CONTRACT-BASED DESIGN IN MODEL-BASED SYSTEMS ENGINEERING

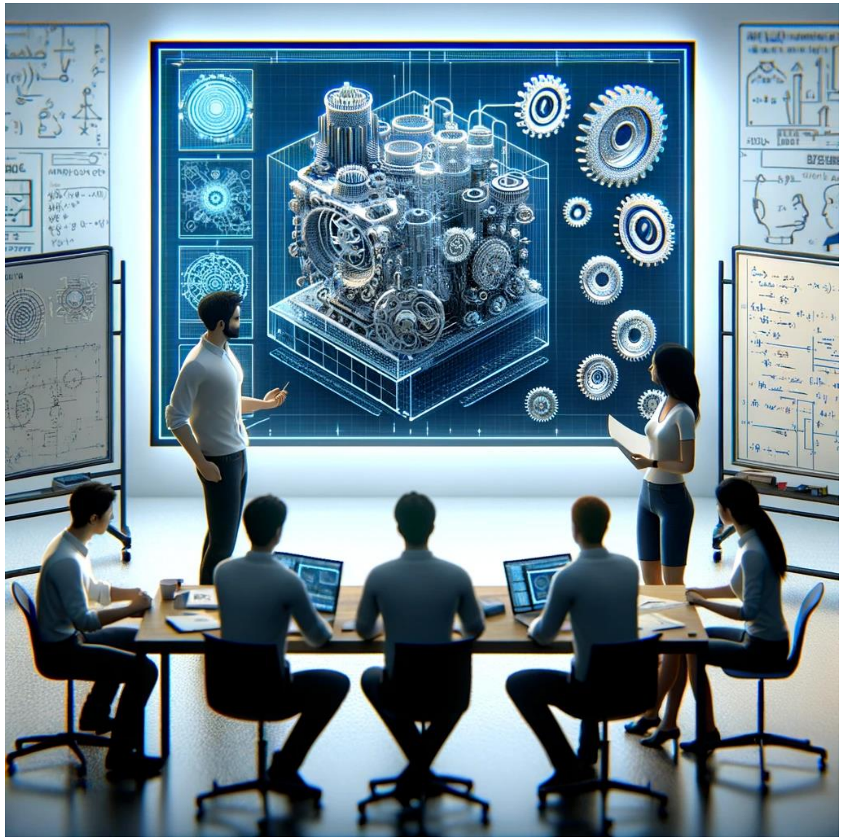
Ingo Stierand

DLR – Institute Systems Engineering for Future Mobility

Why Model-Based (Systems) Engineering

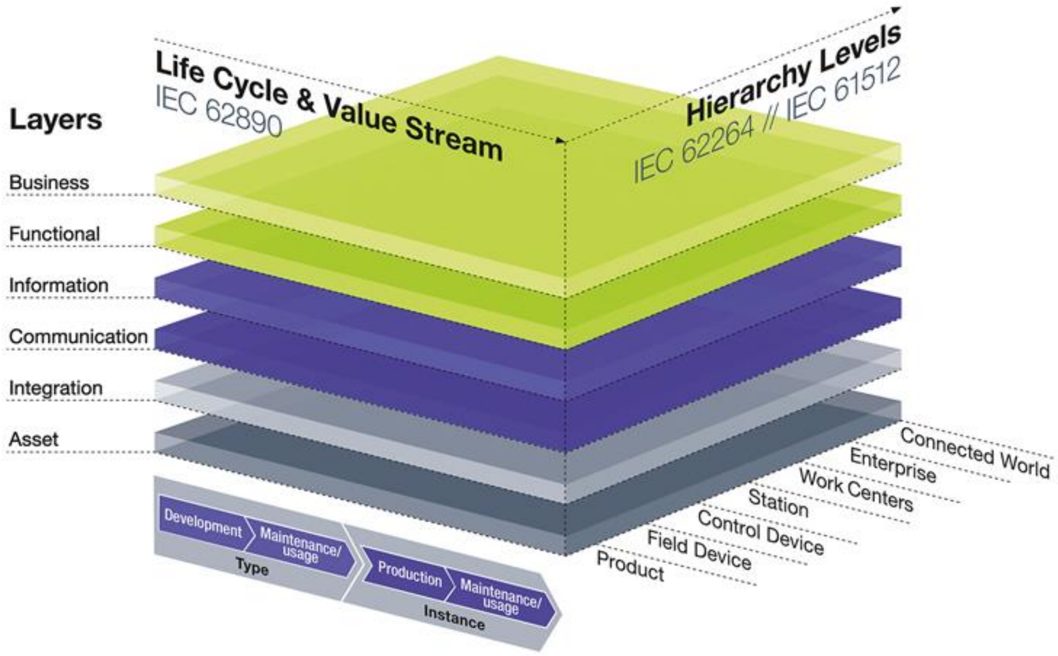
Model-Based Systems Engineering (MBSE) uses models as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification.

- **Complexity Management**: Comprehensive models that represent all aspects of a system help in managing complexity by providing a clear and consistent representation.
- **Collaboration**: Models as a common language provide a shared framework that everyone can understand and contribute to.
- **Adaptability**: Models can be updated more easily than traditional documents, which enables engineers to more rapidly adapt to evolving requirements.
- **Requirements Management**: MBSE enables linking system requirements directly to elements within the model. This ensures traceability and helps to see the impact of changes.
- **Risk Management**: By simulating and analyzing models, MBSE allows engineers to identify potential issues and risks early in the design process.
- **Regulatory Compliance and Safety**: Thoroughness and accuracy of MBSE models assist in ensuring that systems meet regulations and safety requirements.
- **Data Management**: MBSE models can integrate data from various sources and maintain data consistency throughout the system's lifecycle.



Modelling Frameworks

- Each application domain comes with particular demands, regulations and constraints.
- Modelling frameworks help structuring the development process:
  - Layers (perspectives, viewpoints)
  - Life cycle phases
  - Hierarchy levels
  - …
- Many MBSE modelling languages exist to “fill in” elements in those matrixes:
  - Business Process Model and Notation (BPMN)
  - Structured Goal Notation (GSN)
  - UML diagrams
  - …



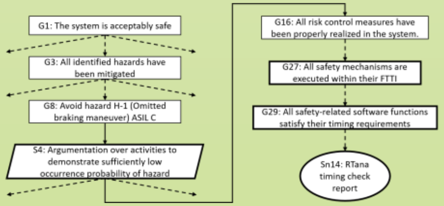
Reference Architecture Model Industry 4.0 – RAMI 4.0 (Source: DKE)

SPES modelling framework

- Focus on (safety-critical) cyber-physical systems.
- Provision of semantically consistent, continuous, traceable MBSE along viewpoints and abstraction levels.

Requirements Viewpoint

- Operational context
- Goals
- Top-level requirements



Lidar Grabber
A_{L1}: sd.lidar occurs every 66 ms.
G_{L1}: Reaction(sd.lidar.OK,(set(mode,Normal),lidar_output)) within [5,60] ms.
G_{L2}: Reaction(sd.lidar.NOT_OK,(set(mode, Recover)) within [5,60] ms.

Camera Grabber
A_{C1}: sd.camera occurs every 66 ms.
G_{C1}: Reaction(sd.camera.OK,(set(mode,Normal),lidar_output)) within [7,60] ms.
G_{C2}: Reaction(sd.camera.NOT_OK,(set(mode,Recover)) within [7,60] ms.

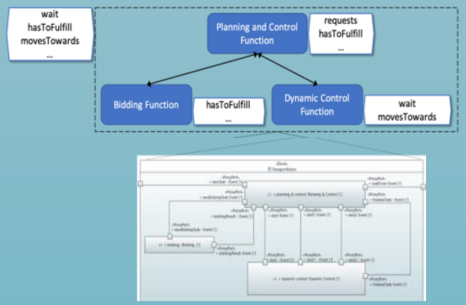
Object Localization via Lidar
G_{OL}: Reaction(lidar_output,object_poses) within [120,795] ms.

Object Localization via Camera
G_{OC}: Reaction(frame,bounding_boxes) within [20,55] ms.

