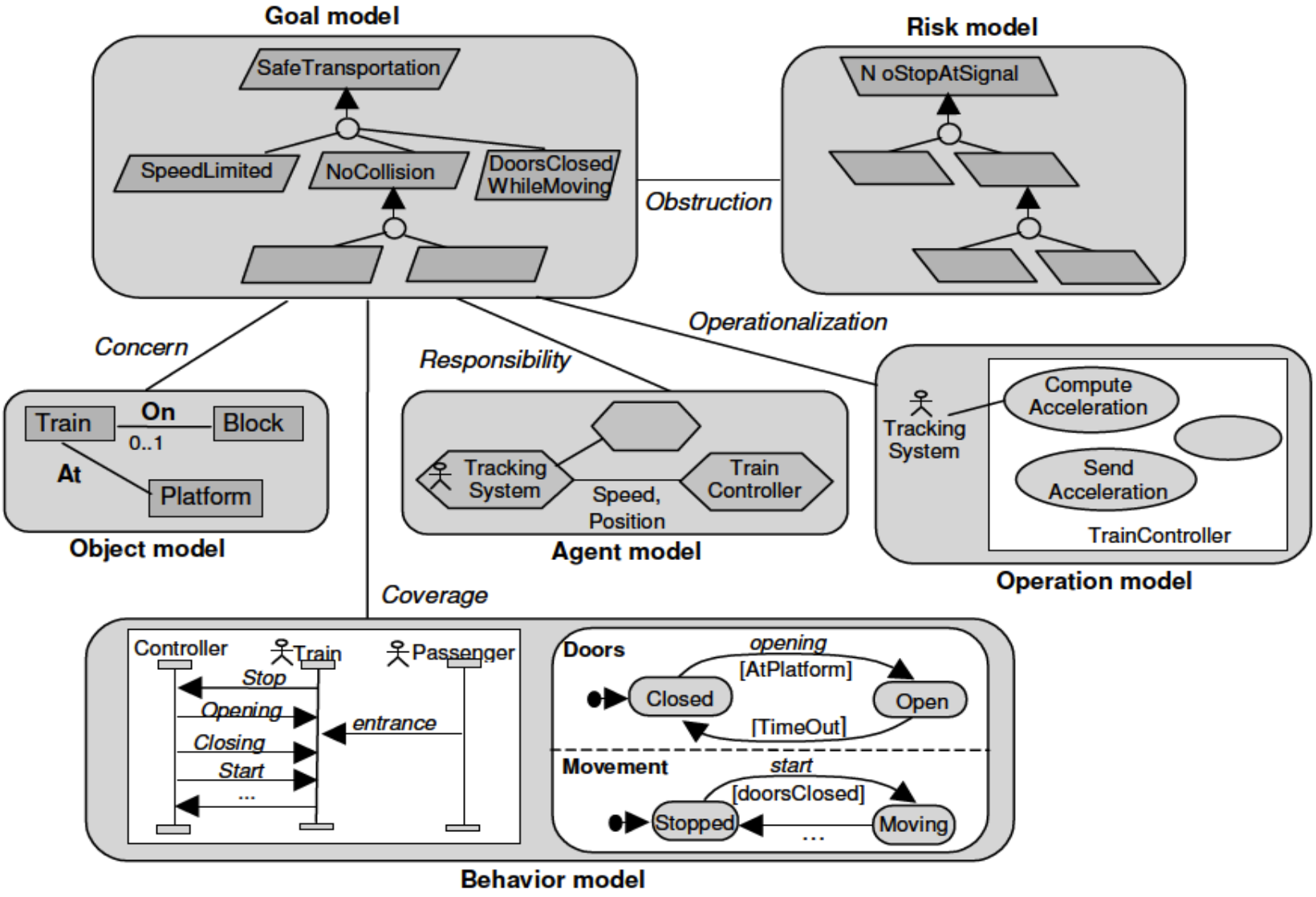
- **TROPOS**

"an agent-oriented software engineering methodology"

→ John Mylopoulos et al.

Department of Distributed and Dependable Systems D3S

KAOS multi-view modeling



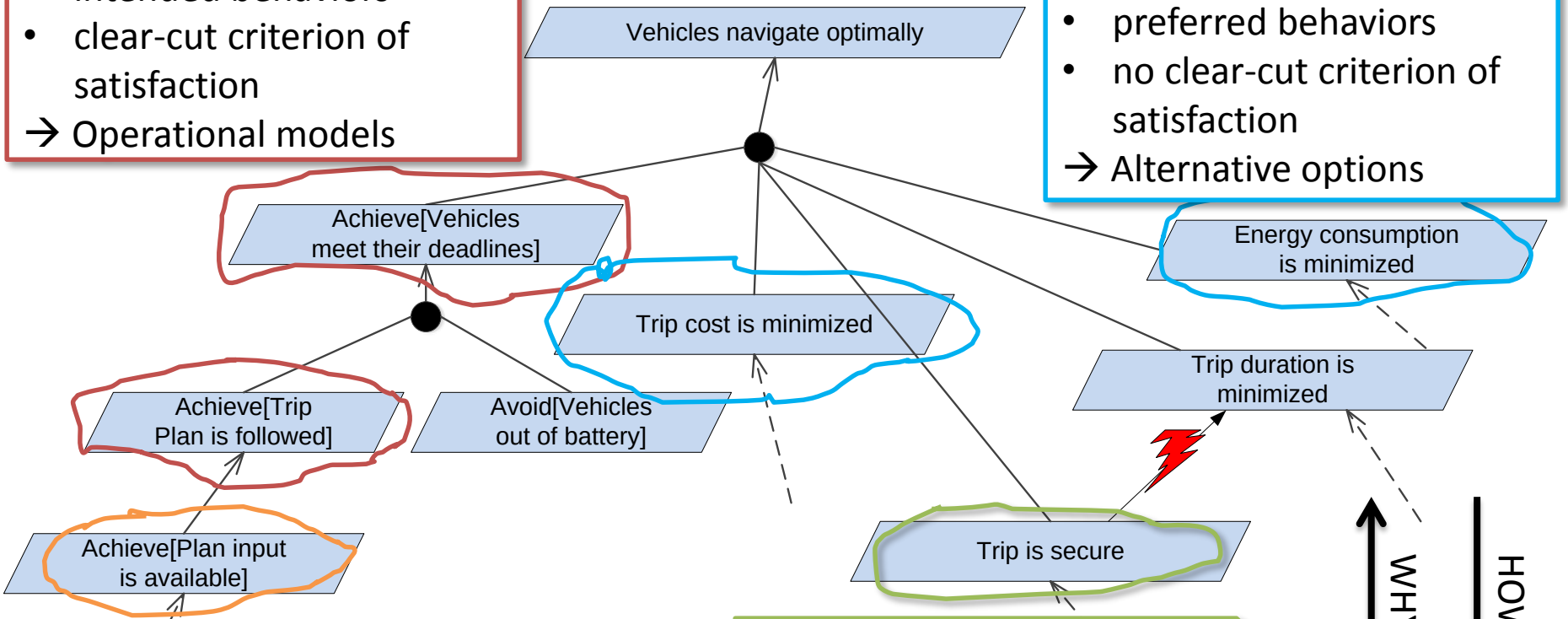
Goal model - I

behavioral (hard) goals:
- intended behaviors
- clear-cut criterion of satisfaction

