Safe control for systems with value- and time-dependant deviations

Dieter Zöbel * Christian Weyand ** Christian Schwarz ***

* University Koblenz-Landau(e-mail: zoebel@uni-koblenz.de).
** University Koblenz-Landau(e-mail: weyande@uni-koblenz.de).
*** University Koblenz-Landau(e-mail: chrschwarz@uni-koblenz.de).

Abstract: Real-time scheduling is both based on a broad theoretical background and available through a multitude of tools and infrastructures. The central input parameters to this discipline are the demand for execution time and the real-time conditions given as deadlines or periods. The former has attracted a lot of research efforts, mainly in the scope of worst case execution time analysis. However, the latter did not gain the same degree of attention up to now. The derivation of real-time conditions is deeply application dependent taking into account the technical facilities as well as the specified requirements. Despite this heterogeneity a canonical process model is proposed to verify program code against specified requirements. The corresponding approach is based upon a dedicated calculus to backtrack and propagate the maximal and minimal imprecisions in time and value of the data available to the instant of executing the canonical processes.

Keywords: Real-time systems, technical processes, consistency, control systems, precision

1. INTRODUCTION

In various of our projects concerning the design and implementation of embedded control systems has been a remarkable experience that it is not straightforward at all to derive the application specific real-time conditions. This is particularly pitiful in the context of real-time education where we intend to present a well-structured and self-contained approach to software-hardware co-design.

The theory of real-time scheduling is strongly process-oriented and presumes the knowledge of various process-dependant parameters. The most important ones are the demand for execution time by a given process and the temporal constraints on the starting and completion times of this process. The former is typically derived from experimental measurements or more sophisticated - from a worst case execution time analysis of the program code executed by this process. A lot of research efforts have been spent to estimate the demand for execution time and several tools to perform this calculation for a given hardware are available today.

In contrast it seems to us that the question of finding the time bounds in form of deadlines or periods (in Xiong et al. (2008) called the period assignment problem) has not attracted the same degree of attention. The reason may be that real-time conditions for processes are not unique by themselves and depend upon a heterogeneity of criteria, e.g.:

- The behavior of the technical system which has to be controlled.
- The temporal constraints emerging from the sensor and actuator systems to measure and feed back to the technical system.
- The timely impacts introduced by the computational system and the scheduling policy on the execution of processes.
- The procedure to derive the real-time requirements which have to be satisfied for the technical system and which have to be established by the computational system.

In our laboratory vehicles for exercising autonomous driving (automation with series trucks Zöbel et al. (2005), see figure 1) but also in our driver assistance system (e.g. driver assistance for backing up cars with trailers Bal ter et al. (2004)) more efforts had to be spent to the items listed above than to those topics which have a high degree of presence in scientific literature e.g. worst case execution time analysis or real-time scheduling.

A basic attempt towards the formulation of real-time conditions has been adopted from the modelling techniques applied for database systems. Instead of the value based definition of consistency, real-time applications require some temporal version of consistency, accentuating that real-time processes operate on data objects that are steadily aging (see Audsley et al. (1992) and Raman ritham (1993)). The two decisive definitions – absolute and relative temporal consistency – bound the absolute and relative time since the data has been taken from the real world.

A generalization of this approach to determine real-time conditions distinguishes between the technical system, represented in terms of real-time entities, and its observation, namely real-time images Kopetz (1998). A relation, called...
temporal accuracy, is defined for assigning the real-time image to some real-time entity within a bound history. Based on this knowledge the worst case error when utilizing this real-time image is estimated and can be taken into account for decisions which have to be made by the real-time process.

A more recent approach distinguishes between how fast a computation responds and how fresh the sensed data is (see Kao et al. (2003)). With the knowledge of the current time, the timestamps for the points of time, when a certain data item is read, and given consistency time-bounds the terms absolute and relative consistency are established for continuous and discrete data items.

