in-tech
an Infosys company

Beat the complexity with MBSE

Mastering complexity with early validation and Model-based Development

www.in-tech.com

The Hannover Messe 2025 Showcase

Digital transformation is reshaping industries, driving products toward intelligent, software-defined solutions. However, increasing complexity poses significant challenges. To address these, a structured approach combining Systems Engineering, AI-driven development, and Virtual Validation is essential. Our showcase demonstrates these key concepts using a real-world example: traffic sign recognition in a vehicle.

Content Overview

Introduction

The Hannover Messe 2025 Showcase

Introduction

1 Four Big Challenges Every Systems Engineer Faces Today.

The digital transformation is reshaping industrial and automotive products, transitioning them from electromechanical systems to intelligent, software-driven solutions. Cloud technologies, IoT, and AI enable entirely new functionalities that enhance user experience and set new industry standards. These advancements pave the way for innovative business models, such as digital services and personalized customer experiences. At the same time, customer expectations have evolved. They now demand smart products that are always up to date, easy to use, and customizable to individual preferences.

While these advancements bring enormous opportunities, they also introduce significant challenges. The complexity of developing and maintaining software-defined products has increased dramatically, making traditional development methods insufficient. The challenges associated with this transformation can be categorized into four primary areas.

- **Software defined products**
- System of Systems
- Multi-domain
- **New Technologies (e.g. AL)**
- Variants and Platforms
- **Shorter Product Lifecycles (TTM)**
- Continuous Development
- Agile Engineering
- Circular Economy
- End of Life Management

- **Safety**
- **Cybersecurity**
- Reliability
- Avalability
- Resilience
- Sustainability

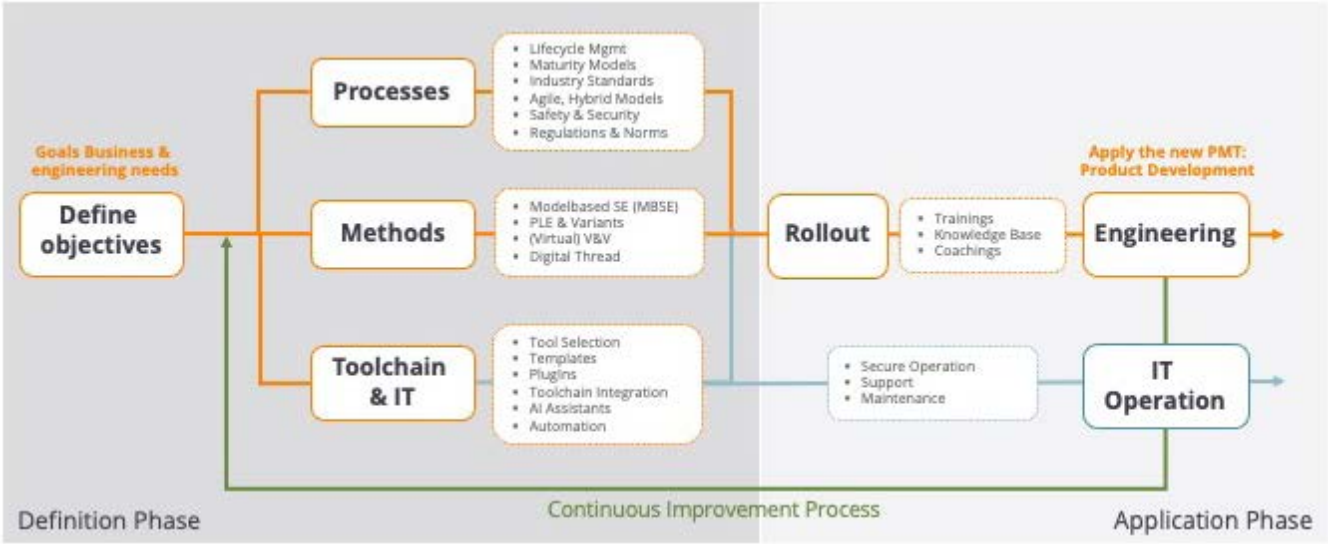
- Large Organisations
- Interdisciplinary Teams
- **Partner and supply chains**
- Global cooperation
- Cultures, Timezones, Languages
- **Agile and hybrid methodologies**
- Product as a service models

- **Regulatories**
- Legal
- **Directives**
- Norms
- Privacy

Managing these complexities with traditional development methods is highly challenging and often leads to inefficiencies and increased risks. The increasing integration of software, evolving product architectures, and accelerating market demands require new approaches to ensure efficiency, reliability, and compliance across the entire product lifecycle.

2 The Transformation Challenge

The introduction of a holistic systems engineering approach is not just about purchasing and integrating new tools. It represents a fundamental transformation of the development process and requires a structured approach tailored to an organization's current maturity level. This transformation encompasses adjustments to processes, methods, toolchains, and IT systems, involving their definition, rollout, and continuous optimization.



The primary focus of this whitepaper is on the aspect of Model-Based Systems Engineering (MBSE), which we will detail in the following sections and demonstrate through our showcase.