→ Operational models

soft goals:
- preferred behaviors
- no clear-cut criterion of satisfaction

→ Alternative options

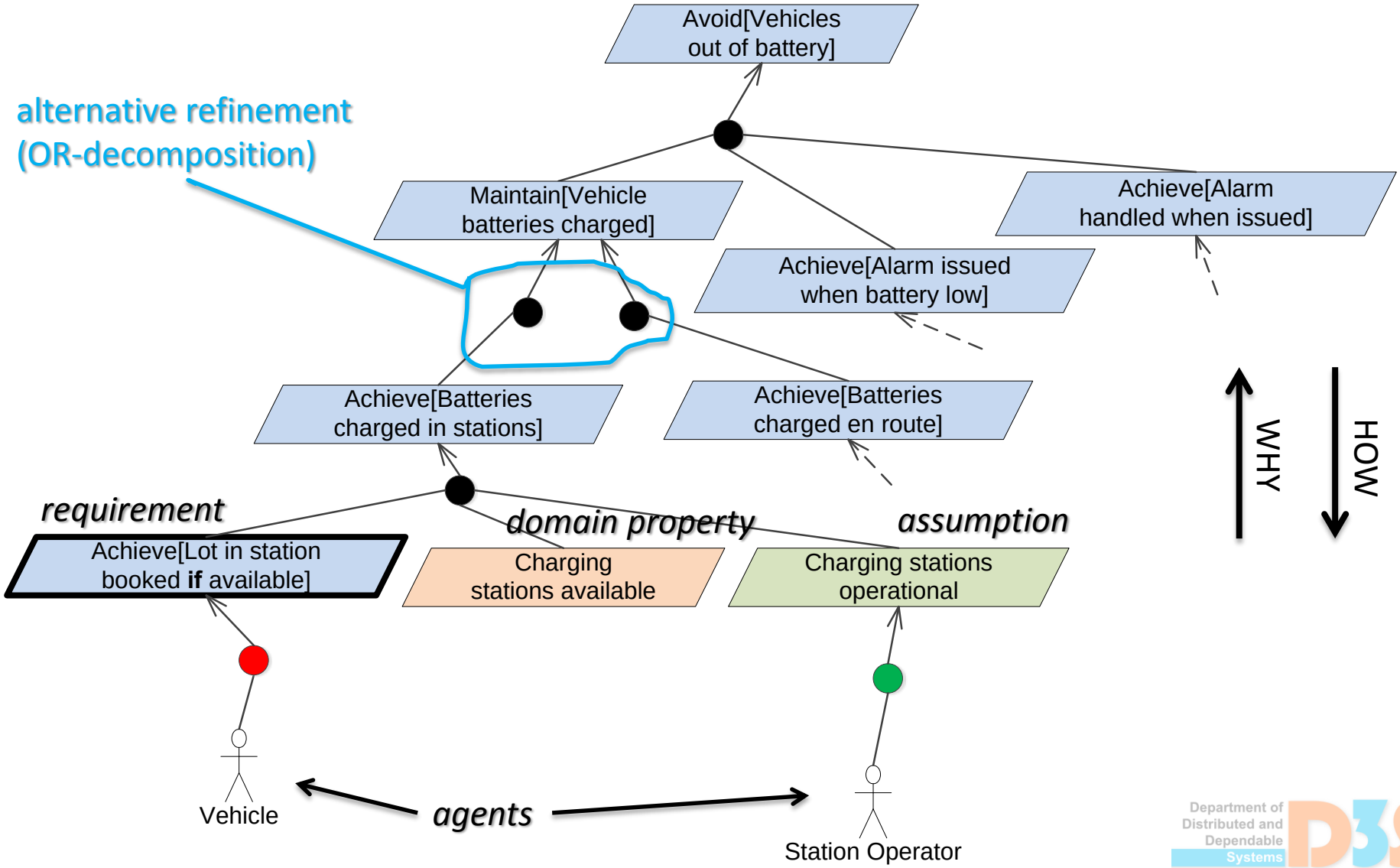


functional goals:
Underlying operation, feature, service, task

non-functional goals:
quality goals e.g. security, accuracy, architectural, development goals, etc.

Department of Distributed and Dependable Systems D3S

Goal model - II



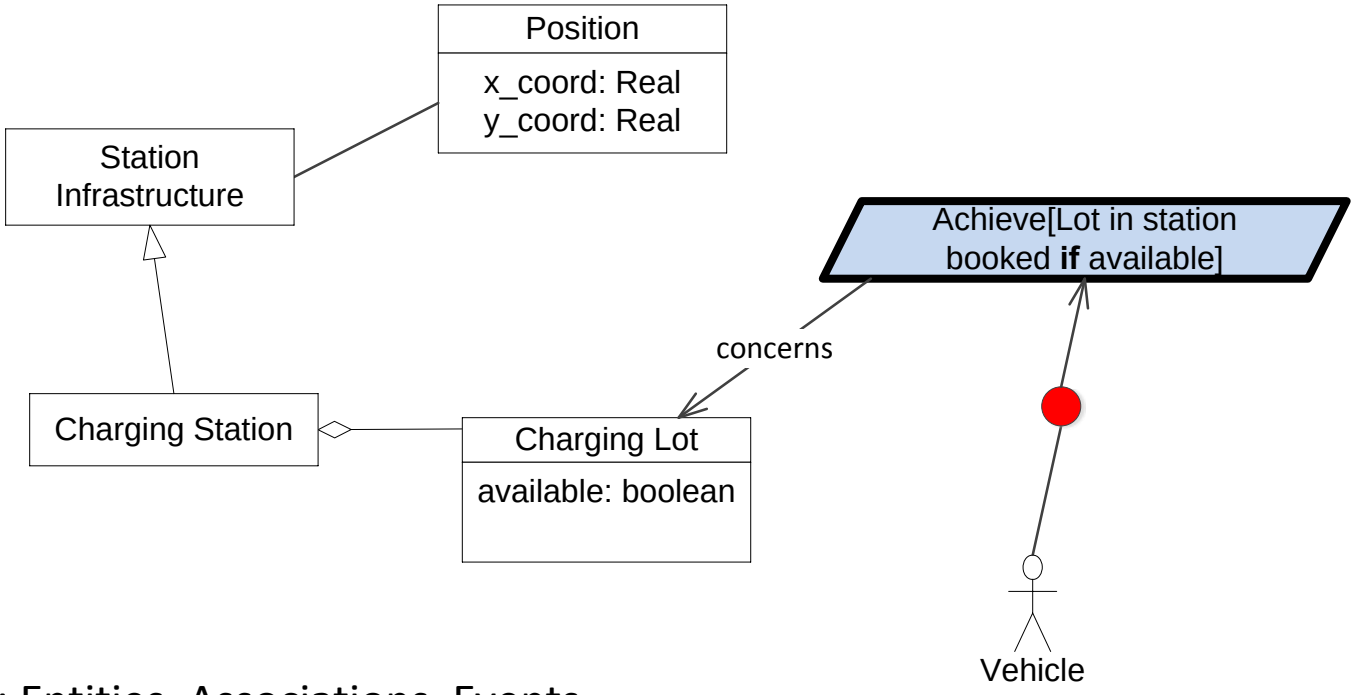
Formal Specification of Goals

Name Lot in Station booked **if** available
Def If a place is available, then it must be booked by the vehicle in order to recharge
Type Achieve
Category Satisfaction
Source interview with VW
Priority Medium
FormalSpec
$\forall$ v: Vehicle, cl: ChargingLot:
LowBattery(v) $\wedge$ Available(cl) $\Rightarrow$
$\Diamond_{\leq T}$ Booked(v,cl)

Achieve[Lot in station booked **if** available]

Real-time linear temporal logic: o P, $\Diamond$ P, P U N, P W N, *and operators on past*

Object model



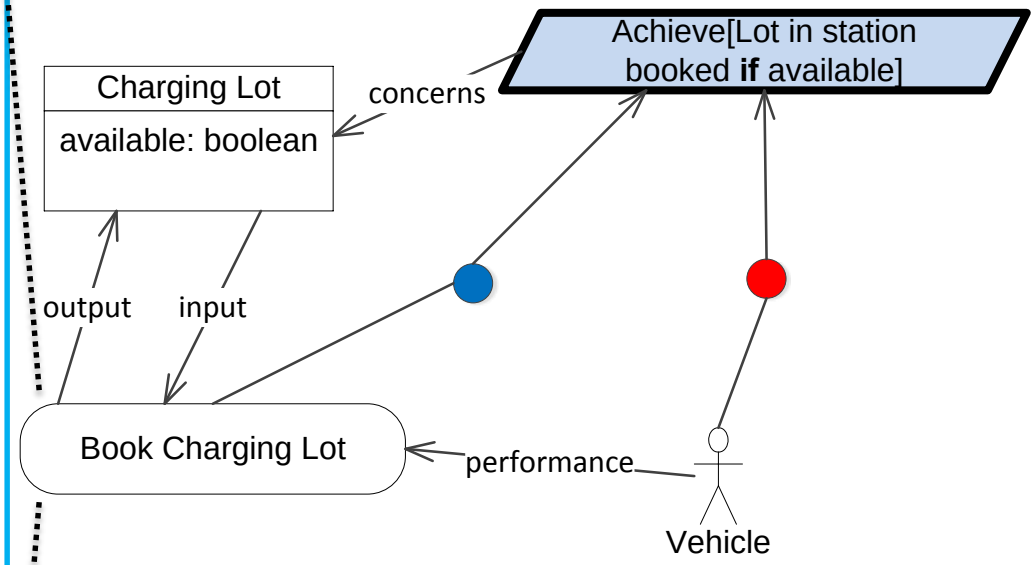
Objects: Entities, Associations, Events

Structure/Object model: UML class diagram notation

Only objects concerned in/referenced by goals are described

Operations model

Operation BookChargingLot
Def If a place is available, then it must be booked by the vehicle in order to recharge
Input cl: ChargingLot, v: Vehicle
Output cl
DomPre cl.available = true
DomPost cl.available = false
ReqPre …
ReqTrig for LotInStationBookedIfAvailable: LowBattery(v) ∧Close(v,cl)
ReqPost …



DomPre, *DomPost*: what the operation means in the domain

ReqPre, ReqTrig, ReqPost: additional strengthening to ensure the associated goal

What Goals provide in KAOS

sufficient completeness criterion:

A specification is complete **with respect to a set of goals** if all the goals can be proven to be achieved from the specification and the properties known about the domain.

pertinence criterion:

A requirement is pertinent **with respect to a set of goals** if its specification is used in the proof of at least one goal.