Our article may be understood as an attempt to continue the approaches cited above. The reason to do so is at least threefold in that we try to overcome shortcomings and extend these approaches to achieve a surplus in the integration into the software development cycle. First, let us have a look at the shortcomings that we feel:

- The knowledge of some current time \( t \) is assumed. However, the execution of processes is prone to jitter. This holds for the starting (and completion) times of subsequent process executions, but even more subtle also for subsequent statements within the same execution of a process. So, the utmost knowledge we assume here is the duration of the time interval where the statements of the process are executed sequentially, but potentially interrupted at any instant of time.
- Only the history up to some current time \( t \) is considered. However, the process controlling the technical system has to make decisions which have consequences for the future. This has to be anticipated with some bound degree of deviation and feedback to the technical system. The methods to be used here are analogous to those used for the history.
- The approaches concentrate on inaccuracies in time. However, the inaccuracy of the values must not be neglected. Furthermore, it has to be considered that there can be a tradeoff between the inaccuracies of time and value.

Besides these shortcomings and before proposing our approach we take a brief look at the development of time-critical software from the viewpoint of a programmer. Typically this viewpoint is program-centric in that the values of variables are processed and evaluated for decision making. Particular in the scope of embedded systems several questions emerge from this view and unsettle the programmer:

- How precise is the value of a variable in correlation to the technical system?
- From which instant of time with respect to the technical system does the value of a variable stem?
- At which instant of time a decision will be made by the program in execution?
- At which instant of time will the decision made by the program take effect in the technical system.

Program code written under these circumstances is aggregated to processes which build up the computational part of the embedded system. These processes are executed concurrently following some real-time scheduling policy. Even though there is a profound theory of scheduling behind, the question remains how to extract the application-specific time bounds in order to apply this theory.

Summing up, in this paper we propose a methodology of engineering real-time applications by combining a structured approach for software development with selected but well-understood disciplines of real-time scheduling. The rest of the article is structured as follows: Section 2 introduces a basic model for real-time control applications built upon canonically structured processes. Of central importance is section 3 where a calculus for limiting deviation in value and time is introduced. This calculus is elaborated into structured sequence of operative steps in section 4. Then, in section 5 the approach is applied in the context of autonomous driver thereby deriving safety conditions for a particularly risky driving maneuver. The paper closes with a summary of results and an outlook to the possibilities for an appropriate tool support.

2. THE BASIC MODEL FOR EMBEDDED CONTROL

The majority of real-time applications center around the control of at least one technical system. The control itself is established by a computational system which gathers information about the technical system, condenses this information in order to assess the status which the technical system is in. Then, depending upon this assessment control decisions have to be taken and corresponding actions have to be initiated. The decision and initiation is henceforth called control action CA.

The control action is piece of program code. So, the question arises on what a software developer can trust on when referring to the variables which are to represent the status of the technical system at the instant of execution. For an accurate consideration we introduce a basic model of control (see figure 2) which distinguishes between the set of value domains \( V = \{ V_1, ..., V_n \} \) describing the technical system and the set \( OV = \{ OV_1, ..., OV_n \} \) of domains containing the observed images of these values. By this model a sensor relates some physical entity \( x \) between the technical system and the computational system:

\[
SR_x \subseteq V_x \times OV_x
\]

Analogously an actuator relates some physical entity \( y \) between the computational system and the technical system:

\[
AR_y \subseteq OV_y \times V_y
\]

Imposing certain rules on the program structure is a common means to reduce complexity. This also holds for the
Fig. 2. Basic model of an embedded control system
design and implementation of real-time systems (e.g. Xu (2003)). Furthermore canonical program structures constitute adequate textual fragments to be used by generic development tools (which actually is out of the scope of this paper). Bound to these deliberations and favoring time triggered policies of scheduling we propose (similar to Lundberg (2002)) the following canonical code fragment for preemptive real-time processes:

```c
while (true){
    wait_until(t);
    if (C(OV))/\n        OVx=sensor.x.get(); // 1
    if (C(OVz)) OVy=A(OVx,OVy); // 2
    actuator.y.set(OVy); // 3
    t=t+period; // 4
}
```

In terms of the basic model the principal purpose of the computational system to assert that the technical system behaves as specified. A simple but effective means to express the desired behavior of the technical system are predicates formulated in terms of the technical system: \( I_{RT}(V) \). In contrast the program is a formal expression entirely based upon variables of \( OV \) (see e.g. condition \( C \) of the code fragment in line 3).