3 MBSE: Your Single Source of Truth for Complex Systems

In today's rapidly evolving product development landscape, both organizational and technical dependencies are dramatically increasing. Products are becoming interdisciplinary, development cycles more agile and faster, and expectations for transparency and end-to-end consistency have never been higher. Traditionally, systems engineering has relied on documents or various discipline-specific tools that describe different aspects - such as specifications, functional descriptions, or interface documentation - in natural language. However, these documents frequently contain redundant information, quickly losing

consistency over time and across multiple contributors. Additionally, natural language is often ambiguous, information is scattered, and administrative overhead remains substantial as different stakeholders require individually tailored views.

How can these challenges be addressed effectively? MBSE offers a powerful solution. With MBSE, all critical information relevant to systems engineering - such as requirements, interfaces, systems, functions, properties, and parameters, - is captured and defined centrally within a single, cohesive database or model. Dependencies among elements are explicitly represented through hierarchical structures, instantiations, or linkages. A standardized notation, such as the Systems Modeling Language (SysML), is utilized. SysML Version 1 extends the Unified Modeling Language (UML), while the upcoming Version 2 introduces further enhancements specifically designed for systems engineering.

This centralized model enables stakeholders to generate multiple tailored views suitable for diverse project phases or specific stakeholder needs. Such views may include various SysML-defined diagrams (e.g., block diagrams, activity diagrams, sequence diagrams), structured lists (such as Bill of Materials or interface definitions), code artifacts, or even traditional documents required by reviewers or regulatory authorities.

The benefits of adopting MBSE are significant:

Consistency: Because each element exists only once within a central model, all derived views and documentation consistently reference the same information, eliminating redundancy and discrepancies.

Transparency: Centralizing information enhances visibility across disciplines and departments, promoting communication, collaboration, and reducing silo mentality.

Traceability: Explicitly modeled dependencies significantly improve traceability, vital for verification processes, safety-critical systems, comprehensive testing, and impact analyses necessary for managing system changes.

Clarity: Using a standardized, structured notation reduces ambiguities typically associated with natural language, ensuring greater precision and clearer communication.

In short, MBSE fundamentally transforms systems engineering by providing a unified, transparent, traceable, and precise approach to managing complexity in product development.

The Hannover Messe 2025 Showcase

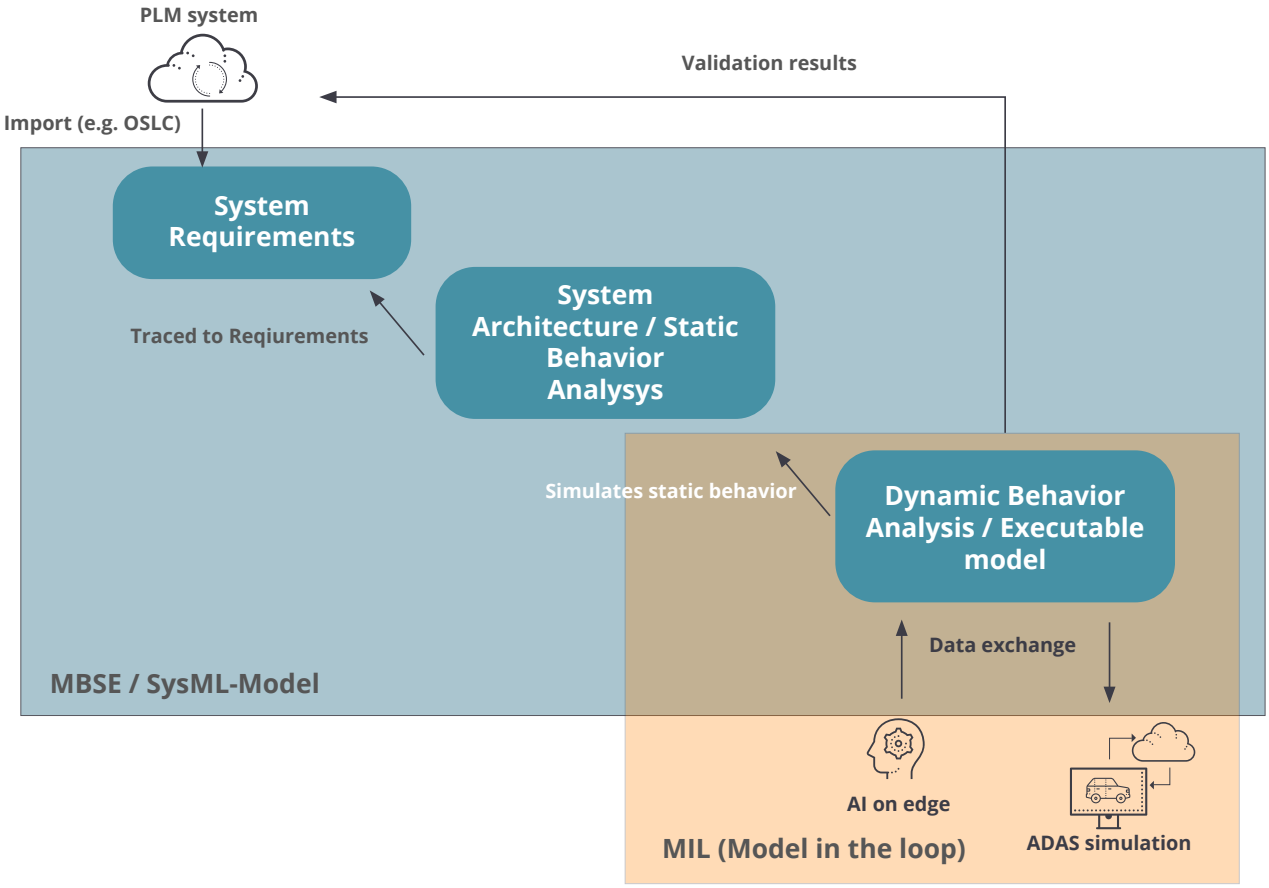
1 Integrating Requirements Engineering into Model-Based Systems Engineering

