Temporal Properties in Component-Based Cyber-Physical Systems

Tobias Sehnke, Matthias Schultalbers, Rolf Ernst

Temporal Properties in Component-Based Cyber-Physical Systems

Tobias Sehnke[1], Matthias Schultalbers[1], Rolf Ernst[2]

[1]Powertrain Mechatronics, IAV GmbH, {tobias.sehnke | matthias.schultalbers}@iav.de
[2]Institute of Computer and Network Engineering, Technische Universität Braunschweig, ernst@ida.ing.tu-bs.de

Abstract– Modern cyber-physical system are often responsible for safety critical control functions. The quality and performance of these function is directly linked to asserted real-time behavior. In the literature requirements on this behavior are considered exclusively in the context of central architectural decision making. This can be problematic because such systems can consist of a large number of reusable software-components, supplied by different stakeholders that are executed on interconnected processing units. In this work, we present a novel approach to obtain temporal requirements for the design of cyber-physical systems in a more distributed way. The significant advantages are the following: First, our approach simplifies the transfer of assertions made during functional design to the system design process which is important when building systems from reusable components. Second, our approach enables us to consider important aspects of temporal properties in an automated form, that otherwise have to be addressed manually. We demonstrate the applicability of our approach using an automotive use case.

Keywords– Cyber-physical systems, Model-based system engineering, Temporal requirements, Component reuse, Control software

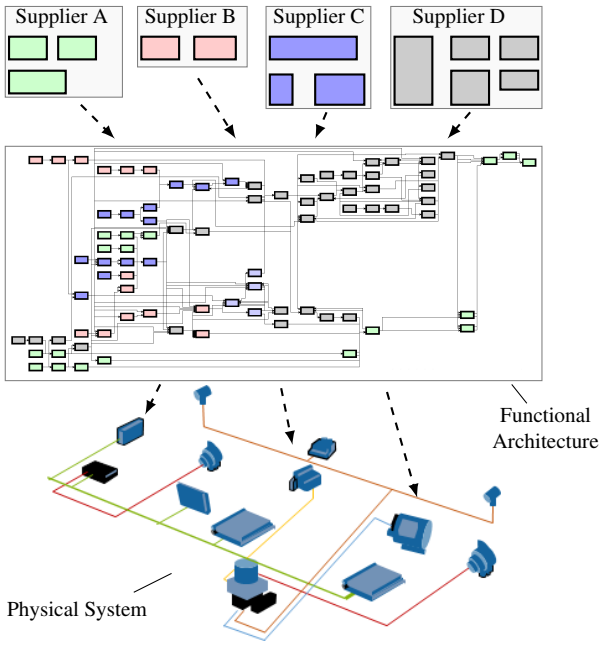

Figure 1: Abstract example for the development of a cyber-physical system from reusable components.

1 Introduction

Context Cyber-physical systems are characterized by a tight integration of a physical mechanism with computer based algorithms that control or monitor these mechanisms. This integration provides several benefits. First, algorithms can regulate systems constantly and accurately, so that challenging requirements can be met. For example modern emission standards on combustion processes can only be met using sophisticated control algorithms. Additionally software can help to reduce maintenance by adapting to wear, aging or defects in the operated system based on the diagnosis of sensors, actuators and the controlled system itself. The increasing responsibilities require a high degree of dependability and quality, as failures may result in either serious damages or expensive fines.

Many of these algorithms originate in control engineering, such as controllers, observers, automata or parameter estimators. The design of such functions is based on physical models of the controlled systems that represent the behavior of the plant. An important aspect for the quality of control applications is the behavior of the executing platform, which has been studied for a long time in the field of networked control systems. There is an extensive amount of literature on the design and test of control functions with known latencies from sensors to controllers known latencies from controllers to actuators, as well as known sampling rates [8, 28]. For example, unaccounted latencies in control loops can reduce the achievable control performance or may even destabilize a controlled system. Therefore it is necessary that requirements on such properties are specified appropriately, to ensure that they are considered appropriately during system design.

Several works propose a specification of these properties through a centralized instance as part of the system design process [11, 24]. In the literature the applicability of these processes are demonstrated for systems that can be comprehended in every detail by a small team.

However, identifying all requirements in regard to time can be problematic for large industrial scale systems due to several reasons. For example, in the automotive industry systems are designed from hundreds of reusable components, developed by specialized teams that are separated in space and time. This separation limits communication between architects and domain experts. Additionally, only a small number of components have access to the physical world, while the majority of components depends on other components for their input data. These data dependencies result in complex signal flows that can depend on the behavior of multiple processing units and communication links. Thus, the central planning instance must be aware of all control theoretic details of the individual components as well as the signal flow to specify appropriate requirements. From experience, we know that this assumption is unrealistic and generally results in systems that are underspecified. Without proper specification, the design process of real-time systems will then either lead very conservative designs or to misbehaving systems.

Thus, improvements to current practices are required to disclose assertions used in control design, so that they can be considered during the design of cyber-physical systems.

Contribution: In this paper, we discuss a component based approach to derive system-level timing requirements from component descriptions. Major goals of this process is to improve the quality of system designs by disclosing requirements that are otherwise easily overlooked. The approach consists of two key parts. The first part are component descriptions based on the temporal semantics model. Thereby, we borrow from the concept of the data-age, where requirements refer to event-chains not yet defined. Second, we demonstrate how these descriptions can be transformed into requirements used by standard models.

While our work is mainly motivated by automotive software, the approach is applicable to any component-based cyber-physical system with control functions.

