Towards a Requirements-Driven Design of Ensemble-Based Component Systems

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Ilias Gerostathopoulos
iliasg@d3s.mff.cuni.cz

Dr. Tomas Bures
Dr. Petr Hnetynka

CHARLES UNIVERSITY IN PRAGUE
faculty of mathematics and physics

Goal of this talk

"To show that requirement-driven design approaches cannot be used out-of-the-box in the design of ensemble-based component systems"

Goal of this talk

"To show that requirement-driven design approaches cannot be used out-of-the-box in the design of ensemble-based component systems"

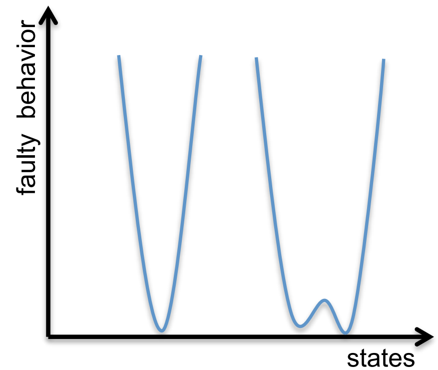
Department of Distributed and Dependable Systems

Target domain

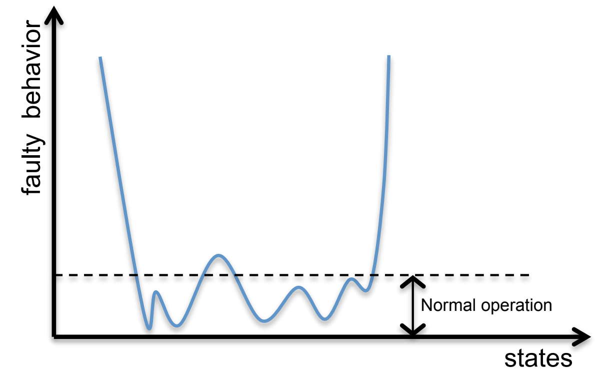
Resilient Distributed Systems

Department of Distributed and Dependable Systems

Resilient Distributed Systems



a. Dependable systems



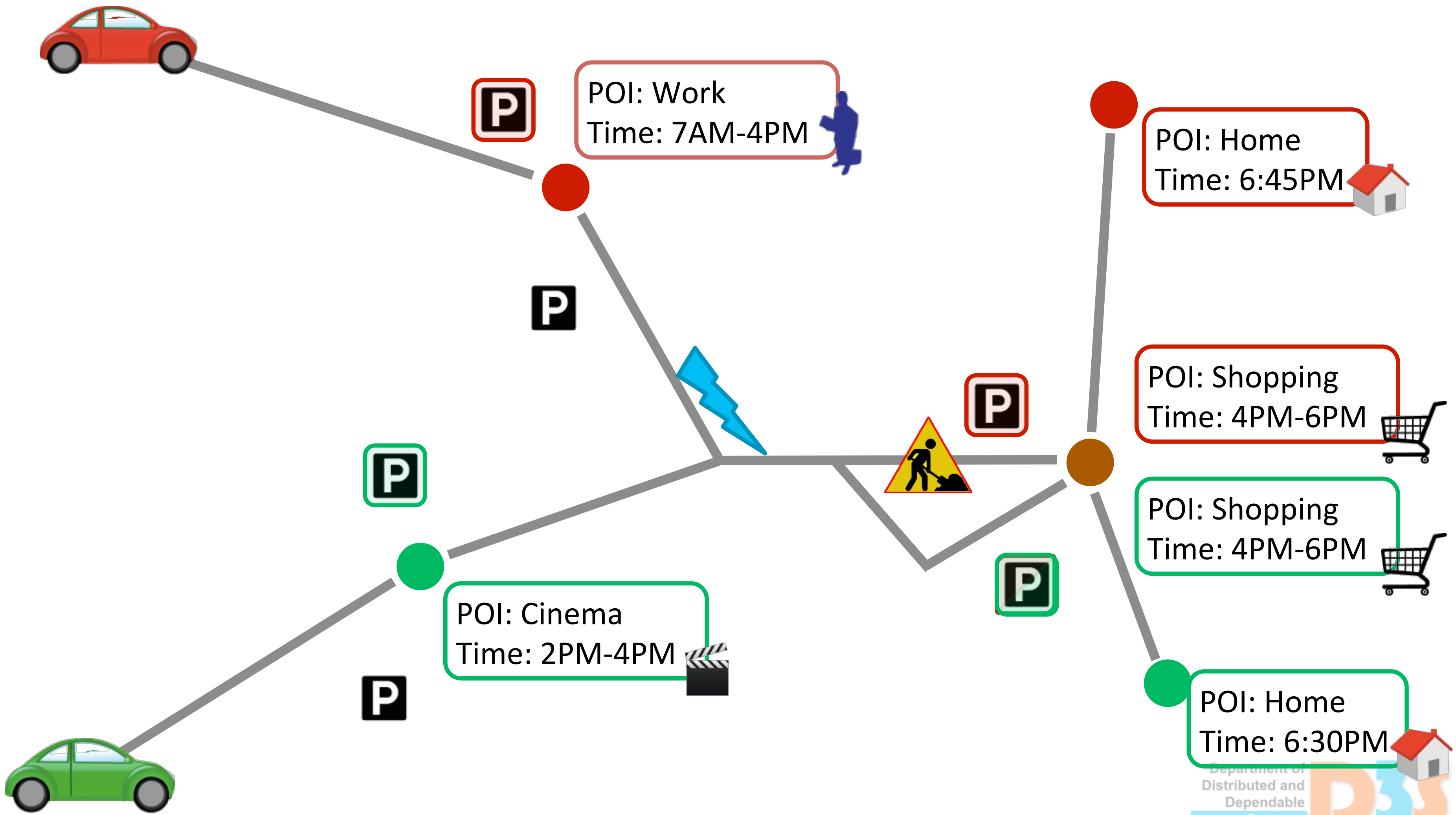
b. Resilient systems

Resilient systems: systems that remain *dependable*, even when facing *continuous change* and are *constantly adapting* to it

- Inherently distributed
- Self-aware services pursuing global goals
- Recurring/dynamic changes in the environment
- **Cloud setting: nodes in ad-hoc, dynamic networks**

