An Architecture for Experiments in Networked Control Systems

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

25-02-2022 / 03-03-2022

CITATION

Kloock, Maximilian; Scheffe, Patrick; Greß, Ole; Alrifaae, Bassam (2022): An Architecture for Experiments in Networked Control Systems. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.19236936.v1>

DOI

[10.36227/techrxiv.19236936.v1](https://doi.org/10.36227/techrxiv.19236936.v1)

An Architecture for Experiments in Networked Control Systems

Maximilian Kloock, Patrick Scheffe, Ole Greß, and Bassam Alrifae

Abstract—Rapid prototyping of a Networked Control System (NCS) is challenging because of the physical distribution of agents. Furthermore, experiments with NCS may be subject to external influences which prevent reproducibility. This article presents an architecture for experimental testing of NCS with focus on decision-making. Our architecture for experiments of NCS is strictly modular and hierarchical, and therefore it supports an easy and rapid exchange of each single controller as well as of optimization libraries. Additionally, the architecture synchronizes the whole network of sensors, computation devices, and actuators. Thus, it achieves deterministic and reproducible results, even for time-variant network topologies. Using this architecture, we are able to include active and passive agents and agents with heterogeneous dynamics in the experiments. The architecture also allows handling communication uncertainties, e.g., data packet drop and time delay. The resulting architecture supports performing in-the-loop tests on the model, software, and hardware levels. We demonstrate the architecture in the Cyber-Physical Mobility Lab (CPM Lab) using 20 agents. The architecture can be applied to other domains.

SUPPLEMENTARY MATERIALS

A demonstration video of the CPM Lab, an open-source test-bed using our architecture, is available at <https://youtu.be/PfM0qdzorCc>.

I. INTRODUCTION

A. Motivation

Testing a Networked Control System (NCS) in real-world environments is challenging. Multiple agents perform individual computations and actions to fulfill a common task. The agent's computations may require a non-deterministic amount of time and; hence, the computation times vary in each test run. Furthermore, in tests that use distributed hardware for distributed computations, the time synchronization may contain inaccuracies and the communication time may be stochastic. This non-determinism affects the reproducibility of the experiments.

An architecture for testing NCS has to enable reproducible experiments and, thus, has to achieve determinism of the timing of all the systems' actions. A modular architecture allows for easy and fast updates and rapid prototyping. Furthermore, we require the architecture to support parallel, sequential, and hybrid computations and to handle time-variant network topologies. Our focus is on measuring the computation times of the decision-making algorithms and the decision quality.

B. Related Work

Various research has addressed experimental architectures and testing of NCS on different levels of software and

hardware. A test-bed for development, deployment, testing, and analysis of NCS is introduced in [1], [2]. The test-bed is able to test the fault tolerance and reconfiguration capabilities of the algorithms, as well as test the system stability. Other test-beds, e.g., the test-bed in [3], focuses on model-based systems engineering with the goal to find new patterns and trends, investigate reusability of models and components, and provide a scenario repository. Nevertheless, these test-beds do not support experiments with physical systems. Test-beds that include the physical systems are, e.g., the works in [4], [5], which focus on ambient intelligence applications. The work in [6] presents a test-bed for performing cyber-attacks and evaluating their effects to networked systems. In [7], an architecture for resource aware computing is presented. A client-server architecture with a time-triggered client and event-triggered server is presented in [8]. This architecture focusses on a specific hardware and software setup to hold a steel ball in position using an electromagnet and an optical sensor. An example test-bed which focuses on the communication network to simulate packet losses and communication delays is the test-bed in [9].

However, the test-beds in [1]–[9] have inflexible architectures which are hard to use in other domains. More flexible architectures make use of a middleware. Middlewares introduce an abstraction layer that makes the architecture available to multiple domains, use-cases, or scenarios. An early developed middleware is the CORBA middleware [10]. This middleware was introduced to make software available to a wide range of applications, such as business, facility, or embedded applications. There exist multiple extensions of CORBA, e.g., for mobile systems [11] and real-time systems [12]. Nevertheless, one implicit assumption of real-time CORBA is that communication overheads are tolerable by applications. Since real-time CORBA does not consider communication delays, it is mostly specified for real-time applications that run on a single node. Another middleware that focusses on wireless control networks is ETHERWARE [13]. ETHERWARE makes use of event-based communication over User Datagram Protocol (UDP). The authors used the UDP [14] instead of the Transmission Control Protocol (TCP) [15] to avoid retransmissions, since they can occupy the network with outdated data. This middleware is flexible due to the ability to change parameters at run-time. This flexibility makes it useful in a wide range of applications. The Cyber-Physical Systems Lab in [16], [17] uses ETHERWARE. They present an architecture for testing of cyber-physical systems. The authors demonstrate their architecture using model-scale vehicles. The hardware architecture includes multiple model-scale vehicles on a driving field,

two cameras to sense the positions of the vehicles, and one laptop per vehicle for external computations. The vehicles themselves do not perform any computations, but write data on their actuators and read their sensor data. The sensor data are communicated to the laptops which send the actuator inputs to the vehicles. The laptops compute the actuator inputs for a trajectory which is given by a central trajectory planner. This architecture enables rapid prototyping of the centralized trajectory planning algorithms, due to the possibility to easily and rapidly change software modules at one place without the need to adapt the rest of the architecture. However, the architecture is vulnerable to communication delays and packet losses, since the vehicles require frequent updates of their actuator signals. Moreover, it only supports centralized algorithms. It is not possible to test distributed algorithms without adaptations to the architecture.

