Event Detection in C-ITS: Classification, Use Cases, and Reference Implementation

Lennart Reiher[*], Bastian Lampe[†], Lukas Zanger[‡], Timo Woopen[§], and Lutz Eckstein[¶]

Abstract: The transition from traditional hardware-centric vehicles to software-defined vehicles is largely driven by a switch to modern architectural patterns of software, including service orientation and microservices. Automated driving systems (ADS), and even more so, Cooperative Intelligent Transport Systems (C-ITS), come with requirements for scalability, modularity, and adaptability that cannot be met with conventional software architectures. The complexity and dynamics of future mobility systems also suggest to employ ideas of the event-driven architecture paradigm: distributed systems need to be able to detect and respond to events in real-time and in an asynchronous manner. In this paper, we therefore stress the importance of data-driven event detection in the context of ADS and C-ITS. First, we propose a classification scheme for event-detection use cases. We then describe a diverse set of possible use cases and apply the classification scheme to a selection of concrete, innovative examples. Last, we present a modular event detection software framework that we publish as open-source software to foster further research and development of complex C-ITS use cases, but also for robotics in general. The framework is published at `github.com/ika-rwth-aachen/event_detector`.

Keywords: Cooperative Intelligent Transport Systems, Event Detection, Software Architecture, Event-Driven Architecture, Use Cases

1 Introduction

The advancement of automated driving technology goes hand-in-hand with the trend towards software-defined vehicles. Software for automated driving, however, does not stop at the vehicle, but quickly encompasses diverse entities in a Cooperative Intelligent Transport System (C-ITS) to power the automated and connected mobility of the future. The shift towards increasingly complex and interconnected software also requires a shift in the design of software architectures for these systems. From individual automated driving systems to C-ITS, the underlying software architectures need to be scalable, modular, and adaptable to the dynamic and complex interactions that define the future mobility system. Conventional automotive E/E architectures are function-oriented and lack

[*]Lennart Reiher is doctoral researcher and Group Leader *Connectivity* at the Institute for Automotive Engineering (ika) at RWTH Aachen University (e-mail: lennart.reiher@ika.rwth-aachen.de).

[†]Bastian Lampe is doctoral researcher and Group Leader *Automation* at ika.

[‡]Lukas Zanger is doctoral researcher at ika.

[§]Timo Woopen is doctoral researcher and Manager of the Research Area *Vehicle Intelligence & Automated Driving* at ika.

[¶]Lutz Eckstein is Professor and Director of ika.

flexibility with respect to adaptation and extension [1]. Instead, modern architectural patterns such as service orientation and microservices propose to only loosely couple isolated components [1, 2].

The complex and dynamic interactions in future traffic also indicate an inherently event-driven nature: if a vehicle is involved in a crash, it automatically records relevant event data leading up to the incident [3]; if an automated vehicle detects emergency vehicle sirens, it automatically reconfigures its trajectory planning to make way; if two connected automated vehicles are approaching an intersection equipped with intelligent roadside infrastructure, the infrastructure automatically shares a fused collective environment model with the vehicles [4].

These are just some examples that motivate the combination of service-oriented and microservice architectures with the event-driven architecture (EDA) paradigm [5]. The occurrence of specific events (e.g., vehicle crashes) is detectable in data (e.g., crash sensors). The detection of an event then triggers a consequent action (e.g., record event data). EDAs enable such real-time event detections and responses, allowing distributed systems, in particular, to take adequate asynchronous actions without the need for continual polling or extensive state machines.

Such capabilities open up a wide range of new possibilities for improving the safety, efficiency, and user satisfaction of traffic. A large part of the *dynamic-ness* of future C-ITS will be reflected in the specific patterns in the data (or events), e.g., a number of vehicles arriving at a particular intersection equipped with intelligent roadside infrastructure. The complexity of such systems in general, but also the complexity of the events that should be detectable in the data, therefore motivate to not treat use cases as the aforementioned ones individually and in an isolated manner, but to think of them with a generic event detection perspective in mind.

This work argues for the importance of data-driven event detection in the context of ADS and C-ITS through the following contributions: (a) We propose an overarching classification scheme for event detection use cases in the context of ADS and C-ITS; (b) we describe a diverse set of possible use cases and present concrete, innovative examples systematically characterized by our classification scheme; and (c) we present a modular event detection software framework and corresponding open-source reference implementation based on the Robot Operating System 2 (ROS 2) [6] to foster further research and development.

2 Background

Service-oriented and microservice architectures for automotive applications have already been proposed and partially implemented, both in academia and industry [1, 7, 8, 9, 10, 11]. So far, only few works of literature are concerned with event-driven architectures in the context of automated driving [12, 13, 14, 15].