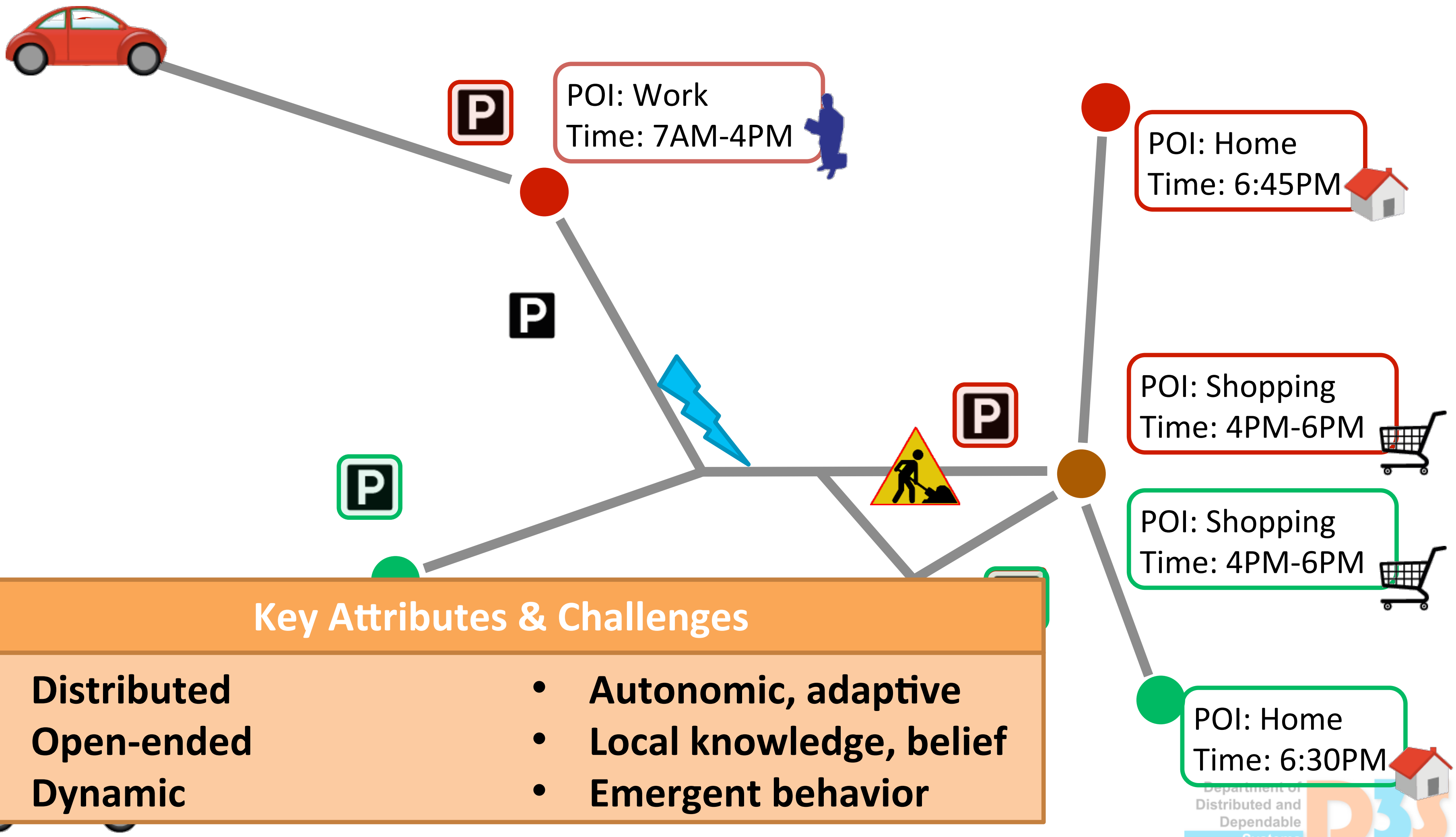
Department of Distributed and Dependable Systems

RDS – an intelligent navigation case study



Department of Distributed and Dependable Systems

RDS – an intelligent navigation case study



Key Attributes & Challenges

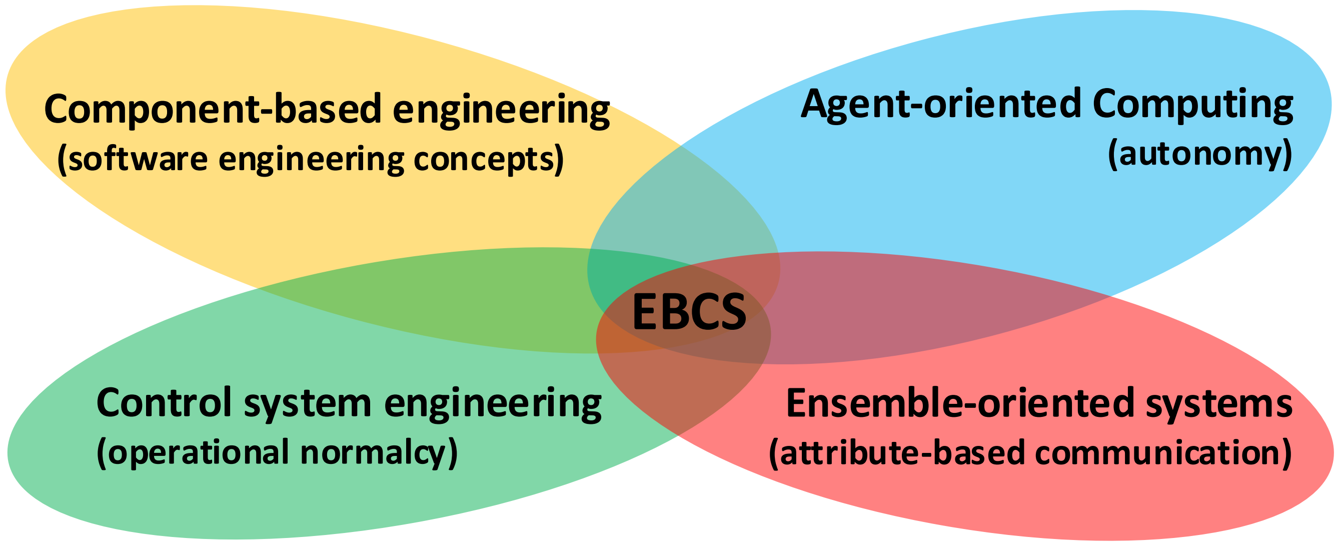
- Distributed
- Open-ended
- Dynamic
- Autonomic, adaptive
- Local knowledge, belief
- Emergent behavior

