





Service-Oriented Model-based Control Dynamic Software for Dynamic Systems

Ole Greß¹ , Markus Zimmer² , Dominik Scheurenberg², *Student Members, IEEE*,
Lorenz Dörschel² , *Member, IEEE*, Bassam Alrifaae³ , *Senior Member, IEEE*

Abstract— Control loops trend towards cyber-physical systems and come with corresponding challenges that complex IT systems from other domains share, such as increased effort to integrate components into a system. Modern service-oriented architectures can help address these challenges by decoupling component development from system integration. This paper presents a concept to model control loop elements as services that are flexibly integrated at runtime using a central entity called orchestrator. Our concept is suitable for a general control system, is capable of running on embedded and general-purpose computers, and can ensure deterministic computations. We apply our approach to an experimental setup of a three-tank system and implement multiple control loops using a set of services and an orchestrator. We show that using our architecture, the service composition is easy to change dynamically at runtime and thus realize a retrofit of the process control.

I. INTRODUCTION

In times of digitalization, control loops are rarely implemented in analog components but almost exclusively consist of digital control elements, and thus software. These cyber-physical control loops trend toward ever-increasing complexity and shortened lifecycles. Keeping *cyber-physical systems* (CPS) up-to-date and maintainable is important to keep up with short lifecycles, but the associated integration effort is becoming more expensive with increased complexity [1]. Traditional control system design is based on cause-effect relationships between the measurement, signal processing, control, and actuation steps, which are often modeled as a functional block diagram. One way to realize this is to use a model-based software development process to implement function blocks in code, define interfaces and interactions between them, and finally assign them to *electronic control units* (ECUs) [2]. While these function blocks can be reused in other systems, their interactions within a system, and thus the flow of information, are fixed after the system's deployment.

In control literature, there are various approaches for creating runtime-flexible control loops. Switching adaptive

control can detect changes in a control loop and switch between a set of controllers [3], [4]. For over-actuated systems, control allocation can be used to compensate for faulty actuators by reallocating the available control values [5]. However, these approaches fix all components and their interactions at design time, and flexibility is only possible between a pre-determined set of options.

The lack of flexibility and maintainability have long been known in software and have resulted in software architectures for distributed systems that can keep flexibility using exchangeable software modules [6], [7]. One of these, *service-oriented architecture* (SOA), is based on reusable services that are combined in a flexible system integration to achieve a complex goal. Dynamic integration of these services, called *service composition*, has received extensive research in the fields of web services and cloud computing [8], [9]. However, a SOA for CPS additionally needs to be real-time capable, ensure deterministic computations, and be compatible with low-cost microcontrollers.

Kugele et al. present a concept for a SOA for an automotive context that provides a failover mechanism that changes the flow of information at runtime; however with a focus on general-purpose computers and cloud-based usage [10]. *Robot Operating System* (ROS2) is known as a widely used service-oriented middleware in the field of robotics and has recently been adapted for use on microcontrollers [11]. However, ROS2 neither provides a mechanism to control the flow of information between nodes, nor a mechanism to coordinate the timing of nodes, hindering computational determinism. Stoll et al. address the former and present a SOA based on ROS2 that adapts the control flow to dynamically switch services at runtime in case of failures or newly added components, but without addressing possible timing issues [12]. Research at RWTH Aachen University resulted in the microcontroller compatible *Automotive Service-Oriented Architecture* (ASOA) that uses an *orchestrator* to centrally manage the service composition and coordinate the timing of the services [13]. While ASOA dynamically chooses service compositions between three pre-defined operating modes, it cannot determine the most suitable operating mode on its own or select suitable service compositions at runtime. Additionally, we introduced a SOA specifically tailored to conducting deterministic and reproducible experiments in the domain of connected and automated vehicles in [14].

Our proposed SOA is based on ASOA but specialized in control systems. Using domain knowledge on control loops, we can monitor a control system at runtime and reason which

This research is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 468483200.

¹Ole Greß is with the Chair of Embedded Software, RWTH Aachen University, 52074 Aachen, Germany (e-mail: gress@embedded.rwth-aachen.de).

²Markus Zimmer, Dominik Scheurenberg and Lorenz Dörschel are with the Institute of Automatic Control, RWTH Aachen University, 52074 Aachen, Germany (e-mail: {M.Zimmer, D.Scheurenberg, L.Doerschel}@irt.rwth-aachen.de).

³Bassam Alrifaae is with the Department of Aerospace Engineering, University of the Bundeswehr Munich, 85579 Neubiberg, Germany, (e-mail: bassam.alrifaae@unibw.de).

service composition is the most suitable.

No generalizable control approach based on SOA has been proposed so far. Therefore the novelty of our approach is to model control loop elements as services and dynamically integrate them at runtime using an orchestrator. We use a retrofitting scenario on a three-tank system to validate our approach. Our findings showcase that our architecture enables the reliable exchange of control loop elements at runtime for retrofitting and upgrade purposes.

First, Section II presents our architecture. Section III elaborates on the evaluation scenario and the services implemented for it. Section IV evaluates the results of this scenario. Section V draws a conclusion on our architecture and presents research topics for future works.