2 Background

In the past the real-time community has developed several formats and methodologies consider test temporal requirements during the development process. Work has focused on *top-down* system engineering according to the V-model [11, 24]. Thereby, it is assumed that the whole cyber-physical system and its algorithms are planned by a central planning instance. This design includes the functional and the dynamical architecture of the system, which is documented and tested using models. Temporal properties are addressed by specifying and testing timing requirements based on event-based timing models, which focus on observable events and their relation to another [4, 20, 11]. Properties that are represented in event-based models are the latency of an event-chain, the event-rate and the synchronicity of events. The latency describes the difference between the tags of events that are causally linked by a signal flow. The event rate describes the difference of the tags of two consecutive events, while the synchronicity can either refer two the difference of the tags or to the difference of the latency of two events.

Several event-based standards for the specification of timing requirements have been proposed such as the AUTOSAR Timing Extensions [1, 11], AMALTHEA [27] or MARTE [6]. These requirements then provide boundary conditions for the mapping of software onto processing units and the design of the scheduling [12]. The implementations can be verified using real-time analysis methodology such as Real-time Calculus [26], Compositional Performance Analysis [9] or automata-based methodologies [5, 14].

However, requiring that timing properties have to be obtained by a central planning instance, will likely lead to underspecified systems unless they are small. Without proper specifications the software-hardware co-design process might lead to expensive and unnecessary errors. This has also been pointed out by [18, 21, 13]. To enhance the number of timing constraints, iterative processes can be used as in [13]. Still, the identified constraints are system specific, which limits reuse. Additionally, the underlying event-based models can not consider all relevant temporal aspects of control software. First, it is not possible to consider software effects in the components themselves on temporal properties, such as delaying or filtering. Second, these models can not represent time related properties that can affect the quality of the control software, such as *aliasing* or the *bandwidth*. Therefore, a lot of manual labor is required to derive real-time requirements from the abstract models used in control engineering An improved description for temporal properties in cyber-physical systems was provided in [25]. The main drawback of the approach is that it can be used for analysis only but not for design.

There exist also several contract based approaches, which document implementation aspects such as execution times and response times [3, 23, 21]. These contracts, based on assume-guarantee reasoning, use interfacing to decompose system into smaller more manageable units. If data data dependencies exist, contracts are obtained by decomposing an event-chain of an existing system-level timing specification obtained from top-down design processes. One drawback is the assumed composability, which is not always given due to interactions as discussed by [17]. Also components may have to assume properties, that may not by functionally relevant to the individual component but for the overall functionality of the system. This is described by [7] as "assumption explosion". Reusing such contracts to obtain timing requirements for a new design, is then likely to result in several unnecessary constraints.

3 The Temporal Semantics Model

In a cyber physical system data will often represent physical states which get their context from the physical system.

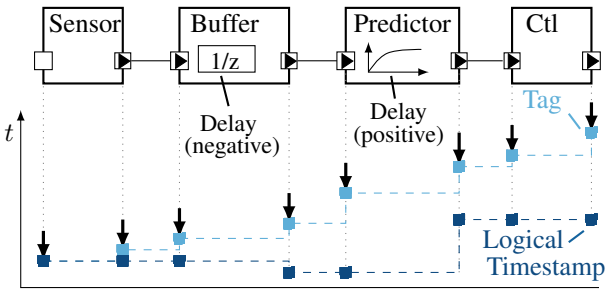

Figure 2: Example showing the propagation of a logical time stamp along the signal path and its alteration by algorithmic delays in the buffer and the predictor.

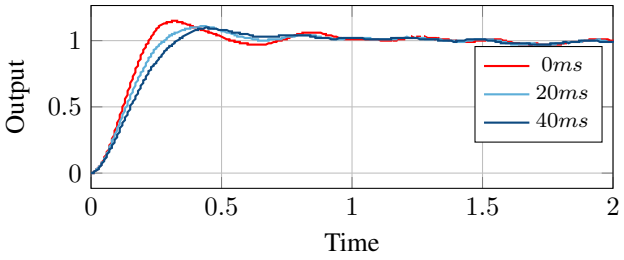

Figure 3: Step response of the simulated system for different time delays.

Also, it is often much more important to know, what point in time the data actually represents rather than knowing the time of the read operation. The general approach of our model is therefore to focus on the represented point in time, which we call the logical timestamp. Additionally we acknowledge, that often the original source of the processed information as well as the signal paths are not necessarily important as long as the value is correct and has the desired properties. Therefore, it is sensible to consider the properties of information at the interfaces, rather than as a property of a signal flow. Based on these considerations we developed the temporal sematics model which will be outlined and motivated in the following. For a more formal description of the model we refer to [25] and to the provided supplementary material[1].

3.1 Events, Signals and Causal Chains

The temporal semantics model provides a means to describe temporal properties of data in a cyber-physical system based on data-events. A data-event is an observable occurrence at an interface. We thereby differ between sampling, actuation, reading and writing, whereas sampling and actuation are operations, where a driver component interacts with its physical environment. Reading and writing are operations at data interfaces whereby a component communicates with other components. In AUTOSAR these interfaces are called VARIABLEDATAPROTOTYPES. The set of events that occurs at a single interface is called a signal. In control theory signals are often represented in the frequency domain. If a signal is represented in the frequency domain, it is called a spectrum of the signal. For each signal, there exists a signal path, which describes the flow of information from a sampling interface to the respective interface. A causal chain describes causal relationship between events.

Each event is described by a triple consisting of a value, the time of the occurrence of the event and the so called logical timestamp, which is the concrete point in time, where the value represents the physical state. The logical timestamp is motivated by the assumption the values are physically based and that they represent a physical state in the physical world at a certain point in time. This logical timestamp can be obtained by propagating time of a sampling instant at a physical interface through along a signal flow. We further assume that the temporal context (logical timestamp) can be altered by so called *algorithmic delays* in the components due to buffering (negative) or predictions based on models (positive). An example for a simple signal flow is shown in Fig. 2. Note, that the timestamp $\hat{t}_x^k$ progresses with the signal flow due to computation. In contrast the logical timestamp is propagated from the sampling event and altered only by the algorithms in the components.