Goals refinement checking

A refinement of goal G into subgoals SG_1, …, SG_n is correct, when it is

- complete: $\{SG_1, \ldots, SG_N, DOM\} \models G$
- consistent: $\{SG_1, \ldots, SG_N, DOM\} \not\models false$
- minimal: $\{SG_1, \ldots, SG_{j-1}, SG_{j+1}, \ldots, SG_n, DOM\} \not\models G$

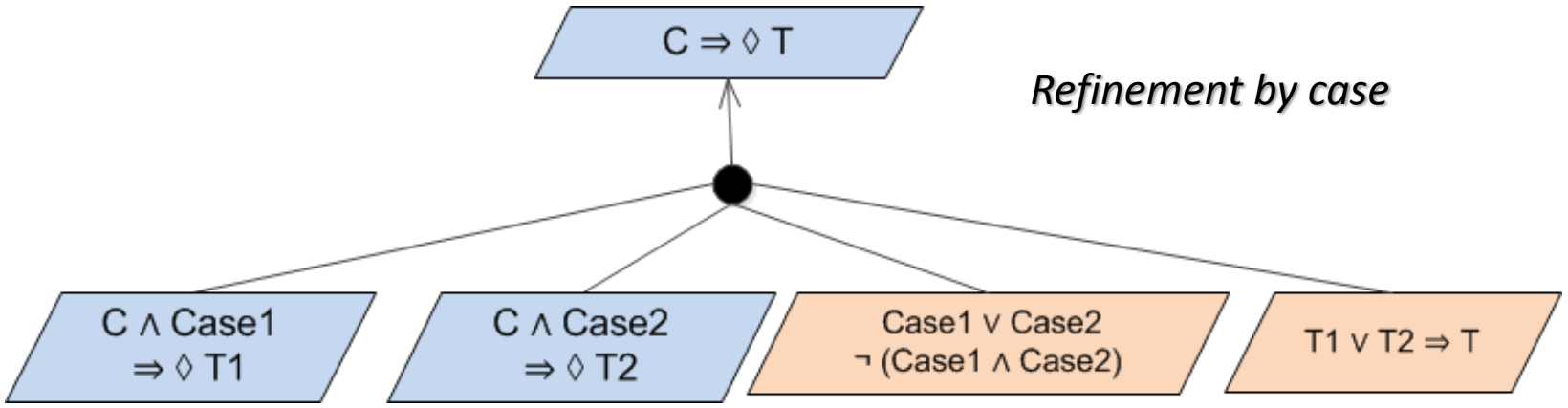
How to check goal refinements?

1. Use LTL theorem prover
   - heavyweight, nonconstructive
2. Use bounded SAT solver
   - input: $SG_1 \wedge \ldots \wedge SGn \wedge Dom \wedge \neg G$
   - incremental check/debug
3. Reuse refinement patterns

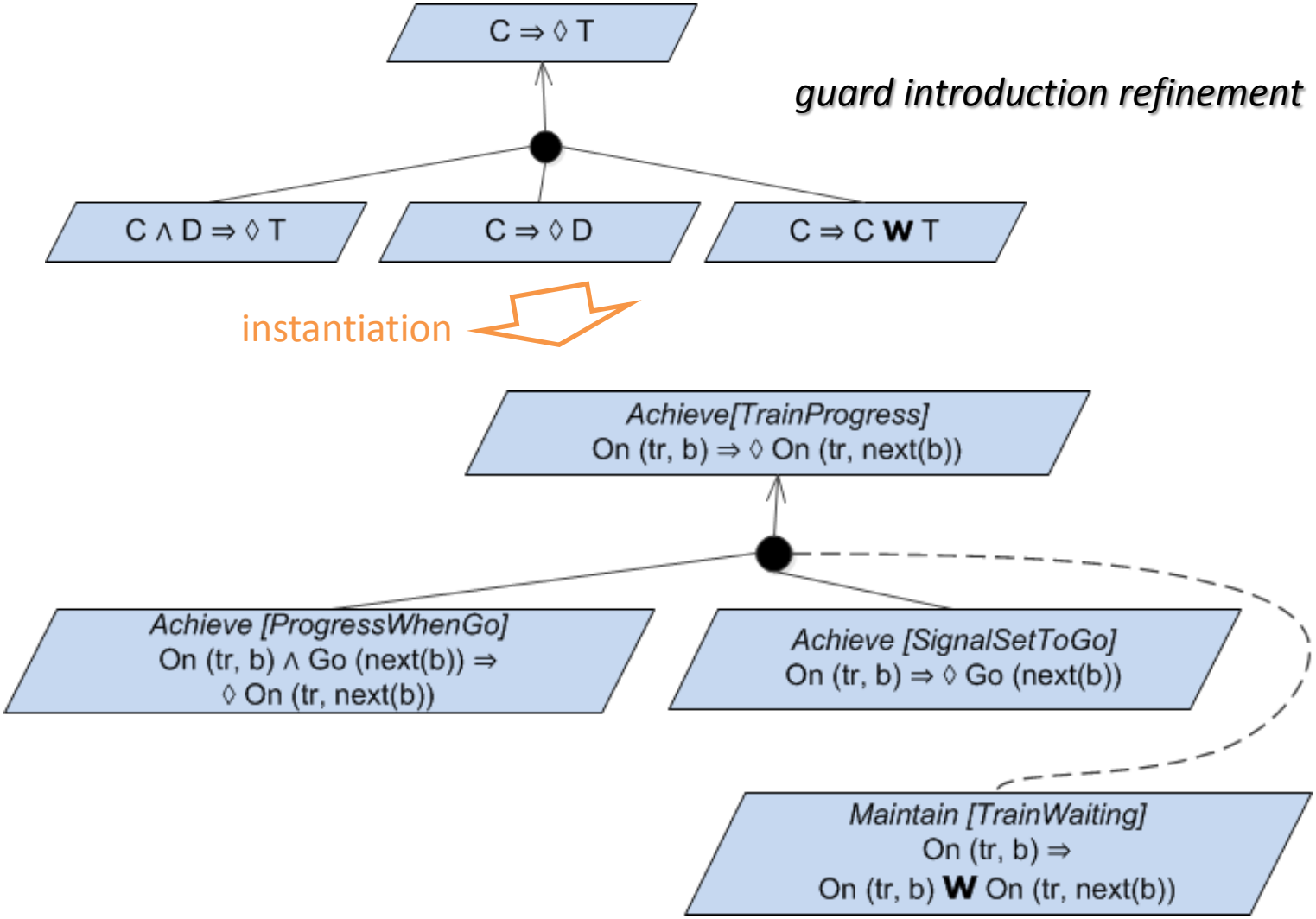
Refinement patterns - I

- Catalogue of patterns encoding *refinement tactics*
- Generic refinements proved formally, once for all
- Reuse through instantiation, in matching situation

Examples:

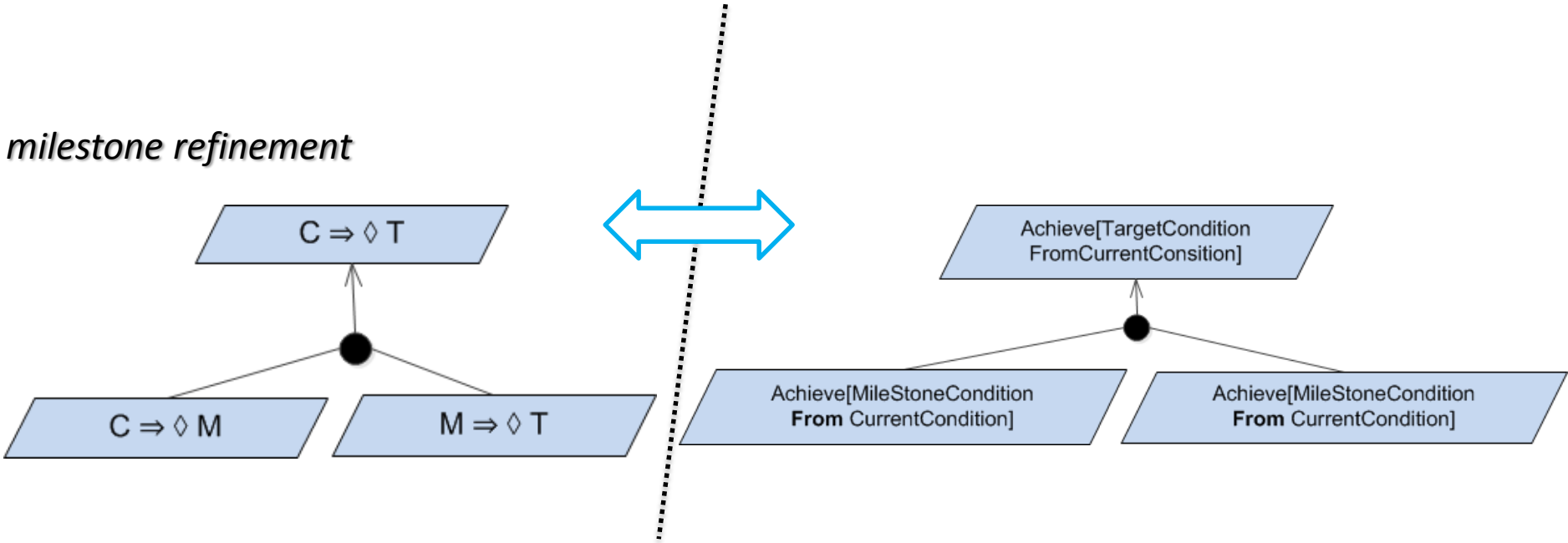


Refinement patterns - II



Refinement patterns - III

milestone refinement



Formal pattern vs informal guideline

Apart from goal refinement, patterns can be applied:

