Design by Uncertainty: Towards the Use of Measurement Uncertainty in Real-Time Systems

Peter Ulbrich, Florian Franzmann, Fabian Scheler,
Wolfgang Schröder-Preikschat
Chair in Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg
Erlangen, Germany
{ulbrich, franzman, scheler, wosch}@cs.fau.de

***Abstract*—Real-time systems usually incorporate a wide variety of challenges: A control engineer, for example, aims for the highest possible control quality achievable. Here, one key element is to minimise the *uncertainty of the measurements*. This, to put it simple, is the noise of sensor data, which has a negative effect on control. Although measurement uncertainty is well treated in control theory, it is usually ignored in common real-time architectures where temporal properties are the prevalent criteria. Consequently, the communication between control engineers and real-time specialists is rather one way, revolving around deadlines and sampling periods. However, on closer examination, many real-time properties are derived from the measurement uncertainty aspired by the control engineer. Conversely, temporal variations, virtually inevitable in practice, can be represented as measurement uncertainty as well.**

Tackling the measurement uncertainty should therefore be an interdisciplinary task: The control system respects the actual run-time conditions instead of estimating them. Likewise, the real-time system considers measurement uncertainty rather than blindly sticking to deadlines. In this paper we present an uncertainty-centric approach to leverage measurement uncertainty in real-time architectures, not only at design time but also at run-time. Using measurement uncertainty as an explicit interface minimises the gap between real-time specialists and control engineers and facilitates a modular and flexible system design. Our preliminary results are promising and show the ease of use and the applicability to existing systems.

***Keywords*-Measurement Uncertainty, Software Architecture, Covariance, Scheduling, Frameworks, Operating Systems**

I. Introduction

Nowadays digital controllers are almost omnipresent in embedded cyber-physical systems and control theory is a well-known and well-understood engineering discipline. Although applications employing digital controllers are structured in a relatively simple manner – sample, compute, and act – their implementation is anything but easy. Traditional control theory, which still is the predominant paradigm for many controllers, assumes that all inputs are sampled instantaneously and equidistantly, providing a temporally consistent snapshot of the physical environment [1]. It is, however, almost impossible to actually meet this requirement in practical implementations of digital control loops. One reason is the non-zero execution time in computing systems. Thus, two sensors attached to the same computing node cannot be sampled at the same instant but have to be queried sequentially.

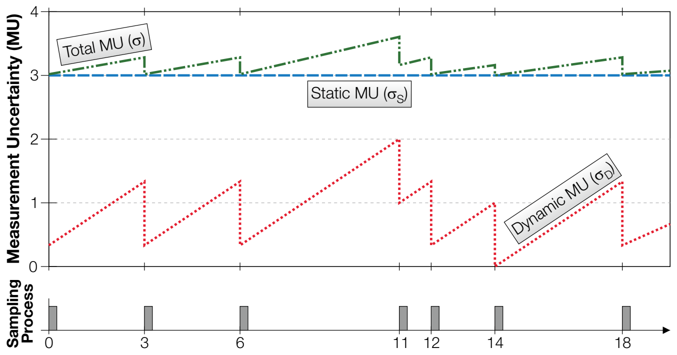

Fig. 1: The total MU consists of SMU and DMU. The former usually is constant and set at design time. The latter results from run-time behaviour and varies with the sampling instants.

The lack in temporal precision results in dephased measurements, which suffer from run-time jitter and impair the quality of the measured values. This degradation manifests itself as noise [2] and is often called *measurement uncertainty* (MU). Computer science and control theory already came up with a number of ways to tackle that issue. On the one hand, computer science tries to optimize the temporal behaviour by ensuring equidistant sampling to minimise run-time jitter and, thereby, reduce MU to a manageable level. On the other hand, control engineers use Kalman filters [3], which are somewhat robust w. r. t. jitter. While each of these approaches is valid, we think that they are imperfect because they lack an interface that connects computer science and control engineering. Current solutions are tailored to individual problems, are monolithic and not reusable.