The notion of event-driven software architectures revolves around the detection of and response to so-called *events*. An *event* is typically defined as a *significant change in state*. There are two types of components in EDAs: event emitters detect an event's occurrence and emit an event notification (commonly also referred to as *event*); event consumers receive these notifications and act in response. Notably, event emitters and consumers are only loosely coupled: emitters do not need to know what consumers there are and

vice-versa. EDAs facilitate asynchronous, scalable, and real-time processing of events and responses in distributed systems. [5]. The service-oriented and event-driven architectural patterns can complement each other by triggering specific services in response to specific events [16].

In the context of automated driving, *Object and Event Detection and Response (OEDR)* is considered one fundamental component of the *Dynamic Driving Task (DDT)* as defined by the SAE for the definition of its levels of driving automation [17]. ISO 21448 and ISO 34501 define an *event* as an *occurrence at a point in time* [18] or, equivalently, a *relevant state change of an entity within a scenario* [19], e.g., a traffic light turning green at a given time. For the purpose of crash reconstruction, EU legislation requires the installation of event data recorders in all new passenger cars and light commercial vehicles from July 2024 and in all buses, coaches and heavy-duty vehicles from January 2029 [3].

Building on top of these definitions, in the context of this paper, we define an *event* as a *developer-defined change in state of a C-ITS (including individual entities and the environment) that is associated with the occurence of certain patterns in the data exchanged in the C-ITS. Event detection* then is defined as the *process of analyzing said data in order to identify the patterns signaling the event.* An *action* is defined as the *developer-defined direct consequence triggered by the detection of an event.*

With respect to the use cases that are enabled by future C-ITS, several international standards institutes, the cellular community, and industry- and academia-spanning consortia have already proposed a multitude of concrete use cases [20, 21, 22].

3 Event Detection Use Cases

In light of the many C-ITS use cases already proposed in literature (cf. Section 2), our goal in developing a classification methodology is to take a step back and create a detailed structure for C-ITS event detection use cases. The data-driven event detection view aims to provide a common ground for deriving, conceptualizing, structuring, and implementing these use cases in future event-driven architectures.

3.1 Classification Scheme

To develop a classification scheme for event detection use cases, we first identify a set of classifying dimensions. We strive for these dimensions to be as mutually exclusive and collectively exhaustive as possible, without making the scheme overly complex. Next, for each classifying dimension, we identify a set of distinct characteristic values that can be used to classify use cases. The *incidence* of a use case, e.g., may be characterized as *frequent*, *common*, *rare*, or *unexpected*. The classification of a use case along the proposed scheme helps to conceptualize and implement use cases, or to derive innovative new ones.

For the classification scheme, we identify 13 diverse dimensions and 39 characteristic values along these dimensions. An overview of the classification dimensions and characteristic values is given in Fig. 1. A full definition of the entire classification scheme, including a definition of all classifying dimensions and characteristic values, is given in Table 1. Note that not all combinations of characteristic values along different dimensions are meaningful. Also note that some use cases may fall onto multiple characteristic values along a single dimension.