Engineering RDS via Service Components

Ensemble-Based Component Systems

Department of Distributed and Dependable Systems

Ensemble-Based Component Systems



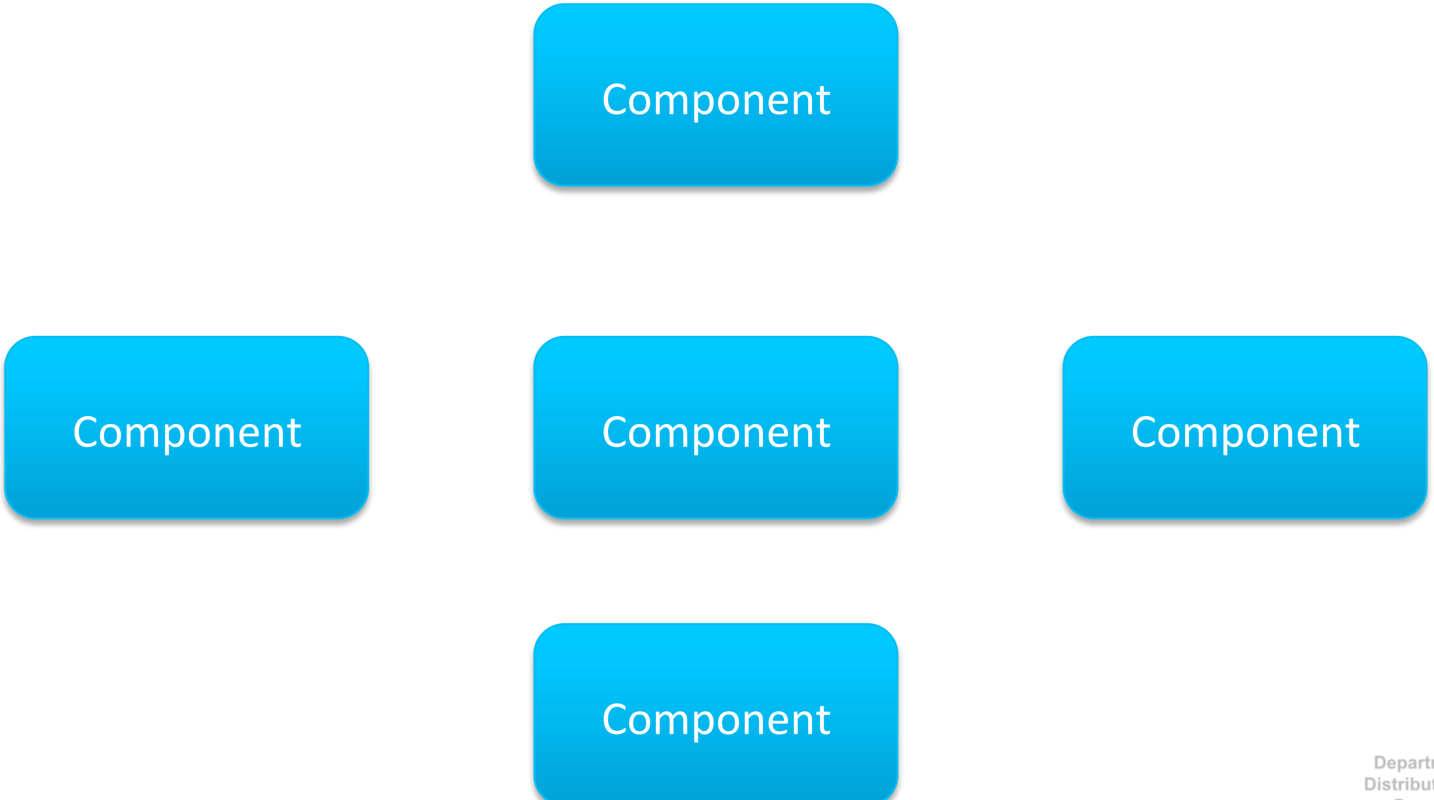
→ Integrate ideas & concepts from different areas
→ Provide abstractions for engineering RDS