- *Goal operationalization*
- *Obstacle analysis*

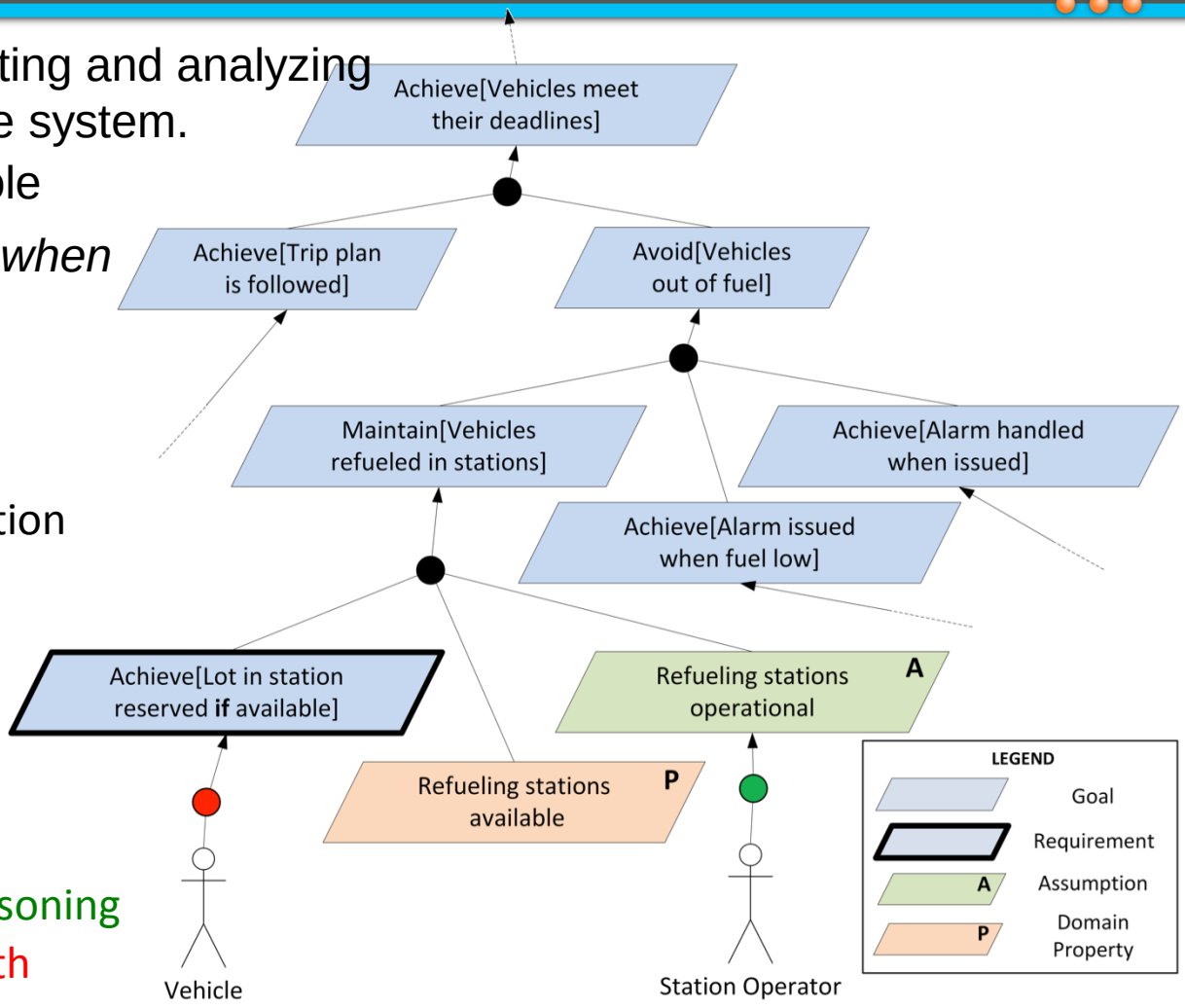
Requirements modeling – KAOS

Goal-oriented method for eliciting and analyzing the requirements of a software system.

- Goals have a prominent role
- Formal methods are used *when* and *where* needed

Goal model
Agent model
Object model
Operation model
Behavior model

} KAOS specification



Applicability in design of EBCS:

\+ captures the (intended) system behavior at a high level

\+ allows for automatic formal reasoning

– does not align requirements with architecture

– is intended for requirements analysis and documentation, not system design

Department of Distributed and Dependable Systems D3S

Requirements modeling – Tropos

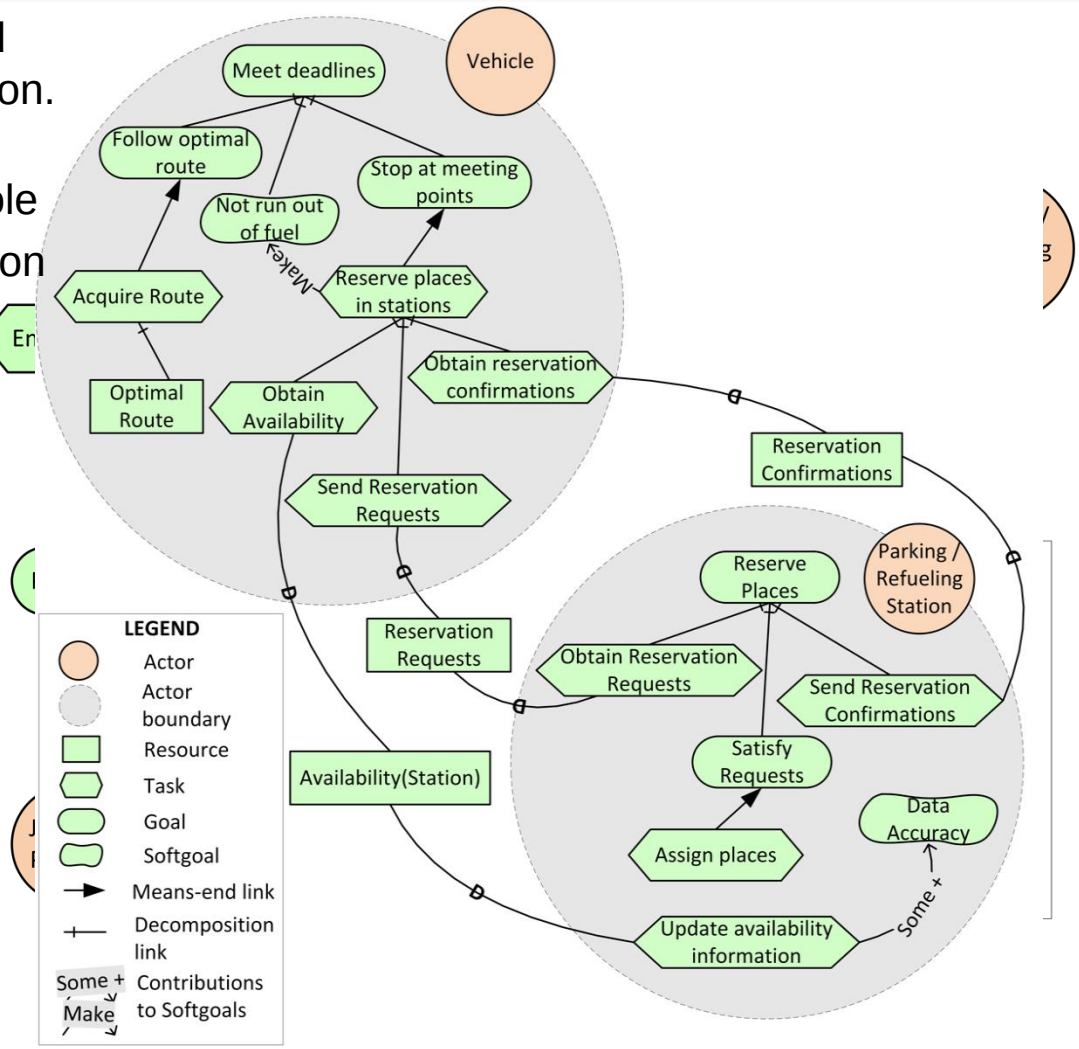
Methodology for building agent-oriented software systems that uses the i* notation.