We believe that two components of MU – as illustrated in Fig. 1 – should be distinguished: The first component is known a-priori and is determined by properties of the sensor, the targeted environment and by the sampling rates chosen, and thus can be handled at design time. Typical examples are dephased sampling instants due to different sampling rates. We refer to this component as *static measurement uncertainty* (SMU). The second component is not known a-priori and can only be accounted for at run-time, thus, we call it *dynamic*

This work was partly supported by the Bavarian Ministry of State for Economics, Traffic and Technology under the (EU EFRE funds) grant no. 0704/883 25 and the German Research Foundation (DFG) under grant no. SCHR 603/9-1.

measurement uncertainty (DMU). In this paper, we regard run-time jitter as primary source for DMU.

Current control algorithm designs do not distinguish SMU from DMU explicitly, endorsing rather complex digital control loops dealing with SMU as well as with DMU. We, however, believe that using e. g., the Kalman filter is just one solution to cope with DMU. Strictly speaking, Elmenreich's approach [4] to eliminate execution-time jitter by choosing a time-triggered execution environment is another, albeit indirect one.

Since DMU seems to be a common ground for computer scientists and control engineers, we propose a) to explicitly distinguish SMU and DMU as opposed to the 'combined' MU that is used in current control algorithms and b) an architecture that explicitly makes the DMU as well as SMU accessible.

II. Related Work

Since MU is not a novel problem in *digital signal processing* (DSP) and control theory, several methods have been developed to handle it.

A well known technique making use of SMU is sensor fusion. Its purpose is to combine measurements of diverse sources or of different sampling in a consistent way. Since sensor fusion is important in a wide variety of applications, the topic is well treated in literature and a number of sensor fusion models have been developed to accommodate the problem.

Approaches comparable to ours are the sensor fusion architectures proposed by Elmenreich [5], Hightower et al. [6] and Zug et al. [7]. Elmenreich's approach was developed in the context of the *Time-Triggered Architecture* [8] and does not take jitter into account.

Hightower's approach is inspired by the ISO/OSI model [9] and divides the DSP system into multiple layers (sensors, measurements, fusion, ...). While this is helpful w. r. t. modularization, the approach does not target real-time systems, rendering it inappropriate for our purposes. Zug et al. focus on fault detection in distributed sensor systems. Their architecture provides building blocks for reliable communicating through messages but does not regard jitter either.

Our approach goes beyond the presented architectures by making the DMU explicitly available, promoting it to an interface between DSP and control theory. This is necessary, as we target event-driven real-time systems where execution-time jitter is an intrinsic problem.

III. Approach

We aim for modularisation of the *process control system* (PCS) and consequently our approach is based on the separation of concerns: each component should be implemented by the respective domain expert. Using measurement uncertainty as interface makes this feasible. Therefore, we propose an MU-aware system architecture that: a) establishes the MU as an interface, b) makes its individual parts (SMU and DMU) available, c) integrates well into existing systems, and d) is lightweight in terms of resource requirements. In the remainder of this section, we outline our architectural approach and its basic components and discuss the resulting advantages.

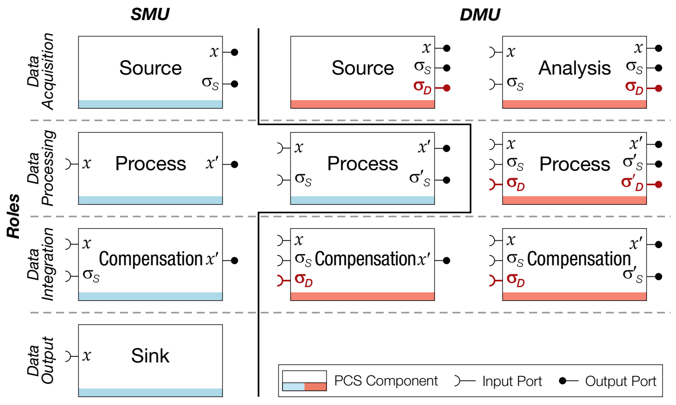

Fig. 2: Roles and components of the uncertainty-aware real-time architecture.

