Applying LTTA to guarantee flow of data requirements in distributed systems using Controller Area Networks[*]

Marco Di Natale
ReTiS Lab.
Scuola S. Anna, Pisa, Italy

Albert Benveniste
IRISA/INRIA Campus de Beaulieu
35042 Rennes cedex, France

Paul Caspi
VERIMAG/CNRS 2 avenue de Vignate
38610 Gières, France

Claudio Pinello
Cadence Research Labs
2150 Shattuck Ave, Berkeley, CA, USA

Alberto Sangiovanni Vincentelli
EECS Dept. U.C. Berkeley
Berkeley, CA, USA

Stavros Tripakis
Cadence Research Labs
2150 Shattuck Ave, Berkeley, CA, USA

Abstract

Most automotive applications today consist of a set of interacting tasks. These applications are implemented on a distributed architecture consisting of several ECUs connected by buses. Although algorithms are designer to be robust, several communication flows between tasks should be designed in such a way that there is no loss of data. Furthermore, safety critical applications depend on correct data being delivered on time. We address this problem when the buses of the implementation architecture are Controller Area Networks (CANs). Flow-of-data requirements are addressed using an intermediate mapping solution where the LTTA protocol is used on some of the communication links. Then the problem becomes one of mapping LTTA links to CAN buses. We give conditions under which this mapping is feasible and show how to optimize the design solution with respect to performance.

1. Introduction

In automotive applications, the propagation of information from one end to the other of a functional chain is typically implemented by a set of periodically activated tasks and messages. The execution platform is a distributed architecture consisting of several ECUs connected by buses. In several functions, there are flows of data that should be preserved, either because they carry status-change information (such as, for example, a gear shift) or because the value of each datum is a sample that is used to reconstruct the history of a physical variable in the environment. An example of the latter is a stream of radar samples used in active safety functions to detect objects in the environment, and their trajectories.

These streams can be identified by the fact that messages are sent every time a value changes or contain a *rolling counter* that allows the receiver to detect a missing value. Robustness is build into the function(s) by transforming state change events in status variables and sampling them periodically or by building robust controls that can estimate (with some loss of accuracy) the missing values in a stream. Each strategy is typically accompanied by some amount of oversampling, which results in some waste of resource utilization on the bus and CPU side.

In addition, as demonstrated in [9], the implementation of stream preserving flows allows the semantics preserving implementation of synchronous reactive models of computation. Being able to preserve the semantics of a software model at implementation time, especially in distributed architectures, is not a trivial task, but brings along the added value that the implementation can be trusted to faithfully match the results of simulation and the properties proven by formal analysis of the software model.

Furthermore, real-time constraints may be defined on the computation and communication latencies. We address the problem of the system analysis and design optimization when the buses of the implementation architecture are CAN buses, a common choice due to cost and component availability. The precise definition of the problem and our approach are described next (Figure 1).

The Problem

- *Functional Model.* Consider a set of interacting tasks, labeled as $\tau_i : i = 1 \dots n$. Tasks can be writers, readers or possibly both, when they sample incoming data from a communication link, perform their computations and then forward the results. A superscript can be used to label a task as a reader or a writer.

- *Constraints and Assumptions.* Define a link $l_{i,j}$ for each writer-reader relation between two tasks that ideally requires no loss of data. We assume *acyclic communication graphs*. A *path* from τ_i to τ_j, or

[*] This research was supported in part by the European Commission under the projects IST-2001-34820 ARTIST and IST-004527 ARTIST2 Networks of Excellence on Embedded Systems Design, by the NSF under the project ITR (CCR-0225610), and by the GSRC.

$p_{i,j} = [\tau_i, \ldots, \tau_j]$, is an ordered sequence of tasks, such that there exists one link between any two consecutive tasks. The end-to-end latency $L_{i,j}$ associated to $p_{i,j}$ is defined as the largest possible time interval that is required for a value written by τ_i to be read by τ_j. A deadline d^i can be defined for the completion time of τ_i or for an end-to-end computation on $p_{i,j}$ as $d^{i,j}$.

- *Mapping.* The architecture consists of several ECUs $(P_1, \ldots, P_m)$ connected by buses $(B_1, \ldots, B_c)$. Mapping the functional model to the architecture defines a deployment relation between tasks and ECUs, where $\pi_{i,c}$ equals TRUE if τ_i is executed on P_c and FALSE otherwise and between links and buses, where $\gamma_{i,j,b}$ equals TRUE if the communication support of $l_{i,j}$ is provided by B_b.

The problem, typical of distributed systems, is *to find an architecture and a deployment relation such that the constraints are all satisfied, possibly optimizing some efficiency criteria*.