Sensor Fusion
G_{F1}: Reaction(object_poses,fused_objects) within [5,25] ms.
G_{F2}: Reaction(bounding_boxes,fused_objects) within [5,25] ms.

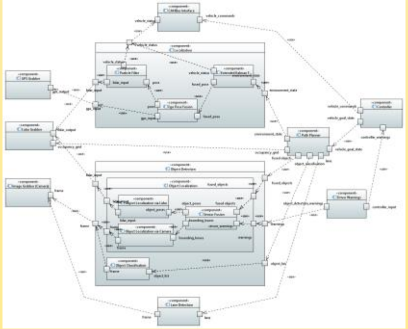
Functional Viewpoint

- System behavior, and
- Dependencies
- Decomposition of functions



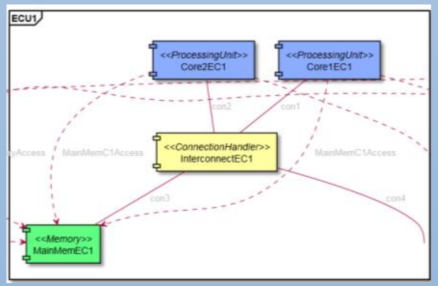
Logical Viewpoint

- Decomposition into sub-systems and components
- Assigning functions to logical units



Technical Viewpoint

- Hardware/Software Design
- Scheduling/middleware
- Optimization of resources
- Communication

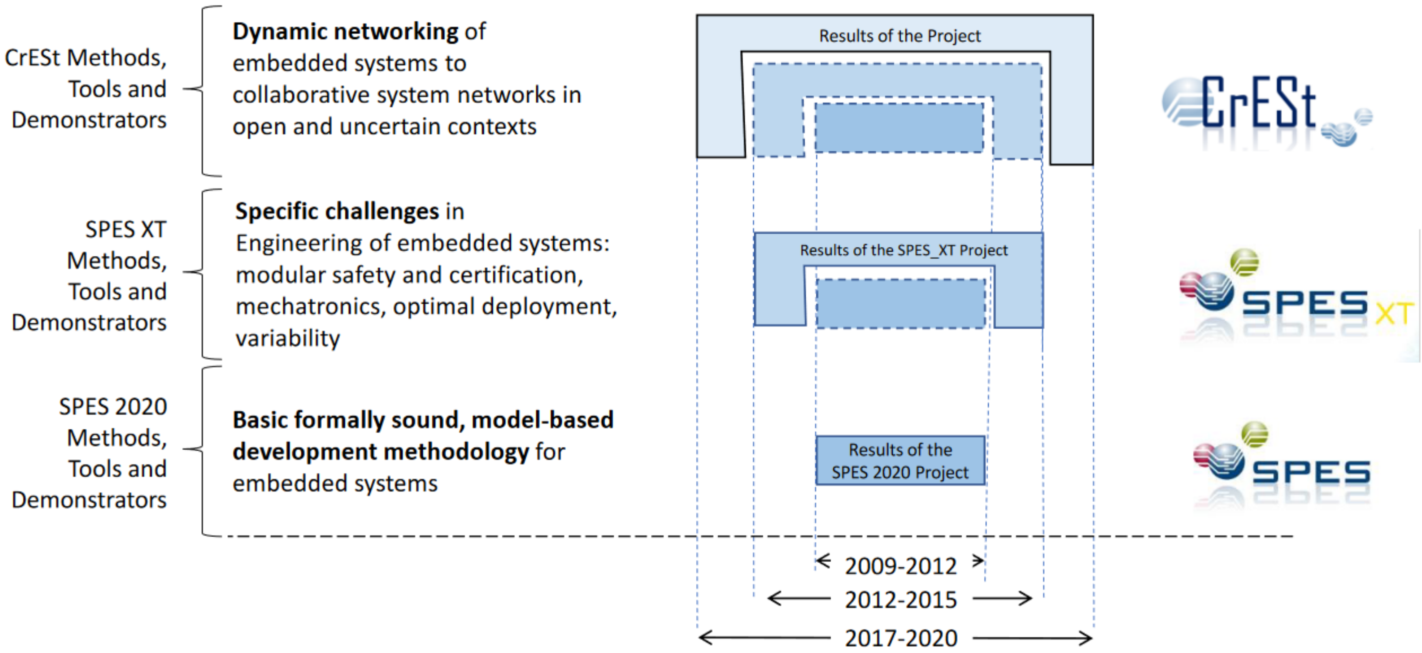


Abstraction levels

History: From SPES 2020 to CrESt

The Vision of SPES 2020

The development of software intense systems (CPS) can be accomplished through a set of **integrated modeling techniques**, their use and integration in the development is fully understood.