In the realm of system development, defining system requirements is a crucial step that is often facilitated by specialized requirements engineering tools. These tools play a vital role in capturing, analyzing, and managing requirements to ensure that the final system meets the intended needs and specifications. However, as the complexity of systems increases, there is a growing need to integrate requirements engineering into a model-based environment, thereby ensuring the continuity of the digital thread throughout the development process.

Model-Based Systems Engineering (MBSE) offers a powerful approach to system development by providing a structured framework for modeling and analyzing complex systems. While MBSE does not replace traditional requirements engineering, it enhances its capabilities by making it an integral part of the digital thread. This integration allows for seamless traceability and verification of requirements, ensuring that they are consistently linked to architecture model elements.

One of the key benefits of shifting to model-based requirements engineering is the ability to demonstrate standard-compliant traceability. By linking requirements directly to architecture model elements, organizations can provide proof of traceability that meets industry standards. This is particularly important in regulated industries where compliance with standards is mandatory.

Requirements can be directly modeled using languages such as SysML, which provides a standardized way to represent system requirements within the model-based environment. Additionally, requirements can be imported or reflected into the model-based environment using lifecycle management tools and standardized interfaces like Open Services for Lifecycle Collaboration (OSLC). This flexibility ensures that requirements are accurately captured and maintained throughout the system development lifecycle.



Showcase implementation

The development of an Advanced Driver Assistance System (ADAS) controller, specifically designed for traffic sign recognition, is a complex process that involves several critical steps. This showcase highlights the creation of an ADAS controller, referred to as ADS, using the example of a traffic sign recognition system as part of the system functionality.

The initial phase of the development process is the definition of system requirements, as illustrated in figure 1. These requirements serve as the foundation for the subsequent model-based architecture and behavior specification. They establish the functional framework necessary for the traffic sign recognition system and outline the intended reactions of the ADAS controller. The showcase uses basic SysML notation for the modeling of the requirements directly in the MBSE toolchain.

1	REQ_001	The ADS shall only be used on motorways (Autobahnen) and main roads (Bundesstraßen).
10	REQ_010	When the ADS detects a traffic sign indicating a hazard, the ADS shall deactivate.
11	REQ_011	When the driver takes over control of the longitudinal behavior of ego vehicle, the ADS shall subordinate its actuation to the driver's input.
12	REQ_012	The ADS must be authorised for road traffic in Germany.
13	REQ_013	The ADS must comply with the StVO.
14	REQ_014	The ADS shall react to vehicles ahead and ensure a save distance.
15	REQ_015	Das ADS shall react to cut-ins and ensure a save distance.
16	REQ_016	When ADS is activated and no authorised maximum speed was detected, the ADS shall comply with the recommended speed.
17	REQ_017	The ADS shall control the longitudinal movement.
18	REQ_018	The ADS shall control the lateral movement.
19	REQ_019	The ADS shall adapt to the environmental conditions.
2	REQ_002	The ADS shall be activated and deactivated by the driver.
20	REQ_020	The ADS shall be developed in accordance with ASPICE, ISO26262, ISO21448, ISO21434.
21	REQ_021	The ADS shall meet the following non funtional requirements: ·Software update over the air ·Safety according the state of the art ·ODD Coverage
22	REQ_022	The ADS shall detect the traffic signs using two independent sensors.
24	REQ_024	When the ADS detects a traffic sign whose meaning can not be derived, the ADS shall deactivate.
25	REQ_025	When the ADS deactivates, the ADS shall visually infrom the driver about the deactivation.
3	REQ_003	The ADS shall detect the following traffic signs indicating instructions on longitudinal movement: - Authorised maximum speed 5 km/h (Zeichen 274-5) and corresponding end of the maximum authorised speed (Zeichen 278-5)- Authorised maximum speed 10 k...
4	REQ_004	The ADS shall detect the following traffic signs indicating hazards: - Uneven road surface (Zeichen 112)
5	REQ_005	The ADS shall detect the following traffic signs indicating instructions on lateral movement: - No overtaking for motor vehicles of all kinds (Zeichen 276)- End of the overtaking ban for motor vehicles of all kinds (Zeichen 280)
6	REQ_006	The ADS shall visually inform the driver about taffic signs indicating hazards and instructions longitudinal movement.
7	REQ_007	When the ADS detects a traffic sign indicating restrictions on the authorised maximum speed, the ADS shall adapt the ego speed to this authorised maximum speed.
8	REQ_008	When the ADS detects a traffic sign indicating the end of a maximum authorised speed, the ADS shall adapt the ego speed to recommended speed 130 km/h.
9	REQ_009	When the ADS is not capable to recognize a traffic sign, the ADS shall be deactivated.