DEECo component model:
Dependable Emergent Ensembles of Components

→ Brings separation of concerns to the extreme

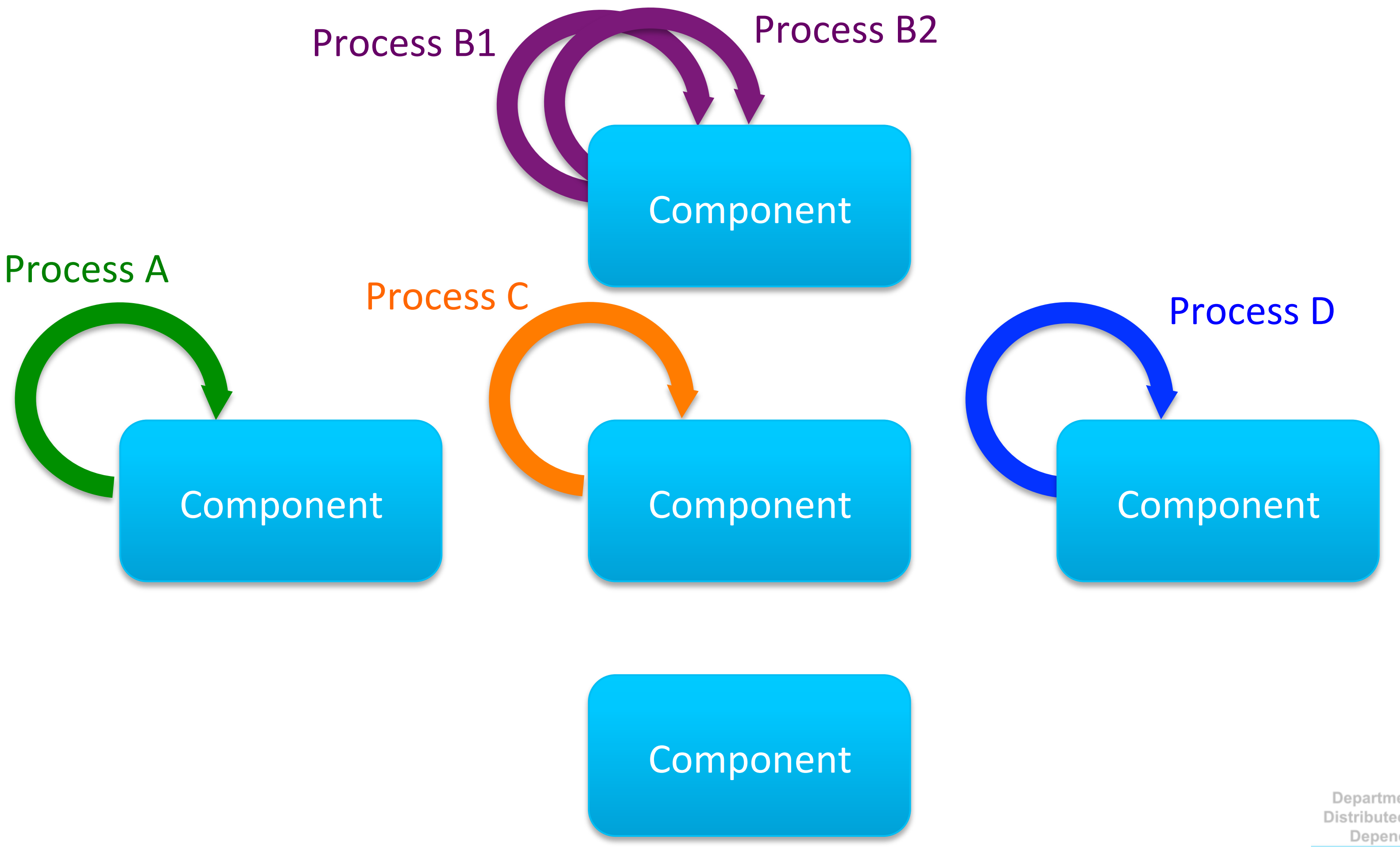
DEECo model – Component

Components are **strictly autonomous** units featuring cyclic periodic execution.



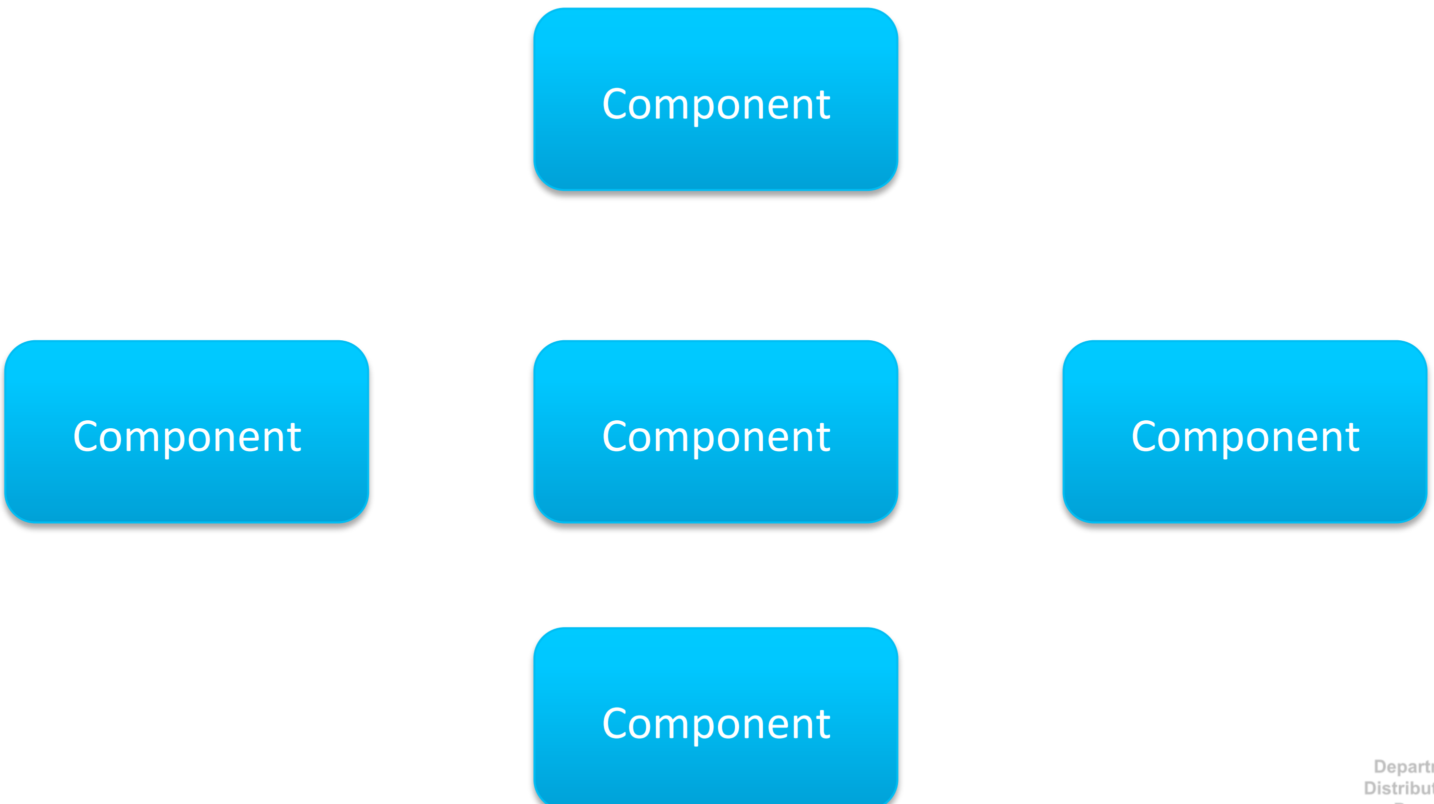
DEECo model – Component

Components are **strictly autonomous** units featuring cyclic periodic execution.