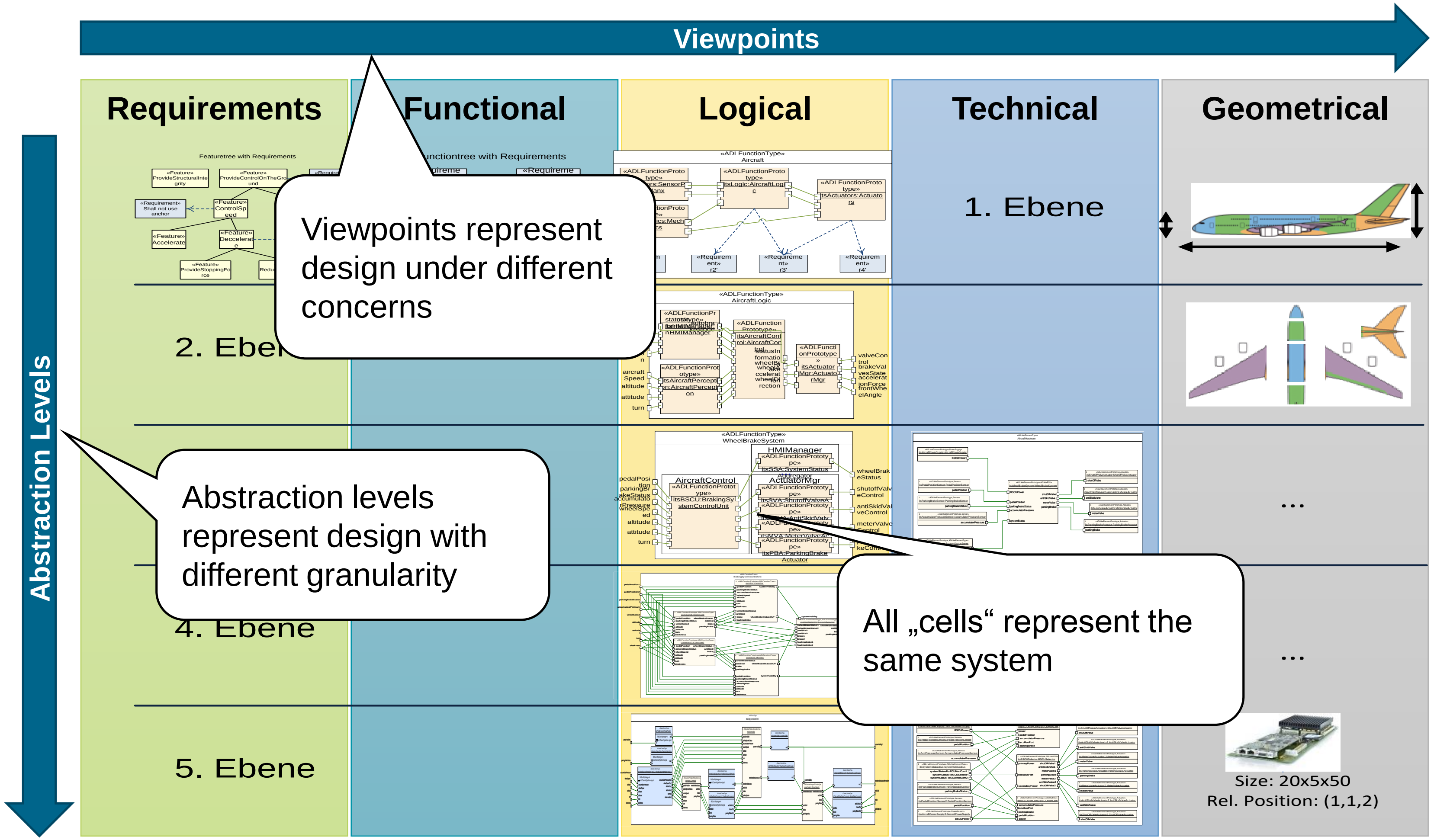


Agenda

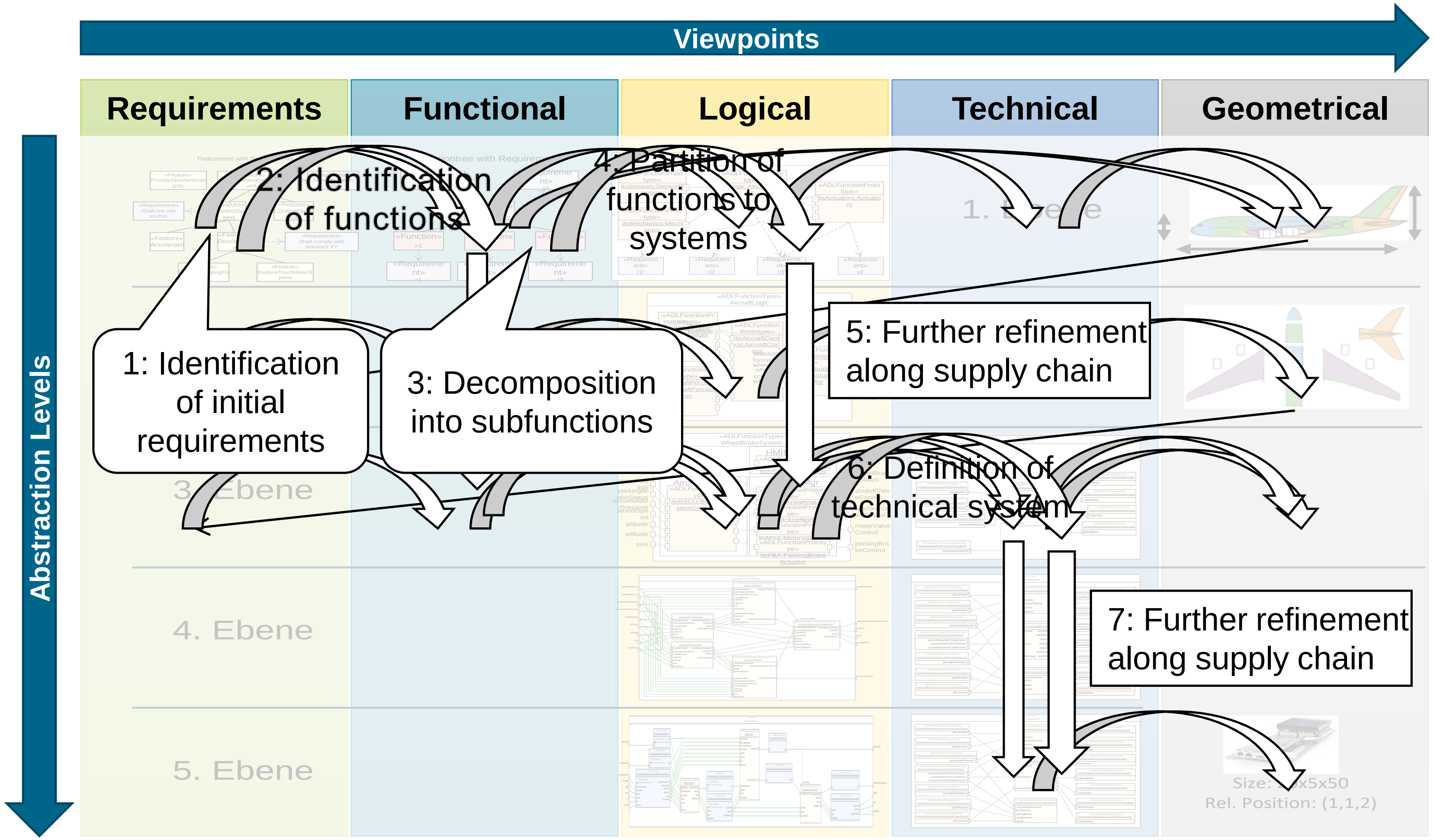
- **Compositional Semantic Framework**
- A "Meta Theory" of Contract Based Design
- Example

Compositional Semantic Framework

Abstraction Levels and Viewpoints



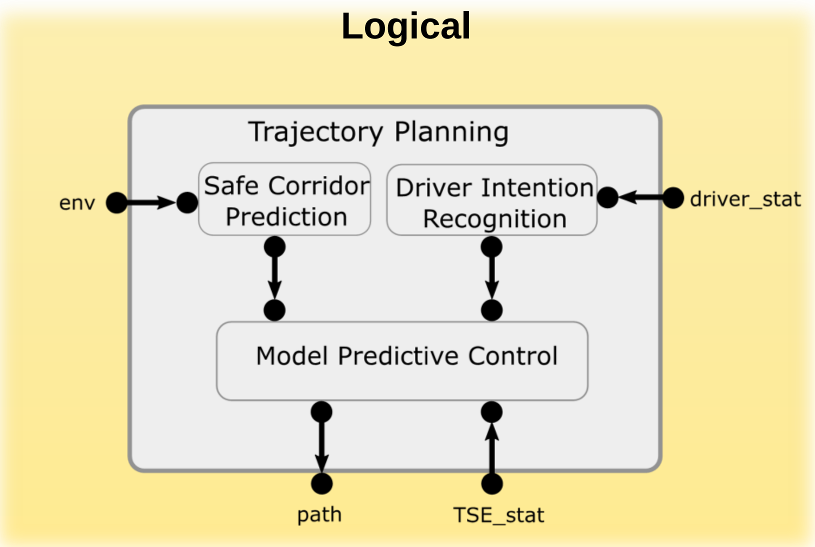
Compositional Semantic Framework
Design Steps (Examples)

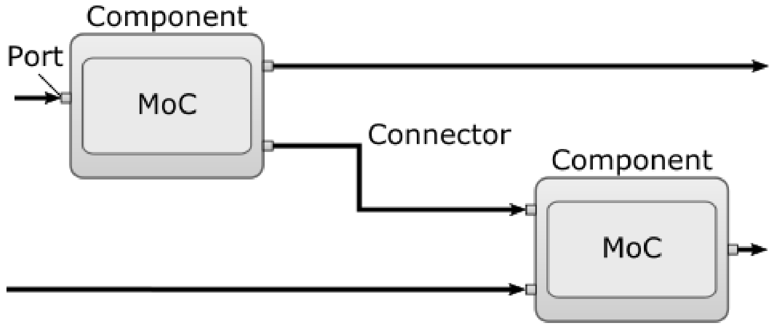


Compositional Semantic Framework

Components and Hierarchy

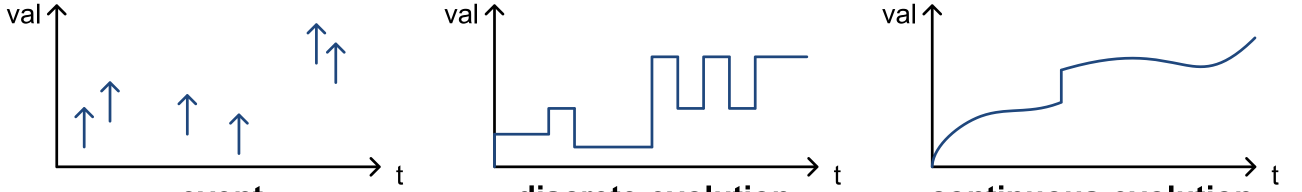
- Components
  - Basic design entity to structure models
  - Well defined interfaces
  - Can be reused in different design contexts
- Typed Ports
  - Define syntactical interface to adjacent components / environment
- Hierarchy and Composition
  - Allows deeply nested component hierarchies
  - Supports top-down and bottom-up design
- Connectors
  - Component interaction via port connections
  - Simple and complex connectors