3.2 Temporal Signal Properties

Using this model, we can define a set of measurements that are based on the logical timestamp. These measurements, which we call temporal signal properties, provide an improved description for the timing behavior of such signal flows compared to existing descriptions. They are called the *logical data age*, *data synchronicity*, *logical sampling rate*, *logical band limit* and *aliasing*. In the following we present the individual properties and discuss why they improve the description for the timing behavior of such signal flows compared to existing descriptions.

Logical Data Age is a measurement for input delays. Input delays are an important aspect as they can reduce the performance of a controlled system significantly [19]. Effects of time delays are overshooting behaviors or an overall loss of stability. Input delays may also reduce the ability of a monitor to notice a defect in the system within a given timeframe.

In order to demonstrate the effect of a time delay on a control system, we set up a small experiment. For these experiments we build a Simulink model of a PID controller which controls a stable plant. The plant itself can is modeled by a second order transfer function $G(s) = 1/(s^2 + s + 1)$. The output of the plant is sampled at a rate of $1ms$. This signal is then filtered by a first-order filter. This filtered signal is then used by a controller at a rate of $10ms$. The controller parameters are $P = 70$, $I = 10$, $D = 10$. The absolute value of the control output is limited to 100. To show the effect of the time delay we add a constant time delay block into the system and

[1] http://arxiv.org/abs/1711.09130

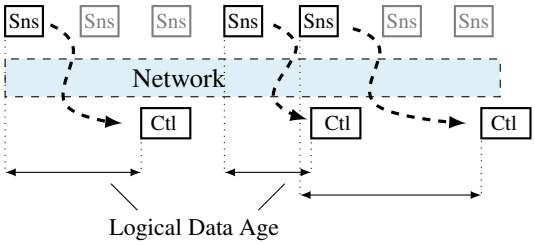

Figure 4: Logical data age of an abstracted signal flow.

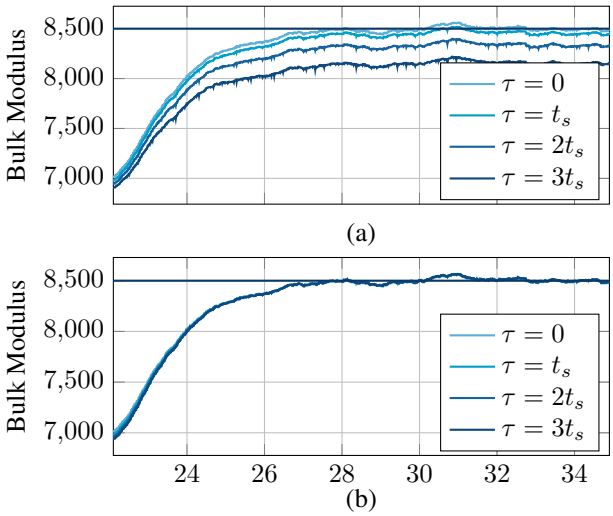

Figure 5: Simulation results for the estimated bulk modulus when a) the pressure p_{cr} is delayed by τ; b) the pressure p_{cr} and the angle of the crank shaft α are delayed by τ.

simulate the step response. The results of these experiments are shown in Fig. 3. As expected, does the increasing delay worsen the behavior of the controlled system by increasing the overshoot.

In event based models such as AUTOSAR, time delays are measured by end-to-end latencies. Still, component may delay values internally using algorithms or buffers. Such delays can not be measured by the end-to-end latency and will therefore be ignored. Thus, the latency is not always a proper measure of the time delay. We address these effects, by measuring the logical data age which is the difference between the tag and the logical timestamp of an event. An interpretation of the logical data age of a controller is shown in Fig. 4. Note, that the concrete signal flow that may include several functions is abstracted into a point-to-point communication through an abstract network.

Data Synchronicity is an is a measure of the temporal coherence of data, which can be important when dealing with data fusion algorithms or for functions that compare states. These algorithms will often compute a state based on several input values. If these values are not temporally coherent, the processed output will differ from the true physical state.

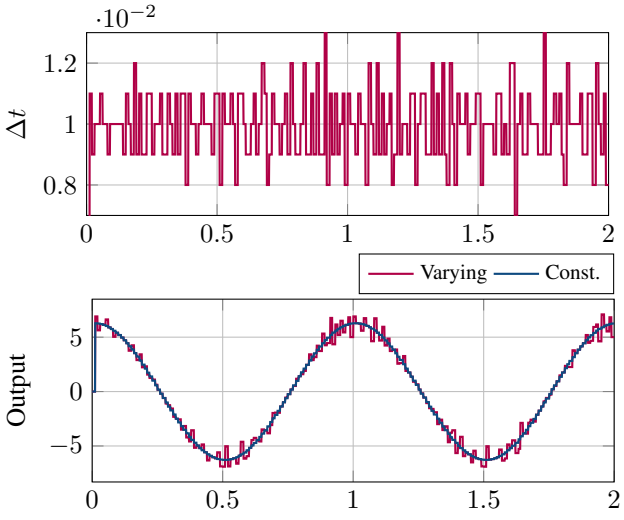

Figure 6: Output signal of the differentiator for constant and varying logical sampling rate.

To emphasize the importance of that temporal coherence, we simulate an application function which measures the states of an injection system presented in [2]. The function computes the states of fuel in the common rail of a combustion engine using an estimator. This estimator uses a zero dimensional model of the common rail to estimate the bulk-modulus based on the inflow and outflow and the measured pressure in the common rail. For a more detailed description of the algorithm we refer to [2].