Figure 1 Excerpt from the list of system requirements displayed in a table format

The primary function of the system is to accurately recognize speed limitations and identify potential danger areas on the road. Recognized speed limits are then used as input for the Adaptive Cruise Control (ACC) system. This integration ensures that the vehicle adjusts its speed according to the detected traffic signs, enhancing safety and compliance with road regulations.

In cases where errors occur in the detection of traffic signs or when warning signs, such as upcoming road works, are correctly identified, the system is designed to deactivate the ACC. Additionally, a warning message is issued to the driver, alerting them to the situation and prompting them to take appropriate action.

	React_to_Traffic_Signs
REQ_012	
REQ_013	
REQ_014	
REQ_015	
REQ_016	REQ_016
REQ_022	
REQ_017	REQ_017
REQ_018	
REQ_019	
REQ_020	
REQ_021	
REQ_001	
REQ_002	
REQ_003	REQ_003
REQ_004	REQ_004
REQ_005	
REQ_006	REQ_006
REQ_007	REQ_007
REQ_008	REQ_008
REQ_009	REQ_009
REQ_010	REQ_010
REQ_011	
REQ_024	REQ_024
REQ_025	REQ_025

Figure 2 Trace linkage of the use cases to the system requirements

To ensure traceability, the system requirements are directly linked to use case elements within the model through SysML trace relationships, as depicted in figure 2. These use cases form part of the forthcoming static behavioral model, providing a clear and structured representation of the system's intended functionality. For the showcase only one use case for the reaction to recognized traffic signs is used.

Modeling use cases can be interpretated as part of the system context definition or a black box behavioral view. Using SysML Use Case diagrams, the involved actors, which means elements from the system context interacting with the system of interest, such as parts of the vehicle or the vehicle's surroundings are identified. A graphical representation including the relationships between actors (active or passive) and the use case is shown in Figure 3.

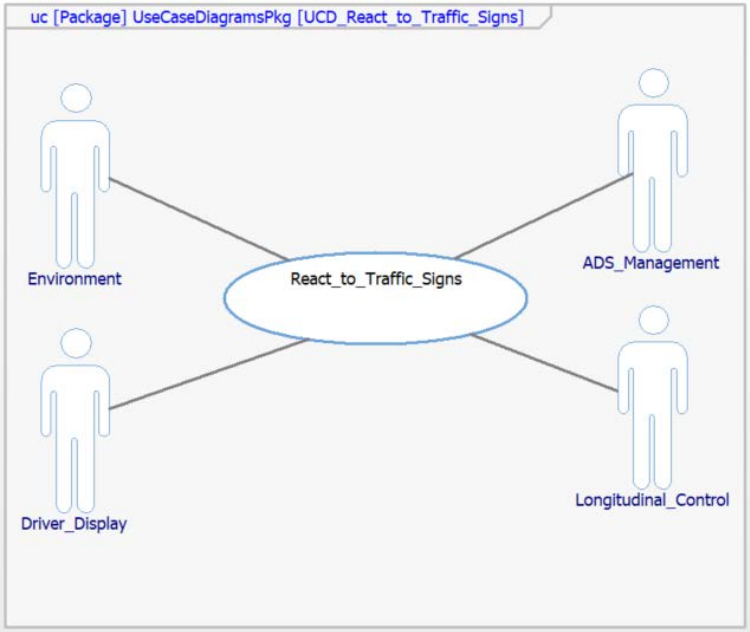


Figure 3 Modeling of the use case under consideration

Analyzing the requirements gives us an indication of which actors are involved in addition to the system itself.

The **Environment** plays a vital role in the use case. It surrounds the vehicle and contains the traffic signs and delivers the input upon which the system will react. **ADS Management** is superordinate to the system under consideration and controls whether the system is active or inactive. **Driver Display** and **Longitudinal Control** consume the output in form of the velocity setpoint and messages to the driver. It is important to mention that the actor Environment is substituted by the external AI on edge device.

2 Model-Based Specification of System Architecture and Static Behavior

The development of complex systems necessitates a structured approach to defining system architecture, which is grounded in the initial system requirements. This process involves creating model-based specifications that encompass various architectural representations,

including functional, logical, and physical architectures. Each representation serves a distinct purpose, and their necessity must be aligned with specific project or product constraints.

Functional and logical architectures are solution-independent, focusing on the essential functions and logical operations of the system without delving into specific solutions. In contrast, physical architectures are solution-dependent, detailing the tangible components and their interactions within the system. The decision to include each type of architecture in the specification process depends on the unique requirements and limitations of the project or product.

Static, or non-executable, system behavior is often incorporated into the solution-independent functional architecture. This aspect of the architecture provides a clear understanding of how the system is expected to function under various conditions, without specifying the exact implementation details.

The SysML language plays a crucial role in this model-based approach by offering standardized diagrams and modeling options for both architecture and behavior modeling. SysML facilitates essential linking and traceability relationships throughout the architecture and behavior model, ensuring that all elements are interconnected and can be traced back to the original system requirements.