- Agent and related notions (goals, plans, intentions) have prominent role
- Focus on early stages of SWD and on the organizational context

Goal models ⇨ Enhanced goal models

⇩

BDI architecture (JACK, JADE) ⇦ Actor-Agent mapping



Applicability in design of EBCS:

\+ aligns the requirements phase with architecture and implementation phases

\+ preserves a manageable set of concepts throughout the software development phases

– typically assumes static architecture (speaks about fixed instances)

– a bit ambiguous (goal or task?)

Detour: Resilient Systems

"A resilient control system is one that maintains state awareness and an accepted level of operational normalcy in response to disturbances, including threats of an unexpected and malicious nature"

[wikipedia]

Resilience

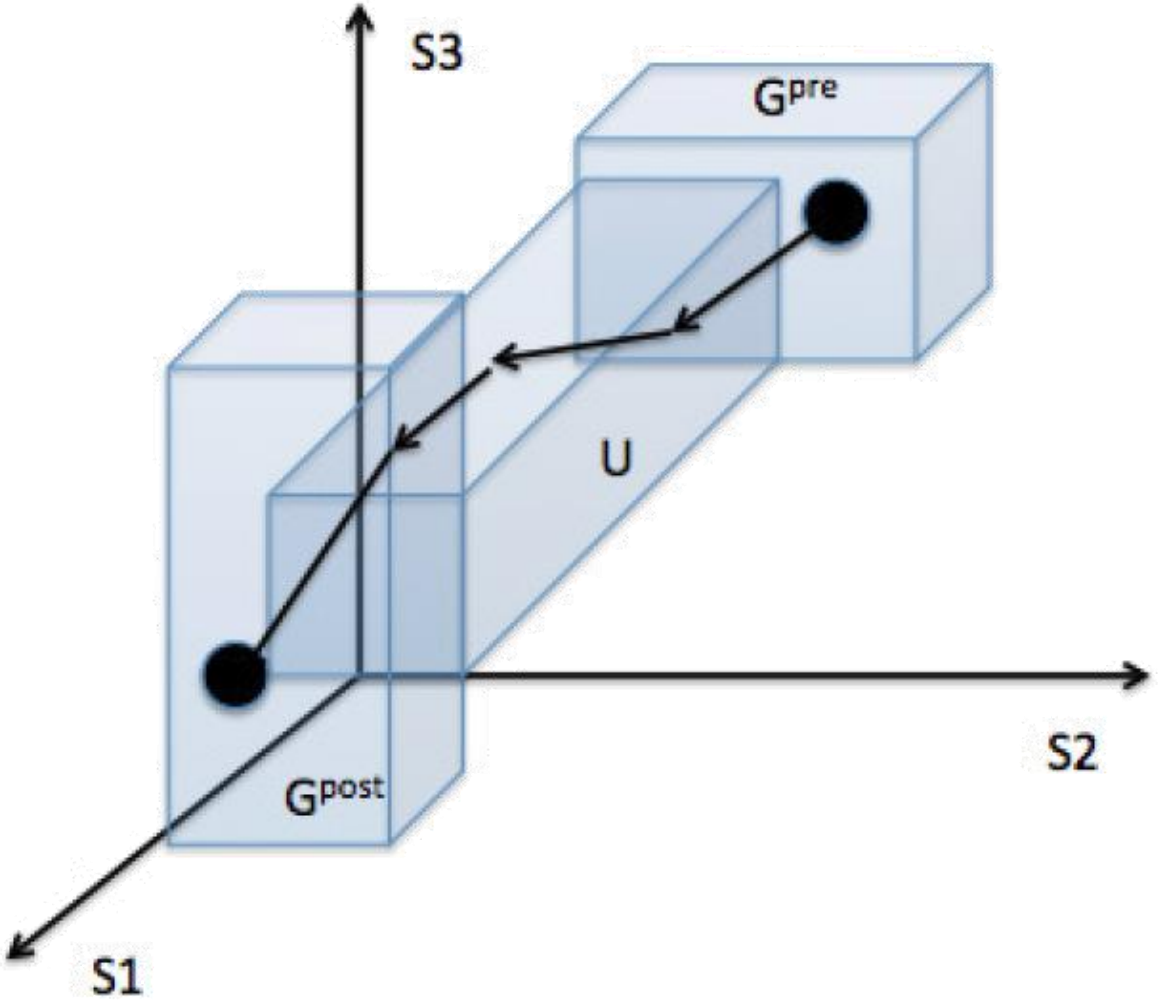
System adaptability

System evolvability

Impact on external environment

- Cooperative aspects

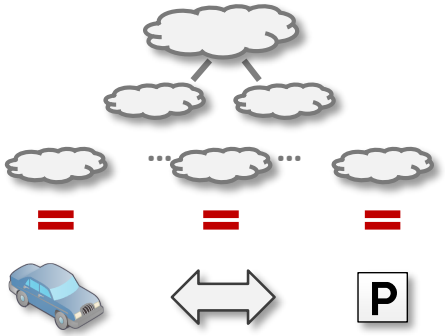
SOTA Model



IRM – Invariant Refinement Method

Systematic gradual refinement

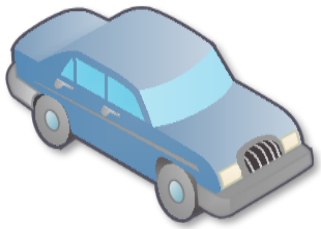
- Architecture design
- Conceptual framework & guidelines
- Borrows from goal-based requirements elaboration
  - KAOS, i*



Invariant (What is to be refined?)

- Describes the **operational normalcy** of a (sub)system
  - i.e., the desired (global) state of the system that should be preserved as the knowledge valuation evolves in time
- Suitable for expressing **both goals and low-level concepts**
- Syntactically a condition on knowledge valuation of a set of components

Refinement



All vehicles meet their route/parking calendars

P

A

Up-to-date V's **plan** w.r.t. P's availability reflecting V's calendar is available

V's position is aligned with its **plan**

An up-to-date **plan** can always be followed and it always schedules reaching the destination in time

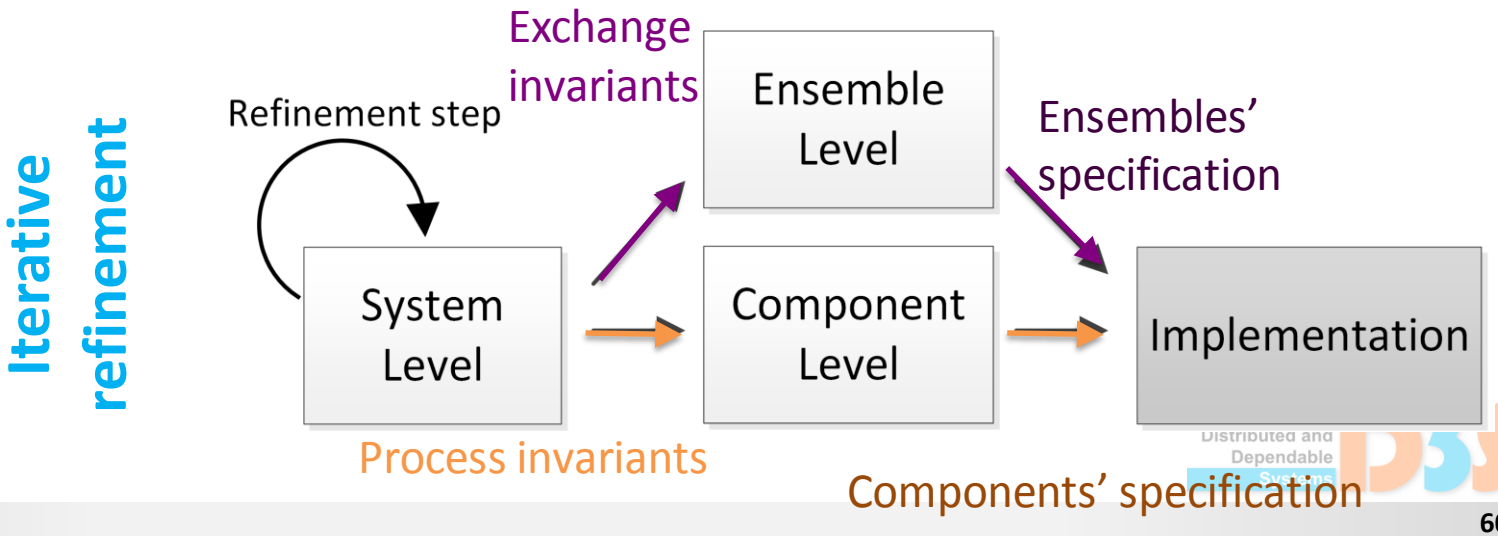
Department of Distributed and Dependable Systems D3S