Compositional Semantic Framework
Component Behaviour (Semantic Domain)

- Behaviour is visible at component ports
  - Common dense time domain: $\mathbb{T} = \mathbb{R}^{\geq 0}$ (for example)
  - Port types and value domains: D_p for port p
  - Behaviour in term of signals: $s_p : \mathbb{T} \to V_p \cup \{\perp\}$
    - $\perp$ means “absent value”
- Allows for specification of different “types” of behaviour:
  - **Discrete event** (absent values except for discrete set of time points)
  - **Discrete evolution** (changes only at discrete time points)
  - **Continuous evolution**



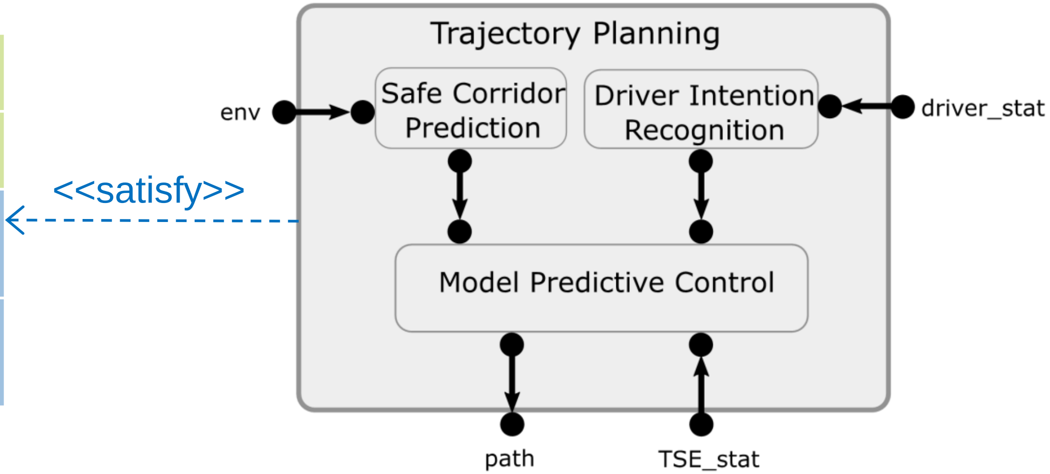
event **discrete-evolution** **continuous-evolution**

Compositional Semantic Framework

Contracts – Assume / Guarantee Reasoning

- **Assumptions** (green):
  - Specify necessary conditions of environment and surrounding components for component to work properly
- **Guarantees** (blue):
  - Specify required behaviour that must be guaranteed by implementation if used in context compliant to assumptions
- Split in assumptions and guarantees allows **compositional** reasoning schemes
  - Implementations can be developed independently
  - Reduces verification complexity
  - Detect integration issues early in design process before developing implementations

Assumption A:	driver_stat occurs every 1s.
	TSE_stat occurs after every path.
Guarantee G:	Reaction(TSE_stat.reached,path) within [150,1500] ms.
	Reaction(TSE_stat.paa,path) within [150,200] ms.

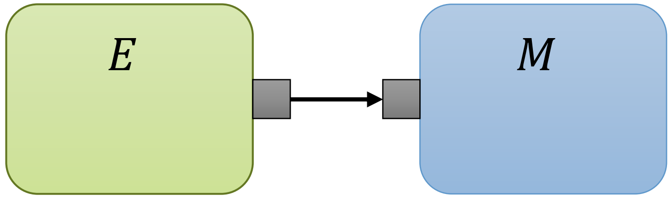


Agenda

- Compositional Semantic Framework
- **A "Meta Theory" of Contract Based Design**
- Example

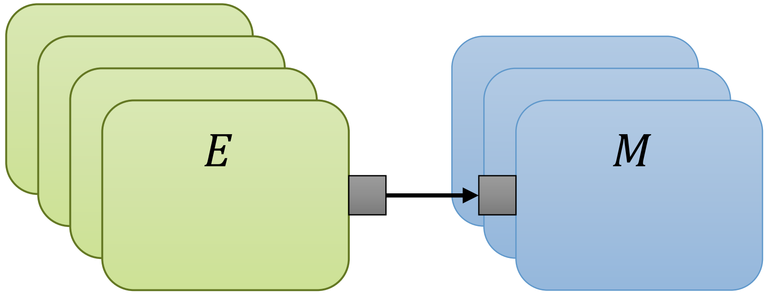
Contract Based Design
Components

- Contract Based Design implies existence of Components
  - A *component* is the basic design entity.
  - Components can be *composed*: $M_1 \times M_2$.
  - Not all components can be composed
    - Components are *composable* if $M_1 \times M_2$ is well-defined.
  - An *environment* of M is a component E such that $E \times M$ is composable.



Contract Based Design
Contracts

- A contract $C = (\boldsymbol{E_C}, \boldsymbol{M_C})$ specifies two sets of components
  - We say, a component $M \in \boldsymbol{M_C}$ is an *implementation* of C: $\mathrm{M} \vDash^M C \Leftrightarrow \mathrm{M} \in \boldsymbol{M_C}$
  - We say, a component $E \in \boldsymbol{E_C}$ is an *environment* of C: $E \vDash^E C \Leftrightarrow E \in \boldsymbol{E_C}$
  - Each $E \vDash^E C$ and $\mathrm{M} \vDash^M C$ must be composable



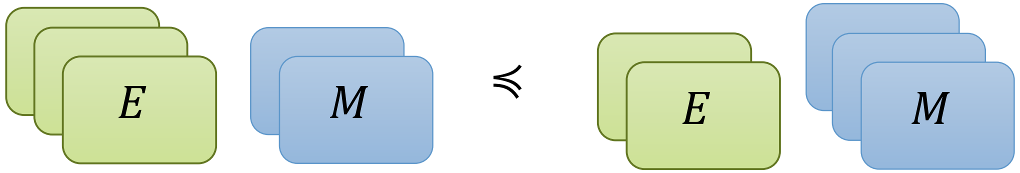
$$C = (\boldsymbol{E_C} , \boldsymbol{M_C})$$

  - We say, C is *consistent* iff it has at least one implementation: $\boldsymbol{M_C} \neq \emptyset$
  - We say, C is *compatible* iff it has at least one environment: $\boldsymbol{E_C} \neq \emptyset$

Contract Based Design
Refinement, Composition

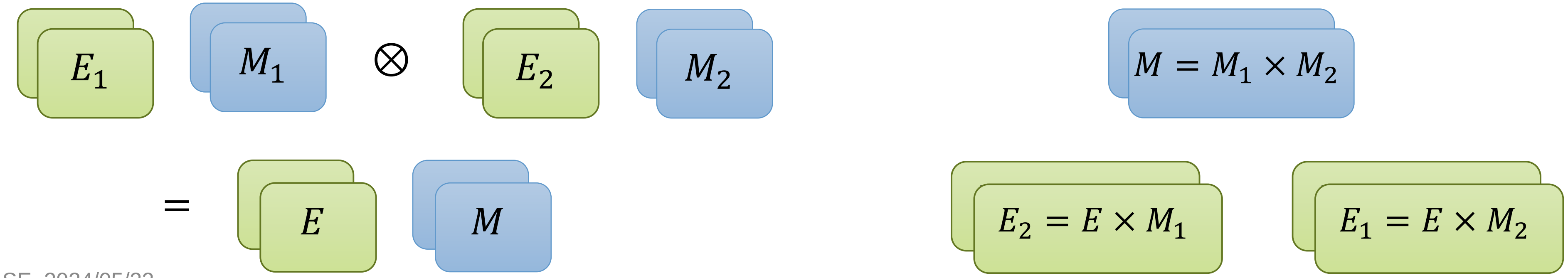
- Refinement

Contract C' refines C, $C' \preccurlyeq C$, iff $\boldsymbol{E}_{\boldsymbol{C'}} \supseteq \boldsymbol{E}_{\boldsymbol{C}}$ and $\boldsymbol{M}_{\boldsymbol{C'}} \subseteq \boldsymbol{M}_{\boldsymbol{C}}$