II. PROPOSED ARCHITECTURE

The proposed architecture entails a model-based control loop, with elements modeled as services. We assume that the system state is not measurable, necessitating a filter that estimates the system state from the measurements. The filter and the controller utilize separate process models. The control loop interacts with its environment via sensors and actuators and requires a reference to follow. This results in six essential elements for the considered general model-based control loop: the filter, the controller, the process model, the sensor, the actuator, and the process.

These elements must remain independent for service implementation, ensuring reusability and accommodating application-specific demands. Although these benefits could also be achieved using standardized code blocks with well-defined interfaces, services have the distinguishing benefit of providing a middleware as well as in-built lifecycle management. It decouples the development of services and the integration of components not just during initial development, but also during runtime when updates to a system become necessary.

Because our architecture is designed specifically for control loops, the orchestrator can utilize domain knowledge to select the correct services and to facilitate stable ad hoc integration of new services. When integrating multiple interdependent periodical computing services into an execution chain, it is beneficial to adapt the starting point of computations so that each service begins its computations as soon as the preceding service's results arrive. These starting points are centrally managed by the orchestrator, as it is the only element with the system-level knowledge necessary to determine the relative timings for an efficient execution chain.

A. Interface Types

To allow transmission of the process model between elements and to represent any operating points of interest of a nonlinear system, as well as possible system parameter variations, in this work, we focus on *linear time-variant* (LTV) systems. Our architecture is based on the linear

state-space formulation

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t), \quad (1)$$

$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t), \quad (2)$$

with the state vector $\mathbf{x} \in \mathbb{R}^{N_x}$, the input vector $\mathbf{u} \in \mathbb{R}^{N_u}$, the output vector $\mathbf{y} \in \mathbb{R}^{N_y}$, the system matrix $\mathbf{A} \in \mathbb{R}^{N_x \times N_x}$, the input matrix $\mathbf{B} \in \mathbb{R}^{N_x \times N_u}$, the output matrix $\mathbf{C} \in \mathbb{R}^{N_y \times N_x}$, and the feed-through matrix $\mathbf{D} \in \mathbb{R}^{N_y \times N_u}$. The transmission of the process model between services is realized using the matrices \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} . For the sake of brevity, we omit the time argument when it is clear from the context.

We define interface types as data representations of the vectors and matrices introduced in Eq. (1) and Eq. (2) as follows:

- control values type τ_u for the control values \mathbf{u} ,
- system states type τ_x for the system states \mathbf{x} ,
- measured values type τ_y for the measured values \mathbf{y} ,
- reference values type τ_{ref} for the reference values \mathbf{y}_{ref} ,
- model type τ_{model} for the system's model matrices \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} .

Each interface type has a fixed size to set a known upper bound on memory consumption. If the complexity of the applied process model is known *a priori*, the maximum sizes can be set per N_u , N_x , and N_y .

B. Service Types

We define *service types* to enable interchangeable control loop elements. All services of a particular service type provide and require the same interface types and can be exchanged with one another. Our architecture defines six service types, each corresponding to one control loop element. Fig. 1 shows an overview of all interface and service types as well as their interactions.

III. IMPLEMENTATION

A. Three-tank system

We demonstrate the proposed SOA using the existing experimental setup of a laboratory-scale three-tank system at IRT, RWTH Aachen University (Fig. 2). Three fluid tanks

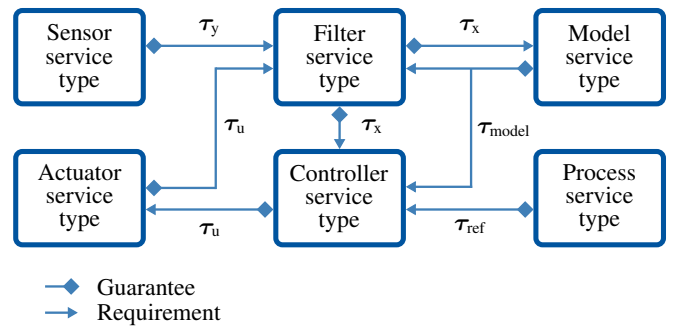


Fig. 1: Overview of the defined interface and service types, e.g., the filter service type has a requirement of measured values type τ_y and model type τ_{model} while offering a guarantee of system state type τ_x .

are connected in series, and the outlet of the third tank is connected to a liquid reservoir. To pump the liquid from the reservoir back into the three-tank system, a pump is connected to the first tank. The pump can supply a maximum flow of $Q_{in,max} \approx 50$, l/min and its dynamic response has a dead time of $T_D \approx 1.2$, s. To regulate the liquid height H_3 of the third tank, a capacitive level sensor is installed in tank three and a flow sensor is placed directly behind the pump. All measurements and control signals are transmitted to a *programmable logic controller* (PLC), which is connected through Profinet to a local network. To allow the process control of the three-tank system with the proposed service-oriented model-based control, a desktop PC is connected to this network, communicating with the PLC using the *user datagram protocol* (UDP).