### 3. LIMITING IMPRECISIONS IN VALUE AND TIME

The basic model above consists of a sensor input, a computational system and an actuator output. The imprecisions which are caused by this model have to be limited in a stepwise fashion. So first, we consider merely the value specific imprecisions introduced by the sensor input. In general for a sensor it has to be taken into account that different values \( v_1 \) and \( v_2 \) of the technical system may be mapped to the same value \( ov_2 \) of the computational system. On the other hand it may be that the same physical value \( v_2 \) is once mapped to \( ov_1 \) and at some other instant of time to \( ov_2 \). Effective control can only be applied if there is some knowledge about the limits of such imprecisions of the sensor, or in other terms, about the relation \( SR_x \):

- For some set of observed values \( M \subset OV_x \) it should be known which physical values may have caused them via the sensor:

\[
DOM(SR_x, M) = \{v_x | (v_x, ov_x) \in SR_x \land ov_x \in M\}
\]

- For some set of physical values \( M \subset V_x \) it should be known which observed values \( v_y \) can cause via the sensor:

\[
IMG(SR_x, M) = \{ov_x | (v_x, ov_x) \in SR_x \land v_x \in M\}
\]

As a further step in limiting the imprecisions caused by the sensor, time-dependent effects have to be taken into account. So, it has to be noticed that sensing itself consumes time, and also the computational process until reaching the control action \( CA \). On the other hand the technical system has its own dynamics and evolves further at the same time. The phenomenon that a value which has been sensed is permanently prone to change is formally captured by the term that \( v_x \in V_x \) is \textit{ perishable} (Audsley et al. (1992)) or a \textit{ current picture of an RT entity} (Kopetz (1997)). For the quantification of this phenomenon the authors above introduced two definitions of temporal consistency. One definition, the absolute temporal consistency \( \Delta t_{act_x} \) which applies to our considerations is a quality measure for the value \( ov_x \in OV_x \). Put into our formalism \( ov_x \) has the absolute temporal consistency \( \Delta t_{act_x} \), if in a certain history \([t-\Delta t_{act_x}, t]\) there has been some \( v_x \in V_x \) with:

\[
(v_x, ov_x) \in SR_x
\]

With respect to our experiences with real-time applications and their implementations the approaches above have several drawbacks (see Zöbel (2005)). Hence, a more sophisticated description of the behavior of the control system is up to this point of discussion restricted to the left half (see figure 2) – is now introduced. In order to be applicable to any time-triggered runtime system of preemptive and periodic processes we assume the knowledge of:

- (for relation \( SR_x \)) \( \Delta t_{sl} \), the minimum and \( \Delta t_{sh} \), maximum age of the data accessed by operation \( sensor.x.get() \).
- The period \( \Delta t_i \) of process \( i \) sensing \( v_x \) overwriting the old value of \( ov_x \) and evaluating condition \( C(OVx) \).
- The technical system evolves over time intervals \( \Delta t \in \Delta T \). This has to be captured by the following mapping \( TS \):

\[
TS_x: 2^{V_x} \times \Delta T \rightarrow 2^{V_x}
\]

Applying these definitions we are able to estimate which set of values \( SSV_x \subset V_x \) of the technical system a certain value \( ov_x \) used in condition \( C(OVx) \) stands for. First, the deviating effects of the sensor have to be considered, and second, all possible evolutions of the technical system, so that we calculate \( up \) to the time of executing the control action \( CA \) for \( SSV_x \):

\[
\bigcup_{\Delta t \in [\Delta t_{sl}, \Delta t_{sh} + \Delta t_i]} TS_i(DOM(SR_x, \{ov_x\}), \Delta t)
\]

So far, a condition \( C(OVx) \) which is true for some value \( ov_x \) must satisfy \( I_{RT} \) for any value \( v_x \in SSV_x \). This mode of consideration is the consequent extension and precision of former approaches (e.g. Audsley et al. (1992) and Kopetz (1997)). However, it is not complete yet, for it does not consider the evolution of the technical system from the action \( A \) which has been taken by setting the value of \( ov_y \) in the program fragment \( up \) to the time when the value

\[
\]
Fig. 3. A condition which changes immediately after the sensing operation will influence the actuator not before the next period.