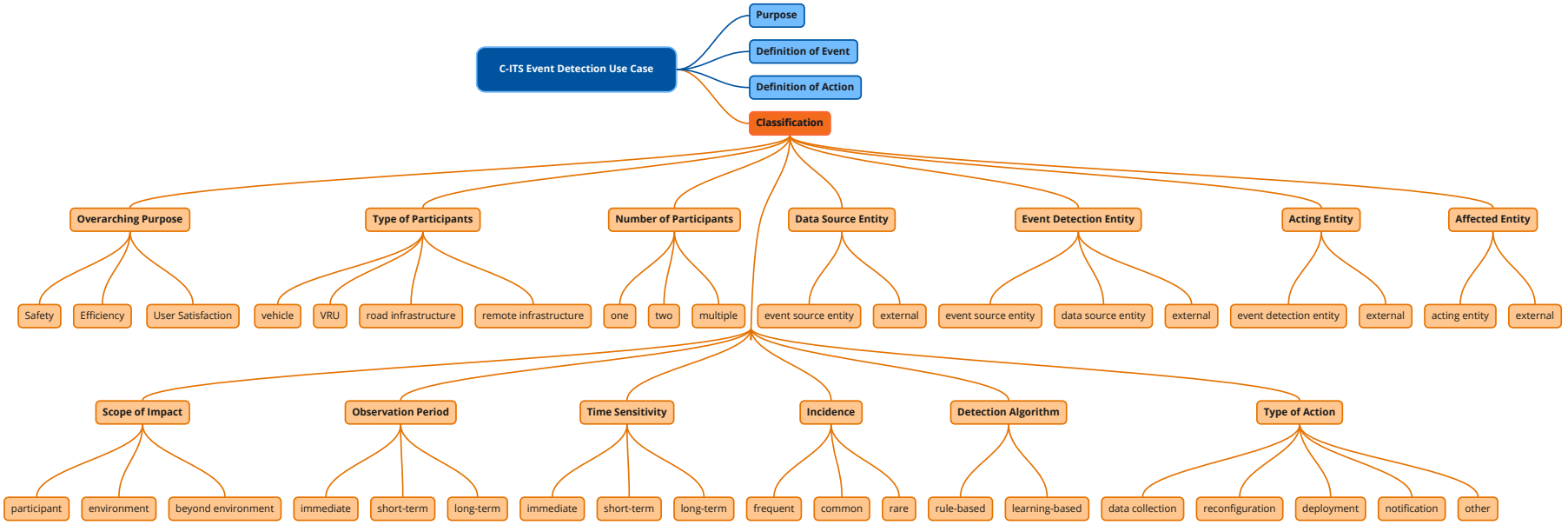

Figure 1: Classifying dimensions and characteristic values for event detection use cases (best viewed digitally)

Dimension	Definition	Characteristic Value	Definition of Characteristic Value
Overarching Purpose	high-level purpose/goal of the use case	safety	improve safety of road transport
		efficiency	improve efficiency
		user satisfaction	improve user satisfaction
Type of Participants	type of participants; participants are entities directly involved in the data processing chain of the use case	vehicle	cars and other motorized vehicles
		VRU	vulnerable road users such as pedestrians, bicyclists, or motorcyclists
		road infrastructure	road/roadside infrastructure such as traffic signs or intelligent roadside ITS stations
		remote infrastructure	remote infrastructure such as control centers or cloud systems
Number of Participants	number of participants	one	single participant
		two	two participants
		multiple	more than two participants
Data Source Entity	entity that initially produces data in which events are detected	event source entity	data is initially produced at the same entity whose state changes give rise to the detected event
		external	data is initially produced at an entity other than the event source
Event Detection Entity	entity that detects events based on identifying patterns in the available data	event source entity	events are detected at event source entity
		data source entity	events are detected at data source entity
		external	events are detected at an entity other than the event source or data source entity
Acting Entity	entity that executes actions directly resulting from event detection	event detection entity	actions are executed at event detection ent.
		external	actions are executed at an entity other than the event detection entity
Affected Entity	entity that is directly affected by consequences of the action	acting entity	direct consequences on the acting entity
		external	direct consequences on an entity other than the acting entity
Scope of Impact	scope of the consequences resulting from the action	participant	consequences on participants
		environment	consequences on participants and their environment
		beyond environment	consequences on entities beyond participants and their environment, e.g., a fleet of vehicles
Observation Period	duration over which data is analyzed to detect an event	immediate	event immediately follows the occurrence of a specific data sample
		short-term	data over a couple of seconds up to minutes is required to detect an event
		long-term	an event is detected based on the analysis of hours of data or even longer
Time Sensitivity	how time-sensitive the action is	immediate	the action leads to immediate consequences
		short-term	the action leads to consequences over the next couple of seconds up to minutes
		long-term	the action leads to consequences after hours or even longer
Incidence	how common events and resulting actions are	frequent	events and actions occur frequently, i.e., they are part of standard operation
		common	events and actions are not frequent, but commonly expected
		rare	events and actions are rare, i.e., the occurrence is special
		unexpected	events and actions are expected to never happen, but deemed possible in theory
Detection Algorithm	type of the algorithm for detecting events	rule-based	rules and heuristics
		learning-based	data-driven and learning-based
Type of Action	type of the action	data collection	data is recorded
		reconfiguration	existing software component is reconfigured
		deployment	new software components are launched
		notification	a notification is sent
		other	any other type of action

Table 1: Classifying dimensions and characteristic values for event detection use cases

3.2 Presentation of Use Cases

We now present a selection of innovative use cases from the field of ADS and C-ITS. We view an event detection use case as a set of the following: (a) a description of its concrete purpose, (b) a definition of the concrete event, (c) a definition of the concrete action, and (d) a classification of the use case along the set of classifying dimensions.

Keeping the extensive classification scheme in mind, concrete event detection use cases are practically endless. We present a selection of diverse example use cases, each serving as a representative instance of one of the higher-level use case clusters. Our goal is to provide inspiration for the development of innovative new use cases that, ideally, can be researched and prototyped using our proposed event detection framework.

In the following, we present one concrete example use case for six diverse use case cluster.

3.2.1 Preventive, Predictive, and Condition-Based Maintenance

This cluster describes use cases that aim at identifying maintenance needs [23] of hardware components before they fail. Preventive, predictive, and condition-cased maintenance helps to prevent failures that could cause harm, to reduce downtime, and to increase the lifetime of hardware components. Use cases rely on monitoring of component state and behavior, and on the gathering and analysis of corresponding historical data.