The UPBOT [18] test-bed provides an architecture for networked algorithms for cyber-physical systems. The architecture is layered in a *body*, *nerves*, *brain*, and *supervisor* layer. The *body* layer has no intelligence and only reads sensor data and writes data on actuators, similar to the vehicles in the Cyber-Physical Systems Lab [16]. The *body* layer provides the sensor values to the *nerves* layer, which provides the actuator signals to the *body* layer. The *nerves* layer translates decisions from the *brain* layer into commands for the *body* layer. Furthermore, the *nerves* layer formats the sensor data and communicate them to the *brain*. The *brain* layer makes decisions and communicate them to the *nerves* layer. Optionally, a centralized supervisor can be used to externally make centralized decisions. In this case, the *brain* layer receives the decisions from the *supervisor* and forwards them to the *nerves* layer. The authors demonstrate their architecture in a test-bed with three robots. The *body* layer consists of the sensors, actuators, and a microcontroller provided by the robot itself. The *nerves* and *brain* layer share a hardware platform which is placed on the robots. The primary use-case of this architecture is to test security threats and to study points of security attacks to NCS. This architecture is not able to perform distributed computations of decision-making algorithms. More related work is presented in the overview papers in [19]–[21]. They underline the need for architectures and test-beds for rapid prototyping, networked computations, reproducible experiments.

C. Contribution of this Article

To the best of our knowledge, there is no flexible architecture for experimental testing of NCS in the literature that addresses networked computations, reproducible experiments, computation times, communication problems and time-variant network topologies. This article presents an architecture to test NCS and focuses on networked control in a reproducible manner on real-world hardware. Each agent in the NCS uses its own sensors, actuators, and computation devices and share information over a communication network. The agents compute in a synchronized way following a logical execution time approach [22]. This leads to deterministic and reproducible tests. Our architecture supports

sequential, parallel, and hybrid computations, depending on the chosen decision-making algorithm. The architecture adapts the mode of operation to the needs of the decision-making algorithm. Furthermore, the architecture is modular and achieves an experimental environment which is suitable for rapid prototyping of decision-making algorithms. The modular architecture consists of four layers. Our demonstration setup in [23] inspired this work.

The four layers of the architecture are the High-Level Controller (HLC), Middleware (MW), Mid-Level Controller (MLC), and Low-Level Controller (LLC). The architecture is capable of rapid prototyping of decision-making algorithms. Our architecture enables multiple agents are able to make their decisions using sequential, parallel, and hybrid computations. Our MW ensures that the agents synchronously compute their decisions and synchronously apply the decisions to achieve determinism and reproducible tests, even with non-deterministic computation and communication times. The MLC implements a decision-following controller and state estimation. The LLC writes data on the actuators and reads the sensor data. This modularity achieves reusability of the architecture and enables adaptations to specific domains.

For decentral communication, the MW uses a publish-subscribe mechanism [24], which is commonly used in distributed systems, e.g., in the Robot Operating System (ROS) [25], ROS2, the Message Queuing Telemetry Transport (MQTT) [26], or service-oriented architectures [27]. However, ROS requires a designated entity for service discovery or binding. In contrast, ROS2 and our MW uses the Data Distribution Service (DDS), a standardized protocol for publish-subscribe communication. The protocol is widely used in safety-critical systems, e.g., in medical devices and air traffic control [28], and in the upcoming AUTOSAR Adaptive platform [29]. DDS offers a variety of configurable Quality-of-Service (QoS) parameters, e.g. the transport protocol.

In contrast to ROS and MQTT, our MW uses UDP instead of TCP, leading to lower communication latencies. This is because UDP does not require an acknowledgment for each data packet. We do not allow retransmissions, because the data are time-critical and become obsolete when a packet is lost. Furthermore, DDS allows to deploy a variable number of agents in the experiments, without having to adapt the underlying communication topology. Additionally, it is easy to extend, adapt, and change the architecture for experiments due to the dynamic coupling of components in the communication topology.

D. Organization of this Article

The rest of this article is structured as follows. Section II defines important terms which are used in this article. Section III introduces our architecture for experiments, starting with a basic architecture for a single agent and extending it to an architecture for sequential, parallel, and hybrid computations. Section IV presents our evaluation on a demonstration platform. At the end, Section V concludes this article.