\(v_y\) is fed back to the technical system and influences the sensed value \(v_x\). Thereby value based deviations have to taken into account once again. So, a formula for the left and right half of the basic model has to consider:

- (a) the value dependent deviations of the sensor
- (b) the age of the sensed values
- (c) the jitter in the response time of process \(i\) executing \(CA\)
- (d) the time elapsed until the actuator influences the technical system
- (e) the value dependent deviations of the actuator

First, notice that the validity of action \(A\) is not checked. This is the task of control theory. However, what we do here is to specify the duration of time for which the validity of condition \(C\) of the program fragment corresponds to the validity of predicate \(I_{RT}\). To capture this, analogously to the left part of the control system the operation \(\text{actuator}_y\text{.set}(0V_y)\) needs some minimum minimum time \(\Delta h_y\) and some maximum time \(\Delta h_{y'}\) until the action \(A\) really gets influence on the system.

Second, when looking forward from the instant of time when evaluating condition \(C\), this decision has to be valid until it is executed once again in the next period (see items 3) and 4) of the enumeration above). Hence, the period \(\Delta e_i\) has to be considered twice within the calculation of the time interval for which condition \(C\) should hold.

Third, for the calculation of the minimum time for the control loop not the execution time \(\Delta e_i\) of process \(i\) is relevant, but the reduced execution time \(\Delta e_i\), starting with overwriting \(0V_x\) and ending with setting a new actuator value \(\text{actuator}_y\text{.set}(0V_y)\).

As depicted in figure 3 there is an enormous jitter between the lower and upper bounds for the \(CA\)-response. So, finally the bounds of the interval are 1:

\[
\begin{align*}
\Delta l_{xy} &= \Delta s_{lx} + \Delta e_x + \Delta a_{y} \\
\Delta h_{xy} &= \Delta s_{hx} + 2 \Delta p_i + \Delta a_{y}
\end{align*}
\]

Now, taking the whole control system into account, a certain value \(ov_x\) used in condition \(C(0V_x)\) induces an action \(A(0V_x)\) which finally effects the technical system. The entity \(x\) meanwhile has adopted some value in \(SSV_x \subset V_x\):

\[
SSV_x = \bigcup_{\Delta \tau \in [\Delta l_{xy}, \Delta h_{xy}]} TS_x(DOM(SR_x, \{ov_x\}), \Delta \tau)
\]

In this context correctness of \(C(0V_x)\) is equivalent to that any value \(v_x \in SSV_x\) satisfies \(I_{RT}\). Furthermore, any \(y \in IMG(AR_y, SSV_x)\) has to be correct in the sense of control.

\(1\) At a closer look for \(\Delta h_{xy}\) a slightly less pessimistic equation calculation is possible. Instead of two periods the longest time of reaction is bound by \(2 \Delta p_i - (\Delta e_i - \Delta r_e)\).

4. DERIVATION OF THE CONTROL ACTION

Unfortunately this is not the operational structure which is needed from the viewpoint of program development. Here typically the requirements in terms of \(I_{RT}\) are given and the control action, particularly condition \(C(0V_x)\) has to be coded. Starting with predicate \(I_{RT}\) we derive \(C(0V_x)\) by applying the formula for \(SSV_x\) step by step in reverse order:

1. \(I_{RT}\) holds for the technical system at any instant \(\tau\) of time.
2. At some instant \(\tau - t\), \(t \in [\Delta l_{xy}, \Delta h_{xy}]\), of time we get the new predicate \(I\) by replacing any value \(v_x\) in \(I_{RT}\) by \(v' \in V_x\) with \(V' = \{v'_x | TS_x(v'_x, i) = v_x\}\), thereby:

\[
I = \bigcup_{v'_x \in V'} (I_{RT})_{v'_x}
\]

3. Predicate \(I\) incorporating all time dependencies of the technical system is transformed into a predicate \(C\) of the computational system by incorporating all value dependent imprecisions due to the sensor. Therefore any \(v' \in V'\) in \(I\) is replaced by a value \(ov_x \in OV_x\) which is the sensor image of \(v'_x\):

\[
C = \bigcup_{ov_x \in IMG(SR_x, \{v'_x\})} I_{ov_x}
\]

4. For \(C\) is already a predicate in terms of \(OV_x\) it has to be transformed into the program code condition \(C(0V_x)\).