- Composition

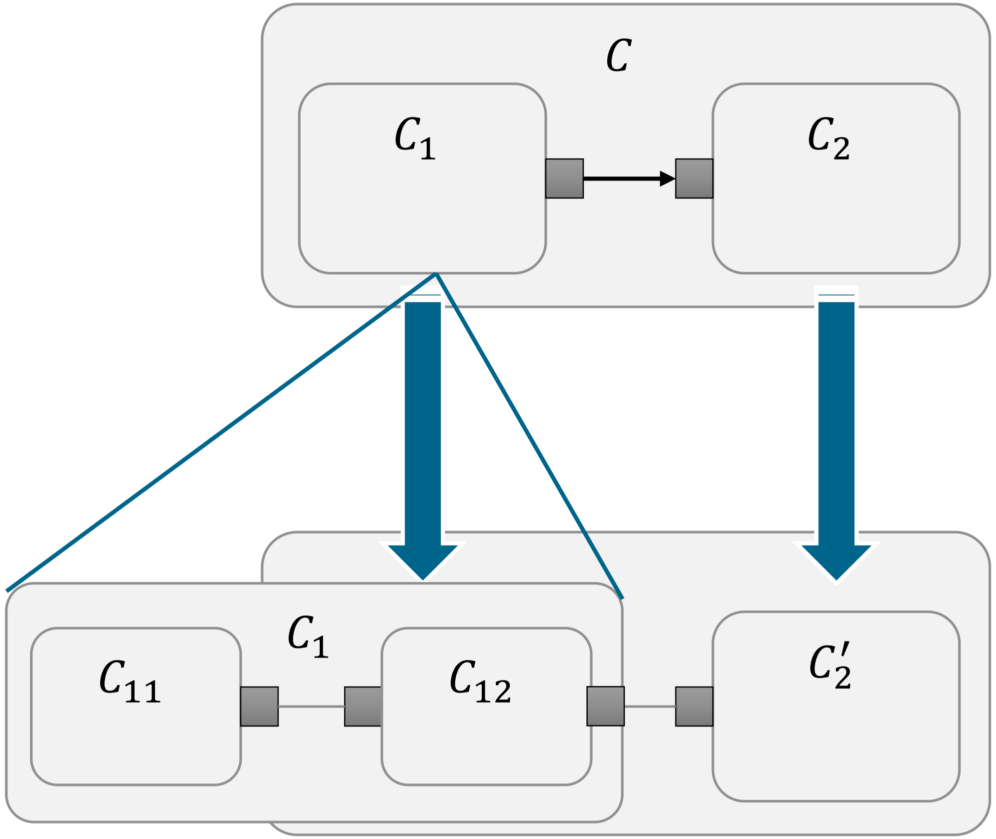
$$C_1 \otimes C_2 = min\left\{ C \;\middle|\; \begin{bmatrix} \forall M_1 \vDash^M C_1 \\ \forall M_2 \vDash^M C_2 \\ \forall E \vDash^E C \end{bmatrix} \Longrightarrow \begin{bmatrix} M_1 \times M_2 \vDash^M C \\ E \times M_1 \vDash^E C_2 \\ E \times M_2 \vDash^E C_1 \end{bmatrix} \right\}$$



Contract Based Design
Important Properties

- Refinement
  - Let be $C' \preccurlyeq C$
  - Any implementation of C' is an implementation of C
$$M \vDash^M C' \Longrightarrow M \vDash^M C$$
  - Any environment of C is an environment of C'
$$E \vDash^E C \Longrightarrow E \vDash^E C'$$
- Independent Implementability
  - Let be $C_1' \preccurlyeq C_1$ and $C_2' \preccurlyeq C_2$
  - The composition of C_1' and C_2' refines the composition of C_1 and C_2
$$C_1' \otimes C_2' \preccurlyeq C_1 \otimes C_2$$

Contract Based Design
Important Properties Applied



$C_1 \otimes C_2 \preccurlyeq C$ Virtual Integration Test

$$\left.\begin{matrix} C_1' \preccurlyeq C_1 \\ C_2' \preccurlyeq C_2 \end{matrix}\right\} \Longrightarrow C_1' \otimes C_2' \preccurlyeq C_1 \otimes C_2 \preccurlyeq C$$

$$(C_{11} \otimes C_{12}) \otimes C_2' \preccurlyeq C_1 \otimes C_2$$

Contract Based Design
Virtual Integration Test

- Recall
  - $C' \preccurlyeq C$ iff $\boldsymbol{E}_{\boldsymbol{C'}} \supseteq \boldsymbol{E}_{\boldsymbol{C}}$ and $\boldsymbol{M}_{\boldsymbol{C'}} \subseteq \boldsymbol{M}_{\boldsymbol{C}}$
  - $C_1 \otimes C_2 = min\left\{ C \;\middle|\; \begin{bmatrix} \forall M_1 \vDash^M C_1 \\ \forall M_2 \vDash^M C_2 \\ \forall E \vDash^E C \end{bmatrix} \Longrightarrow \begin{bmatrix} M_1 \times M_2 \vDash^M C \\ E \times M_1 \vDash^E C_2 \\ E \times M_2 \vDash^E C_1 \end{bmatrix} \right\}$

- Virtual Integration Test
  - In top-down design processes, often C is given as well as C_1 and C_2.
    - Calculating $C_1 \otimes C_2$ and checking $C_1 \otimes C_2 \preccurlyeq C$ is not necessary.
  - For any contract C with

$$\begin{bmatrix} \forall M_1 \vDash^M C_1 \\ \forall M_2 \vDash^M C_2 \\ \forall E \vDash^E C \end{bmatrix} \Longrightarrow \boxed{\begin{bmatrix} M_1 \times M_2 \vDash^M C \\ E \times M_1 \vDash^E C_2 \\ E \times M_2 \vDash^E C_1 \end{bmatrix} \quad \text{Virtual Integration Test}}$$

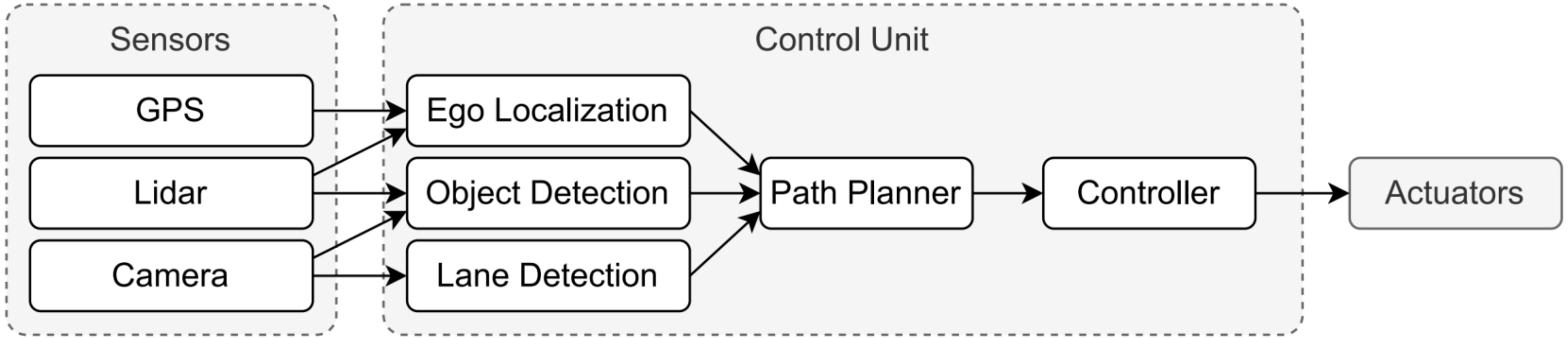
holds that $C_1 \otimes C_2 \preccurlyeq C$.