Again we setup an experiment using a Simulink-model. The estimator depends on several inputs such as the fuel pressure, the rotational angle of the crank shaft as well as the rotational speed of the engine and the parameters of the injection control. To obtain the system behavior we use a reference model and add measurement noise. For our experiment, we vary the age of input properties using time-delay blocks. We discuss two different setups, where the measured pressure p_{cr} and the angle of the crank shaft α are delayed. At first, we delay only the pressure signal p_{cr}. Next, the angle α is also delayed. Note that the reference value for the estimated bulk modulus is set to $E_{Ref} = 8500$.

In the first experiment, the delay of the pressure causes a constant offset in the estimated bulk modulus. For a single sample delay the difference is neglectable. For a delay of two samples, an offset error of 2% (see Fig. 5) is identified. From experience we know that such constant offsets are hard to relate to timing and have a high chance of not getting noticed. If α is also delayed, the offset error vanishes as shown in Fig. 5. This small setup demonstrates that it can be important for some values to be synchronous while the absolute delay may be irrelevant.

In our model we measure data synchronicity as the difference of the logical timestamps of values that are computed simultaneously. This is advantageous to measuring the difference of tags of the input values, because that measure can

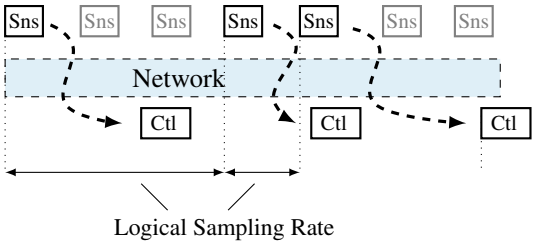

Figure 7: Logical sampling rate of an abstracted signal flow.

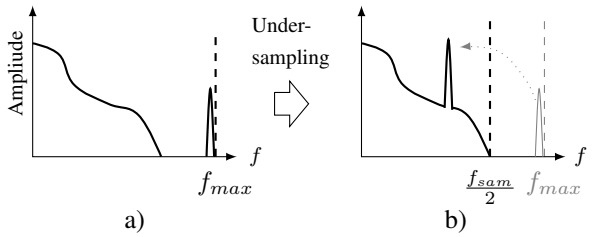

Figure 8: Representation of a signal with measurement noise that is undersampled.

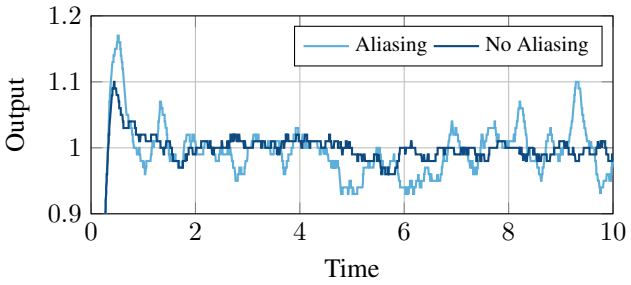

Figure 9: Step response of the simulated system with and without aliazing.

not represent the properties of signal flows. The advantages to compared to synchronicity descriptions based on event-chains are the same as for the logical data age.

Logical Sampling Rate is a measure of how fast the information can change at an interface. This is important when dealing with periodically sampled data.

To demonstrate the effects that can be caused by a falsely assumed sampling rate we provide an example. This example includes a function, which reads values from an external sensor and then differentiates this value. Such functions are used for example to compute an acceleration from a speed. We execute this function with a constant period and design the differentiator, assusming a constant sampling rate of the information of $10ms$. The nominal behavior for this function is shown in Fig. 6. Next, we vary the sampling rate of the input values using a random jitter of up to $2ms$ as shown in Fig. 6. As a result, the differentiator produces a signal with undesirale noise, which indicates a behavior that does not exist.

The variation described in the example can be caused either by a variation in the latency of the signal flow or by a jitter in the sampling itself. This is also shown in Fig.7 for an abstract signal flow. To represent such effects, we measure the sampling of the information as the difference of the logical timestamps of two consecutive events. This improves on current models as they describe either the event rates an events or the latency of the event chain but not both simultaneously.

Logical Band Limit is a measure for the range of frequencies, that a signal represents. More specifically it describes the highest frequency $f_x^{\max}$ in which a signal x that can be represented as a spectrum can have an amplitude that is nonzero. An example for a spectrum of a signal is shown in Fig. 8 a). The maximum frequency is determined by the cutt-off frequency of filter algorithms in the signal path as well as the logical sampling rates. This is because a sampled signal cannot represent frequencies that are larger than the Nyquist frequency [15], which is determined by the sampling rate. If there exists no spectrum (e.g. if the signal represents a discrete state), the band limit describes a lower bound on the time, in which the signal does not change its values. The logical band limit improves the description of the information content of a signal compared to using only the event rate. For example, assume a signal has been filtered with bandwidth that is much lower than the sampling frequency. In that case the event rate would indicate a much wider spectrum than the one that is actually available.

Aliasing describes a distortional effect effect than occurs when a signal with high frequencies is sampled at a lower resolution. From Shannon's sampling theorem we can derive that aliasing will occur if an input reads a signal whose frequency spectrum that is greater than the Nyquist frequency of the input. This occurs exactly, when the logical band limit of the output signal is larger than the logical sampling rate of the input signal. Note, that once aliasing occurs, it will be propagated along the signal path.