Refinement

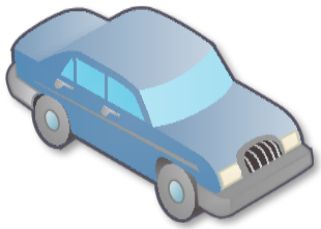
- **Decomposition** into a **conjunction**
- Formal interpretation
  - $Child_1 \wedge \cdots \wedge Child_n \Rightarrow Parent$ (entailment)
  - $Child_1 \wedge \cdots \wedge Child_n \not\Rightarrow false$ (consistency)
  - i.e., “traditional” refinement
- Involves **design decisions**
  - **New knowledge** can be added
  - Knowledge **dependencies** can occur

Leaves of Refinement (When to Stop?)

- Stop when an invariant is
  - Assumption
  - Can be “easily” mapped to a low-level execution concept
- **Process invariant**
  - Condition on knowledge of a single component
- **Exchange invariant**
  - A **belief** of a component vs. **knowledge** of another



Leaves of Refinement



P

All vehicles meet their route/parking calendars

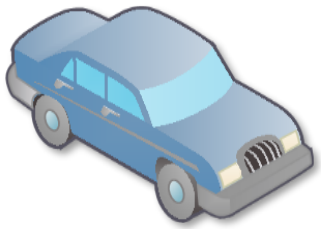
A

Up-to-date V's plan w.r.t. P's availability reflecting V's calendar is available

An up-to-date plan can always be followed and it always schedules reaching the destination in time

V's position is aligned with its plan

Leaves of Refinement



Up-to-date V's plan w.r.t. **P's availability** reflecting V's calendar is available

P

...

Up-to-date V's plan w.r.t. **V's belief over P's availability** reflecting V's calendar is available