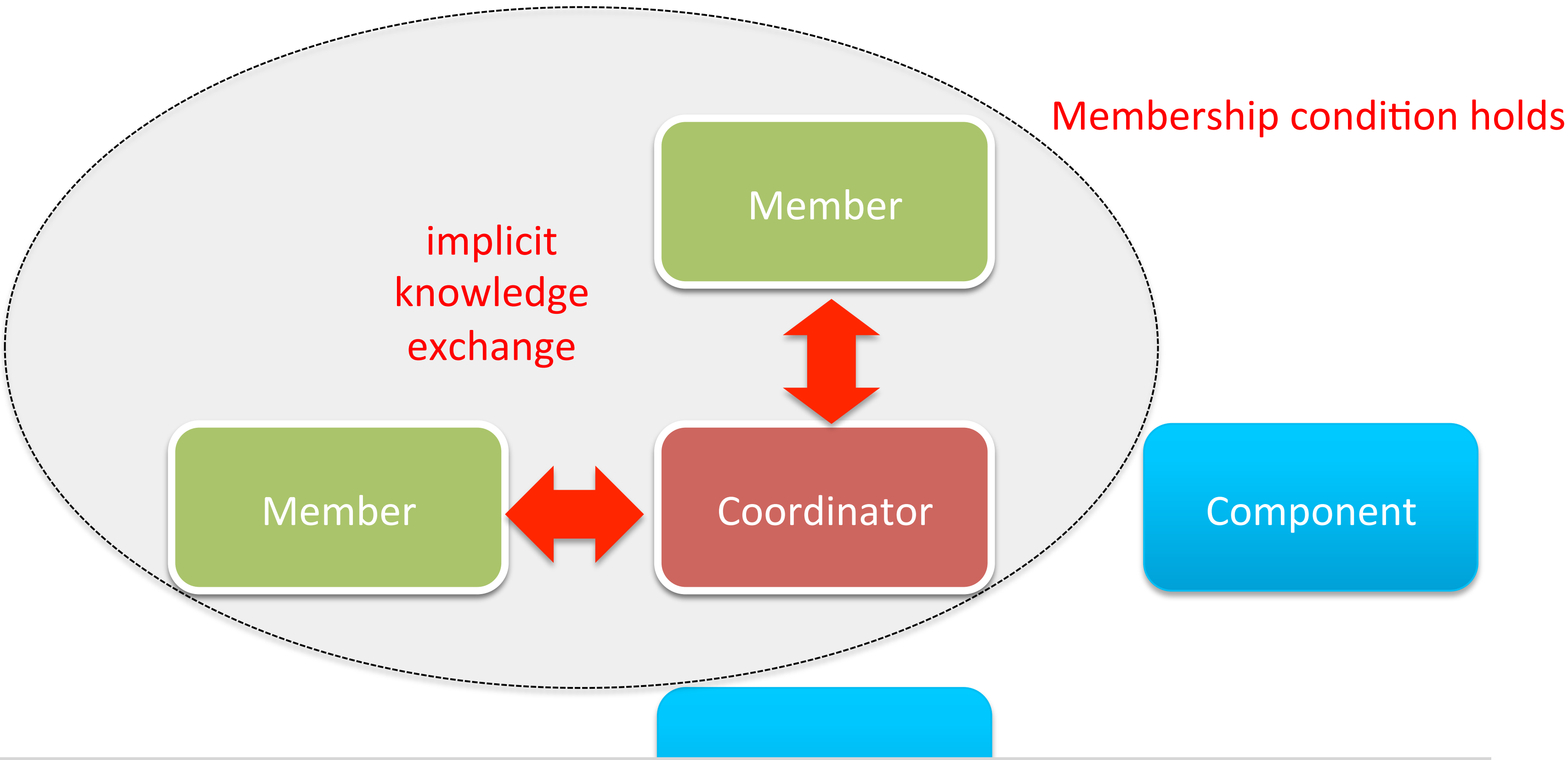
DEECo model – Ensemble

Ensembles are (stateless) **interaction templates** that describe
a) **When** to communicate b) **What** to communicate



DEECo model – Ensemble

Ensembles are (stateless) **interaction templates** that describe
a) **When** to communicate   b) **What** to communicate



T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "DEECo – an Ensemble-Based Component System," to appear in *Proc. of CBSE '13*, 2013.

Issues & Challenges

EBCS design

Design of EBCS: The process of identifying and specifying the desired components (knowledge & processes) and ensembles.

Challenge

How to design EBCS in a systematic and effective way?

- How to come up with the desired components and ensembles, according to system requirements?
- How to achieve design validation and traceability?

Problem

Classic requirement-driven design approaches do not suffice.

- Use-cases, user stories, …
- Goal-oriented approaches

Department of Distributed and Dependable Systems

EBCS design

Design of EBCS: The process of identifying and specifying the desired components (knowledge & processes) and ensembles.

Challenge

How to design EBCS in a systematic and effective way?

- How to come up with the desired components and ensembles, according to system requirements?
- How to achieve design validation and traceability?

Problem

Classic requirement-driven design approaches do not suffice.

- ~~Use-cases, user stories,~~ …
- Goal-oriented approaches

Describe "how" instead of "what"
Inherently less adaptable/evolvable.
Require event anticipation.

Department of Distributed and Dependable Systems

EBCS design

Design of EBCS: The process of identifying and specifying the desired components (knowledge & processes) and ensembles.

Challenge

How to design EBCS in a systematic and effective way?

- How to come up with the desired components and ensembles, according to system requirements?
- How to achieve design validation and traceability?

Problem

Classic requirement-driven design approaches do not suffice.

- ~~Use-cases, user stories,~~ …
- Goal-oriented approaches

Describe "how" instead of "what"
Inherently less adaptable/evolvable.
Require event anticipation.

Can the high-level concepts (goals, actors, …) provide a remedy?