II. DEFINITIONS

This section introduces definitions which are used in this article.

Definition 1 (Architecture): According to [30], an architecture divides a system into modules, interactions between modules, and properties of modules and their interactions. We use the term architecture to refer to software and hardware modules, interactions, and properties. We use the terminology *hardware* architecture, if we refer to an architecture that contains only hardware modules and *software* architecture, if the architecture contains only software modules.

An architecture consists of interacting modules. We use the term *agent* for an encapsulation of modules to an autonomous subsystem within the NCS.

Definition 2 (Agent): An agent is a dynamic subsystem of a NCS that interacts with the environment and other agents using sensors, actuators, and communication. We divide the set of agents into active agents and passive agents.

a) *Active Agent:* An active agent implements networked control and reacts to decisions of other agents.

b) *Passive Agent:* Passive agents implement no networked control and do not react to other agent's decisions. Active agents have to consider passive agents.

An example of active agents are networked and autonomous vehicles in mixed autonomous and manual traffic. Passive agents are manual driven vehicles. All vehicles form a NCS in which the networked and autonomous vehicles have to consider the manual driven vehicles to not collide. Nevertheless, the manual driven vehicles may not communicate with the networked and autonomous vehicles.

When multiple agents run their control tasks, the networked control may require communication between the agents. Definition 3 introduces different communication schemes for *networked control*.

Definition 3 (Networked Control): We distinguish centralized and distributed control as part of networked control, according to [31].

Fig. 1. An illustration of networked control. The cogs indicate the entities involved in the decision-making.

a) *Centralized Control:* In centralized control, a central device computes the decisions for all agents. To this end, all agents communicate their states to the central device. Then, the central device communicates the resulting decisions to the corresponding agents.

Fig. 2. An example adjacency matrix A and the corresponding coupling graph G .

b) *Distributed Control:* In distributed control, each agent makes its local decisions and considers its own objectives and other agents. Distributed control implements communication between agents.

Fig. 1 illustrates centralized and distributed control. In distributed control, the communication between agents depends on their coupling. We model the coupling of agents using a *coupling graph*.

Definition 4 (Coupling Graph): A coupling graph is a graph $G = (V, E)$, where the vertices $V = \{v_1, \dots, v_n\}$ represent the agents and the edges $E \subseteq V \times V$ represent the couplings between the agents. An edge is denoted as $e_{ij} = (v_i, v_j)$. A coupling graph is represented by an adjacency matrix A with elements

$$a^{(i,j)} = \begin{cases} 1, & \text{if } e_{ij} \in E \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Fig. 2 shows an example adjacency matrix A and the corresponding coupling graph G . A NCS can be tested using different testing methods. Definition 5 introduces different methods for testing of networked control.

Definition 5 (Networked XiL): X-in-the-Loop (XiL) refers to model, software, processor, and hardware in-the-Loop. There are different definitions for XiL, e.g., in [32]–[35]. We define model, software, and hardware in-the-loop as follows.

- In Model-in-the-Loop (MiL) testing, the controller and plant are modelled and executed on a regular desktop computer in a simulation loop. This testing mode is able to test the functionality and logic of the controller.
- In Software-in-the-Loop (SiL) testing, the controller and plant are executed on a regular desktop computer. In contrast to MiL, the control software is not modeled, but consists of the actual code in SiL. This testing mode is able to test software and implementation related functionality.
- In Processor-in-the-Loop (PiL) testing, the control software is executed on the destination hardware. This allows to test the integration of the controller's hardware and software and to analyze the runtime, e.g., the runtime of the decision-making. The plant is simulated on a regular desktop computer.
- In Hardware-in-the-Loop (HiL) testing, the control software is executed on the destination hardware, as in PiL. The plant is simulated to be able to test real-

Fig. 3. The architecture of each agent. The LLC, MLC, and MW run in real-time. The MW triggers the HLC, which may run on a regular (non-real-time) computer.

time capabilities of the controller. To this end, the plant model and the computer running the plant model need to be real-time capable. The plant may also be simulated by a demonstration platform that is real-time capable and convenient for XiL tests.

XiL testing refers to tests using all these testing methods; networked XiL refers to testing of NCS. An important aspect of testing NCS is determinism. If determinism is ensured, simulations and experiments become reproducible.

III. EXPERIMENTAL ARCHITECTURE

This section introduces our architecture for experimental testing of NCS. Section III-A presents the architecture for a single agent and Section III-B extends this architecture to a networked architecture for multiple agents.

A. Agent Architecture

Our experimental architecture consists of multiple agents which share the same hierarchical architecture and additional elements for interaction. Each agent possesses of a HLC which is connected to a MLC using a MW. A LLC implements the hardware abstraction layer and basic functionalities, e.g., reading sensor data and writing data to the actuators. Fig. 3 shows the agent's architecture. The following subsections introduce the architecture along its hierarchy.