A simple example of this effect is shown in the bode plot in Fig. 8 b). Note, that the high-frequency peak is undersampled. Thereby it is folded into the lower end of the spectrum. We also use the previous control example to demonstrate the behavioral effects of aliasing. Therefore we introduce measurement noise to the output signal. Noise is added to the measured signal using a band limited white noise with a noise sampling of $2e$-3 and a noise power of $5e$-6. In one experiment, the filter runs in the same rate as the sensor (1ms). In the second setup the filter runs in the rate of the controller, which results in aliasing. The results of this experiment are shown in Fig. 9. The experiments show that the controlled system reacts in less than $0.5s$ with a small overshoot and settles at the reference value for the faster execution rate. The measurement noise has only a minor effect on the tracking behavior. In contrast, the slower execution rate, the system

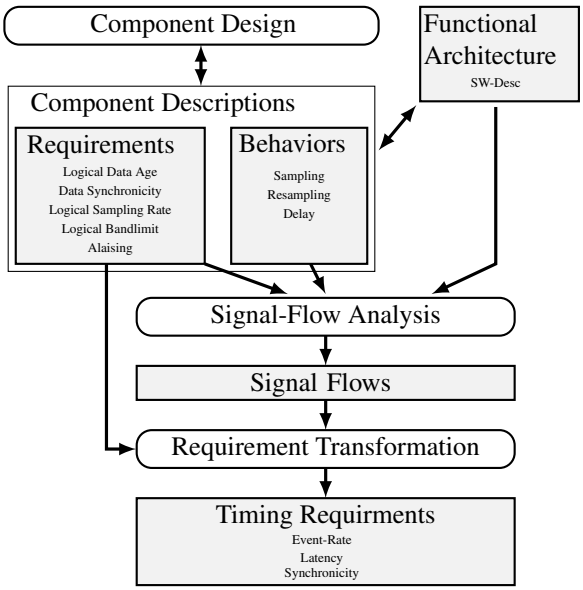

Figure 10: Overview of proposed proposed process.

```
<COMPONENT="Controller">
    ...
  <SIGNAL–REQUIREMENTS>
    <ALIASING–REQUIREMENT>
      <DATA–ELEMENT–REF>Ctl_y </DATA–ELEMENT–REF>
      <ALLOWED>FALSE</ALLOWED>
    </ALIASING–REQUIREMENT>
    <LOG–DATA–AGE–REQUIREMENT>
      <DATA–ELEMENT–REF>Ctl_y </DATA–ELEMENT–REF>
      <MAX>20.0</MAX>
    </LOG–DATA–AGE–REQUIREMENT>
  </SIGNAL–REQUIREMENTS>
    ...
  <BEHAVIORS>
    <ALG–DELAY>
      <DATA–ELEMENT–REF>Ctl_u </DATA–ELEMENT–REF>
      <SOURCE–ELEMENT–REF>Ctl_y </SOURCE–ELEMENT–REF>
      <ABS>0.0</ABS>
    </ALG–DELAY>
  </BEHAVIORS>
    ...
</COMPONENT>
```

Figure 11: Example for a component specification.

reaches the set point value at the same speed, but oscillates afterward. This is because aliasing causes the high-frequency measurement noise to be interpreted as a low-frequency oscillation. Since low frequencies are hard to suppress, the controller follows them, resulting in significant limitations of the achievable control performance.

4 Application to the Design-Process

As shown by the examples, it is possible to test behavioral effect on the functionality of the individual components in regard to temporal signal properties without knowing the actual signal paths in the system. Testing methodologies, other than simulations, include linear analysis and model checking [10]. The obtained results retain valid, as long as the concerned physical system does not change severely. Therefore, it is sensible to annotate known requirements on temporal signal properties as part of the component description. These requirements are truly reusable, because they do not reference to a specific architecture. This can be important, when using the same function on control units of different suppliers.

Using these component descriptions, we propose a process to improve the timing requirement-set of system designs. Thereby, we first identify the related signal paths of the specified interfaces. Next we transform the signal requirements into requirements that can be represented by are standard event-based models. This is necessary, as companies have invested heavily into tooling concerning their respective domain models such as AUTOSAR.

Our work differs from other work in the form that existing work uses component annotation only for properties that are related to the implementation of the real time system, such as execution time, response time or service curves, but does not address requirements on signals. We also do not rely on interface algebra, which means that component description only have to include requirements that are a concern of the component. Thereby we lower the amount of assumptions compared to the approaches discussed in [7].

4.1 Extended Component Descriptions

In order to apply our metrics to the design process, we specify requirements on these metrics as part of the component description. Note that our approach is similar to the *Age-Constraint* constraint used in AUTOSAR. Thereby we define constraints on event-chains that are not yet defined. Thus, our proposed extension can be added to existing software-component descriptions in a similar way as the AgeConstraint. Our description is applicable to either sender-receiver interfaces as well as client-server interfaces. In the following we will discuss the general frameworks for the so called *SIGNAL-REQUIREMENTS* and *BEHAVIORS*. An example for the proposed description is shown in Fig. 11 for a controller which includes two requirements and a behavior.

4.1.1 Behaviors

Behaviours abstract the internal design of a component in order to disclose changes in signal properties, without having to understand each component in its detail. Currently our model includes the description of the Sampling, Resampling and Delays as so called BEHAVIORS. These behaviors are formulated as a relation of the properties of an output to the properties of an input.

Delay is a change in the temporal context of the information by the component. They can be either positive or negative

depending on the algorithm. Foramlly a delay is defined as the difference between the logical timestamps of an input and a related output interface. A positive delay may result from buffering insider of components. Negative delays may result from a prediction of future states.

Sampling and Resampling is a change of logical band limit of a signal with a defined band limiting frequency. Various methods are known that can alter the sampling rate of signals. In general these methods can be described as a filter [16]. Assuming that the filter is ideal, the band limit of a signal is equal or less than the cutoff frequency. For non-ideal filters, we can define a criteria for the cut-off frequency such as the $-3dB$ mark of the filter. Sampling indicates that the data is read through a sensor interface, while resampling indicates that the data is read through a read interface.

4.1.2 Requirements

To disclose assumptions on signal properties we also document requirements. Generally, the properties of any of the defined signal properties can be specified. These requirements are specified either as bounds, using the values MIN or MAX or as an absolute values ABS. Additionally, we imply that the values of the corresponding signals are correct according to a chosen norm.