Agenda

- Compositional Semantic Framework
- A "Meta Theory" of Contract Based Design
- **Example**

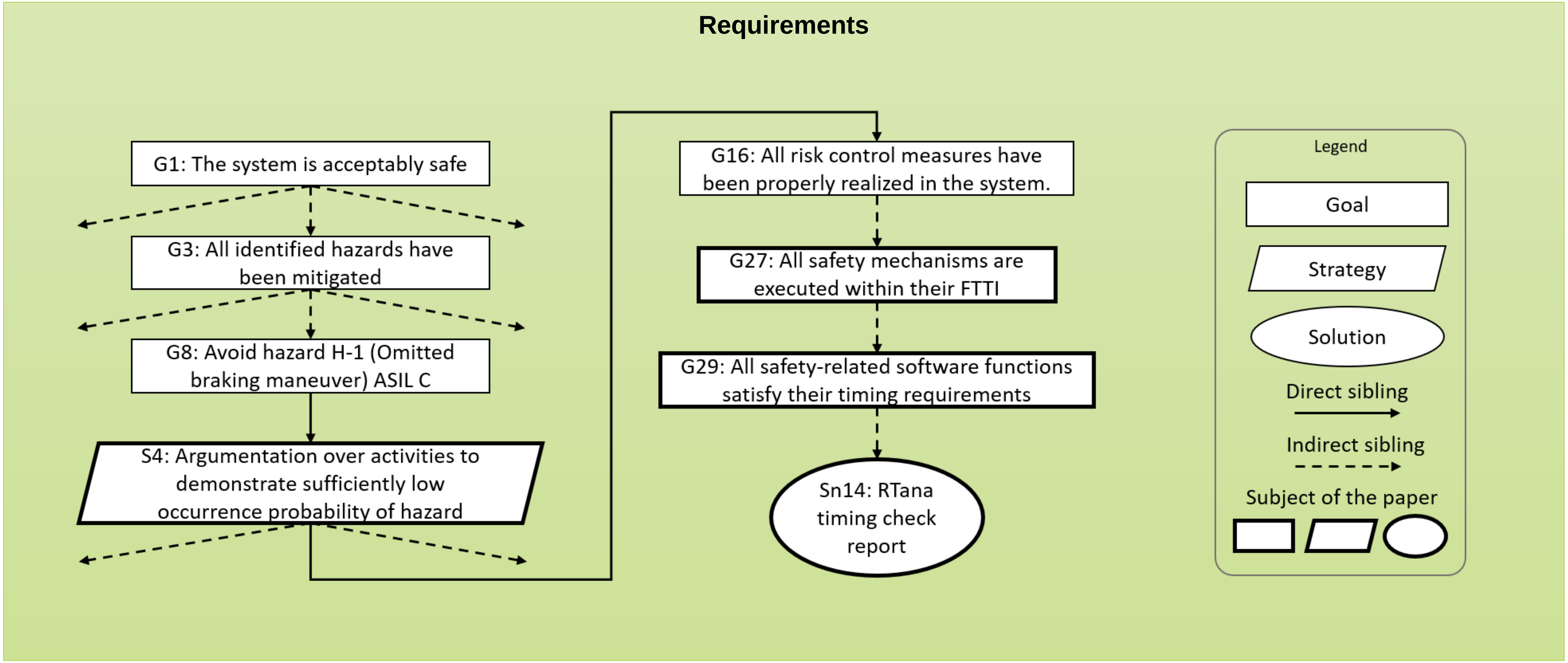
Example – Overview

- (Part of) a highly automated vehicle that shall perform two main functions:
  1. following a predefined route,
  2. avoiding collisions with traffic participants / obstacles.
- Sensors, Control Unit, Actuators:
  - Ego Localization (localize the ego vehicle),
  - Object Detection (detect, localize, and classify traffic participants and obstacles),
  - Lane Detection (detect lane boundaries),
  - Path Planner (plan maneuvers),
  - Controller (and calculate actuator commands).



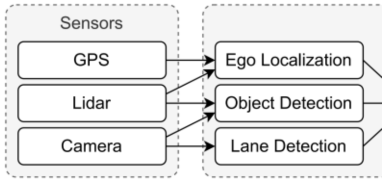
[Becker, Stierand, Koopmann, Westhofen: Providing Evidence for Correct and Timely Functioning of Software Safety Mechanisms, ASE 2023]

Safety Case (Excerpt)



HARA – Safety Goals – Causes – Safety Concept

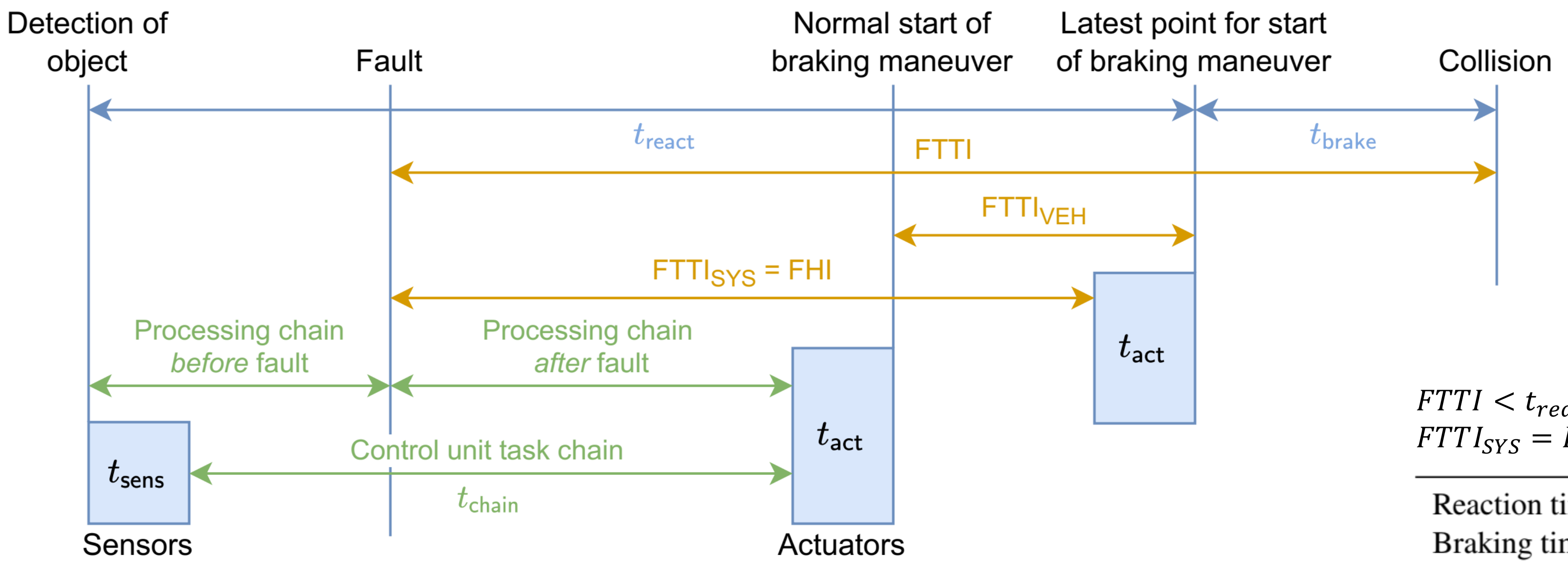
- HARA: Identification of hazards and corresponding risks
  - Here: ***H-1: Omitted braking maneuver***
    (a necessary braking maneuver of the ego vehicle is not performed (in time))
- Two failure modes are considered:
  1. One of *Lidar* or *Camera* fails permanently.
  2. One or both of *Lidar* and *Camera* fail for a limited amount of time.



- Safety Requirements:

ID	Description
SG-1	The system shall prevent omitting required braking maneuvers.
SR-1-1a	The system shall use lidar and camera for object detection.
SR-1-1b	The system shall ensure that objects are detected if lidar or camera fails.
SR-1-1.1	The system shall identify sensor failures.
SR-1-1.1.1	The system shall identify when the lidar sensor has failed.
SR-1-1.1.2	The system shall identify when the camera has failed.
SR-1-1.2	The system shall mitigate sensor failures.
SR-1-1.2.1	The system shall detect objects using lidar.
SR-1-1.2.2	The system shall detect objects using camera.
SR-1-1.2.3	The system shall fuse the objects detected by lidar and camera.
SR-1-1.3	The system shall use only information from working sensors.
SR-1-2	. . .