Example: **Threshold-based condition monitoring of battery health**
Purpose: Enable fleet operators to conduct adequate maintenance measures before a battery failure occurs.
Event: Battery-internal diagnostics report that the battery's state of health is below a certain threshold.
Action: Automatically generate a report with relevant data and notify the fleet operator.
Classification: See Table 2.

Dimension	Char. Value	Dimension	Char. Value
Overarching Purpose	safety, efficiency	Scope of Impact	participant
Type of Participants	vehicle	Observation Period	short-term
Number of Participants	one	Time Sensitivity	long-term
Data Source Entity	event source entity	Incidence	rare
Event Detection Entity	event source entity	Detection Algorithm	rule-based
Acting Entity	external	Type of Action	notification
Affected Entity	external		

Table 2: Classification of use case *Threshold-based condition monitoring of battery health*

Another concrete example use case in this cluster is: *Predictive maintenance of a vehicle's braking system.*

3.2.2 Orchestration of Safety Measures

This cluster describes use cases that aim at detecting hardware or software faults and failures (cf. ISO 26262 [24]) as well as triggering conditions for functional insufficiencies and corresponding hazardous behavior (cf. ISO 21448 [25]). Upon detection, appropriate

safety measures are triggered to orchestrate system reconfigurations that prevent harm, potentially maintaining system functionality or transitioning to a safe state.

***Example:* Reconfiguration of perception system on camera fault**
Purpose: Prevent potential harm caused by invalid data produced by a camera sensor affected by a hardware fault.
Event: The camera sensor produces invalid data, e.g., all pixel values are zero for a certain period of time.
Action: The system orchestrator reconfigures the system architecture by removing the camera from the data processing chain. The orchestrator may determine additional necessary actions, e.g., to reduce the maximum speed of the vehicle or to transition the vehicle to a safe state.
Classification: See Table 3.

Dimension	Char. Value	Dimension	Char. Value
Overarching Purpose	safety	Scope of Impact	participant
Type of Participants	vehicle	Observation Period	short-term
Number of Participants	one	Time Sensitivity	immediate
Data Source Entity	event source entity	Incidence	rare
Event Detection Entity	event source entity	Detection Algorithm	rule-based
Acting Entity	event detection entity	Type of Action	reconfiguration
Affected Entity	acting entity		

Table 3: Classification of use case *Reconfiguration of perception system on camera fault*

Another concrete example use case in this cluster is: *Transitioning to safe state in case of uncontrollable functional insufficiency.*

3.2.3 ADAS/AD System Reconfiguration under Normal Operation

This cluster describes use cases that aim at an automated reconfiguration of advanced driver assistance and/or automated driving systems (ADAS/AD) in automated vehicles in response to specific events. This cluster focuses on event-action pairs that are part of the normal operation of the system, i.e., they are not triggered by faults or failures. Such use cases are largely ego-centric, i.e., the main participant is a single automated vehicle that is analyzing its own state and its environment.

***Example:* Level 3 request to intervene [17]**
Purpose: Issue a request to intervene to the driver of a Level 3 automated vehicle if the vehicle is about to leave the operational design domain (ODD) of the automation system.
Event: A Level 3 automated vehicle detects is about to leave the ODD of the automation system, e.g., due to taking a highway exit coming up in the next kilometers.
Action: Issue a request to intervene to the driver of the vehicle.
Classification: See Table 4.

Another concrete example use case in this cluster is: *Situation- and weather-aware perception.*

Dimension	Char. Value	Dimension	Char. Value
Overarching Purpose	safety	Scope of Impact	participant
Type of Participants	vehicle	Observation Period	immediate
Number of Participants	one	Time Sensitivity	short-term
Data Source Entity	event source entity	Incidence	frequent
Event Detection Entity	data source entity	Detection Algorithm	rule-based
Acting Entity	event detection entity	Type of Action	notification
Affected Entity	acting entity		

Table 4: Classification of use case Level 3 request to intervene

3.2.4 C-ITS Reconfiguration

This cluster describes use cases that aim at an automated reconfiguration of a C-ITS which includes numerous vehicles, infrastructure, and other connected entities that react to specific events.

Example: **Emergency vehicle prioritization**
Purpose: Minimize travel time for emergency vehicles in emergency operations.
Event: A C-ITS control center is informed about a new emergency operation, including the route of an emergency vehicle.
Action: Vehicles' navigation systems are informed by the control center to avoid specific roads.
Classification: See Table 5.