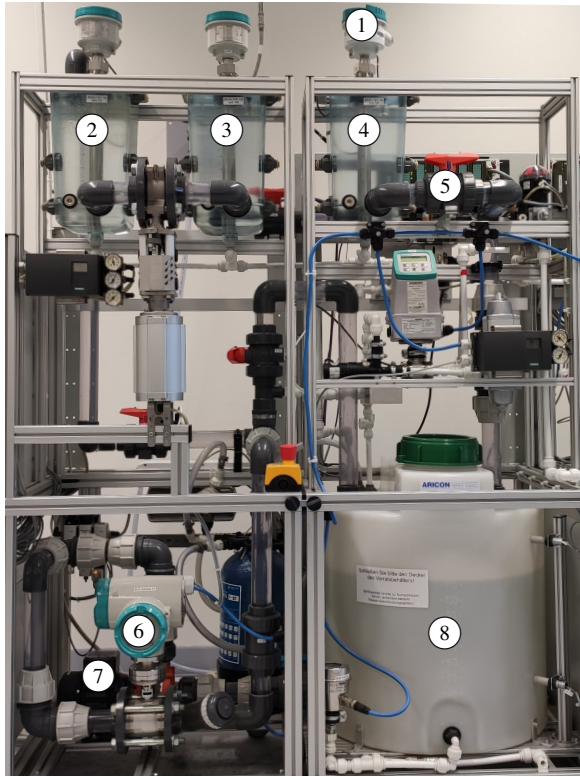


Fig. 2: Experimental setup of the three-tank system with ① liquid level sensor, ② - ④ water tanks, ⑤ outlet orifice, ⑥ flow sensor, ⑦ pump and ⑧ liquid reservoir.

To evaluate our architecture, we consider the scenario of retrofitting the process control of the three-tank system. Initially, we implement cascaded control using PI controllers for regulating the liquid height H_3 in the third tank. We start with only the initial process control services deployed. The retrofit involves two steps:

- 1) Introducing an *extended Kalman filter* (EKF) to reduce the measurement noise for the feedback loops of the cascade controller.
- 2) Upgrading the cascade controller from *proportional-integral* (PI) controllers to *model predictive control* (MPC) to account for the pump's dead-time

and system constraints.

For this evaluation, the deployment of a new service corresponds to starting it as a new application on the desktop PC. The following gives a detailed description of the services implemented within our SOA for the given retrofit scenario.

B. Implementation of Service Types

1) *Model Service Type*: As shown in the schematic illustration in Fig. 3, the three liquid tanks are connected by orifices and the dynamic behaviors of the liquid levels are modeled by the following differential equations

$$\dot{H}_1 = \frac{1}{A_0} (Q_{in} - \psi_{12} A_{12} \sqrt{2g \Delta H_{12}}), \quad (3a)$$

$$\dot{H}_2 = \frac{\sqrt{2g}}{A_0} (\psi_{12} A_{12} \sqrt{\Delta H_{12}} - \psi_{23} A_{23} \sqrt{\Delta H_{23}}), \quad (3b)$$

$$\dot{H}_3 = \frac{1}{A_0} (\psi_{23} A_{23} \sqrt{2g \Delta H_{23}} - Q_{out}), \quad (3c)$$

with difference in liquid height $\Delta H_{ij} = H_i - H_j$, the orifice cross-sectional areas A_{ij} and the corresponding flow coefficients ψ_{ij} . The input volume flow Q_{in} into the first tank is supplied by the controllable pump, whose dead-time free dynamics are modeled as

$$\dot{Q}_{in} = \frac{1}{T_p} (Q_{target} - Q_{in}), \quad (4)$$

with the time constant T_p and the target volume flow Q_{target} . The outgoing volume flow Q_{out} depends on the liquid height H_3 in the third tank and the outlet orifice with its cross-sectional areas A_{out} and flow coefficients ψ_{out} , given as

$$Q_{out} = \psi_{out} A_{out} \sqrt{2g H_3}. \quad (5)$$

The required system outputs for controlling the level of H_3 are measured of the input volume flow $Q_{in,m}$ and the measured liquid height $H_{3,m}$.

To obtain a mathematical model of the three-tank system for linear control theory with the system states $\mathbf{X} = (H_1 \ H_2 \ H_3 \ Q_{in})^T$, the system input $U = Q_{target}$ and the system output $\mathbf{Y} = (Q_{in} \ H_3)^T$, the nonlinear differential equations $\dot{\mathbf{X}} = \mathbf{f}(\mathbf{X}, U)$ (cf. Eq. (3), (4)) are linearized by applying a first-order TAYLOR series approximation for an

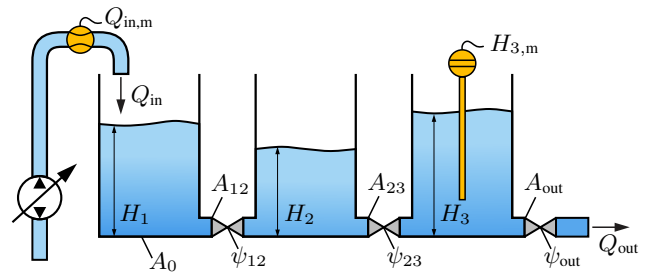


Fig. 3: Tank system consisting of three tanks connected through static orifices, a water supply with a controllable pump, one flow sensor to measure the input volume flow $Q_{in,m}$ and one capacitive sensor to measure the liquid height $H_{3,m}$ in tank three.