Department of Distributed and Dependable Systems

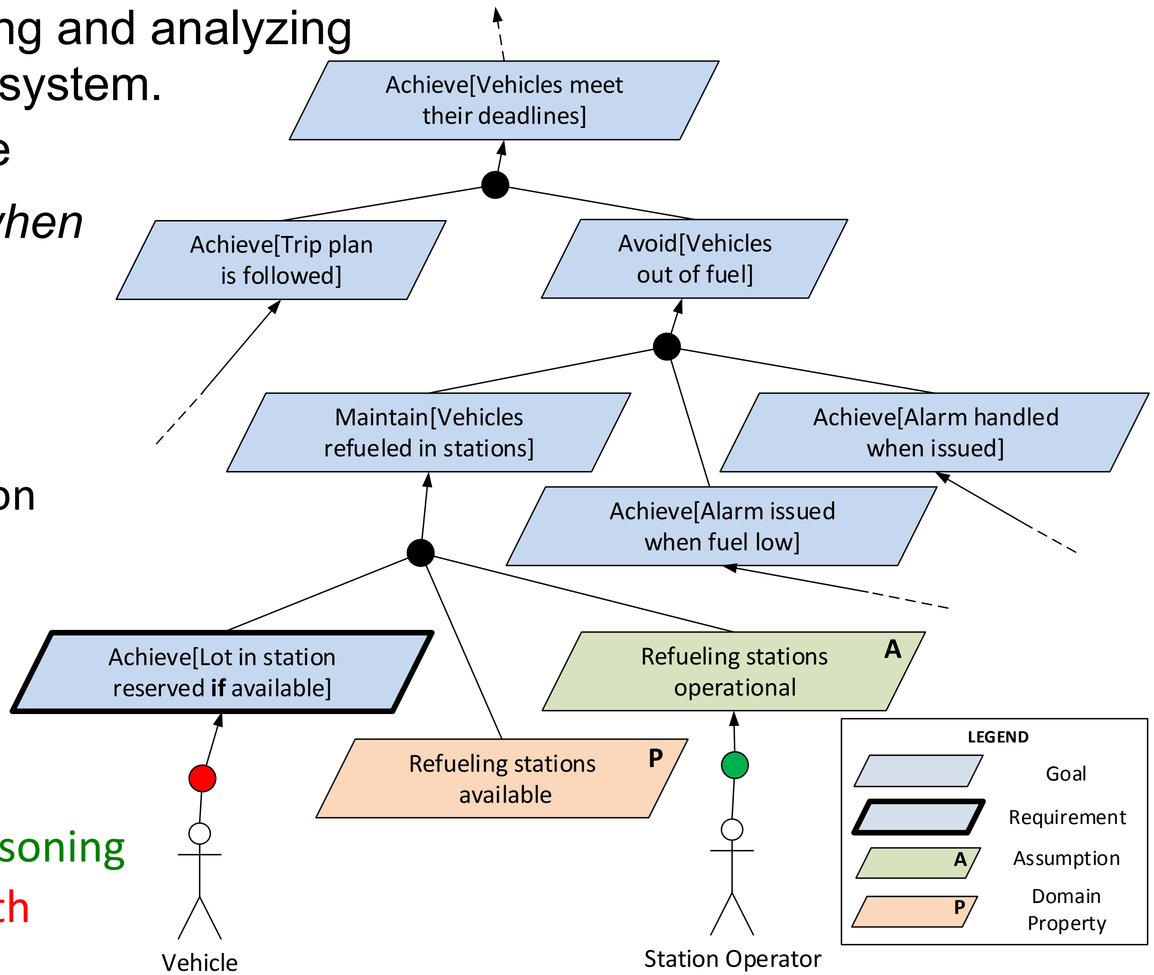
Requirements modeling – KAOS

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



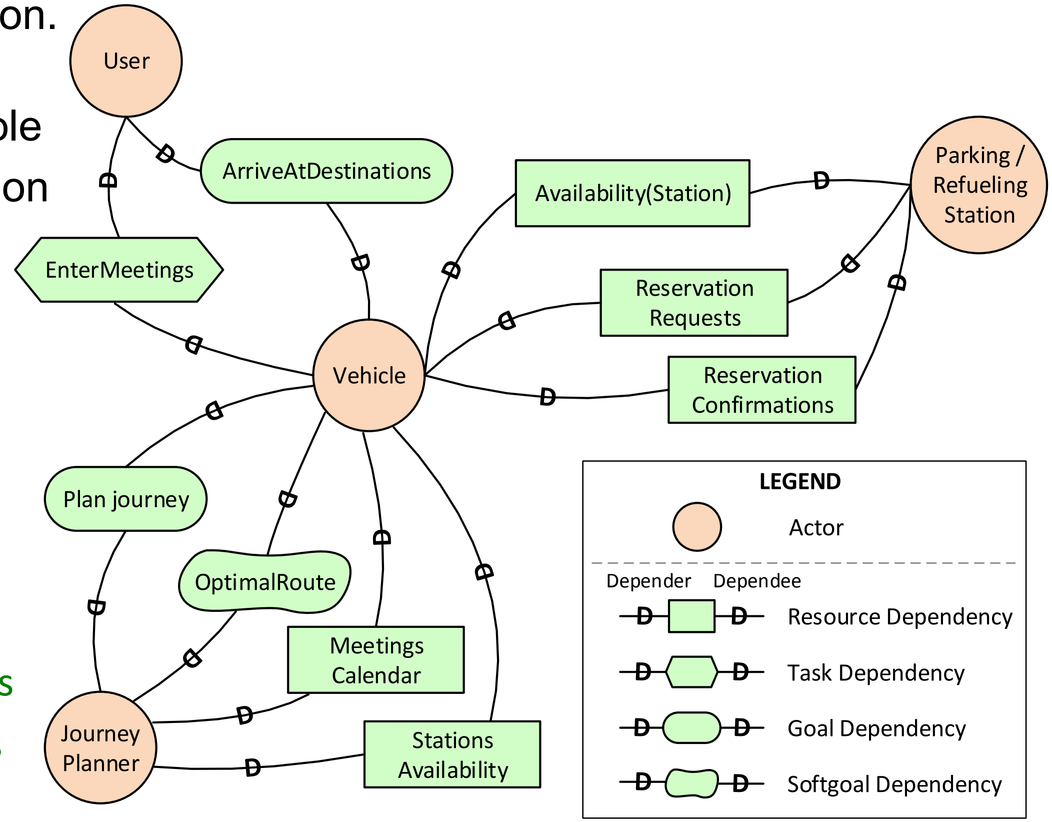
Applicability in design of EBCS:

+ 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

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 ⇩ Actor-Agent mapping ⇨ BDI architecture

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

_ comprises a number of models with manual mappings between them

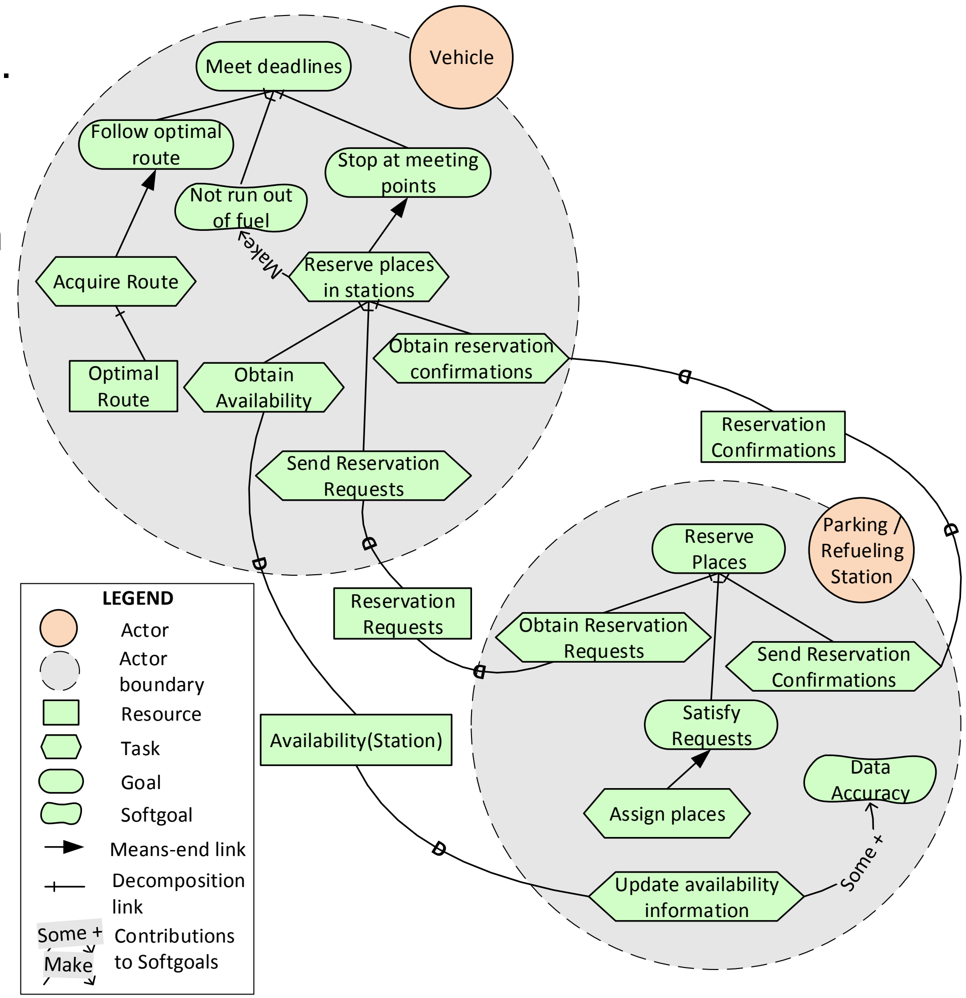
_ does not cope with the emergent architecture requirement