Fault Tolerant Time Interval (FTTI)

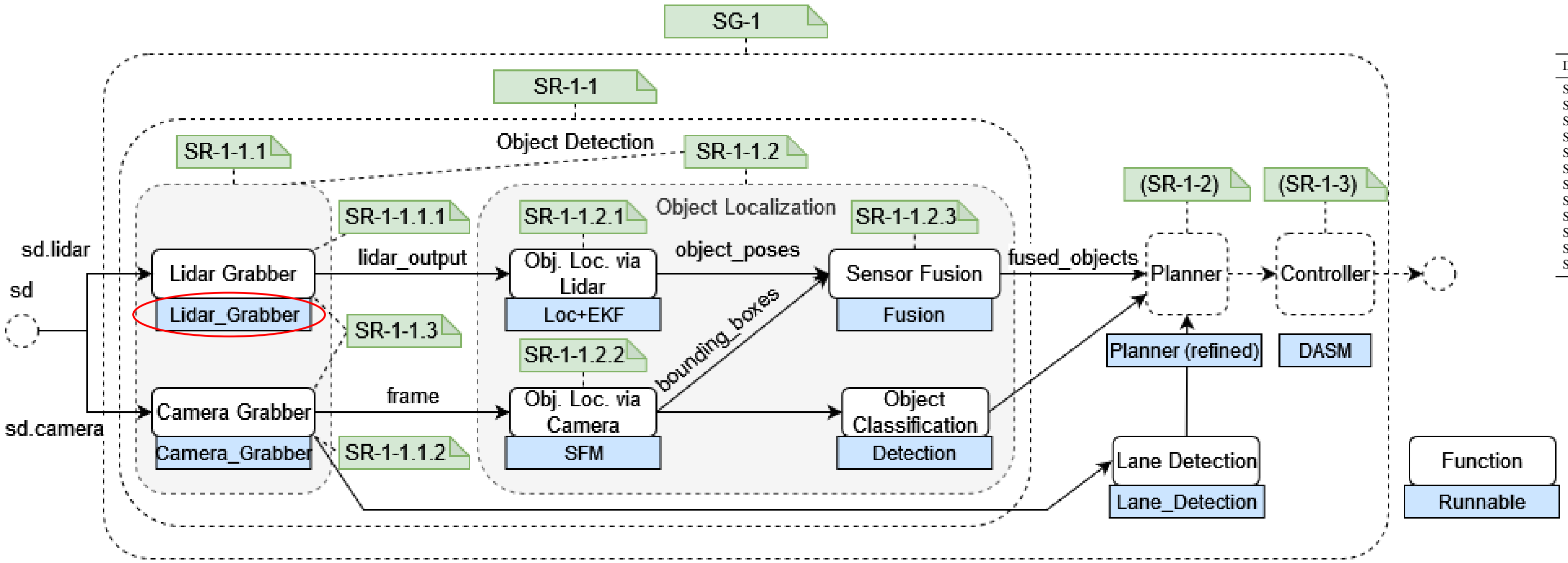


$$FTTI < t_{react} + t_{brake} \approx 4.57s$$
$$FTTI_{SYS} = FHI < t_{react} + t_{act} + t_{sense} \approx 2.43s$$

Reaction time	t_{react}	2.57 s
Braking time	t_{brake}	2.0 s
Sensor processing time	t_{sense}	50 ms
Actuator time	t_{act}	100 ms
Control unit WCRT	t_{chain}	< 800 ms

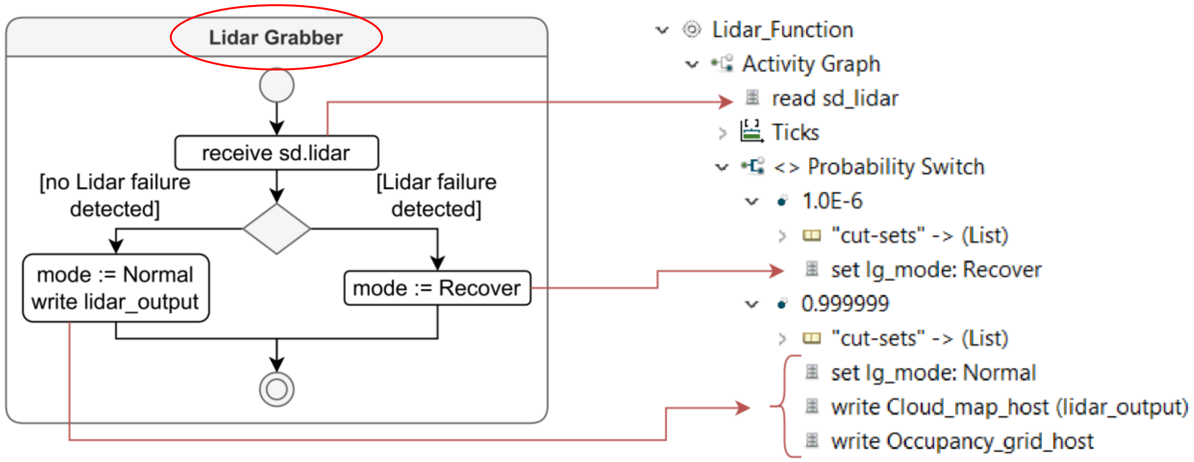
Remark: We consider a *safety mechanism implemented with emergency operation* [ISO26262:2018]

Mapping Requirements to Technical Architecture



ID	Description
SG-1	The system shall prevent omitting required braking maneuvers.
SR-1-1a	The system shall use lidar and camera for object detection.
SR-1-1b	The system shall ensure that objects are detected if lidar or camera fails.
SR-1-1.1	The system shall identify sensor failures.
SR-1-1.1.1	The system shall identify when the lidar sensor has failed.
SR-1-1.1.2	The system shall identify when the camera has failed.
SR-1-1.2	The system shall mitigate sensor failures.
SR-1-1.2.1	The system shall detect objects using lidar.
SR-1-1.2.2	The system shall detect objects using camera.
SR-1-1.2.3	The system shall fuse the objects detected by lidar and camera.
SR-1-1.3	The system shall use only information from working sensors.
SR-1-2	. . .

Implementation in APP4MC:

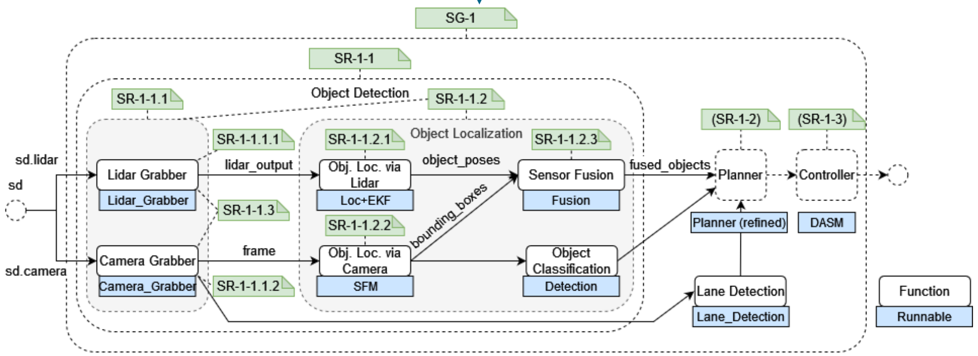


Mapping Requirements to Technical Architecture
Timing Contracts

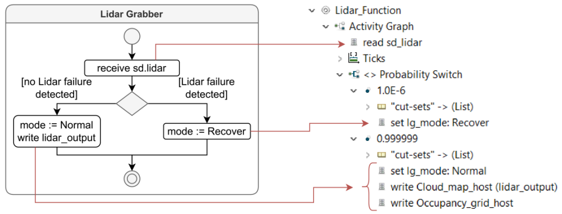
A: sd occurs every 66 ms. G1: Reaction(sd.OK,fused_objects) within [100,1006] ms in mode Normal. G2: Reaction(sd.NOT_OK,set(mode,Recover)) within [5,66] ms in mode Normal. G3: Reaction(sd.OK,fused_objects) within [100,940] ms in mode Recover.