operating point $\mathbf{X}_{\text{OP}}, U_{\text{OP}}$. The linearized model is then time-discretized for the use within the model-based approaches (cf. Section III-B.4, Section III-B.5), yielding the general linear time-discrete state-space representation of the three-tank system

$$\mathbf{x}_{k+1} = \mathbf{A}_d(k) \mathbf{x}_k + \mathbf{B}_d(k) u_k, \quad (6a)$$

$$\mathbf{y}_k = \mathbf{C}_d(k) \mathbf{x}_k + \mathbf{D}_d(k) u_k, \quad (6b)$$

with k specifying a discrete point in time and \mathbf{x}, u and \mathbf{y} defined as relative variables, i.e., $\mathbf{x} = \mathbf{X} - \mathbf{X}_{\text{OP}}$. For the considered system $\mathbf{D}_d = 0$ holds.

2) *Process Service Type*: The evaluation process service implements the process service type and provides a reference value for the liquid height H_3 . The process service chosen for this scenario iterates through the ordered list of reference values $H_{3,\text{ref}} \in (10 \text{ cm}, 17 \text{ cm}, 10 \text{ cm}, 20 \text{ cm}, 10 \text{ cm}, 20 \text{ cm})$, changing the reference value every 120 s.

3) *Actuator and Sensor Service Type*: A PLC offers UDP interfaces to actuate the pump, and read out the liquid level and flow sensors. The actuator and sensor services interface with this UDP interface on one side, and with the proposed architecture on the other side. Additionally, they convert the sensor's output and actuator's input to the physical quantities and units expected by other services. To validate a control loop configuration in a simulation, these services may easily be replaced with simulated actuator and sensor services without affecting other services.

4) *Controller Service Type*: The implemented controller services are a cascaded control service utilizing PI controllers (cf. [15]) and an MPC service. The cascaded control contains two feedback loops. The outer loop is used to regulate the liquid height H_3 and the inner loop is designed to control the volume flow Q_{target} . Compared to MPC, the cascaded control service is a model-free control approach, i.e., it does not require a process model.

MPC is a class of model-based control methods that explicitly use a mathematical model of the process to predict its system's states. A cost function containing relevant process variables is defined and evaluated over the prediction of the system dynamics. The minimization of the cost function yields optimal control inputs. The advantages of MPC are the simple consideration of dead time of the system's dynamics and an explicit accounting for limitations of the control inputs as well as the system states. To account for the pump's dead time behavior, the time-discrete state-space (cf. Eq. 6) is augmented with additional state variables realizing a shift register [16]. As a result, the system input does not affect the dynamics of the system until the dead time T_D has elapsed.

The optimization problem of MPC results from a general continuous-time *optimal control problem* (OCP) [17]. We define the cost function of the OCP for a tracking error problem which reads

$$J = \int_{t_0}^{t_0+T_p} (\|h_3(t) - h_{3,\text{ref}}(t)\|_{Q_y}^2 + \|\dot{u}(t)\|_{Q_u}^2) dt, \quad (7)$$

using the quadratic weighting notation $\|c\|_{\mathbf{Q}}^2 = \mathbf{c}^T \mathbf{Q} \mathbf{c}$. The first cost term penalizes the system output's er-

ror (cf. Eq. (6b)) regarding a reference trajectory $h_{3,\text{ref}}(t)$ and the second cost term penalizes the control input's alteration rate $\dot{u}(t)$. In general, the OCP cannot be solved online due to the infinite number of optimization variables. Since the controller model in Eq. (6) is considered to be linear, the OCP can be approximated as a *quadratic programming* (QP) problem by applying a discretization. For this purpose, various strategies can be found in the literature [17]. The resulting QP problem can be solved by standard QP solvers and reads

$$\min_{u(\cdot|k)} \sum_{k=1}^{H_p} \|h_{3,k} - h_{3,\text{ref},k}\|_{Q_y}^2 + \|u_k - u_{k-1}\|_{Q_u}^2 \quad (8a)$$

subject to:

$$u_{\min} \leq u_k \leq u_{\max}, \quad \forall k \in [1, \dots, H_p], \quad (8b)$$

$$\Delta u_{\min} \leq \Delta u_k \leq \Delta u_{\max}, \quad \forall k \in [1, \dots, H_p], \quad (8c)$$

$$\mathbf{x}_{\min} \leq \mathbf{x}_k \leq \mathbf{x}_{\max}, \quad \forall k \in [1, \dots, H_p], \quad (8d)$$

with the prediction horizon $H_p = T_p/T_s$, sampling time T_s and the control horizon $H_u \leq H_p$. If $H_u < H_p$, $u(j|k) = u(H_u|k)$ holds for $j \in [H_u + 1, H_p]$. The matrices Q_y and Q_u are used to weight the individual penalization terms in the cost function Eq. (8a). Furthermore, box constraints for the control input Eq (8b), the alteration rate of the control input Eq. (8c) and the system states Eq. (8d) are applied.