1) *High-Level Controller (HLC)*: The HLC runs high-level computations of the decision-making just when it is triggered by the MW. It receives the data required by the decision-making algorithm from the MW, runs the algorithm

as fast as possible, and sends the computed decisions back to the MW. It then waits for the next trigger and new data to start its computations.

The HLC runs on a development computer running any operating system and does not need to be real-time capable. Therefore, the HLC is independent of the programming language. Our architecture currently supports MATLAB/Simulink and C++. An advantage of the HLC is that the decision-making algorithm and depending libraries can be rapidly replaced or updated. Therefore, the architecture is capable of rapid control prototyping of MiL and SiL.

2) *Middleware (MW)*: The task of the MW is to synchronize the computations of the decisions and to perform the communication between the HLC and the MLC. It triggers the HLC after each MW period time T_{MW} . The MW runs on a real-time computer, which may be a hardware separated from the HLC's hardware, and achieves logical execution time [22] to enable deterministic experiments. The MW achieves reproducible results due to determinism. As described in Section I-C, our MW uses DDS for decentralized communication to achieve a flexible architecture.

3) *Mid-Level Controller (MLC)*: The MLC runs on a real-time capable microcontroller and performs light-weight mid-level computations, e.g., light-weight state estimation. The MLC sends the control inputs to the LLC at a predefined MLC period time T_{MLC} . The MLC also receives the sensor signals from the LLC and performs a state estimation. The estimated states are communicated to the MW at each MLC period.

4) *Low-Level Controller (LLC)*: The LLC provides the hardware abstraction layer. It handles the access to the physical system, i.e., it writes data on the actuators and reads the sensor data in real-time. The LLC receives the control inputs from the MLC and sends sensor data to the MLC.

The LLC handles all hardware-dependent implementations. Therefore, exchanging the physical system requires just modifying the LLC-software and not that of the MLC and HLC. This makes changes of the physical system fast and easy and allows for HiL testing.

5) *Architecture Workflow*: Fig. 4 shows the timing of our layer composition. The MW gathers the estimated states from the MLC and triggers the HLC in a predefined frequency. The HLC then computes decisions in a best-effort strategy. The variable τ_{c_1} denotes the time from reading the sensor data in the LLC to starting the HLC computation. The variable τ_{HLC} denotes the computation time of the HLC. τ_{HLC} may vary for different time steps. After computing the decisions, the HLC communicates the decisions to the MW, which forwards them to the MLC. The MW labels when the MLC should use the new decision and overwrites the old decision. This ensures a logical execution time, which makes sure that even for different and non-deterministic communication and computation times at different time steps in the HLC, the MLC deterministically applies its decision in a predefined frequency. The MLC then sends the computed control inputs to the LLC, which writes data on the actuators and reads the sensor data. τ_{c_2} denotes the time from the made

Fig. 4. The workflow of the agent's architecture.

decision to the actuators. The sensor data are then used for state estimation in the MLC which communicates them to the MW.

B. Architectures for Networked Control Systems

The architecture for testing NCS consisting of multiple agents follows the agent architecture of Subsection III-A. Each agent has the same MW period time T_{MW} , MLC period time T_{MLC} , and LLC period time T_{LLC} . There are multiple possibilities for networked architectures, depending on the communication scheme for networked control. The resulting networked architectures are introduced in the following sections. Section III-B.1 presents the centralized architecture and Section III-B.2 presents distributed architectures for sequential, parallel, and hybrid computations.

1) *Architecture for Centralized Control:* Fig. 5 shows the architecture for testing centralized decision-making algorithms. This architecture consists of one HLC and MW for all agents. Each agent only implements its MLC and LLC. The agents communicate their estimated states from their MLCs to the central MW. The MW aggregates these states and triggers the HLC. After the HLC computations, the MW sends each computed decision to the corresponding agent's MLC. In each agent, the MLC and LLC work as described in Section III-A.

Fig. 6 illustrates the workflow of the centralized architecture for two agents. For each agent, only the MLC is shown. The workflow between the MLC and the corresponding LLC is the same as in Fig. 4. The MLCs communicate their estimated states to the MW. The MW triggers the HLC in the predefined frequency $1/T_{MW}$ by sending the aggregated states. After the HLC computations, the HLC sends the computed decisions for all agents to the MW. The MW then forwards each decision to the corresponding agent labelled with further information when to apply the decision. All MLCs apply their decisions at the same time after the MW period.

2) *Architecture for Distributed Control:* There are different architectures for testing distributed control systems.

Fig. 5. The architecture for centralized computations. All agents communicate to a central MW which triggers a central HLC.

Fig. 6. The workflow of centralized computations for two agents. The communication with the LLCs are omitted, but work as illustrated in Fig. 4.