⊩ ? (VIT)

Lidar Grabber A_{L1}: sd.lidar occurs every 66 ms. G_{L1}: Reaction(sd.lidar.OK,(set(mode,Normal),lidar_output)) within [5,60] ms. G_{L2}: Reaction(sd.lidar.NOT_OK,(set(mode, Recover)) within [5,60] ms.
Camera Grabber A_{C1}: sd.camera occurs every 66 ms. G_{C1}: Reaction(sd.camera.OK,(set(mode,Normal),lidar_output)) within [7,60] ms. G_{C2}: Reaction(sd.camera.NOT_OK,(set(mode,Recover)) within [7,60] ms.
Object Localization via Lidar G_{OL}: Reaction(lidar_output,object_poses) within [120,795] ms.
Object Localization via Camera G_{OC}: Reaction(frame,bounding_boxes) within [20,55] ms.
Sensor Fusion G_{F1}: Reaction(object_poses,fused_objects) within [5,25] ms. G_{F2}: Reaction(bounding_boxes,fused_objects) within [5,25] ms.



⫤ ?

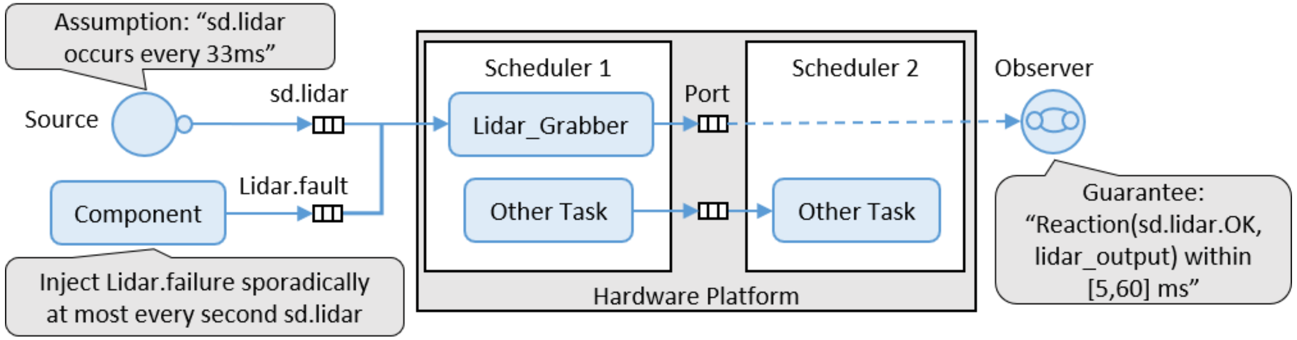


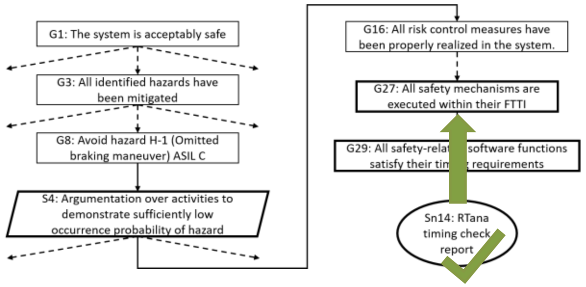
Analysis

Approach and Structure of the Rtana$_{2sim}$ Model

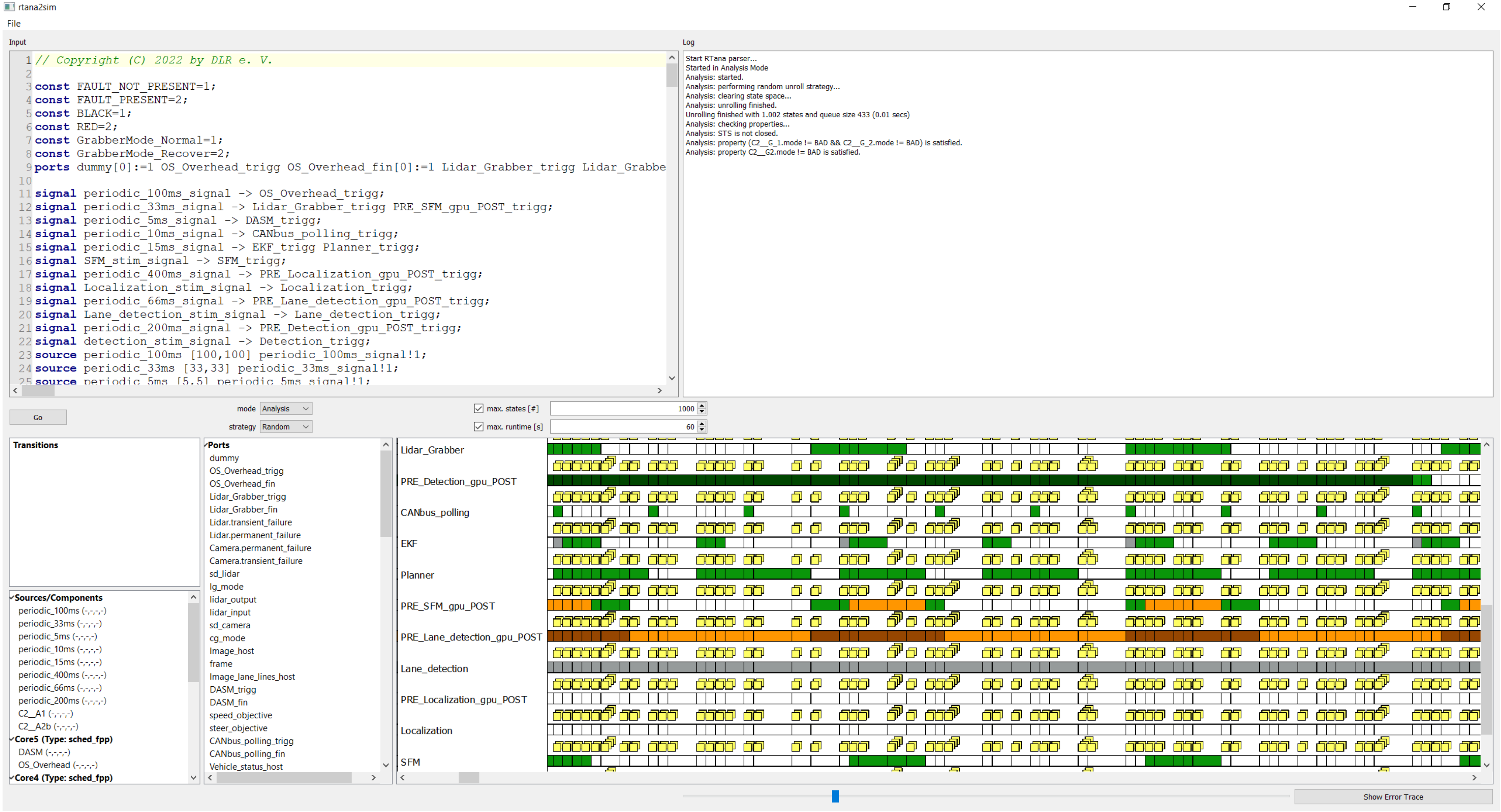
- APP4MC model is translated into RTana$_{2sim}$ model.
- Assumptions are translated into event sources.
- Guarantees are translated into observer automata.
- Additional components provide for fault injection.

A: sd occurs every 66 ms.
G1: Reaction(sd.OK,fused_objects) within [100,1006] ms in mode Normal.
G2: Reaction(sd.NOT_OK,set(mode,Recover)) within [5,66] ms in mode Normal.
G3: Reaction(sd.OK,fused_objects) within [100,940] ms in mode Recover.





Analysis
Example



Conclusion

- Model-based systems engineering helps in solving many challenges in engineering processes.
- The SPES modelling framework aims at supporting engineering of (safety-critical) CPS.
- Contract-based design provides formal design and engineering support:
  - Correctness of key design steps becomes verifiable.
  - Enables tool support in verification tasks.
- Industrial example demonstrates applicability.