Dimension	Char. Value	Dimension	Char. Value
Overarching Purpose	safety, efficiency	Scope of Impact	participant, environment
Type of Participants	vehicle, remote infrastructure	Observation Period	short-term
Number of Participants	multiple	Time Sensitivity	short-term
Data Source Entity	event source entity	Incidence	common
Event Detection Entity	external	Detection Algorithm	rule-based
Acting Entity	event detection entity	Type of Action	reconfiguration
Affected Entity	external		

Table 5: Classification of use case *Emergency vehicle prioritization*

Other concrete example use cases in this cluster are: *Local on-demand sensor fusion through roadside infrastructure* and *Latency-aware edge offloading of ADAS/AD applications.*

3.2.5 Shadow Mode Testing

This cluster describes use cases that aim at testing new applications in a shadow mode, i.e., running an open-loop alternative of an existing software component in parallel to test it under real-world conditions.

Example: **Discrepancy mining for a perception system in shadow mode**
Purpose: Run a perception system in shadow mode to enable an assessment of performance differences between the existing version and a new version.
Event: Relevant discrepancies between the outputs of the existing and shadow mode perception systems occur, e.g., an object is only present in the environment model produced by the shadow mode system.
Action: Record inputs and outputs of both perception systems for offline analysis and evaluation.
Classification: See Table 6.

Dimension	Char. Value	Dimension	Char. Value
Overarching Purpose	safety	Scope of Impact	participant, beyond environment
Type of Participants	vehicle	Observation Period	short-term
Number of Participants	one	Time Sensitivity	short-term
Data Source Entity	event source entity	Incidence	common
Event Detection Entity	data source entity	Detection Algorithm	rule-based, learning-based
Acting Entity	event detection entity	Type of Action	data collection
Affected Entity	acting entity		

Table 6: Classification of use case *Discrepancy mining for a perception system in shadow mode*

3.2.6 Collective Learning

This cluster describes use cases that aim at a long-term optimization of selected performance indicators of data-driven software components by leveraging data from a fleet of connected vehicles, infrastructure, etc. The concept is covered in more detail in [26].

Example: **Multi-perspective label generation for collectively learnt object detection**
Purpose: Generate autolabeled training samples from collectively gathered fleet data to iteratively improve object detection models based on supervised learning.
Event: Two (or more) sensor-equipped vehicles observe the same scene from different perspectives and their object detection models produce conflicting environment models.
Action: Record sensor data and conflicting environment models for offline autolabeling.
Classification: See Table 7.

Another concrete example use case in this cluster is: *Multi-timestep label generation* [26].

4 Event Detection Software Framework

The previous section illustrates how C-ITS use cases can be thought of as event-action pairs. Motivated by the recurrence of this event-driven theme in the presented use cases, we propose a generic, modular, and data-driven event detection framework for C-ITS. A unified event detection framework can be the backbone for a wide range of C-ITS applications, as it allows for the reuse of components and the development of new applications

Dimension	Char. Value	Dimension	Char. Value
Overarching Purpose	safety	Scope of Impact	participant, beyond environment
Type of Participants	vehicle	Observation Period	short-term
Number of Participants	multiple	Time Sensitivity	short-term
Data Source Entity	event source entity	Incidence	common
Event Detection Entity	data source entity	Detection Algorithm	rule-based, learning-based
Acting Entity	event detection entity	Type of Action	data collection
Affected Entity	acting entity		

Table 7: Classification of use case *Multi-perspective label generation for collectively learnt object detection*

by combining existing components. Such a system further integrates well with the idea of automated and connected systems that can access and process any data at any time, facilitating the development of new applications and services.

As part of this work, we open-source a ROS 2-based implementation of our proposed generic event detection framework called `event_detector`[1] to foster research and development in the field of automated and connected driving.

4.1 Architecture of the Event Detector

The main purpose of the event detector is to act upon developer-defined events that are associated with patterns in data. Irrespective of any particular use case, the event detector fulfils three main tasks: (a) it buffers incoming data, (b) it periodically analyzes available data in order to detect events, and (c) it triggers actions in response to detected events.

These three main tasks are handled by three main components of the event detector, respectively: (a) the data buffer, (b) the data analysis for event detection, and (c) an action plugin system for triggering different types of actions. The high-level architecture of the event detector is shown in Fig. 2, complemented by a more detailed description of all components in the following subsections.

A key design element of the event detector is its modularity, positioning it as a generic event detection framework that can be easily extended and adapted to new use cases. This is achieved through the use of a plugin system for the data analysis and action components. Different so-called *action plugins* exist for different types of actions, e.g., recording data upon event detection. New action plugins can easily be integrated with the core event detector framework. Within these action plugins, the data analysis for event detection is implemented in fully-customizable developer-defined so-called *analysis rules*.

The core of the event detector framework is implemented as a high-performance C++ ROS 2 component node. It covers generic data buffering and implements a generic framework for periodic data analysis. Action plugins are dynamically loaded at runtime using ROS 2's `pluginlib` library [27].