One of the significant advantages of adopting a model-based architecture and behavior specification is the ability to integrate variant management within a Product Line Engineering (PLE) approach. This integration can occur on a functional basis, allowing for the management of different functional variants within a product line. Additionally, it supports trade-off studies to evaluate various physical realization variants, enabling organizations to make informed decisions about the most suitable physical implementations.

Showcase implementation

We now analyze the functional aspects of the “React to Traffic Signs” use case. The functional analysis is separated from the foundational model and contains replicates of the use case and actor elements. This avoids unintended interactions when multiple analyses are

performed. The upcoming static behavioral modeling is clearly traceable to the respective use case and its related system requirements.

The static behavior modeling process is divided into **Activity Modeling** and **Interaction (Sequence) Modeling**. Static behavior modeling focuses on analyzing and representing a system's behavior and structure without execution. It describes how components relate to each other and what possible states exist, but it does not simulate changes over time or interactions in real execution. The primary goal is to understand and document the system's potential behavior based on predefined rules and relationships.

Activity Modeling consolidates textual requirements into a unified behavioral context, identifying dependencies and behavioral cases. It emphasizes the logical sequence of activities while disregarding timing aspects. By modeling a single SysML activity diagram, we capture an overarching view of system functions and interactions between the system under consideration and actors. The step allows capturing a white box view of the system function or use case. Figure 4 illustrates the activity diagram. The control flow goes from top to bottom in a linear manner. Four diamond-shaped elements are located on the diagram. The first two form the Decision Nodes based on which the behavioral cases are differentiated. **First node** distinguishes whether the system is currently active or inactive. Nevertheless, three basic actions are performed right from the beginning in both cases. If the system is inactive, the action sequence is immediately terminated (Right-pointing path). If the system is active, the **second node** further differentiates upon which traffic sign was detected. A distinction is made between traffic signs indicating speed limitations, end of speed limitations and hazards. Additionally, the possible case of non-interpretability is considered. Subsequently individual actions are taken. The last two rhombic elements represent the Merge Nodes, at which the control flows converge.

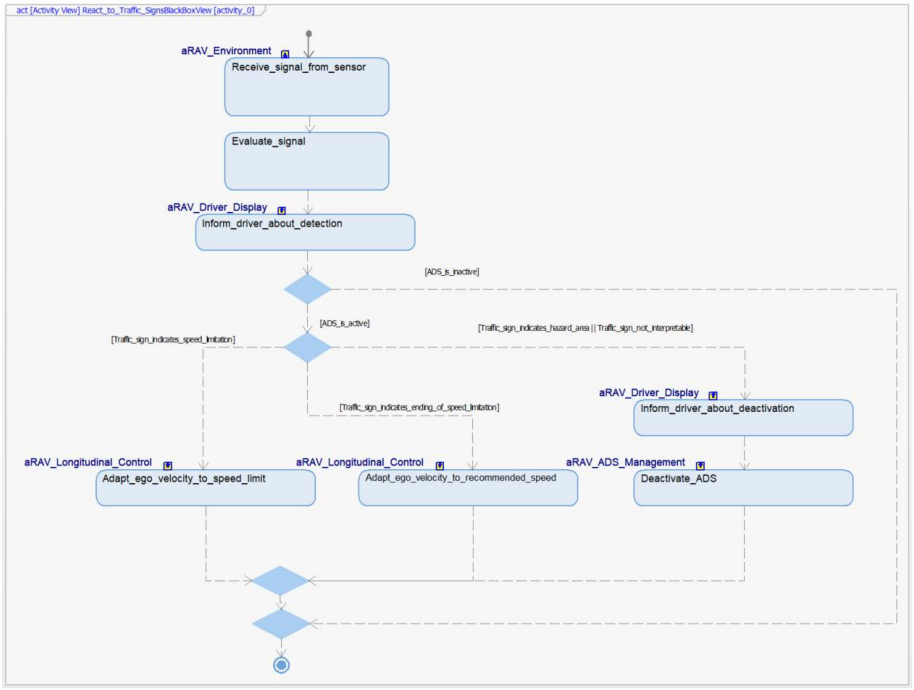


Figure 4 Activity modeling of selected use case

Interaction Modeling examines each possible information flow from the activity diagram, indicated by the decision nodes. This leads to a set of scenarios directly derived from the activity modeling represented by SysML sequence diagrams, incorporating relative or absolute timing where needed. For verification and validation aspects, these scenarios can be used a basis for the test case definition.

As a bridge to dynamic (or executable) behavior modeling, this step introduces operations (functions performed by an element) and receptions (functions triggered by events). Arguments can utilize complex SysML data structures, extending beyond primitive data types. Figure 5 depicts the left path of the activity diagram, where operations appear as self-messages, and receptions as inter-lifeline messages. Condition Marks, representing activity diagram Guards, aid case identification.