4.2 Graph-Based Search for Signal Paths

Signal properties refer only to interfaces, but not to the respective signal flows. Therefore signal flows need to be determined in order to formulate timing requirements for the respective system. In the following we discuss methodologies to identify such signal paths. Thereby we assume that the functional architecture is known, including a description of the components and the communicating interfaces. At first we construct the so called signal flow graph. Based on this graph, we can then identify the signal paths.

4.2.1 Constructing the Graph

The signal flow graph is a mathematical representation of the data dependencies in the software. It consists of nodes, which describe the interfaces and edges, which describe the dependencies between these interfaces. We differentiate between so called inter-component edges, which represent communication between components and intra-component edges, which represent dependencies inside of components. The first type can be obtained from the architecture description which relates the input and output interfaces of the components. Next we determine the intra-component edges. Thereby it is important to represent all relevant dependencies while ensuring that the number of dependencies in the graph do not grow too large. That would be problematic, because that would result in a large amount of possible signal paths for each interface.

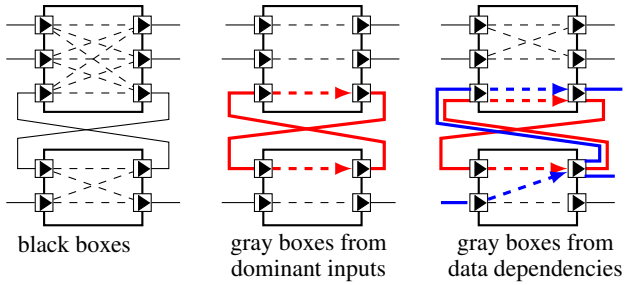

Figure 12: Considered approaches to represent dependencies inside of components.

In the following, we discuss three different approaches to address this problem and discuss their advantages and disadvantages.

Black Boxes At first, we assume that no information about the internals of the components are known during the construction of the graph. Thereby, the dependencies are over-approximated by mapping each output to each input of a component. A small example for such a so called black box interpretation of a single component is shown in Fig. 12 a). The problem of this over-approximation is, that it adds a large number of edges to the signal flow graph. Given a software with hundreds of components, this approach may lead to a large number of possible paths in the graph. The so called path explosion hinders the following search for signal paths, limiting the overall applicability of our approach.

Gray Boxes from Dominant Inputs Second, we use an approach based on so called dominant inputs. Thereby, we define exactly one input interface, which we consider as relevant. These relations can be obtained from the linking information included in the specified behaviors. The motivation for the dominant input The use for this approach can be justified by the fact the logical timestamp of an output signal that is computed from multiple input values has different possible interpretations such as medians or bounds. By choosing the dominant input, we decide on an interpretation where the logical timestamp of the output depends only on the logical timestamp of a single interface. The advantage of the approach is that the final graph will have only a very limited amount of paths. A problem of the approach is the occurence of cycles. Such cycles are actually self-references, which means that we will not able to find a sensor interface from which to derive a physical time. An example for such a dependency is shown in Fig. 12 b), where the cyclic dependency is highlighted in red.

Gray Boxes from Data Dependencies At last, we analyze the internals of the component and relate data dependencies between the input and output interfaces. These dependencies may be obtained from documentations or code-analysis.

The advantage of the approach is that it prevents unnecessary over-approximations for the relations of inputs and outputs. At the same time we might not be able to prevent that inputs, whose influence on a value may not be very important will still be represented as a dependency. At the same time we prevent, that the following path search will get stuck in a single cycle since there will always exist other paths to derive the logical timestamp from. For our example we highlight this additional path in blue in Fig. 12 c).

4.2.2 Obtaining Signal Paths from the Signal Graph

After the construction of the signal flow graph, we identify the signal paths. The specific task thereby is to identify all relevant signal flows that end at an interface of interest. In order to solve this task we can rely either on a fully or a partially automated approach based on graph-based search methodologies.

To identify each signal path in a fully automated form, we use a path search algorithms that iteratively searches all paths that end at the interface of interest. Such automated search can only be used, if the solution space is limited. This requirement can be fulfilled by constructing the signal flow graph using the dominant-input based approach because it contains a very limited number of possible paths. Additionally fully automated search is practical, if the number of dependencies between the interfaces is low. Else the number of possible search results can be excessive, which would render the results useless.

If the solution space is to large we employ a partially automated approach, where a system engineer interacts with graph search techniques. This interaction is carried out with a graphical tool that we have created for this application. Thereby the system engineer separates the signal flows that are considered as relevant from the other ones. Our methodology uses the tool to backtrack the graph starting from the specified interface. Thereby, the parents of a respective node are identified and shown to the user. These results are analyzed by the system engineer, whereby the signals whose influence can be considered as unimportant, are removed from the solution space. For example when a controller has two inputs, where one is a slowly changing scheduling parameter, then its effect on the signal properties of the control value can be neglected. Once deselected, the node will not be considered in future steps. Next, we search for the parents of the nods in the solution space and add them to our results. These results are then again shown and analyzed. We repeat this process until all relevant paths are identified.

4.3 Transformation of Signal Requirements

In the following we outline the relationship of constraints on signal properties to requirements on conventional real-time properties such as the event rate, and the end-to-end latency which can be modeled and analyzed by existing tooling. We assume that signal requirements are formulated in a bounded form. For the interested reader we provide formal the relationships as supplementary material[2].

At first we discuss the data age requirement. It can be shown, that such a requirement will be satisfied, if we can ensure that the latency stays in certain bounds, which can be computed by subtracting the delays in the path from the requirement. Thus the data age requirement provides an implicit requirement on the end-to-end latency of the respective causal chain. A requirement on the data synchronicity will be satisfied if the relative latency of the corresponding event chains stays inside of certain bounds. We can compute these bounds from the requirement by subtracting the delays in the path. Thus, we derive a requirement on the synchronicity of the respective event-chains. We can also show that a requirement on the logical sampling rate can be satisfied, if the sum of difference of two consecutive latencies and the difference of two consecutive tags stays inside of certain bounds. We therefore require in this case, that sum of these two values must be inside of specified bounds.