The implementation based on the the open and widely used ROS 2 ecosystem makes the event detector a versatile reference for the research on development of complex event-

[1] `github.com/ika-rwth-aachen/event_detector`

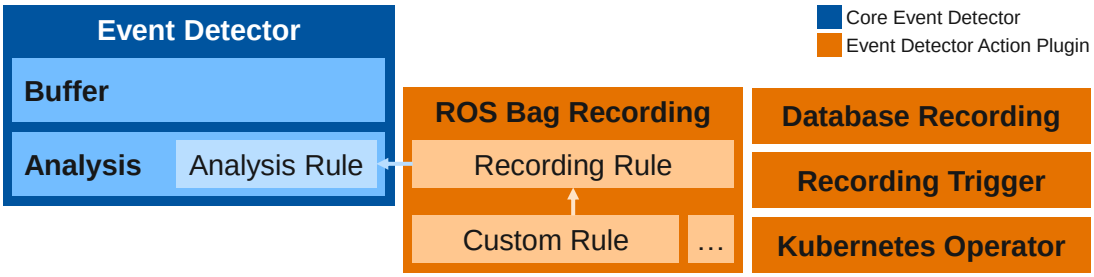

Figure 2: Architecture of the event detector framework: the core event detector manages the data buffer and periodically executes data analysis based on an abstract analysis rule interface; the core event detector is complemented by action plugins; action plugins implement a specific action functionality such as recording to a ROS bag file; specific data analysis for event detection and a specific action consequence in the end are implemented in developer-defined custom analysis rules.

and data-driven C-ITS applications. Outside of the context of automated and connected driving, the event detector framework also enabled similar event-driven use cases in other domains, such as robotics.

4.1.1 Data Buffering

The detection of events in data, in general, is not restricted to a single point of time, but often requires access to a history of data. For this reason, all incoming data is buffered for a configurable amount of time. As a ROS 2-based framework, data is received and buffered in the form of ROS messages. The buffer and the event detector in general are agnostic to specific data types in the sense that the framework can easily be extended to cover arbitrary ROS message types. Relevant data to be buffered is specified on a ROS topic basis with customizable buffer lengths per topic. Buffered topics are assigned to so-called clients, underlining the possible deployment of the event detector in a complex C-ITS environment incorporating data from multiple sources, e.g., vehicles, connected VRUs, and roadside infrastructure.

Given the extent of data being produced in automated driving systems, it is infeasible and undesirable to store all data indefinitely. Therefore, the buffer is realized as a ring buffer covering a configurable duration of the immediate past. New data is continually added to the buffer, overwriting the oldest data once the buffer is full.

The data buffering functionality is completely implemented in the core event detector framework. An excerpt of a possible configuration of the data buffer is part of the example configuration file shown in Listing 1.

4.1.2 Data Analysis for Event Detection

Given its name, a core functionality of the event detector is to detect events identified by patterns in the buffered data. For this purpose, the buffered data is periodically analyzed by the data analysis component. Any particular analysis is use case-specific, naturally, and is developer-defined in the form of so-called *analysis rules*. This customizability is achieved through an abstract base class for analysis rules that developers can extend to implement their specific analysis logic, cf. Fig. 2. The abstract base class provides a

Listing 1: Condensed configuration file of the event detector

```yaml
event_detector:
  ros__parameters:
    # [...]
    buffer:
      default_time: 10.0
    analysis:
      default_period: 5.0
    clients:
      - ego
    client_params:
      ego:
        client_data_list:
          - geometry_msgs__PoseStamped
          - sensor_msgs__Image
        data:
          geometry_msgs__PoseStamped:
            topics:
              - /ego/pose
          sensor_msgs__Image:
            topics:
              - /ego/camera/image
    rules:
      - event_detector_db_recording_plugin::RecordAllRule
    rule_params:
      event_detector_db_recording_plugin::RecordAllRule:
        enabled: true
        geometry_msgs__PoseStamped: true
        sensor_msgs__Image: true
    # [...]

```

common interface for the event detector to interact with the analysis rules and vice-versa. A developer-defined analysis rule can freely request data from the buffer, analyze it, and act upon the detection of a specific event.

The detection of any specific event is closely tied to the action that should be triggered upon its detection. Together, the event-action pair describes a use case to fulfil a specific purpose, cf. Section 3. For this reason, any developer-defined custom analysis rule is associated with a specific action plugin, as described in Section 4.2.

A single instance of the event detector can be configured to run multiple analysis rules in parallel.

An excerpt of a possible configuration of the data analysis is part of the example configuration file shown in Listing 1.