5) *Filter Service Type*: Since direct measurement of the system states is either not possible or too costly, the application of a state estimator, also known as *soft sensors*, has been proven successful. Furthermore, state estimators can be applied for measurement noise reduction, parameter or disturbance estimation, and provide a measure of model accuracy through the covariance matrix \mathbf{P} .

We implement an EKF service and a measurement bridge service. The measurement bridge passes-through the measurements guarantee τ_y of the sensor service to the system state requirement τ_x of the controller service without further processing. State estimators reconstruct the system states based on a process model and taking into account the past control inputs and measurable system outputs. The process dynamics are considered as a LTV system, which allows the application of an EKF. The following equations are based on the discretized state-space representation of [18]. The estimators model is based on the discretized state-space model in Eq. (6) which is considered to be disturbed by the noise terms $\boldsymbol{\mu} \in \mathbb{R}^{N_x}$, $\boldsymbol{\nu} \in \mathbb{R}^{N_y}$ and reads

$$\mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d u_k + \boldsymbol{\mu}_k, \quad (9a)$$

$$\mathbf{y}_k = \mathbf{C}_d \mathbf{x}_k + \mathbf{D}_d u_k + \boldsymbol{\nu}_k. \quad (9b)$$

The noise terms are assumed to be uncorrelated, mean-free, normally distributed white noise. Measurement noise is represented by $\boldsymbol{\nu}$ and state noise is represented by $\boldsymbol{\mu}$. The estimation of the state vector \mathbf{x}_{k+1} is portioned into two steps: *time update* and *measurement update*, cf. Fig. 4. First, the algorithm is initialized with an initial guess of the state vector \mathbf{x}_0 and the covariance matrix $\mathbf{P}_0 \in \mathbb{R}^{N_x \times N_x}$.

Following the initialization of the time-discrete state-space model, the two update routines are executed recursively. To account for the covariances of the noise terms, two constant positive definite matrices $\mathbf{Q}_{\text{KF}} \in \mathbb{R}^{N_x \times N_x}$ and $\mathbf{R}_{\text{KF}} \in \mathbb{R}^{N_y \times N_y}$ are defined. Assuming uncorrelated noise terms $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, the matrices $\mathbf{Q}_{\text{KF}}, \mathbf{R}_{\text{KF}}$ are given as diagonal matrices. The deviation of the measured system output \mathbf{y}_k and the predicted output $\mathbf{C}_d \mathbf{x}_k$ is weighted with the Kalman gain matrix \mathbf{K} , cf. Fig. 4. Higher measurement noise results in smaller Kalman gain values, leading to a greater reliance on prediction over measurement.

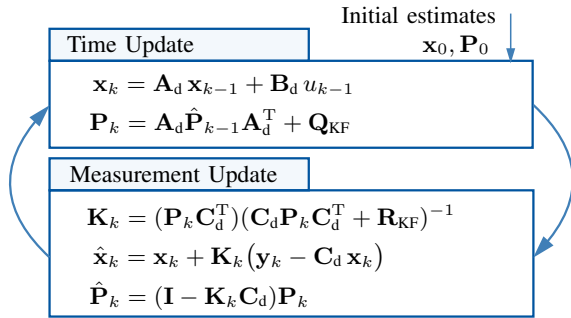


Fig. 4: EKF algorithm: Scheme of the one-step recursive update equations.

C. Orchestration

The orchestration is the central component of the control loop and the only component with system-level knowledge of all services in the network. It connects services into service compositions, herein called control loop configurations. The control loop configuration is implemented depending on the available services. In this scenario, the orchestration prefers newer services. Fig. 6 depicts the applied control loop configurations with respect to our architecture (cf. Fig. 1). In other scenarios, the orchestrator could choose a control loop configuration based on user-specified criteria. High frequent switching between model and controller configurations may destabilize the control loop. In particular, the time between setpoint changes should not be faster than the settling time of the closed-loop system. In our study, due to the user-defined retrofitting of the three-tank system's process control, high switching oscillations are avoided.

D. Scheduling of services

Each service in this scenario runs its computations with a period of $T = 100$ ms. If every service's periodic computation started at the same time, the order of completion for service computations would be non-deterministic and unpredictable. This would lead to an unknown delay between measurements and the corresponding actuator inputs, making the system hard to control at best and unpredictably unstable at worst. To counteract this, we use fixed timing offsets for each service type, as shown in Fig. 5 to achieve a predetermined order of computations. Under the assumption of known upper bounds on the communication latency between services as well as on the services' computation times, this

ensures deterministic system behavior with a known fixed delay. Because actuator and sensor service are scheduled at the same time, all other services start their computations after measurements are received by the sensor and finish them before they are enacted by the actuator, resulting in a delay of one period T . We also schedule the orchestrator in the same way as the services, because an unsynchronized orchestration can lead to nondeterministic system behavior.