Compared to the centralized architecture in Section III-B.1, all distributed architectures consist of one HLC, MW, MLC, and LLC per agent. Each LLC communicates with its corresponding MLC. The MLCs send their states to all MWs. Each MW communicates with its corresponding HLC and the HLCs share data between each other dependent on the chosen decision-making algorithm. After the HLC computations, the MLCs receive the decisions of their corresponding HLCs. We now introduce the architectures for sequential, parallel, and hybrid computations.

a) *Architecture for Sequential Computations:* Fig. 7 shows the sequential architecture. In sequential computations, only one HLC computes its decision at each time. To this end, the agents are ordered. Each agent communicates the computed decisions to all successors and receives all decisions of its predecessors in the coupling graph. The communication of decision-making related data is required only between the HLCs.

Fig. 8 illustrates the workflow of the sequential architecture for two agents. The MWs of the two agents trigger the corresponding HLCs at the start of the MW period. Then,

Fig. 7. The architecture for sequential computations. Each agent has its own HLC. The HLCs share their decisions in a sequential order.

Fig. 8. The workflow of sequential computations for two agents. The communication with the LLCs are omitted, but work as illustrated in Fig. 4.

only HLC₂ starts immediately to compute decisions as fast as possible. Afterwards, HLC₂ communicates the decisions to MW₂ and to HLC₁. HLC₁ is triggered by receiving the decisions from HLC₂ and starts its computations and communicates the resulting decision to MW₁. Both MWs label information for the MLCs when the decision has to be applied. After the MW period ends, both MLCs apply their decisions at the same time.

b) Architecture for Parallel Computations: Fig. 9 shows the architecture for parallel computations. It differs from the architecture for sequential computations in the way that the communication between the HLCs is possible in both directions. In the parallel computations, all HLCs

Fig. 9. The parallel architecture for networked computations. Each agent has its own HLC. The HLCs share their decisions with coupled agents.

compute their decisions simultaneously and communicate with one another according to the coupling graph. Depending on the networked control strategy (cooperative or non-cooperative), each HLC may receive the states of multiple MLCs (cooperative), or of only one MLC (non-cooperative). The HLCs communicate their decisions with one another (non-cooperative). The HLCs may also share other algorithm data with one another (cooperative). Fig. 10 illustrates the workflow of the parallel architecture using an example of two agents. The MLCs send their estimated states to the MWs, which trigger the HLCs at the same time. The HLCs compute their decisions in parallel. After the computation of the decisions, the corresponding HLC sends its decision to its MW. The MW labels information when the MLC should apply its decision. The MW forwards the decision to the corresponding MLC. At the beginning of the next MW period, the MLCs apply their decisions and the MWs trigger the next HLC computations.

c) Architecture for Hybrid Computations: The architecture for hybrid computations combines the architectures for sequential and parallel computations. The HLCs are grouped by their computation dependencies. All HLCs in a group compute at the same time, as in the parallel architecture. The groups, nevertheless, compute in a sequential manner, as in the sequential architecture. Fig. 11 illustrates hybrid computations using an example with 5 HLCs which are assigned into 3 groups. The overall computation time is the sum of all maximum computation times of each group.

Fig. 12 shows the workflow of the first two groups of the example in Fig. 11. The MW triggers HLC₂ and HLC₃ to start their computations. Since they are both members of the group 1, they compute in parallel. When HLC₂ and HLC₃ finished their computations, they trigger HLC₁, since it is in the second group. Since HLC₁ is the only member of group 2, no other HLC computes at the same time. The MWs label

Fig. 10. The workflow of parallel computations for two agents. The communication with the LLCs are omitted, but work as illustrated in Fig. 4.

Fig. 11. The hybrid architecture for three agents. Each agent has its own HLC. The HLCs share their decisions between one another in the same group. The communication between multiple groups works in a sequential order.

the information for the MLCs when to apply their decisions. At that time, all MLCs synchronously apply their decisions.

IV. EVALUATION

This section evaluates our architecture using our Cyber-Physical Mobility Lab (CPM Lab) [36]. The next subsections introduce our demonstration setup and a timing analysis of our architecture.

A. Demonstration Setup

Fig. 13 illustrates the components of our CPM Lab as a demonstration platform for the presented architecture for networked and autonomous vehicles. The CPM Lab consists of

Fig. 12. The workflow of hybrid computations for three agents. The communication with the LLCs are omitted, but work as illustrated in Fig. 4.

20 model-scale vehicles (μ Cars) [37], a camera for the indoor positioning system [38], external computation devices (the HLCs), a main computer to control and monitor experiments, a map containing the road network, and a router for wireless communication. The camera of the indoor positioning system takes pictures of the driving area and detects the positions and orientations of the vehicles and communicates them to the computation devices. The computation devices plan trajectories using the computed positions and orientations. The trajectories are then sent to the μ Cars. Each computation device is assigned to one μ Car and represent the HLC and MW for this μ Car. Due to space and weight requirements, the μ Cars implement only the MLC and LLC. However, logically, each μ Car is assigned to an HLC for high-level trajectory planning. Such remote decision-making is commonly used in rapid prototyping approaches, e.g., in experimental tests of algorithms for the mars rover [39], [40]. Each μ Car is equipped with a Raspberry Pi which runs the MLC and an ATmega microcontroller which runs the LLC. Fig. 14 shows the architecture of the μ Car. It makes use of the architecture presented in Section III-A. The HLC plans the trajectory for the vehicle and communicates the trajectory via the MW to the MLC. The MLC implements a trajectory following controller and computes the system inputs for the vehicle. The system inputs, consisting of torque and steering

Fig. 13. A schematic overview of the CPM Lab.