When discussing the logical band limit, we first have to acknowledge that the sampling rate can not raise the band limit. Therefore a guaranteed upper bound must be realized determined by the cut-off frequency of filters in the path. Still, we can at least guarantee the lower bound of the band limit. If we ensure the sampling rate is small enough to represent the frequencies. Hereby, we have to require that the logical sampling rate at this interface is smaller or equal than the required band limit. Therefore, the requirement on the band limit can be reformulated into a requirement on the sampling rate.

By definition, the band limit can only be increased without aliasing by a resampling operation. Also we know that the timing behavior can not decrease the band limit. In order to satisfy an aliasing constraint, it must be ensured, that for each read interface in the signal path the sampling rate is not larger than the band limit. It can be shown, that is sufficient to test this requirement for a subset of interfaces, namely input interface, which are linked to a resampling behavior as well as the specified interface itself. In order to respect these constraints, we therefore can formulate requirements on the sampling rate, which then result in the corresponding real-time requirements for the respective interfaces.

4.4 Use Case

In order to demonstrate the applicability of our approach we present an automotive use case that is derived from an existing engine control software. Our use case consists of 13 components that are responsible for the control of the fuel system of a combustion engine (see Fig. 13). It includes a control system similar to the aliasing example including a sensor (Sns), a preprocessing unit (Prep), a control function (Ctl) and an actuator. Additionally, this system contains several models (Mdl) and some monitoring functions (Mon).

[2] http://arxiv.org/abs/1711.09130

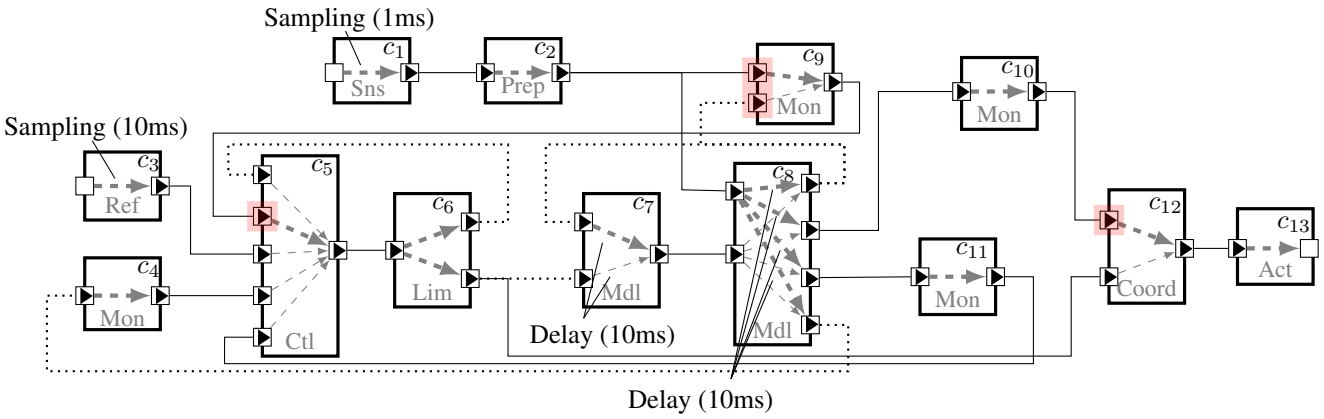

Figure 13: Setup of the example control system.

Type	Interface	Format	Value
log. sampling rate	$\underline{\mathbf{u}}_{(5,2)}$	MIN MAX	8 12
aliasing	$\underline{\mathbf{u}}_{(5,2)}$		False
data sync.	$\underline{\mathbf{u}}_{(9,1)}, \underline{\mathbf{u}}_{(9,2)}$	MAX	2
log. data age	$\underline{\mathbf{u}}_{(12,1)}$	MAX	200

Table 1: Specified requirements on signal properties

For our example we examine each component for their internal behavior. The results are annotated in Fig. 13. We obtain four requirements by analyzing the components individually. The respective interfaces are highlighted in Fig. 13. The first requirement specifies that logical sampling rate at second input of c_5 has to be between $8ms$ and $12ms$. Additionally no aliasing is allowed at that interface. We require a data synchronicity at the first two interfaces of c_9 smaller than 2ms. The last requirement specifies that the logical data age of the first interface of c_{12} must be smaller than $200ms$.

Next, we identify a dominant input for each output interface of the components. We indicate the assignment of these dominant input using bold arrows in Fig. 13. When building the signal flow graph, we do not obtain any cycles so that this approach can be used. This then enables us to identify the signal flows in an automated form. Once we know the signal paths for the specified interfaces, we can evaluate the guarantees in the components and the specified properties and determine system-level timing constraints.

As a result of this process we identify a set of timing requirements. The first condition requires that the sum of the event-rate at the sampling interface of c_1 and the latency jitter of the respective causal chain has to be in the limits of $8ms$ and $12ms$. Assuming a constant event rate of $10ms$ with negligible jitter at the read interface, we specify for the latency jitter of the causal chain to be in between $8ms$ and $12ms$. The aliasing requirement results in a limit on the sum of event-rate and latency jitter to be smaller than $1ms$. Note that the requirements derived from condition (1) and (2) can not be satisfied simultaneously. Further analysis shows, that this is due to a deactivated sensor in c_2. If we reactivate the filter and add a resampling guarantee with a band limit of $10ms$ condition changes to $10ms$. For the third criteria we determine that the latencies of the causal chains must differ by $12ms$ due to the delay in c_8. The same delay also results in a required latency for the causal chain of the first input first of c_{12} to be smaller than $190ms$.