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 ⇓ Actor-Agent mapping ⇒ BDI architecture



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

_ comprises a number of models with manual mappings between them

_ does not cope with the emergent architecture requirement

EBCS design

Proposed Solution

Operational normalcy & Invariant

Operational normalcy: the property of being within the limits of normal operation

⇩

The valuation of the components' knowledge evolves as a result of their autonomous behavior and of knowledge exchange

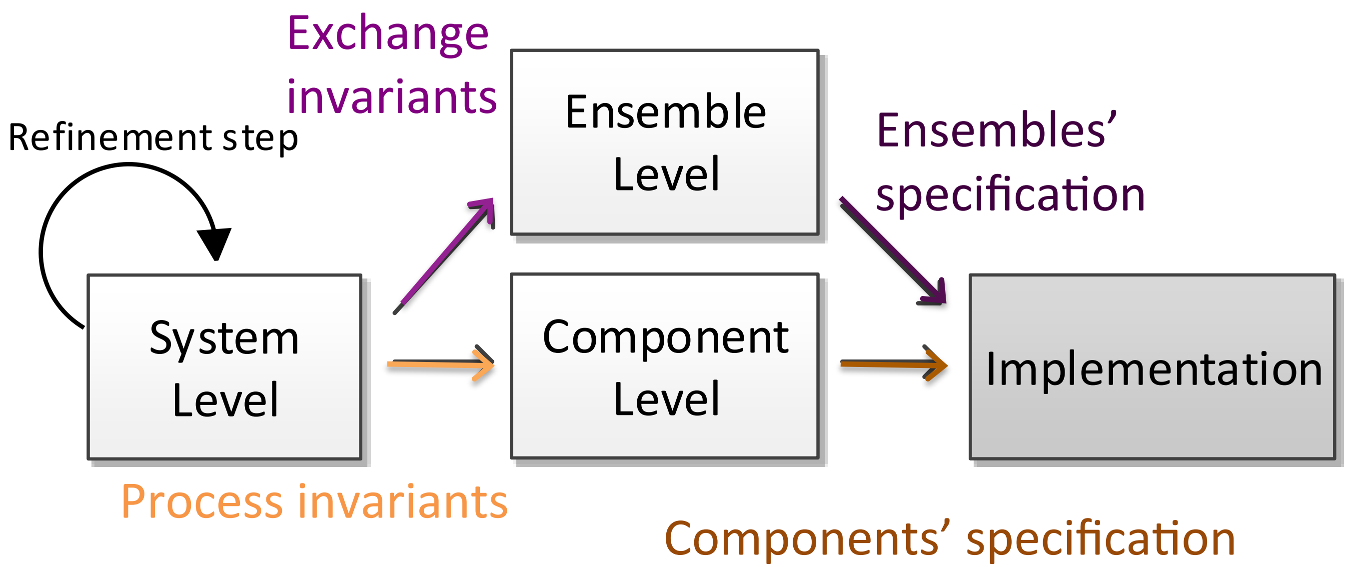
⇩

Invariant: condition on the knowledge valuation of a set of components that captures the operational normalcy to be maintained by the system-to-be

Invariant Refinement Method

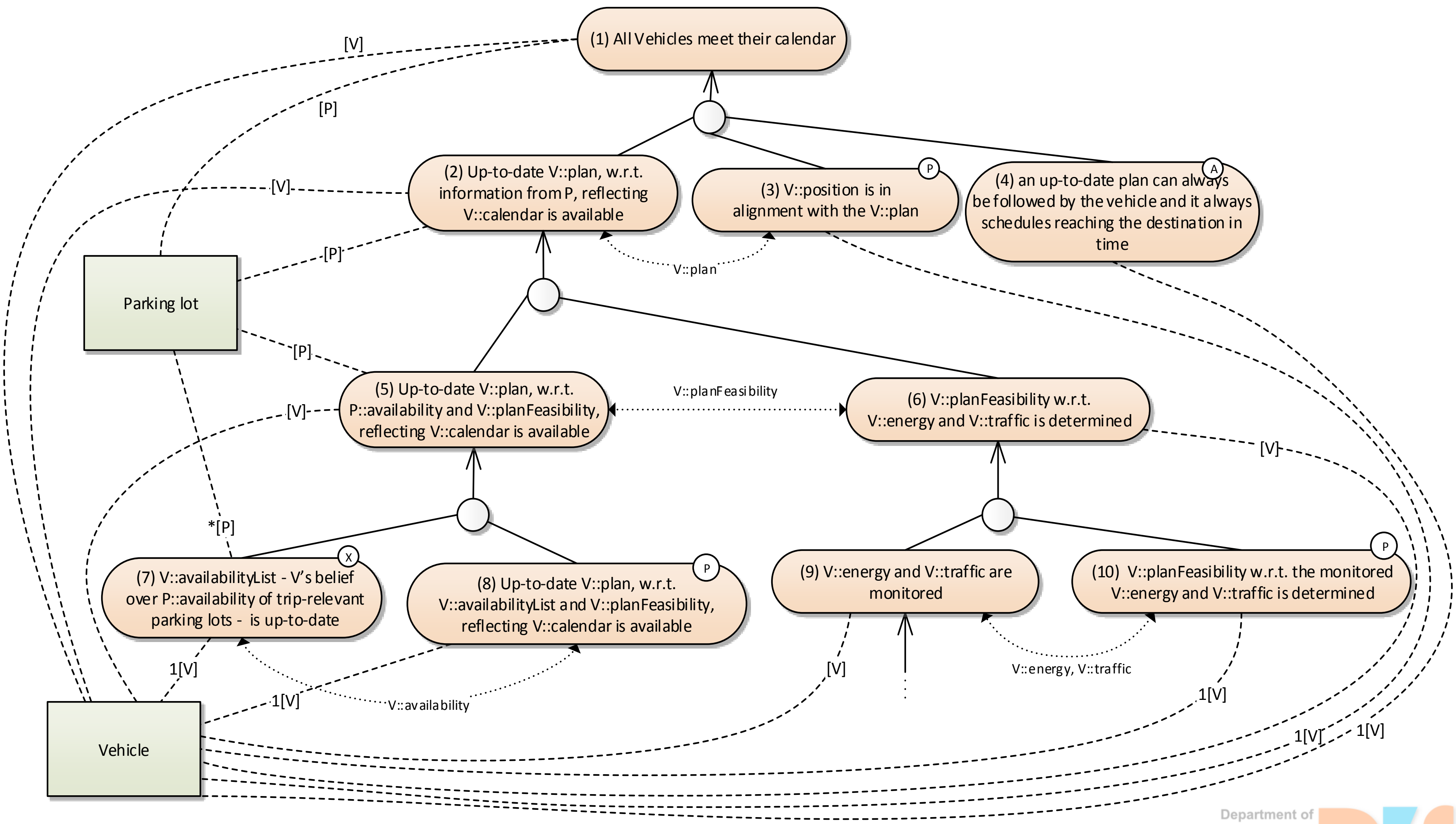
Idea: Iteratively refine and concretize invariants at the system level up to the point where they can be mapped to:

- Component processes (process invariants)
- Knowledge exchange (exchange invariants)

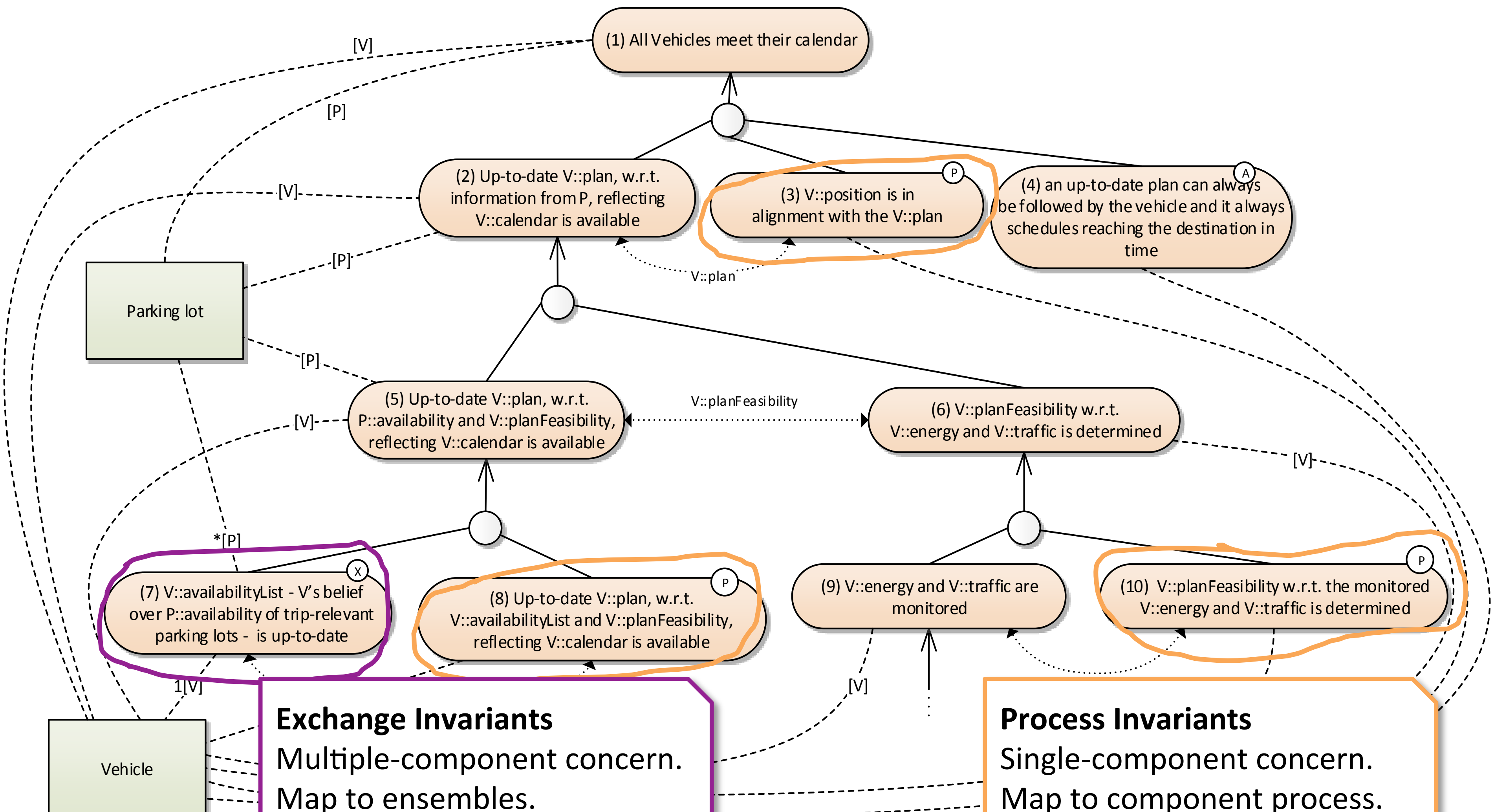


Department of Distributed and Dependable Systems

IRM refinement tree



IRM refinement tree



Exchange Invariants
Multiple-component concern.
Map to ensembles.

Process Invariants
Single-component concern.
Map to component process.

J. Keznikl, T. Bures, F. Plasil, I. Gerostathopoulos, P. Hnetynka, and N. Hoch, "Design of Ensemble-Based Component Systems by Invariant Refinement," to appear in *Proc. of CBSE '13*, 2013.

Conclusion

	KAOS	TROPOS	IRM
High-level modeling	✔	✔	✔
Alignment of requirements with architecture	✖	✔	✔
Dynamic, emergent architecture support	✖	✖	✔

Novelty: "IRM allows reasoning along the line of what needs to *hold* in the system at every time instant (invariants), instead of what needs to *be performed* (actions) or *achieved* (goals)"

Future Work

IRM designer: model-based tool implementing the IRM features

Support for dynamic adaptation based on alternative refinements (when different assumptions hold)

Design rules and constraints for guiding the developer in design decisions and structural validation

Formal structure and semantics of IRM → system analysis

Department of Distributed and Dependable Systems

Thank you!

Questions?