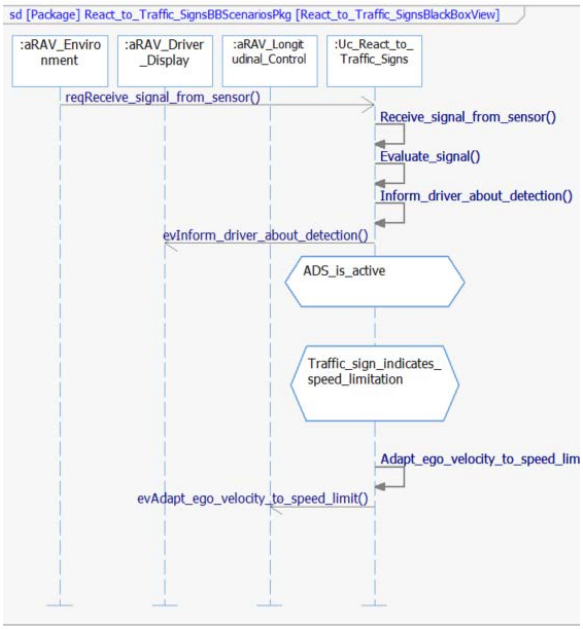


Figure 5 Sequence diagram

Finally, an SysML **Internal Block Diagram** is created as part of the system architecture, taking all actors and exchanged information between them and the system under consideration into account. These interfaces between the system and actors are modelled as proxy ports, typed by interface blocks. To maintain model consistency, interface blocks are assigned functions identified as receptions in the interaction modeling phase. Element instances are then connected using connectors, forming the foundation for dynamic behavior modeling. Figure 6 presents the architectural view reduced to the elements of the traffic sign recognition function, including the respective interface definitions.

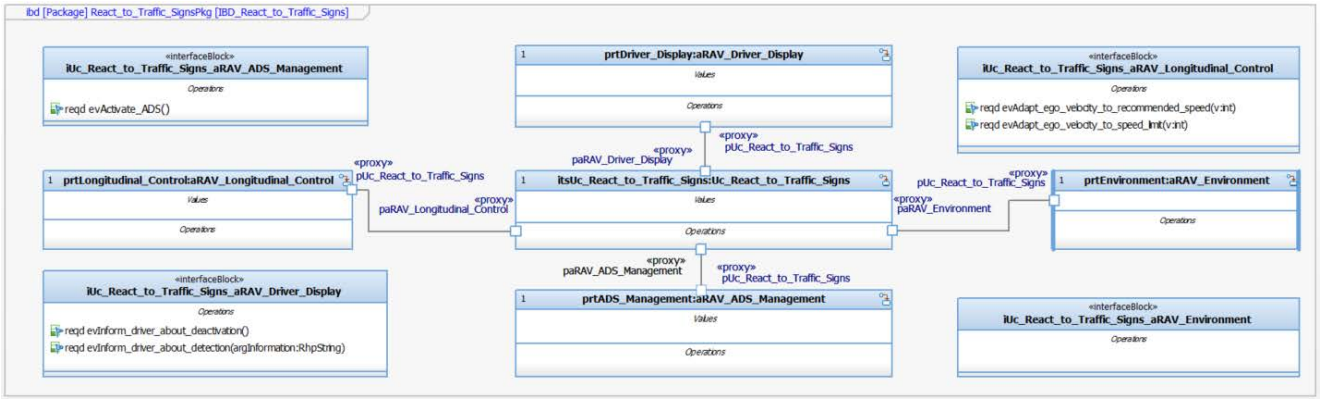


Figure 6 Internal Block Diagram

3 Leveraging Dynamic System Behavior for Early Validation in Model-Based Development

In the realm of system development, validating system requirements at early stages is crucial for ensuring that the final product meets its intended specifications. Dynamic, or executable, system behavior plays a pivotal role in this early validation process, as it allows for the simulation and testing of system requirements while the model-based specification remains virtual.

Executable system models are derived from the system architecture and static behavioral models, providing a functional basis for simulating system behavior. These models enable developers to explore how the system will perform under various conditions, offering insights into potential issues and areas for improvement before physical prototypes are created.

The SysML language offers diagrams for behavior modeling, including activity diagrams, sequence diagrams and state machines. These diagrams can be executed using appropriate tools, transforming static models into dynamic simulations that mimic real-world operations. By executing these SysML models, developers can validate system requirements and refine system designs in a controlled, virtual environment.

A key aspect of this approach is the integration of inputs and outputs from executable SysML models with simulated sensors, such as image recognition systems, and actors, like vehicle dynamic simulations. This integration establishes a Model-in-the-Loop (MiL) environment, which serves as a foundational step towards creating digital twins. MiL environments facilitate comprehensive system validations and verifications, allowing developers to test system behavior in a simulated setting that closely resembles real-world conditions.

The results obtained from MiL simulations can be transferred back to lifecycle management tools, ensuring traceability for testing aspects. This feedback loop guarantees that all testing activities are documented and linked to the original system requirements, providing a clear audit trail and supporting continuous improvement throughout the development process.

Showcase implementation

Dynamic behavior modeling, on the other hand, involves executable simulation, meaning

the model can be run to observe and analyze system behavior in real-time or through controlled scenarios. It allows for time-dependent changes, interactions, and execution-based validation. This approach helps in testing system functionality, verifying correctness, and analyzing behavior under different conditions.