5. AUTONOMOUS DRIVING CASE STUDY

A challenging case study to guarantee for high precision in presence of deviations in value and time may be seen in maneuvers for automated guided commercial vehicles. In our special case a trailer of an articulated series truck (model) has to maneuvered under a container swap body (see figure 4). At a first glance the criterion of safety is clear: Collisions must be avoided.

First of all the relevant parameters regarding the technical and computational system have to be collected and ordered corresponding to the enumeration (a) to (e) in section 3. Second, the safety criterion has to be defined in terms of an invariant and third, the relevant control actions have to be identified in the program code.

Representative for the sensor-actuator interface we focus on the laser scanner which  mounted on the truck and periodically provides its position and orientation computed.
by a triangulation algorithm based on at least three visible beacons. From the white papers of the producer but also from our series of experiments we know that deviations in value are less than ±5 mm in x- and y-direction and less than ±0.5° in orientation. However, this only will be true if the laserscanner does not move. In the case of motion the triangulation inherently causes deviations which are complicated to estimate ².

Hence, the velocity of the vehicle is a prominent scaling factor for deviations. More by experiment than by mathematical analysis we found out that at a speed of up to 30 mm/s the absolute deviation in position is less than 10 mm and in direction less than 1°. This tradeoff can be converted into the strategy that just before and during the tunneling of the swap body the vehicle drives at its minimal speed which is really somewhere between 25 mm/s and 30 mm/s.

Let us assume that the lat-direction is lateral to the vehicle and the long-direction longitudinal to the vehicle. When driving straight backward the lateral velocity is small, always less than ±1 mm. Notwithstanding these slight deviations may cause a collision with the stands of the swap body.

Even at the minimum velocity the progress of the trailer is reasonably large with respect to the value and time dependant deviations. Put more precisely the beacons are scanned 10 times a second. After that the laserscanner packs a telegram with all angles and sends it to the computational system where it is available between 10 ms and 50 ms later. The routine similar to the code fragment above (section 2) which is executed as a periodic process reads the telegram and then computes the position and the orientation of the vehicle. Depending on this data several control actions are performed. One of them is dedicated to apply an emergency brake to avoid a collision with a stand.

The computation of position and orientation lasts at least 20 ms and is performed in every period which is executed every 60 ms. If the control action decides for an emergency brake it still lasts another 50 ms to 90 ms to stop the trailer. Thereby we are able to bound the time span $\Delta \tau$ of the intermediate proceeding technical system:

\[ \Delta l_{xy} = 10 \text{ms} + 20 \text{ms} + 50 \text{ms} \]
\[ \Delta h_{xy} = 150 \text{ms} + 120 \text{ms} + 90 \text{ms} \]

On the other hand we have to specify in terms of the technical system, what is the safety issue in the case of this emergency brake. This is expressed by invariant $I_{emer}$ which is formulated as an an inequality:

\[ I_{emer} \equiv rt - fs > 0 \]

In this formula $fs$ represents the position of the front stand of the swap body and $rt$ the rear position of the trailer (see figure 5).

The invariant $I_{emer}$ is correlated to a a fragment of program code. In this case we may have a boolean variable

² There are lots of publications considering these kinds of deviations and methods to confine them in a tractable way, see e.g. Liagoves (2007).