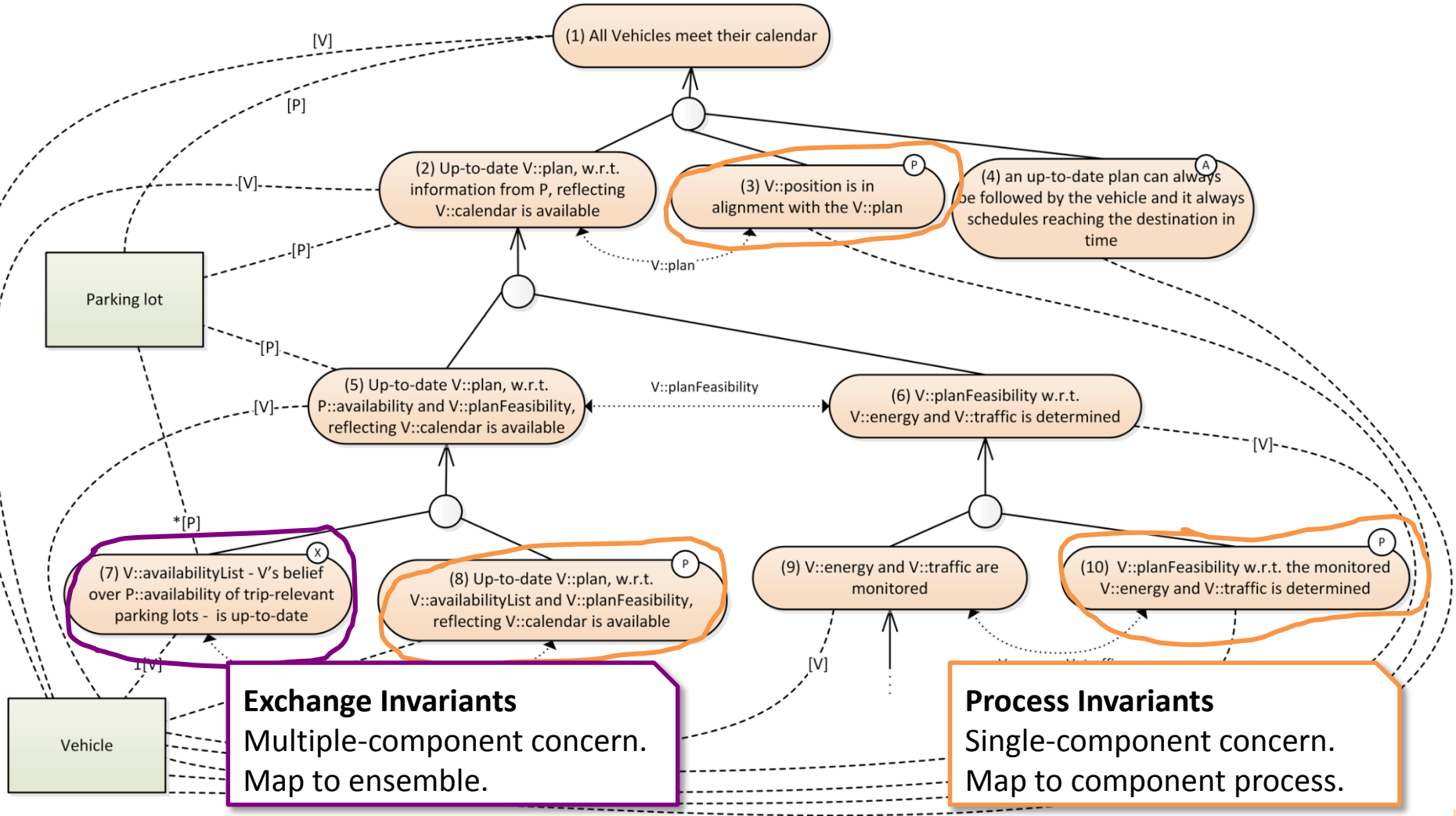
P

V's belief over P's availability corresponds to up-to-date **P's availability**

X

Department of Distributed and Dependable Systems D3S

IRM refinement tree

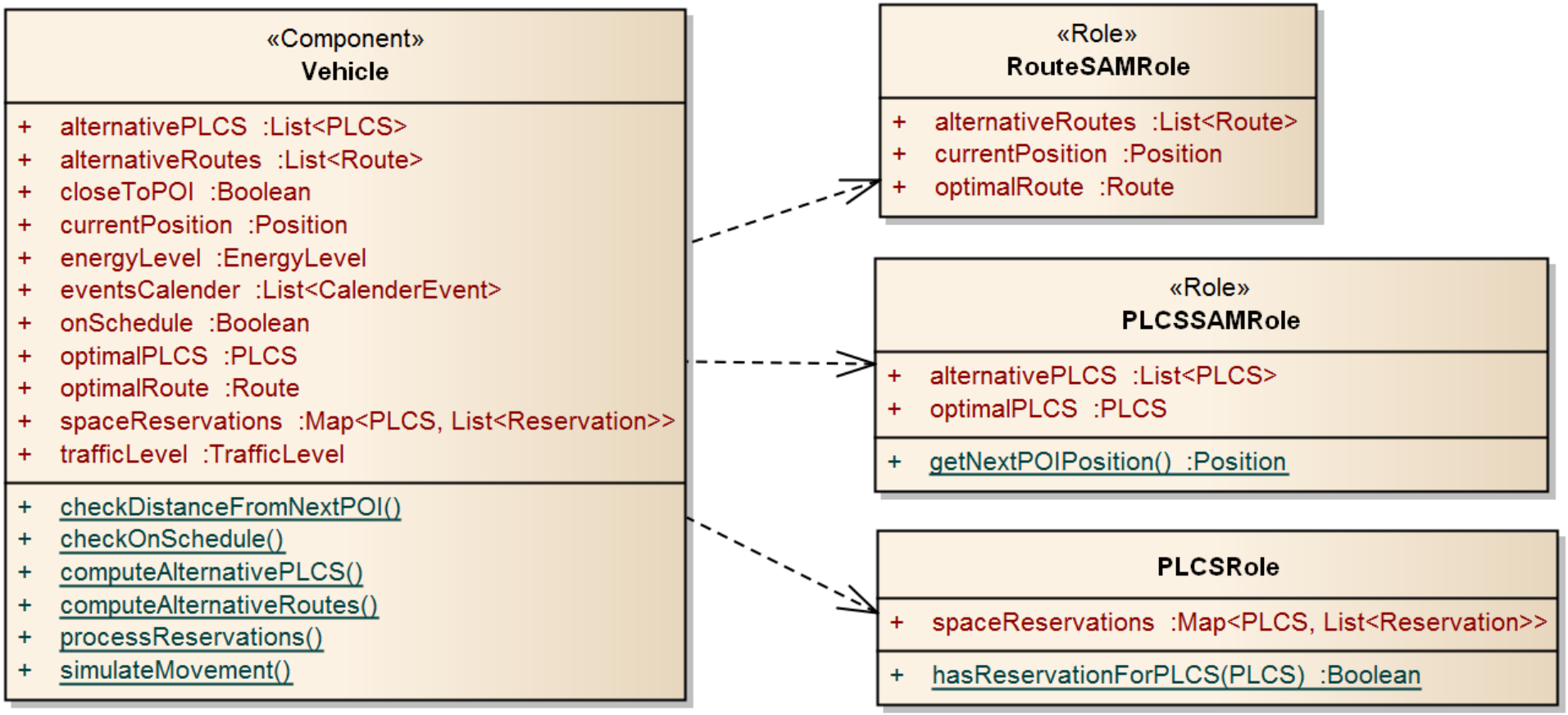


From Leaves to Detailed Design/Code

- Straightforward conversion
  - **Cyclic execution of processes/knowledge exchange maintains operational normalcy** (described by invariants)
- **Process**
  - all the inputs/outputs
  - post-condition/guarantee of the process
- **Ensemble**
  - the components/knowledge involved
  - the membership condition
  - post-condition/guarantee of the knowledge exchange

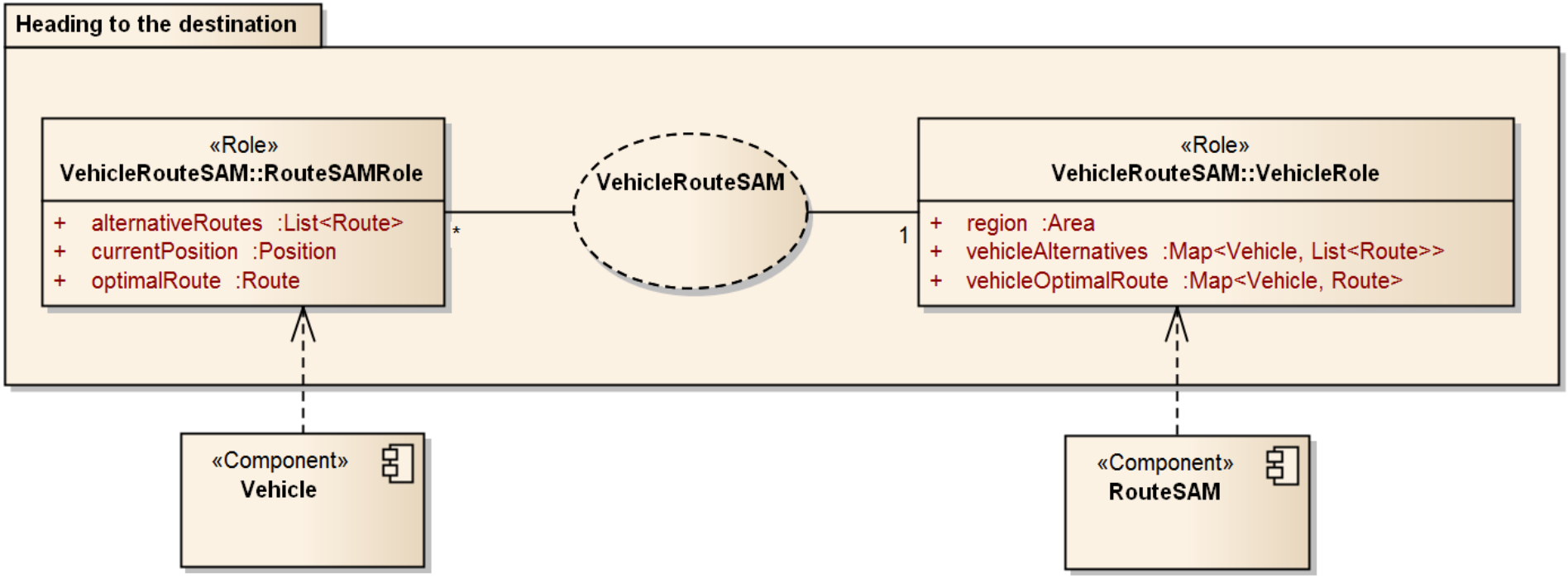
Design of components / ensembles

- Design of components and their roles (i.e. knowledge interfaces)

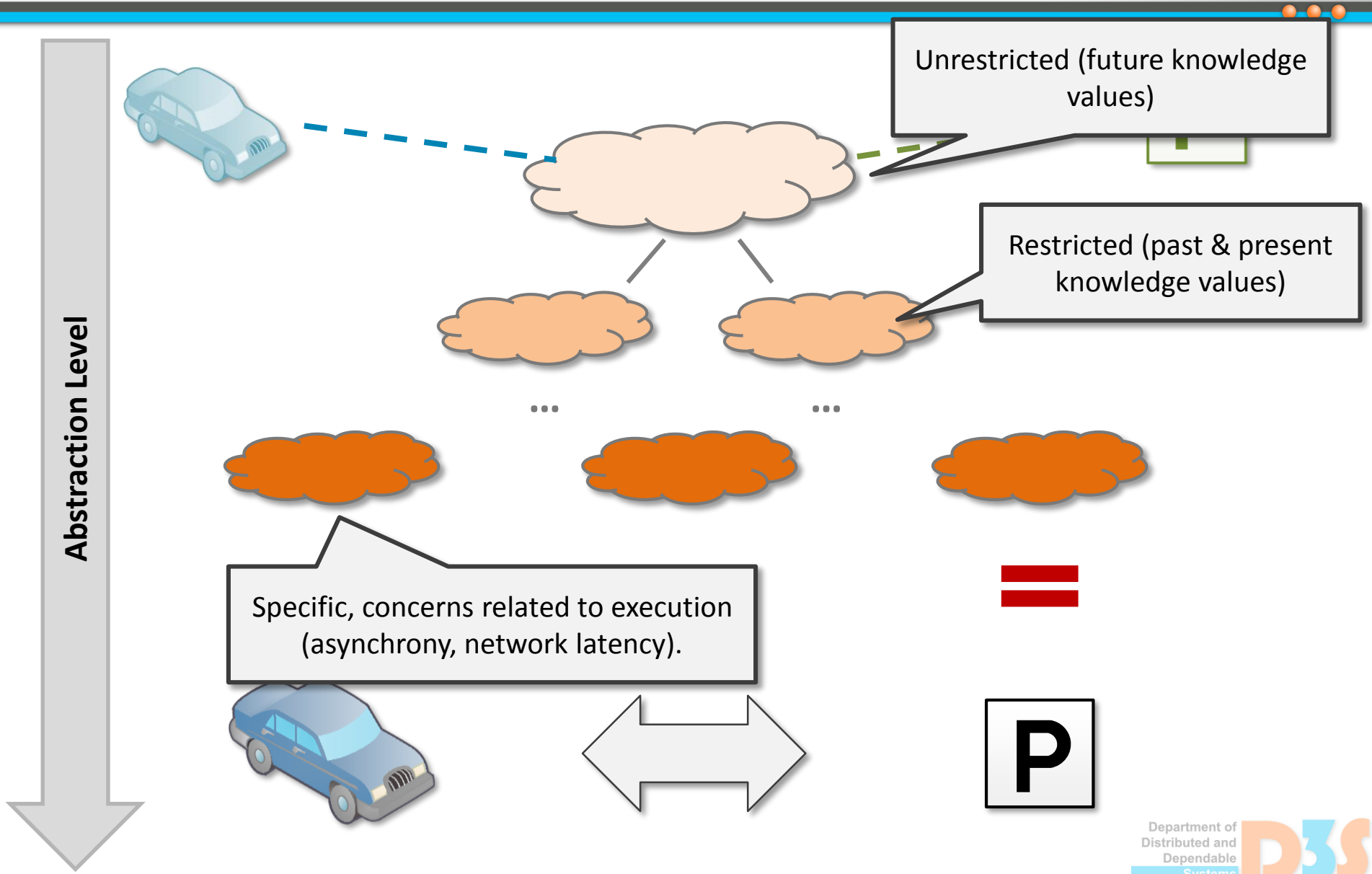


Design of components / ensembles

- Design of component interaction patterns (i.e. ensembles)
  - Captured as partial explicit architecture
  - Valid in a particular situation