The process consists of the state machine diagram modeling itself and the subsequent execution. A comprehensive simulation of the system and the actors requires behavioral representation of all involved elements. Each involved element owns a state machine diagram. The cross-diagram interactions are realized by Send Actions, which can be recognized by the arrow-like elements in Figure 7. The respective arrivals are located on transitions of the state machine diagram of another element.

As indicated in the Activity Diagram in the previous chapter, the system can either be active or inactive. This is realized by the two main states on the right and left in Figure 7. Both states have a substate, in which the received information is processed. The user-friendly information sent to the actor Driver Display, which takes place in both states. The active state extends this behavior by two substates and corresponding Send Actions. The system interacts with the actor Longitudinal Control by sending velocity setpoints. It can either be in the state, where the velocity is restricted to the current speed limitation, or in the state, where the velocity is restricted to recommended speed. Being in one of these substates, the system waits for a new signal.

After structurally modeling the state machine diagram, operations, receptions, and their corresponding events are involved. These are reused from sequence modeling or newly defined, when necessary. Operations do not have function bodies so far, which are added here. The return type is defined in this step as well. Operations are placed on the state's entry or exit or act as an effect to a transition. Events are used in Send Actions to enable interaction, as mentioned above. The respective receptions are typically located on transitions, where they act as triggers. Otherwise, they can be received by exclusive Accept Event Actions.

As indicated in the previous sections, the input would be normally provided by the actor Environment, since it contains the traffic signs. The whole showcase consists of three sub-showcases, MBSE, AI on edge and ADAS. The data is provided by the camera (AI on edge) and received by the operation "Receive signal from sensor". The ADAS vehicle simula-

tion is controlled by the output of the model and send in the operations “Adapt ego velocity to speed limit” and “Adapt ego velocity to recommended speed”.

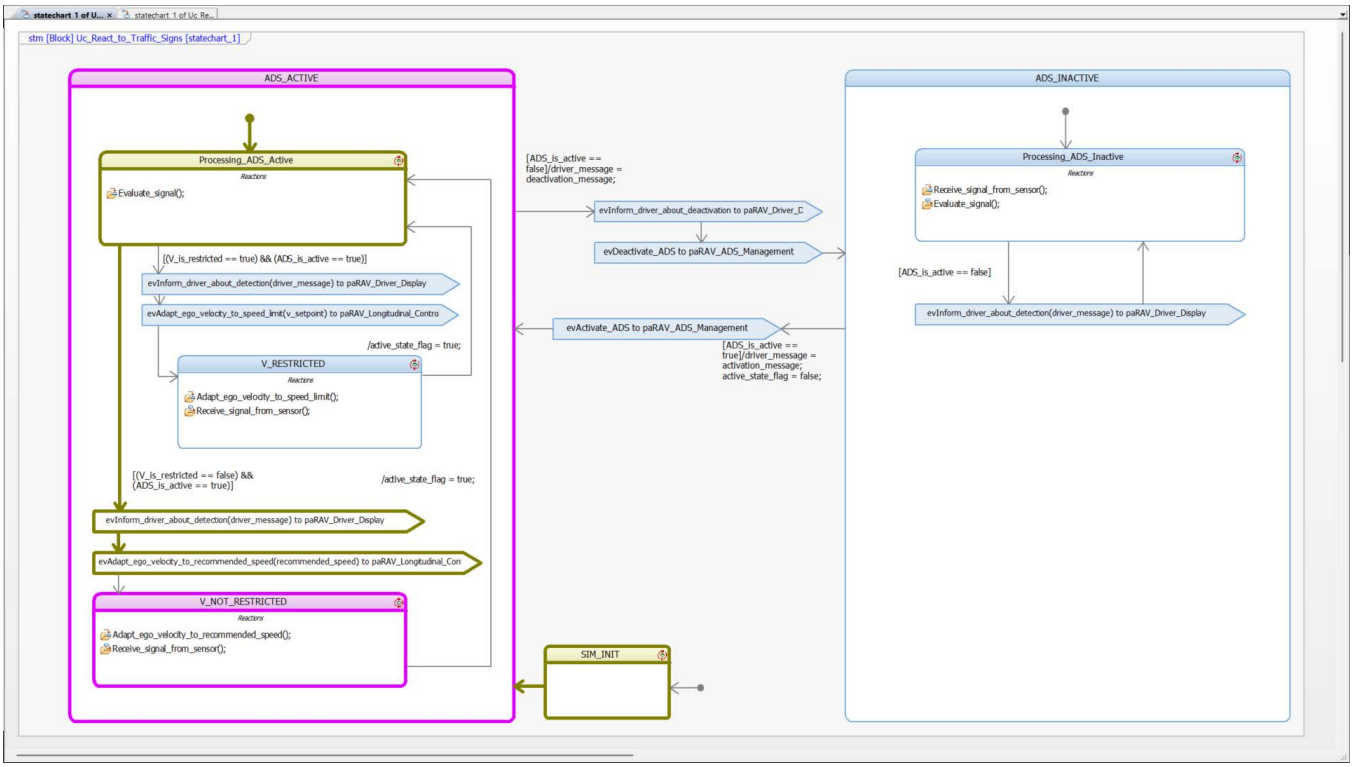


Figure 7 State machine diagram of the use case under consideration

Your Ideal Partner for MBSE Implementation

Digital transformation powered by emerging technologies unlocks unprecedented opportunities – but also increases complexity due to new collaboration models, dependability requirements, and regulatory compliance challenges. Successfully implementing Model-Based Systems Engineering (MBSE) demands comprehensive expertise across processes, methods, tools, and IT, paired with practical experience both in implementation and in day-to-day applications.

in-tech is driving your transformation end-to-end with unique MBSE expertise, a practical engineering mindset, and integrated Systems, Software, and IT skills, enabling sustainable change with lasting impact.