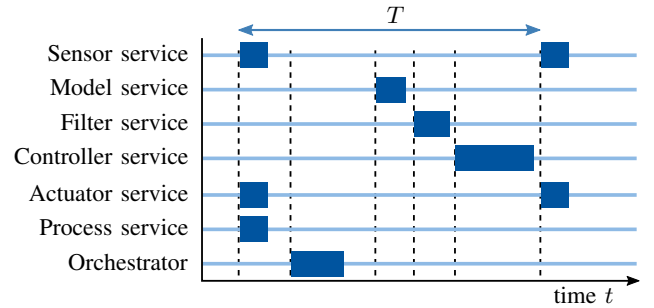


Fig. 5: Qualitative timing offsets of the implemented services. Time axis and computation times are not to scale. Services of the same service type of our demonstrator have the same offset and are consolidated in this figure. In this scenario the period T is identical for every service.

IV. RESULTS

The results of the retrofit scenario applying our service-oriented model-based process control are depicted in Fig. 7. They are divided into three time segments I, II and III based on the selected control loop configurations (cf. Fig. 6). The time segments are separated by black vertical dashed lines. The measured state of the system is represented by solid lines and the results of their state estimates with semicolon lines. The red horizontal lines indicate the three-tank system limits, e.g., maximum/minimum flow rate of the pump and the maximum liquid height of the tanks. Although not used for control, liquid heights H_1 and H_2 are displayed as well. In addition to the overview of the entire experiment, the two bottom illustrations show a detailed view of the point in time when EKF is retrofitted and the point in time when the process control is upgraded from PI controller to MPC. We tuned the MPC, EKF and PI controller in experiments.

In segment I, the cascaded controller is applied for level control of the liquid height H_3 of the third tank. According to the references values $H_{3,\text{ref}}$ of the process service type (cf. III-B.2), level control is conducted in a range of $H_{3,\text{ref}} \in [10 \text{ cm}; 17 \text{ cm}]$. As can be seen in Fig. 7, the controller bandwidth is limited, such that the transient behavior of the controlled variable is rather slow and overshooting occurs. However, the reference values $H_{3,\text{ref}}$ can be reached within 120 s and the three-tank system's limits are not violated.

In segment II, the Kalman Filter service is deployed at time $t \approx 300$ s to retrofit the process control and to reduce measurement noise for the feedback loops of the cascade controller. In this work, the estimates \hat{Q}_{in} and \hat{H}_3 are initialized using the previous measurements $Q_{\text{in,m}}$ and

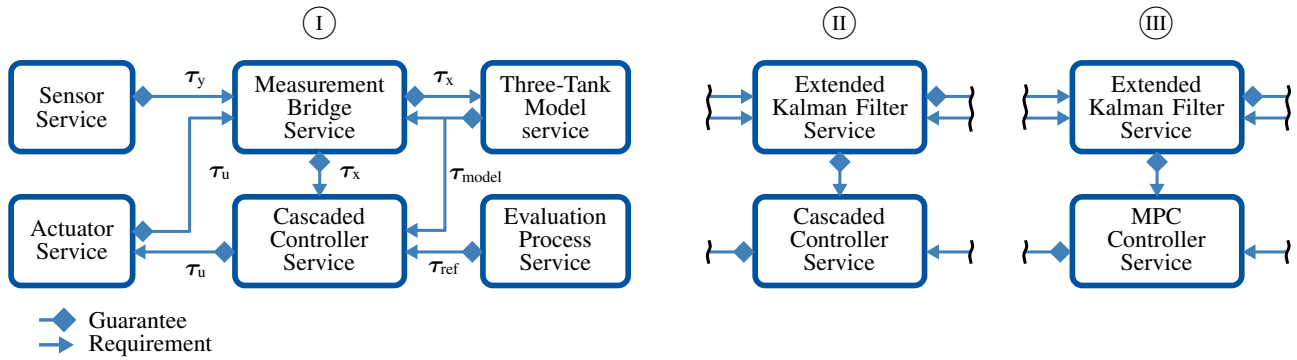


Fig. 6: Representation of the control loop configurations, which are selected within our service-oriented architecture by the orchestrator to retrofit the three-tank system’s process control during operation.

$H_{3,m}$. Initially, the rest of the system states are guessed. Due to the initialization error of the EKF, switching from the measurement values to the estimate values \hat{Q}_{in} and \hat{H}_3 induces a discontinuity to the closed loop. This affects the accuracy of the controlled variable H_3 and leads to a transient control error $e_{H_3} < 0.5$ cm, (cf. detailed view in Fig. 7). After approximately 25 s the state estimates have converged to the actual system states, and the control error has decayed.