Fig. 14. The architecture of the μ Cars in the CPM Lab, implementing the architecture of Fig. 3.

values, are then applied in the LLC, which writes data on the actuators and reads the sensor data. The measured sensor data are communicated to the MLC. The MLC performs a state estimation and communicates the estimated states to the MW, which triggers the HLC at its next time period. Multiple HLCs can be used for sequential, parallel, or hybrid computations, as described in Section III-B.

B. Demonstration Scenario

In order to demonstrate the architecture in the CPM Lab, the μ Cars drive in an intersection and motor way scenario, see Fig. 15. The map contains a highway, on- and off-ramps and a four way intersection. At the intersection, the HLCs distributively compute the μ Car's trajectories in sequence, as described in Section III-B and [41]. The μ Cars follow the paths with a given reference speed. The μ Cars randomly choose their paths each time they enter the intersection. In case of potential collisions, the μ Cars avoid the collisions by reducing their speed in a priority-based manner inspired by [42]. After avoiding the collisions, the μ Cars accelerate to the reference speed. Our methods presented in [41], [43], [44] are further examples of distributed applications.

Fig. 15. The test track of the CPM Lab with 10 μ Cars driving in the intersection and motor way scenario.

C. Timing Analysis

This section demonstrates a case study with 10 vehicles using the demonstration scenario on the map illustrated in Fig. 15.

Table I shows the time-stamps of two consecutive planning phases recorded in the case study experiment to show the effectiveness of our architecture. Due to space reasons, only the timing for 5 trajectories are presented per time step. The MW period T_{MW} has a length of $T_{MW} = 400\text{ms}$ and the time $t \in \mathbb{R}$ starts at $t = 0$, when all HLCs are triggered. The columns show the following information:

- Column 1 (time step): The time step, beginning at time step 1
- Column 2 (trajectory ID): The trajectory ID, which corresponds to the HLC that plans the trajectory, i.e., HLC_x plans for μCar_x .
- Column 3 (HLC triggered): The time when the MW triggers the HLC
- Column 4 (HLC comp. finished): The time when the HLC finishes the computation of the trajectory
- Column 5 (MW received): The time when the MW receives the trajectory
- Column 6 (valid-after time): The valid-after time, i.e., the MW label when the MLC should apply its decision
- Column 7 (MLC received): The time when the MLC receives the trajectory
- Column 8 (LLC received and applied): The time when the LLC receives the trajectory, i.e., when the trajectory is applied in the vehicle

Each row shows the timings for one step from trajectory planning to application of the control inputs in the vehicles. Each trajectory is shown in a separated row. Rows 1 to 5 show the first planning phase and rows 6 to 10 show the second planning phase.

The computation times of HLC_1 to HLC_4 in planning

TABLE I

TIME-STAMPS (IN ms) OF TWO CONSECUTIVE PLANNING STEPS OF THE CASE STUDY EXPERIMENT. THE TIMINGS ARE RELATED TO FIG. 8.

time step	μ Car ID	HLC triggered	HLC comp. finished	MW received	valid-after time	MLC received	LLC received and applied
1	1	000	014	015	400	021	400
	2	000	018	019	400	024	400
	3	000	016	017	400	036	401
	4	000	015	016	400	021	401
	5	000	041	042	400	048	400
2	1	400	528	528	800	537	800
	2	401	532	533	800	539	800
	3	401	529	530	800	542	802
	4	400	526	527	800	536	800
	5	400	524	525	800	537	801

phase one varies in 4 ms. The maximum computation time in planning phase one is 41 ms required by HLC₅ for its computation, because it has to wait for HLC₃ and has to avoid a collision with μ Car₃. All trajectories have the same valid-after time stamp which is 400 ms. The MWs received all trajectories about 1 ms after the corresponding HLCs finished their computations. The received times of the MLCs show a higher variation than the MW received times, due to the non-deterministic communication times of the HLCs. The MLCs have an on-board cycle time of $T_{MLC} = 20$ ms. At the next MLC cycle after $t = 400$ ms, i.e., when the trajectories are assigned to be applied, the MLCs synchronously apply the new trajectories by forwarding the trajectories to the LLCs. At this time, if the valid-after time would not be used, vehicle 5 would apply the new trajectories at $t = 60$ ms, i.e., one cycle time after the other vehicles apply the trajectories at $t = 40$ ms. With a higher variance in the computation times of the HLCs, the difference of MLC cycles of the time when the trajectories are applied may be higher. As this may lead to unexpected behavior, we mitigate this phenomenon by the common valid-after time. Therefore, all vehicles synchronously apply the new trajectory at the same point in time. This leads to deterministic timing of experiments; hence, the experiments are reproducible. In case the valid-after time already passed the MLC receive time, the MLCs immediately apply the new trajectory. In the second planning phase, the MWs trigger the HLCs at time step 400. The HLC computations show only little variations, since no collisions had to be avoided.