A. Architectural Concepts

As already mentioned, the basic structure of a PCS is quite simple: sample, compute, and act. The necessary building blocks can be categorised into four dedicated roles. Besides the obvious *Data Acquisition*, *Processing*, and *Output* roles, *Data Integration* is necessary to remove the MU where it is no longer relevant or acceptable. Data flows can be described as a sequence of basic components that implement these roles: a measurement originates from Data Acquisition (e. g.,, sensor). Subsequently, the measurement endures a number of Data Processing steps (e. g., filter, fusion). Finally, the measurement is fed into Data Integration, which eliminates the MU and passes the result to Data Output (e. g., actuator).

Fig. 2 gives an overview of the resulting architecture: orthogonal to the roles, are the components which implement the architecture. Depending on the interface, the components are further subdivided on the basis of support for *static* and *dynamic uncertainty*. In the following, we detail the individual components, with respect to the SMU and DMU specifics.

1) Data Acquisition: This is the origin of each data flow (x), and therefore, the only source for both SMU (σ_s) and DMU (σ_d). Any subsequent stage only adjusts or compensates MU. To enrich the PCS with run-time information, the architecture introduces a DMU-aware *Source*.

Certainly, the *Analysis* component is an exception. It is used if the necessary temporal properties are indeterminable during the execution of the *Source* component. Subsequently, *Analysis* obtains temporal information, which it converts to DMU. Typical examples are aperiodic sources such as remote sensor nodes, where temporal correlations emerge throughout the communication.

2) Data Processing: DSP steps, such as filtering, fusion, and observers, are encapsulated by Processing components. Hence, they only adjust MUs and do not add new information.

3) Data Integration: The Data Integration role corresponds to the control part (but may also include fusion or observer) of the process control system. From this point on, the measurement uncertainty is no longer needed and therefore compensated.

One component deviates from the others: as an inverse operation to *Analysis*, a third *Compensation* component only compensates for dynamic uncertainty. This way, even DMU-unaware controllers can benefit from DMU compensation.

4) Data Output: finally, Data Output is the sink of a PCS.

B. Uncertainty as an Interface

Using MU as an interface links the design-time and the run-time aspects of a PCS – thus ultimately closing the gap between control engineer and computer scientist.

At design-time, the SMU is used to describe the controller requirements on the one hand and the capabilities of DSP on the other hand. In a nutshell, the control engineer expresses his expectations of the measurement quality. If everything fits, the processing chain is dimensioned and arranged accordingly. The interface supports this by providing SMU information at inputs and outputs where it is used in the same way to describe requirements and capabilities. Hence, each component at design time is a *static component*.

At run-time any temporal deviations, for example sampling jitter, jeopardize the compliance with the design. Executed components that are subject to run-time deviations provide the DMU at their interface, thus, facilitating its specific handling either by means of DSP or scheduling.

C. Modularisation

For many reasons, the PCS and the filters in particular are, as mentioned earlier, often monolithic: control engineers usually work in terms of mathematical models where modularization is usually not an issue. Likewise, the uncertainty information necessary for the filter design is derived from all sensors leading to a global covariance matrix. However, this approach considerably limits the options of the real-time system, for example to prioritize jitter sensitive parts of the PCS at run-time. Our approach simplifies the modularisation of such PCSs by leveraging its uncertainty-centric interface: for independent sensors, the covariance equals the MU. Thus, the individual modules can be derived directly from the matrix. Moreover, this eliminates the cumbersome derivation of individual temporal properties as well.

IV. Implementation and Evaluation

In order to demonstrate the usefulness of our approach, we decided to implement it prototypically in the *I4Copter*'s [10] flight control system. The *I4Copter* is a quadcopter and therefore intrinsically unstable, requiring a fairly complex controller to stabilize its attitude. Most of the code that was already available in an earlier version of the *I4Copter* could be reused, thus showing that our approach acts as a thin wrapper. The transition proved to be straightforward. Furthermore we conducted smaller and more controlled experiments which clearly showed the validity of our method. We implemented two artificial signal sources that generated the same signal. Then we subjected one of the two sources to increasing levels of jitter, while the other remained jitter-free. The two signals were then fused using the MUs as weights. When we compensated for jitter, the mean square error was bounded, while it increased without limit if we did not compensate for jitter (see Fig. 3).