We support you:

Tailored Processes & Methods

- Customer-specific adaptation ensures effective and efficient MBSE implementation.
- Expertise in integrating Product Line Engineering and variant management optimizes your product portfolio.
- Incorporating Safety and Security directly into MBSE practices guarantees compliance and minimizes risk.

Focused Training & Rollout Support

- Specialized training and coaching to foster seamless adoption and maximize acceptance.
- Comprehensive rollout support for smooth and efficient transitions.
- Proactive change management and continuous monitoring to ensure long-term success.

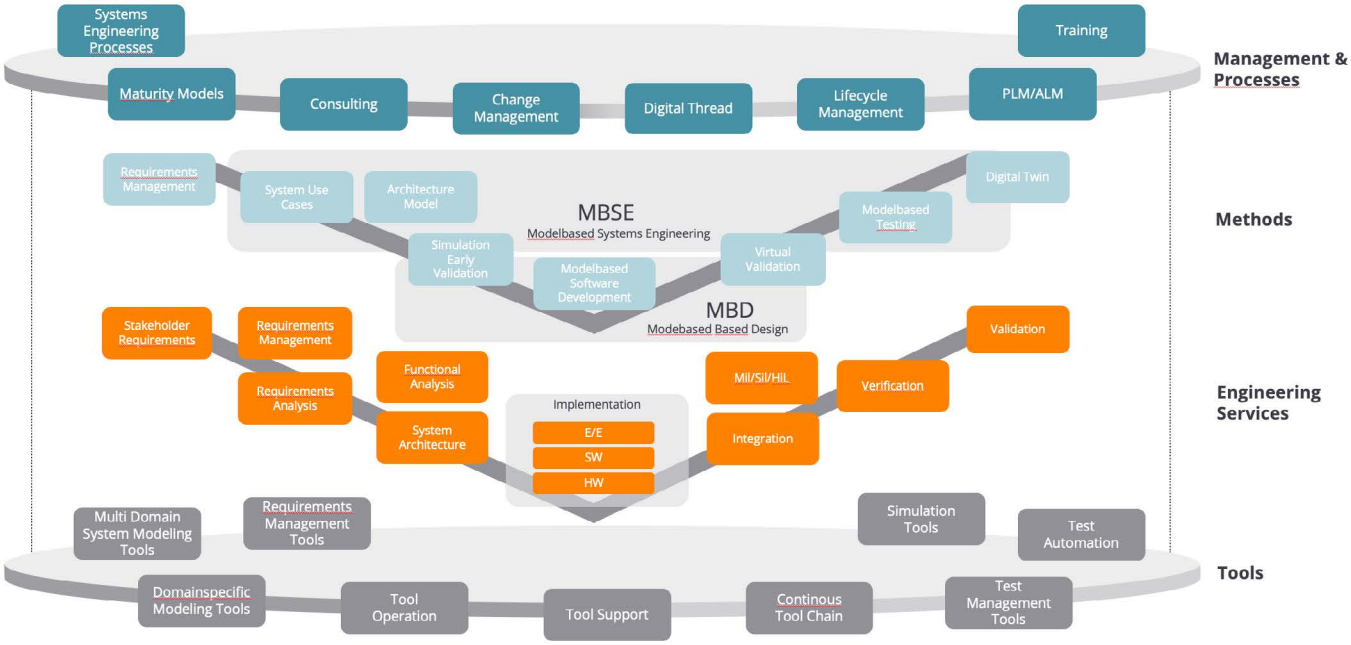
Customized Tool Integration

- Seamless integration into your existing toolchains guarantees compatibility.
- In-house software, UX/UI expertise, and AI-driven automation enable extensive tool customization.
- Data migration expertise ensures continuity and integrity.
- Long-standing partnerships with top tool manufacturers like Dassault, Siemens, and IBM deliver best-in-class solutions tailored specifically to your needs.

Comprehensive Engineering Services

- Hands-on modeling and rapid prototyping during initial phases demonstrate immediate MBSE value.

- Deep methodological expertise across requirements management, architecture and design, safety and security, testing and validation.



in-tech doesn't just understand MBSE theoretically – we implement and apply it practically, ensuring real-world applicability, efficiency, and competitive advantage. Our cross-industry expertise and end-to-end engineering capability make us uniquely positioned to guide your successful MBSE transformation.

Partner with in-tech and turn complexity into clarity, efficiency, and lasting competitive strength.

Contact

If you have any questions, please do not hesitate to contact us. Please contact us via **smart-industry@in-tech.com**

in-tech contact:

André Brückmann
Technical Director Systems Engineering
Email: andre.brueckmann@in-tech.com