V. CONCLUSION

This article presented an architecture for experimental testing of NCS. We introduced an architecture for a single agent and extended it to a central, sequential, parallel, and hybrid architecture for NCS. The modular architecture consists of four layers. This layered approach enables rapid prototyping and experimental evaluation of NCS for different network control schemes and time-variant network topologies. We synchronize networked computations using a logical execution time approach. The agents synchronously apply their decisions due to our architecture. Due to synchronously computed decisions which are synchronously applied, this architecture achieves deterministic and reproducible experiments, especially when dealing with external influences like

non-deterministic computation times and non-deterministic communication delays. Our evaluation on a demonstration platform which follows the proposed architecture underlines these properties.

REFERENCES

- [1] P. S. Kumar, W. Emfinger, and G. Karsai, "A testbed to simulate and analyze resilient cyber-physical systems," in *2015 International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2015, pp. 97–103.
- [2] P. S. Kumar, "Integrated timing analysis and verification of component-based distributed real-time systems," Ph.D. dissertation, Vanderbilt University, 2016.
- [3] A. M. Madni, M. Sievers, S. Purohit, and C. C. Madni, "Toward a mbse research testbed: Prototype implementation and lessons learned," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 2939–2945.
- [4] O. Gabel, L. Litz, and M. Reif, "NCS testbed for ambient intelligence," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1. IEEE, 2005, pp. 115–120.
- [5] R. Obukata, T. Oda, and L. Barolli, "Design of an ambient intelligence testbed for improving quality of life," in *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 2016, pp. 714–719.
- [6] B. Potteiger, W. Emfinger, H. Neema, X. Koutosukos, C. Tang, and K. Stouffer, "Evaluating the effects of cyber-attacks on cyber physical systems using a hardware-in-the-loop simulation testbed," in *2017 Resilience Week (RWS)*. IEEE, 2017, pp. 177–183.
- [7] R. Gupta and R. Shyamasundar, "Reactive framework for resource aware distributed computing," in *Annual Asian Computing Science Conference*. Springer, 2004, pp. 452–467.
- [8] W.-j. Kim, K. Ji, and A. Ambike, "Real-time operating environment for networked control systems," *IEEE transactions on automation science and engineering*, vol. 3, no. 3, pp. 287–296, 2006.
- [9] A. Mukherjee, A. Pakhira, S. Das, I. Pan, and A. Gupta, "Embedded network test-bed for validating real-time control algorithms to ensure optimal time domain performance," in *2011 International Conference on Process Automation, Control and Computing*. IEEE, 2011, pp. 1–6.
- [10] J. Siegel, "Omg overview: Corba and the oma in enterprise computing," *Communications of the ACM*, vol. 41, no. 10, pp. 37–43, 1998.
- [11] S. Adwankar, "Mobile corba," in *Proceedings 3rd International Symposium on Distributed Objects and Applications*. IEEE, 2001, pp. 52–63.
- [12] D. Schmidt and F. Kuhns, "An overview of the real-time corba specification," *Computer*, vol. 33, no. 6, pp. 56–63, 2000.
- [13] G. Baliga, S. Graham, L. Sha, and P. Kumar, "Etherware: Domainware for wireless control networks," in *Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2004. Proceedings*. IEEE, 2004, pp. 155–162.
- [14] "User Datagram Protocol," RFC 768, Aug. 1980. [Online]. Available: <https://rfc-editor.org/rfc/rfc768.txt>
- [15] "Transmission Control Protocol," RFC 793, Sep. 1981. [Online]. Available: <https://rfc-editor.org/rfc/rfc793.txt>
- [16] S. Graham, G. Baliga, and P. Kumar, "Abstractions, architecture, mechanisms, and a middleware for networked control," *IEEE Transactions on Automatic Control*, vol. 54, no. 7, pp. 1490–1503, 2009.