4.2 Action Plugins

As described above, the core event detector is complemented by a set of action plugins. Action plugins implement the specific actions that should be triggered upon the detection of a specific event. As part of the event detector framework, we publish an initial set of action plugins that cover common use cases in the context of C-ITS. Nevertheless,

we highlight that the event detector is designed to be easily extensible with new action plugins or new developer-defined analysis rules in existing action plugins to cover new use cases.

Specific analysis rules are associated with specific action plugins. For most conceivable action plugins, it makes sense to consolidate the core action logic in yet another abstract base class that developers can extend to implement their specific analysis logic and action configuration, cf. Fig. 2.

An excerpt of a possible configuration of an action plugin (for database recording in this case, cf. Section 4.2.2) is part of the example configuration file shown in Listing 1.

In the following, four specific action plugins are presented. Three of them are part of the initial set of action plugins that we publish as part of the event detector framework. Concrete examples of use cases enabled by the event detector and its action plugins are published in a dedicated repository[2].

4.2.1 ROS Bag Recording

The `event_detector_bag_recording`[3] action plugin allows to write data from the buffer to a ROS bag file upon the detection of a specific event. An exemplary use case is the event data recorder, i.e., the recording of relevant data upon the detection of a crash incident. The action plugin already provides two built-in concrete analysis rules: one for recording all data on specified topics, another one for recording data on specified topics only if a specified quantity surpasses a specified threshold.

In principle, this action plugin is similar to the snapshot functionality of `rosbag2` [28]. The event detector, however, offers automated snapshotting based on developer-defined events and has a time-based buffer instead of restricting the number of buffered messages.

4.2.2 Database Recording

The `event_detector_db_recording`[4] action plugin allows to write data from the buffer to a database upon the detection of a specific event. Use cases are similar to the ROS bag recording action plugin (cf. Section 4.2.1), but with the advantage of a more scalable and established storage solution. MongoDB is selected as the database backend, allowing to store ROS messages in their original JSON-like format. The action plugin already provides two built-in concrete analysis rules: one for recording all data on specified topics, another one for recording data on specified topics only if a specified quantity surpasses a specified threshold.

4.2.3 Recording Trigger

The `event_detector_recording_trigger`[5] action plugin allows to trigger the start and stop of a data recording upon the detection of a specific event. It differs from the ROS bag recording and database recording action plugins (cf. Sections 4.2.1, 4.2.2) in that it does not record data itself, but triggers the start and stop of a recording in a separate recording

[2] `github.com/ika-rwth-aachen/event_detector_examples`
[3] `github.com/ika-rwth-aachen/event_detector_bag_recording`
[4] `github.com/ika-rwth-aachen/event_detector_db_recording`
[5] `github.com/ika-rwth-aachen/event_detector_recording_trigger`

system. This separate recording system may, e.g., be a remote ROS bag recording action plugin.

4.2.4 Kubernetes Operator

The `event_detector_operator`[6] action plugin allows to request the deployment or reconfiguration of applications in a Kubernetes cluster upon the detection of a specific event. This action plugin opens up a practically endless range of complex use cases in the context of C-ITS. Kubernetes is well suited as a research platform for the development of complex and highly distributed C-ITS applications, requiring coordination and orchestration of software modules across multiple connected entities. An exemplary use case is the dynamic deployment of an application for the fusion of multiple environment models on an intelligent roadside infrastructure station, when two connected vehicles are approaching an intersection equipped with such infrastructure.

This action plugin is complemented by a so-called application manager component. The application manager is responsible for the actual deployment and reconfiguration of applications in the Kubernetes cluster via the Kubernetes control plane. It receives a task description from an event detector with operator action plugin and translates it into specific Kubernetes workload definitions.

4.3 Related Work

An orchestration framework based on the event detector has already been presented in another work of the authors [10]. The proposed *RobotKube* approach aims at orchestrating containerized microservices for large-scale cooperative multi-robot cyber-physical systems based on Kubernetes. As such, the operator action plugin of the event detector plays a central role in the RobotKube architecture. Multiple event detectors for both operator and recording applications are combined in a reference use case implementation for the use case of collective learning through multi-perspective label generation, cf. Section 3.2.6.

5 Conclusion

The work at hand identifies the importance of event-driven software architectures as an enabler for complex ADS and C-ITS applications. By describing multiple diverse use cases grounded in event detection and subsequent action, we illustrate the omnipresent need for data-driven event detection. The proposed classification scheme for event detection use cases provides a structured framework for thinking about a broad range of use cases from a common perspective. Especially for application in the research and prototyping phase, the common event detection perspective motivates the existence of a flexible software framework for generic event detection use cases in the context of automated and connected driving. The generic and modular `event_detector` software framework proposed in this work fills this gap and, as open source software, also allows other researchers and developers to implement and test their own event-driven use cases.