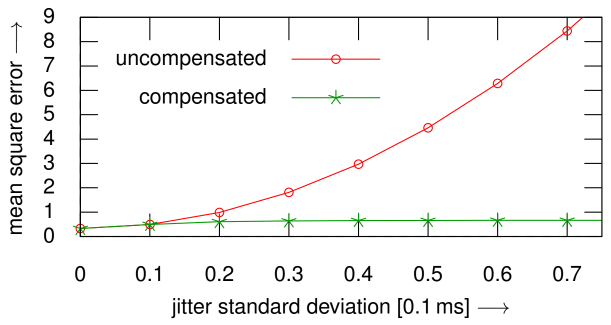

Fig. 3: Jitter experiment: The curves compare the mean squared error if the DMU is compensated or not during sensor fusion.

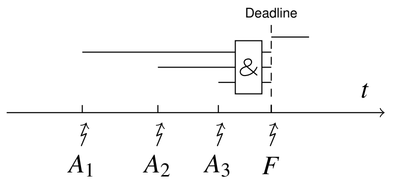

Fig. 4: Bounding the measurement uncertainty via deadlines

In the rest of this section we will focus on the way we compensate for jitter: Walden [11] introduces a straightforward way of calculating the DMU σ'_d from the maximum amplitude A, the sampling interval T and the jitter's standard deviation σ_τ:

$$\sigma'_\mathrm{d} = \frac{\pi \cdot A \cdot \sigma_\tau}{2T}$$

This estimate, however, seems overly pessimistic. In our opinion it is sufficient if only those spectral components are taken into account, which are below the cut-off frequency of the filters that are involved. The DMU then becomes $\sigma_\mathrm{d} = \pi \cdot f_\mathrm{c} \cdot A \cdot \sigma_\tau$ and the combined MU σ' can be calculated from the DMU and SMU σ_s as $\sigma' = \sqrt{\sigma_\mathrm{s}^2 + \sigma_\mathrm{d}^2}$. We use this relationship to prefer more reliable over less reliable values during fusion.

V. Outlook and Future Work

Promoting the SMU and the DMU to an interface opens some interesting opportunities. Here, we briefly discuss two ways to benefit from that additional knowledge by exploiting the SMU in an integrated toolchain and feeding it into a scheduling service in order to optimise the DMU at run-time.

A. Integrated Toolchain

Specifying the SMU and the DMU within the design tools commonly used by control engineers, such as MATLAB/Simulink, is the first step towards an integrated tool support. The toolchain could subsequently exploit the MU information to automatically arrange sampling instants and vary the sampling rates of the different sensors. Likewise, filters or fusion algorithms could be parametrized automatically according to the acceptable MU. This would ultimately release the control engineer from cumbersome manual mapping of MU to deadlines. In addition, the control engineer can even leverage the real-time system feedback (in terms of DMU), for example, in future filter, observer, and controller designs.

In the *I4Copter* we already annotated sensors and the controller with the metadata necessary to derive the PCS. These annotations are evaluated automatically for each sensor and the achievable MU and value range is computed and the sensor fusion is parametrised. Currently, the main purpose of these tools is to check if the achievable values match those specified by the control engineer.

In the future we want to extend our tool-support in two directions. On the one hand, we want to integrate these annotations directly into MATLAB/Simulink-models. Hence, the control engineer does not have to switch to an unfamiliar tool. On the other hand, we want to automate the mapping of driver sampling to threads and the arrangement of sampling instants through use of the *Real-Time Systems Compiler* (RTSC) [12]. The RTSC is a compiler-based tool that arranges and manipulates control flows (i. e., threads and interrupt service routines) through the control flow graphs.

B. Uncertainty-aware System Software

In event-triggered systems, MU is controlled mainly by ensuring dense *artificial deadlines* for real-time jobs sampling sensor values. As long as these deadlines are met, the overall MU certainly stays below a threshold. The aim of these deadlines is to keep the sampling jitter reasonably small, thereby limiting the DMU as much as necessary. This situation is illustrated in Fig. 4: sampling jobs $A_{1..3}$ have to be finished before sensor fusion job F.