- [17] W. H. Ko, "Middleware and architecture for advanced applications of cyber-physical systems," Ph.D. dissertation, 2017.
- [18] T. L. Crenshaw and S. Beyer, "Upbot: a testbed for cyber-physical systems," in *Proceedings of the 3rd international conference on Cyber security experimentation and test*, 2010, pp. 1–8.
- [19] X. Zhou, X. Gou, T. Huang, and S. Yang, "Review on testing of cyber physical systems: Methods and testbeds," *IEEE Access*, vol. 6, pp. 52 179–52 194, 2018.
- [20] O. Salunkhe, M. Gopalakrishnan, A. Skoogh, and Å. Fasth-Berglund, "Cyber-physical production testbed: literature review and concept development," *Procedia manufacturing*, vol. 25, pp. 2–9, 2018.
- [21] B. Vaagensmith, V. Kumar Singh, R. Ivans, D. L. Marino, C. S. Wickramasinghe, J. Lehmer, T. Phillips, C. Rieger, and M. Manic, "Review of design elements within power infrastructure cyber-physical test beds as threat analysis environments," *Energies*, vol. 14, no. 5, p. 1409, 2021.
- [22] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *International Workshop on Embedded Software*. Springer, 2001, pp. 166–184.
- [23] B. Alrifaae, "Networked model predictive control for vehicle collision avoidance: Vernetzte modellbasierte prädiktive regelung zur kollisionsvermeidung von fahrzeugen," Ph.D. dissertation, RWTH Aachen University, 2017.
- [24] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, pp. 200–206.
- [25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [26] I. B. M. C. (IBM) and Eurotech, *MQTT V3.1 Protocol Specification*, 2010.
- [27] A. Kampmann, B. Alrifaae, M. Kohout, A. Wüstenberg, T. Woopen, M. Nolte, L. Eckstein, and S. Kowalewski, "A dynamic service-oriented software architecture for highly automated vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2101–2108.
- [28] Object Management Group, "Who's Using DDS?" <https://www.omgwiki.org/dds/who-is-using-dds-2/>, 2019, [Online].
- [29] S. Fürst and M. Bechter, "Autosar for connected and autonomous vehicles: The autosar adaptive platform," in *2016 46th annual IEEE/IFIP international conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 215–217.
- [30] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: views and beyond," in *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 740–741.
- [31] J. Lunze, *Control theory of digitally networked dynamic systems*. Springer, 2014, vol. 1.
- [32] D. Abel and A. Bollig, *Rapid control prototyping*. Springer, 2006.
- [33] M. C. Reiter, B. Rumpe, and D. Abel, "Environment and tools for the development of navigation-and map-based driver assistance and vehicle guidance systems," *Lehrstuhl und Institut für Regelungstechnik*, Tech. Rep., 2017.
- [34] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in *2008 1st international conference on software testing, verification, and validation.* IEEE, 2008, pp. 485–493.
- [35] J. Schäuffele and T. Zurawka, *Automotive software engineering*. Springer, 2010.
- [36] M. Kloock, P. Scheffe, J. Maczjewski, A. Kampmann, A. Mokhtarian, S. Kowalewski, and B. Alrifaae, "Cyber-physical mobility lab: An open-source platform for networked and autonomous vehicles," in *2021 European Control Conference (ECC)*. IEEE, 2021, pp. 1937–1944.
- [37] P. Scheffe, J. Maczjewski, M. Kloock, A. Kampmann, A. Derks, S. Kowalewski, and B. Alrifaae, "Networked and autonomous model-scale vehicles for experiments in research and education," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 17 332–17 337, 2020.
- [38] M. Kloock, P. Scheffe, I. Tülleners, J. Maczjewski, S. Kowalewski, and B. Alrifaae, "Vision-based real-time indoor positioning system for multiple vehicles," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 446–15 453, 2020.
- [39] R. Castano, T. Estlin, D. Gaines, A. Castano, C. Chouinard, B. Bornstein, R. C. Anderson, S. Chien, A. Fukunaga, and M. Judd, "Oppor-
- tunistic rover science: finding and reacting to rocks, clouds and dust devils," in *2006 IEEE Aerospace Conference.* IEEE, 2006, pp. 16–pp.
- [40] T. Estlin, D. Gaines, C. Chouinard, R. Castano, B. Bornstein, M. Judd, I. Nesnas, and R. Anderson, "Increased mars rover autonomy using ai planning, scheduling and execution," in *Proceedings 2007 IEEE International Conference on Robotics and Automation.* IEEE, 2007, pp. 4911–4918.
- [41] M. Kloock, P. Scheffe, S. Marquardt, J. Maczjewski, B. Alrifaae, and S. Kowalewski, "Distributed model predictive intersection control of multiple vehicles," in *2019 IEEE intelligent transportation systems conference (ITSC)*. IEEE, 2019, pp. 1735–1740.
- [42] B. Alrifaae, F.-J. Heßeler, and D. Abel, "Coordinated non-cooperative distributed model predictive control for decoupled systems using graphs," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 216–221, 2016.
- [43] M. Kloock, L. Kragl, J. Maczjewski, B. Alrifaae, and S. Kowalewski, "Distributed model predictive pose control of multiple nonholonomic vehicles," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 1620–1625.
- [44] M. Kloock, P. Scheffe, L. Botz, J. Maczjewski, B. Alrifaae, and S. Kowalewski, "Networked model predictive vehicle race control," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 1552–1557.