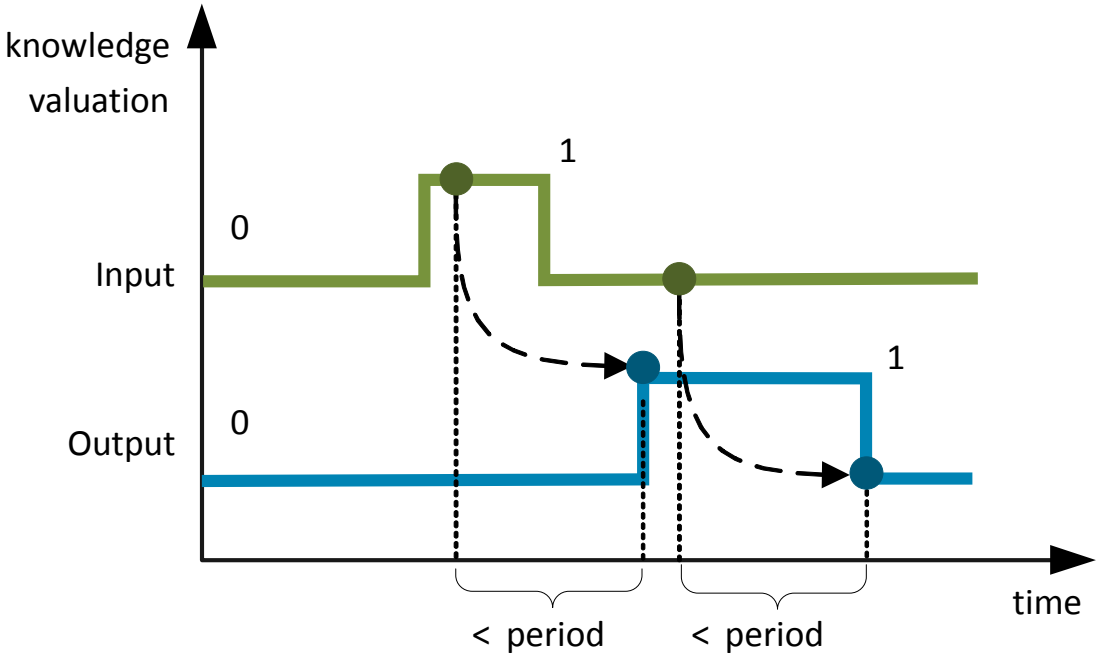


Invariants on Different Levels of Abstraction



Challenge of Invariant Design

- Formulating invariants/refinement rigorously is a challenge
- Knowledge evolves
  - Asynchrony, delays due to distribution



- High-level invariants need to capture the evolution **differently** than low-level invariants

Part 5

Showcase (DEECo + IRM) – intelligent parking

Example 3

- Vehicle component
  - Has a given destination to go to
  - When it gets close to a destination, it chooses a place close-by where it can park
  - To know where it can park
    - it monitors free parking spaces – as it passes by them
    - it exchanges the information about free parking spaces with vehicles close-by

All examples along with instructions how to launch them can be found at:
https://github.com/d3scomp/cbse-tutorial

Assignment

- Adapt the Exercise 3 so as
  - components avoid going to a place which is likely to be taken by another component
- How to refine this requirement?
- How to reflect it as processes and knowledge exchange?

Part 6

Going further: Analysis and simulation

Conclusion

On-going – Belief

- Knowledge of component A about component B is always outdated ~ belief
  - We look into how to quantify the level of inaccuracy caused by the outdatedness
  - Provide modeling and programming abstractions
  - Allow for adaptation based on the level of inaccuracy

```
component Vehicle
  knowledge:
    leaderPosition, leaderVelocity, ...
  state-space-models:
    [leaderPosition, leaderVelocity]:
// Tables for  lookup of min./max. acceleration based on current speed.
// The tables consist of tuples (velocity (in m/s) -> acceleration (in m/s2))
      inAccTable = {0 -> -6, 35 -> -5, 51 -> -3}
      maxAccTable = {0 ->  4, 35 ->3, 51 -> 0}
      Fmin(pos, vel) = (vel, interpolate(minAccTable , vel))
      Fmax(pos, vel) = (vel, interpolate(maxAccTable, vel))
  ...

  process computeAccelerationCACC(in pos, in leaderPos, in desiredDistance, out targetAcc):
    mode-trigger: inaccuracy(distance(position, leaderPosition)) <= THRESHOLD
  ...

  process computeAccelerationACC(in distance, in desiredDistance, out targetAcc):
    mode-trigger: inaccuracy(distance(position, leaderPosition)) > THRESHOLD
  ...
```

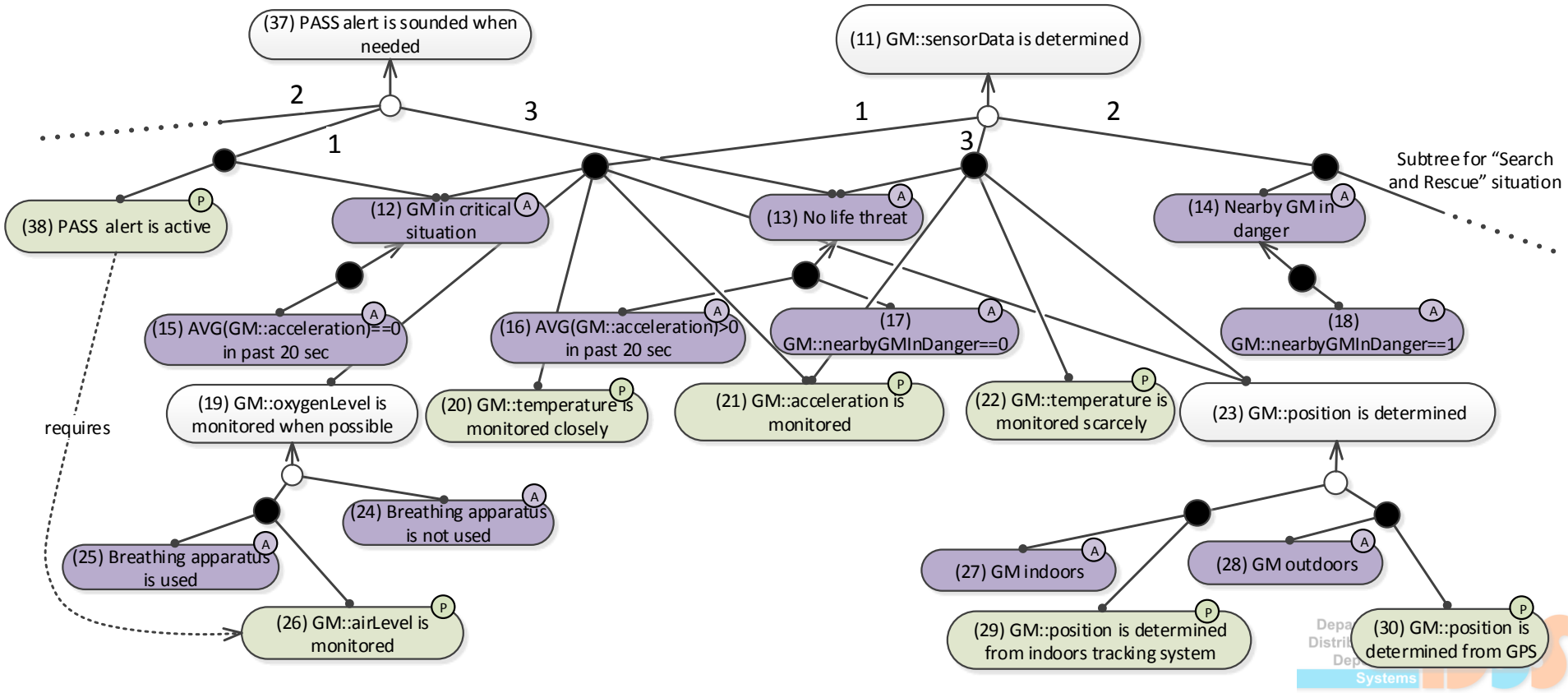
On-going – Peer-to-peer wireless networks

- CPS often rely on peer-to-peer wireless networks (e.g., WPANs)
- Building IP stack over the network is costly and contraproductive
- DEECo computation model can cope well with unreliability of the network
- Ensembles may be used to specify geographical-like routing
- Prospective use of gossip protocols

- Simulation of the network communication
  - JDEECo + OMNet++

On-going – IRM SA

- Allows alternatives (OR-decompositions) in the IRM design
- Thus captures different strategies
- Allows monitoring a system at runtime and selecting a feasible strategy – adapting to the situation in the environment



Interesting Challenges (instead of conclusion)

- Component **self-awareness** and adaptation based on **high-level goals** and strategies
- Techniques and models with a proper level of abstraction for feasible testing and verification of **correctness** of components with **emergent behavior**
- **Prediction** and **optimization** techniques for achieving efficient use of **resources** by distributed adaptive components
- **Security aspects**

Department of Distributed and Dependable Systems D3S