Thereafter, in segment II, the cascaded controller is operated in a range of $H_{3,ref} \in [10 \text{ cm}; 20 \text{ cm}]$. As shown in Fig. 7, the control behavior yields an unsatisfactory response. The reason for this is the controller parameterization, which leads to a too aggressive control behavior for the reference value $H_{3,ref} = 20$ cm. Due to this, at time $t \approx 380$ s the liquid level of the first tank violates the three-tank limit causing the pump to be deactivated (cf. gray marked area). As a result, the fluid levels of the three-tank system drop. As soon as the pump is enabled again, the cascaded controller attempts to reach the reference value $H_{3,ref}$, yielding to a repeat overshooting. To extend the bandwidth of the cascaded controller, gain scheduling could be applied or a model-based controller, such as the *linear-quadratic regulator* (LQR) could be used. In this scenario, the initial cascade controller is upgraded to MPC to account for both, the system constraints and the pump’s dead-time behavior.

The process control MPC service is deployed at time $t \approx 460$ s, which is indicated by the black vertical dashed line. As can be seen from the detailed view in Fig. 7, no discontinuities result from the upgraded process controller compared to the previous retrofitting by the EKF. The MPC smoothly takes over the level control and transfers the controlled variable H_3 to the reference value $H_{3,ref} = 20$ cm. As can be seen from the results in segment III, the MPC achieves a significantly better control result, as expected. Overshoot is reduced significantly and the time to reach a new reference value shows distinctly lower convergence times. It is noteworthy that the gray-marked area is at time $t \approx 620$ s. This area clearly shows that the state estimation for transient system behavior tends to overestimate the tank levels H_i . Consequently, the MPC does not fully utilize

the upper limit of $H_{3,limit} = 30$ cm and stops at a value of $H_3 \approx 28$ cm.

For the given scenario, overestimating the upper level limit leads to more conservative control performance, which is unproblematic. However, this demonstrates the weakness of a model-based control approach. In general, model mismatch can lead to poor control performance using MPC. In such a case, our proposed SOA enables us to switch back to the cascaded controller and thus revert to the initial control.

V. CONCLUSION AND FUTURE WORK

This paper presented a *Service-oriented architecture* (SOA) for CPS, that enables runtime-flexible control loops by modeling control loop elements as services and combining them into service compositions using an orchestrator. To evaluate our architecture, we retrofitted the process control of an experimental three-tank system setup. The scenario considered three different process control configurations, which required exchanging of various control loop elements. With our architecture, we successfully demonstrated the exchange of services at runtime.

The ability to dynamically exchange services at runtime shows, that the system integration was successfully decoupled from the service implementation. We also show that the ability to dynamically exchange services benefits control systems, in particular by enabling retrofits to keep good control performance despite changing requirements. We conclude that future CPS cannot benefit just from SOA in their development and maintenance, but also in their performance by becoming more flexible and robust.

Future research will analyze the transition period between services, how it affects the system’s dynamic behavior, and how this effect can be minimized or compensated. We will also research orchestrators that increase robustness by dynamically responding to component failures. For arbitrary switching topology, rigorous switching conditions must be developed to guarantee stable system behavior.

ACKNOWLEDGMENT

The authors wish to express their sincere gratitude to A. Kampmann and J. Beerwerth for the discussions.