Handling the DMU explicitly, however, could significantly relax the situation if this information is made available to the underlying real-time operating system. The primary goal of the scheduler in such a system would no longer be to meet deadlines but to keep the DMU below a given threshold, thereby maintaining a suitable *quality of control*. One possible way is to speed up or delay threads having great or low influence on the DMU by adapting their priority similar to a priority exchange [13] or sporadic server [14]. Obviously, just providing the scheduler with the DMU is not sufficient for this purpose. The scheduler also needs to predict how much the DMU increases if a sensor is sampled late or even dropped and how much of this DMU could be compensated for by the following sensor fusion and control algorithms.

Such a scheduler, however, would simplify the design and implementation of PCSs significantly as the maximum acceptable measurement uncertainty no longer has to be mapped to the mentioned *artificial deadlines*. Thus, the MU would be the major measure for the quality of control.

VI. Discussion and Conclusion

In this paper we presented an integrative approach for the development of embedded real-time control systems. We aim to bridge the semantic gap between the main domains involved: Control Engineering and Computer Science. Therefore, we based our approach on measurement uncertainty (MU) as a common interface to establish a two-way communication between control and system. Accordingly, we explicitly distinguish between the static and the dynamic measurement uncertainty (DMU), and thus between design-time and runtime. We implemented our approach by an uncertainty-aware real-time architecture that leverages the interface to cope with the DMU and to escape the bonds of bearish deadlines.

Why, however, should a computer scientist switch from temporal properties to measurement uncertainty? First of all, it replaces the inflexible deadlines as the primary means of communication with the control engineer. This might seem insignificant for individual sensors, where the DMU can be directly mapped to deadlines. However, it may turn out to be a powerful approach when considering more complex sensor systems. Since the DMU reflects the actual effects of temporal variance within the processing chain, the individual deadlines, and thus the whole system design, become more flexible. To benefit from this flexibility, uncertainty-aware system software, as sketched in Section V, is necessary in addition to an uncertainty-aware architecture.

When using our approach, apart from the modified communication interface, the control engineer stays in his natural habitat. Explicitly dealing with temporal properties is no longer necessary. Instead, ey can decide whether the real-time system should eliminate the DMU or to design the PCS accordingly. In addition, this opens up the chance to explicitly consider the real-time system feedback (in terms of DMU), for example, in future filter, observer, and controller designs.

References

[1] M. Törngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems Journal*, vol. 14, pp. 219–250, May 1998.

[2] S. G. Rabinovich, *Measurement Errors and Uncertainties: Theory and Practice*, softcover reprint of hardcover 3rd ed. Springer, 10 2010.

[3] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[4] W. Elmenreich, "Sensor fusion in time-triggered systems," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2002.

[5] W. Elmenreich and S. Pitzek, "The time-triggered sensor fusion model," in IEEE *Int. Conf. on Intelligent Engineering Systems*, 2001.

[6] J. Hightower, B. Brumitt, and G. Borriello, "The location stack: A layered model for location in ubiquitous computing," in *IEEE W'shop on Mobile Computing Systems and Applications*. IEEE, 2002, pp. 22–28.

[7] S. Zug, A. Dietrich, and J. Kaiser, "An architecture for a dependable distributed sensor system," IEEE *Trans. on Instrumentation and Measurement*, vol. 60 Issue 2, pp. 408–419, 2011.

[8] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[9] International Organization For Standardization, "ISO/IEC 7498-1:1994," p. 59, 1996. [Online]. Available: http://standards.iso.org

[10] P. Ulbrich, R. Kapitza, C. Harkort, R. Schmid, and W. Schröder-Preikschat, "I4Copter: An adaptable and modular quadrotor platform," in *26th ACM Symp. on Applied computing (SAC '11)*. New York, NY, USA: ACM, 2011, pp. 380–396.

[11] R. H. Walden, "Analog-to-digital converter survey and analysis," IEEE *J. on Selected Areas in Communications*, vol. 17, pp. 539–550, 1999.

[12] F. Scheler and W. Schröder-Preikschat, "The real-time systems compiler: migrating event-triggered systems to time-triggered systems," *Softw. Pract. Exper.*, vol. 41, no. 12, pp. 1491–1515, 2011.

[13] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," in *8th Int. Conf. on Real-Time Systems (RTSS '87)*. Washington, DC, USA: IEEE, Dec. 1987, pp. 261–270.

[14] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *Real-Time Systems Journal*, vol. 1, no. 1, pp. 27–60, 1989.