[6] not yet published

Acknowledgements

This research is accomplished within the research projects **6GEM** (FKZ 16KISK036K), **autotech.agil** (FKZ 1IS22088A), and **UNICAR*agil*** (FKZ 16EMO0284K). We acknowledge the financial support by the Federal Ministry of Education and Research of Germany (BMBF).

References

[1] T. Woopen, et al., *UNICARagil - Disruptive Modular Architectures for Agile, Automated Vehicle Concepts* 27th Aachen Colloquium Automobile and Engine Technology, 2018.

[2] C. Mackenzie, et al., *Reference Model for Service Oriented Architecture 1.0* OASIS, 2006.

[3] European Union, *Regulation (EU) 2019/2144 of the European Parliament and of the Council of 27 November 2019 on type-approval requirements for motor vehicles and their trailers, and systems, components and separate technical units intended for such vehicles, as regards their general safety and the protection of vehicle occupants and vulnerable road users* 2019.

[4] T. Woopen, B. Lampe, R. van Kempen, L. Eckstein, *Architecture of a Collective Memory in UNICARagil* 28th Aachen Colloquium Automobile and Engine Technology, 2019.

[5] M. Chandy, *Event-driven Applications: Costs, Benefits and Design Approaches* California Institute of Technology, 2006.

[6] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, *Robot Operating System 2: Design, Architecture, and Uses in the Wild* Science Robotics, Vol. 7, 2022.

[7] M. Wagner, D. Zoebel, A. Meroth, *An adaptive Software and Systems Architecture for Driver Assistance Systems based on service orientation* International Journal of Machine Learning and Computing, 2011.

[8] S. Furst, M. Bechter, *AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform* IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, 2016.

[9] R. van Kempen, et al., *AUTOtech.agil: Architecture and Technologies for Orchestrating Automotive Agility* 32nd Aachen Colloquium Sustainable Mobility, 2023.

[10] B. Lampe, L. Reiher, L. Zanger, T. Woopen, R. van Kempen, L. Eckstein, *RobotKube: Orchestrating Large-Scale Cooperative Multi-Robot Systems with Kubernetes and ROS* IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), 2023.

[11] *Scalable Open Architecture for Embedded Edge (SOAFEE)* `www.soafee.io`.

[12] J. Dunkel, A. Fernandez, R. Ortiz, S. Ossowski, *Event-Driven Architecture for Decision Support in Traffic Management Systems* IEEE 11th International Conference on Intelligent Transportation Systems (ITSC), 2008.

[13] P. Schneider, *Event Detection and Diagnosis for Intelligent Transport Systems* Doctoral Consortium, Challenge, Industry Track, Tutorials and Posters @ RuleML+RR 2017 hosted by International Joint Conference on Rules and Reasoning, 2017.

[14] E. Eryilmaz, F. Trollmann, S. Albayrak, *An Architecture for Dynamic Context Recognition in an Autonomous Driving Testing Environment* IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), 2018.

[15] S. Lee, S. Lee, J. Noh, J. Kim, H. Jeong, *Special Traffic Event Detection: Framework, Dataset Generation, and Deep Neural Network Perspectives* Sensors, 2023.

[16] P. Krill, *Make way for SOA 2.0* `www.infoworld.com/article/2157676/make-way-for-soa-2-0.html`, 2006.

[17] SAE, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles* J3016_202104, 2021.

[18] International Standards Organization, *ISO 21448:2022 Road vehicles - Safety of the intended functionality* 2022.

[19] International Standards Organization, *ISO 34501:2022 Road vehicles - Test scenarios for automated driving systems - Vocabulary* 2022.

[20] F. Haidar, A. Kaiser, B. Lonc, P. Urien, R. Denis, *C-ITS Use Cases: Study, Extension and Classification Methodology* IEEE 87th Vehicular Technology Conference (VTC Spring), 2018.

[21] CAR2CAR, *Use Cases* 2023.

[22] C-ROADS Platform, *C-ITS Roadmap* 2024.

[23] P. Gackowiec, *General Overview of Maintenance Strategies – Concepts and Approaches* Multidisciplinary Aspects of Production Engineering, Vol. 2, 2019.

[24] ISO - International Organization for Standardization, *ISO 26262:2018 Road vehicles – Functional safety* International Organization for Standardization, 2018. Available: `https://www.iso.org/standard/68383.html`.

[25] ISO - International Organization for Standardization, *ISO 21448:2022 Road vehicles – Safety of the intended functionality* International Organization for Standardization, 2022. Available: `https://www.iso.org/standard/77490.html`.

[26] B. Lampe, L. Reiher, T. Woopen, L. Eckstein, *Cloud Intelligence and Collective Learning for Automated and Connected Driving* ATZelectronis Worldwide, 2023.

[27] *pluginlib* `https://github.com/ros/pluginlib`.

[28] *rosbag2* `https://github.com/ros2/rosbag2`.