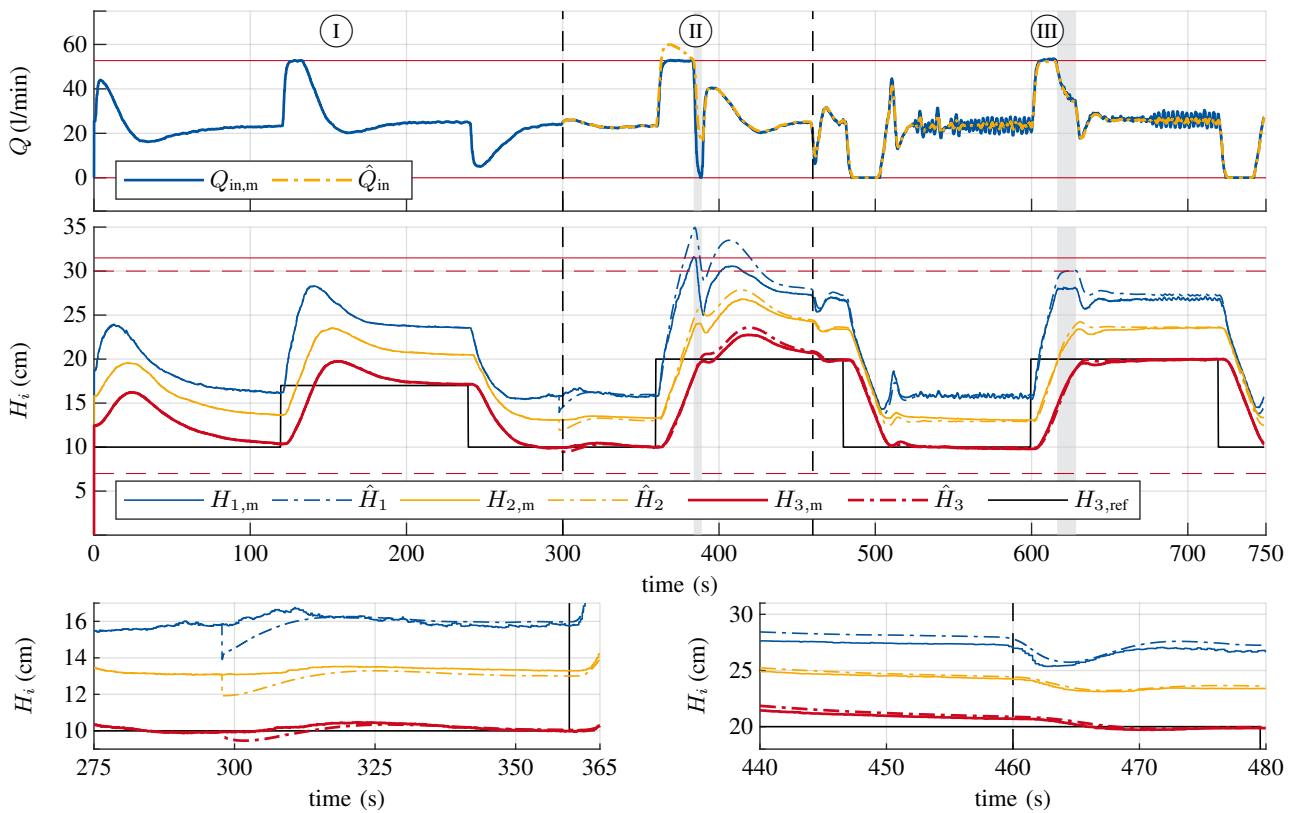


Fig. 7: Experimental level control results of the liquid height H_3 of the third tank. In segment I, the initial process control is applied using a cascade of PI controllers. In segment II, a retrofit to an EKF for measurement noise reduction is realized and in segment III, the process control is upgraded from PI controller to MPC to account for the pump's dead-time behavior and system constraints. The transition from segment I to II, as well as from segment II to III, are magnified at the bottom.

REFERENCES

- [1] M. Törngren and U. Sellgren, "Complexity Challenges in Development of Cyber-Physical Systems," in *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, ser. Lecture Notes in Computer Science, M. Lohstroh, P. Derler, and M. Sirjani, Eds. Cham: Springer International Publishing, 2018, pp. 478–503.
- [2] J. Bach, S. Otten, and E. Sax, "A Taxonomy and Systematic Approach for Automotive System Architectures - From Functional Chains to Functional Networks:," in *International Conference on Vehicle Technology and Intelligent Transport Systems*, 2017, pp. 90–101.
- [3] M. Fu, "Switching Adaptive Control," in *Encyclopedia of Systems and Control*, J. Baillieul and T. Samad, Eds. Cham: Springer International Publishing, 2021, pp. 2261–2266.
- [4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, pp. 60–74, 2022.
- [5] T. A. Johansen and T. I. Fossen, "Control allocation—A survey," *Automatica*, vol. 49, no. 5, pp. 1087–1103, 2013.
- [6] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web services architecture," *IBM Systems Journal*, vol. 41, no. 2, pp. 170–177, 2002.
- [7] M. Lohstroh, Í. Í. Romeo, A. Goens, P. Derler, J. Castrillon, E. A. Lee, and A. Sangiovanni-Vincentelli, "Reactors: A deterministic model for composable reactive systems," in *Cyber Physical Systems. Model-Based Design*. Springer International Publishing, 2020, pp. 59–85.
- [8] A. Jula, E. Sundararajan, and Z. Othman, "Cloud computing service composition: A systematic literature review," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.
- [9] H. Vural, M. Koyuncu, and S. Guney, "A systematic literature review on microservices," in *Computational Science and Its Applications – ICCSA 2017*. Springer International Publishing, 2017, pp. 203–217.
- [10] S. Kugele, D. Hettler, and S. Shafaei, "Elastic Service Provision for Intelligent Vehicle Functions," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 3183–3190.
- [11] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, pp. 60–74, 2022.
- [12] H. Stoll, D. Grimm, M. Schindewolf, M. Brodatzki, and E. Sax, "Dynamic Reconfiguration of Automotive Architectures Using a Novel Plug-and-Play Approach," in *IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, 2021, pp. 70–75.
- [13] A. Kampmann, B. Alrifaae, M. Kohout, A. Wüstenberg, T. Wopen, M. Nolte, L. Eckstein, and S. Kowalewski, "A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 2101–2108.
- [14] M. Kloock, P. Scheffe, O. Gress, and B. Alrifaae, "An architecture for experiments in connected and automated vehicles," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 4, pp. 175–186, 2023.
- [15] K. J. Åström, T. Hägglund, and K. J. Åström, *PID Controllers*, 2nd ed. Research Triangle Park, N.C: International Society for Measurement and Control, 1995.
- [16] J. Lunze, *Regelungstechnik 2: Mehrgrößensysteme, Digitale Regelung*. Berlin, Heidelberg: Springer, 2020.
- [17] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation and Design*, 2nd ed. Santa Barbara: Nob Hill Publishing, LLC, 2020.
- [18] D. Simon, *Optimal State Estimation: Kalman, H [Infinity] and Non-linear Approaches*. Hoboken, N.J: Wiley-Interscience, 2006.