As this small example demonstrates, can our approach be used to derive real time constraints for component based systems from the descriptions of individual components. Such real-time constraints can then be used to apply existing methodologies and tools for the design of the scheduling and mapping of a software. Additionally the methodology can support the detection of false assumptions and design errors in the software of cyber physical systems.

5 Conclusion

In this paper we discuss a process to improve the requirement set of cyber-physical system for temporal properties. We use the temporal semantics model to measure timing behaviors that we call temporal signal properties. Using these properties we annotate component descriptions to disclose temporal requirements that originate from function design. Also, the internal behavior is disclosed in an abstract form. During system design these descriptions can be used to derive system-level timing requirements. The applicability of the approach is shown using a use case. While we outline the general approach, future work will have to demonstrate the specific integration into a domain specific standard, such as AUTOSAR or a more general software engineering standard, such as AADL [22]. Also it would be interesting to study alternative interpretations for intra-component dependencies to simplify the graph-search. While our work has focused on input signals, an extension for output signal could be useful especially when dealing with close loops.

References

[1] AUTOSAR. *Specification of Timing Extensions V2.1.1 (R4.1)*. Mar. 2014.

[2] Remko Baur et al. "Estimation of fuel properties in a common rail injection system by unscented kaiman filtering". In: *2014 IEEE Conference on Control Applications (CCA)*. IEEE. 2014, pp. 2040–2047.

[3] Albert Benveniste et al. *Contracts for system design*. Tech. rep. INRIA, 2012.

[4] Philippe Cuenot et al. "11 the east-adl architecture description language for automotive embedded software". In: *Model-based engineering of embedded real-time systems*. Springer, 2010, pp. 297–307.

[5] Michael Deubzer. "Robust Scheduling of Real-Time Applications on Efficient Embedded Multicore Systems". PhD thesis. Technische Universität München, 2011.

[6] Madeleine Faugere et al. "Marte: Also an uml profile for modeling aadl applications". In: *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*. IEEE. 2007, pp. 359–364.

[7] Marc Förster. *Dependable reuse & guarded integration of automotive software components*. Tech. rep. Forschungsvereinigung Automobiltechnik e. V., 2013.

[8] Rachana Ashok Gupta and Mo-Yuen Chow. "Networked Control System: Overview and Research Trends". In: *IEEE Transactions on Industrial Electronics* 57.7 (July 2010), pp. 2527–2535.

[9] R. Henia et al. "System level performance analysis - the SymTA/S approach". In: *Computers and Digital Techniques, IEE Proceedings* - (2005), pp. 148–166.

[10] J. Kapinski et al. "Simulation-Based Approaches for Verification of Embedded Control Systems: An Overview of Traditional and Advanced Modeling, Testing, and Verification Techniques". In: *IEEE Control Systems* 36.6 (2016), pp. 45–64. ISSN: 1066-033X.

[11] Kay Klobedanz et al. "Timing modeling and analysis for AUTOSAR-based software development-a case study". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*. IEEE. 2010, pp. 642–645.

[12] Hermann Kopetz et al. "Distributed fault-tolerant real-time systems: The Mars approach". In: *IEEE Micro* 9.1 (1989), pp. 25–40.

[13] Steffen Lampke et al. "Resource-Aware Control-Model-Based Co-Engineering of Control Algorithms and Real-Time Systems". In: *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 8.2015-01-0168 (2015), pp. 106–114.

[14] Kim G Larsen, Paul Pettersson, and Wang Yi. "UPPAAL in a nutshell". In: *International Journal on Software Tools for Technology Transfer (STTT)* 1.1 (1997), pp. 134–152.

[15] Farokh Marvasti. *Nonuniform sampling: theory and practice*. Springer Science & Business Media, 2012.

[16] Ljiljana Milic. *Multirate Filtering for Digital Signal Processing: MATLAB Applications: MATLAB Applications*. IGI Global, 2009.

[17] Mischa Moestl and Rolf Ernst. "Handling complex dependencies in system design". In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2016.

[18] Saad Mubeen et al. "End-to-end timing analysis of black-box models in legacy vehicular distributed embedded systems". In: *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE. 2015, pp. 149–158.

[19] Silviu-Iulian Niculescu. *Delay effects on stability: a robust control approach*. Vol. 269. Springer Science & Business Media, 2001.

[20] Marie-Agnès Peraldi-Frati et al. "The TIMMO-2-USE project: Time modeling and analysis to use". In: *ERTS2012 International Congres on Embedded Real Time Software and Systems*. 2012.

[21] Philipp Reinkemeier et al. "Contracts for Schedulability Analysis". In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer. 2015, pp. 270–287.

[22] *SAE-AS5506B: SAE Architecture Analysis and Design Language (AADL)*. Warrendale, PA, USA: International Society of Automotive Engineers, Sept. 2012.

[23] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems". In: *European journal of control* 18.3 (2012), pp. 217–238.

[24] Oliver Scheickl et al. "How timing interfaces in autosar can improve distributed development of real-time software". In: *GI Jahrestagung (2)* (2008), pp. 662–667.

[25] Tobias Sehnke, Matthias Schultalbers, and Rolf Ernst. "Contract-Based Integration of Automotive Control Software". In: *2017 Design, Automation Test in Europe Conference (DATE)*. 2017.

[26] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. "Real-time calculus for scheduling hard real-time systems". In: *ISCAS 2000 Geneva*. Vol. 4. IEEE. 2000, pp. 101–104.

[27] C. Wolff et al. "AMALTHEA Tailoring tools to projects in automotive software development". In: *IDAACS'2015*. July 2015, pp. 515–520.

[28] Lixian Zhang, Huijun Gao, and Okyay Kaynak. "Network-Induced Constraints in Networked Control Systems - A Survey". In: *IEEE Transactions on Industrial Informatics* 9 (2013), pp. 403–416.