---

![Diagram of the trailer approaching the swap body](image)

**Fig. 5.** Diagram of the trailer approaching the swap body

$OV_{lat}$ which indicates when the extrapolated lateral position indicates a possible collision. Corresponding an emergency brake has to be applied if the longitudinal position of the vehicle comes sufficiently near to a stand:

$OV_{lat}$ and $(OV_{rt} - OV_{s} < c)$

Replacing $c$ by 0 would be correct in case of absolute consistency between technical and computational system.

Summing up we have gathered the data for the items (a) to (e):

- (a) ±10 mm in longitudinal deviation by the laserscanner
- (b) the age of the sensed position and orientation is between 10 ms and 150 ms
- (c) the jitter of process $i$ executing $CA$ is between $20 \text{ ms}$ and $120 \text{ ms}$
- (d) the action of braking lasts between $50 \text{ ms}$ and $90 \text{ ms}$
- (e) there is no value dependant deviation of the actuator

The sequel of steps now shows which is the right constant value $c$ to guarantee the invariant $I_{emer}$:

1. The values of the technical system guaranteeing for collision freedom refer to all distances between $rt$ and $fs$ which are greater than zero:

\[ V'_z = \{ v'_z \in V'_z | v'_z > 0 \} \]

2. For determining $V''_z$ all those values have to be included which have been scanned at time $t - \Delta \tau$ and still satisfy $I_{emer}$ at time $t$. Form the considerations above we know:

\[ 80 \text{ms} \leq \Delta \tau \leq 360 \text{ms} \]

During this time span the vehicle may have advanced minimally

\[ 0.080s \frac{25 \text{mm}}{s} = 2 \text{mm} \]

and maximally

\[ 0.360s \frac{30 \text{mm}}{s} = 10.8 \text{mm} \]

Taking into account the maximum motion which is possible in that time we find:

\[ V''_z = \{ v''_z \in V'_z | v''_z > 10.8 \text{mm} \} \]

(3) For the calculation of $OV''_z$ all values have to be taken into account which due to the position deviation of the laserscanner may stem from $V''_z$. The lowest value is still bigger than $10.8 \text{mm} + 10 \text{mm}$.
(4) This leads to the final condition which triggers an emergency brake when variable $0Vx$ which is equal to the difference $0Vet-0Vax$ is less than or equal to $20.8\text{mm}$. So, the constant value $c$ for the final condition is:

$$0Vlat \text{ and } (0Vx <= 20.8)$$

This enumeration describes the immediate way to derive correct fragments of program code. However, the method is much richer and offers other insights which allow to answer several other questions. So, we might ask for the impact that the period $\Delta p_i$ of process $i$ has on the condition computed above. Put in a more general sense the method also allows to enrich sensitivity analysis (see Punnaklat et al. (1997) and Bini et al. (2000)) in that a large variety of impacts are made obvious. This leads to our final vision in the context of autonomous driving which aims at the certification of all relevant safety aspects. Without certifiable methods there will be no approval of driverless commercial vehicles operating at a certain level of autonomy, not even for non-public areas such as logistic centers or harbor areas.

6. CONCLUSION AND OUTLOOK

Relating program code of the computational system to the physical entities of the technical system and thereby deriving deadlines has been the main focus of our approach. In this aspect it is one of many approaches to relate software technique with schedulability analysis. In contrast to the majority of the other approaches we know, our approach can be characterized as one which primarily focuses on the later steps of software development, because we believe that here is an even higher demand for support than in the early phases. Typically, the real-time software is prone to a myriad of changes when implemented, integrated, tested or even when it is running in changing environments. So, there is a great need to keep some kind of congruency (a metaphor used by Wedde et al. (2008)) between the computational and the technical system. Keeping this congruency manually – as we do today – is not acceptable and shows the prototypic status of our approach.

Consequently, we are working for appropriate tools to support this approach. Based on a formal description of all kinds of deviations encountered in the embedded system the semi-automatic derivation of control actions on the level of program code is intended. One approach is based on Borel-algebraic computation rules. However, this approach is limited to scalar values for the read from sensors or written to actuators. For higher dimensional values the estimations become pessimistic up to an intolerable degree. A more advanced approach has been started which applies general constraint based languages and model checking to find the desired program-code conditions. First results will be available soon.

REFERENCES