Buses and Protocols. The Controller Area Network (CAN) [2] is pervasive in automotive systems [10]. Its does not require clock synchronization and it tolerates drifts and jitter in the transmission times of messages. CAN provides guaranteed worst-case bounds on message latencies even when transient faults occur. [3] However, the correct implementation of a function may depend on the definition of message priorities and periods (typically with some oversampling), when the implementation imposes deadlines or requires preserving functional properties (for example, avoiding the loss of data). To cope with these problems, time-triggered protocols have been proposed, in which communications are scheduled to occur at predefined points in time, and each message is assigned a fixed time slot for transmission. TTP's advantage is time determinism in the communication times, with predictable (known) delays at each communication cycle. Their disadvantage are a possible inefficiency in the schedule and the need for strict clock synchronization. FlexRay [11] is a standard time-triggered protocol developed for use in automotive applications by major OEMs and Tier1 suppliers, which will in the future help support highly deterministic and large bandwidth requirement applications. At present, however, the first and only use of FlexRay in commercial vehicles is the planned to be BMW's new X5 Sports Activity Vehicle (SAV) [12], and only for the pneumatic damping system. The Loosely Time-Triggered Architecture (LTTA) has been introduced [1] as a weaker form of the strictly synchronous Time-Triggered Architecture (TTA) proposed by Kopetz [5]. LTTA communication channels are implemented in avionics systems. An LTTA architecture is characterized by the fact that local clocks are not synchronized, and may suffer from relative offset or jitter. Furthermore:

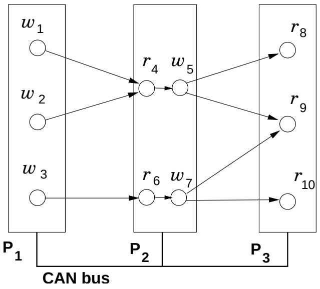

Figure 1. System as a graph of tasks.

- access to the bus occurs quasi-periodically, according to the readings of a local clock, but the different clocks involved for writing to, sending, and receiving from the bus are not synchronized;
- writings and readings are performed independently at all nodes in synchrony with the local clocks;
- the bus behaves like a shared memory, i.e., values are sustained by the bus and are periodically refreshed, based on a local clock owned by the bus.

The clocks are not synchronized, but are bound to deviate from each other with limited drift. In [1], sufficient conditions were given to ensure that an architecture of the above kind is semantics preserving in a certain sense. However, the results of [1] are specific to single-user case and provide no basis for a systematic extension to multi-user and multi-bus communication. In practice, most automotive systems are already operating under the LTTA assumptions, given that nodes and tasks are typically not synchronized, and communication-by-sampling is typically used for the propagation of information among tasks. Hence, a mapping to LTTA does not require any additional implementation and it is more a conceptual than a design and coding operation. At this stage, LTTA results in a set of rules and constraints on clocks (periods) and priority assignments that can provably guarantee preservation of flows among communicating tasks.

The Approach In this paper, we present an approach for solving the mapping problem where the implementation architecture is based on a set of CAN buses by using a two-step process. First, LTTA is used to guarantee that the data are not lost on critical communication links. Then, the LTTA protocol is mapped onto a physical CAN implementation. If this two-step process is successful, we guarantee that the constraints are satisfied and that the implementation is efficient and cost-effective. Efficient and cost-effective here means that the solution includes the amount of oversampling in information communication and processing *that is strictly necessary*. Avoiding unnecessarily high sampling rates can help save CPU and bus utilization or even allow less expensive architecture choices at times.

Of course, not all tasks and deadline constraints can be implemented using LTTA solutions and not always can LTTA protocols be mapped to CAN buses. In this paper, we provide conditions on the periods of the tasks and messages in the system for this approach to be applicable. Further, we define the conditions for checking feasibility with respect to the communication and computation deadlines and the LTTA constraints. *As compared with the formula of [1], our bounds are much tighter for the case of a bus with bounded transmission latencies and where the activation of readers, as well as bus transmissions, are affected by jitter*.

Finally, we formulate the design of a system implementing multiple LTTA channels as an optimization problem, in which task and message **periods** must be assigned such that:

- the LTTA implementation is provably correct for all the links that require flow preservation,
- the worst case message delays are guaranteed,
- the worst case task jitters are guaranteed,
- the end-to-end latencies on the computation paths meet the corresponding deadlines.

This problem can be formulated as a *mixed integer geometric programming* and solved efficiently by approximation to a geometric programming problem.

Related Work The scheduling and response time analysis of CAN messages is discussed in [3], where a flaw in the traditional analysis documented in [6] is reported and a new solution is provided. The limitations in the applicability of worst case analysis to real CAN systems has been discussed extensively in [6]. Support for time-triggered communication is provided by protocols that schedule the messages statically based on tables that define the message that needs to be transmitted at each point in time. TTCAN [13] and the Time-Triggered Protocol (TTP) [14] are examples. TTP uses a generalized time-division multiple-access (GTDMA) scheme with variable sized slots, in which each node has only one opportunity to transmit for each cycle. In FlexRay, slots have the same size, but a node can have more than one transmission opportunity for each cycle. Scheduling techniques for the static segment have been developed by extending the work for scheduling messages in a TDMA bus [15]. In order to accommodate a fraction of traffic that is dynamically activated, flexibility can be added with an additional transmission window reserved to this type of traffic. This is the case of hybrid protocols like Byteflight, introduced by BMW for automotive applications and later superseded by the FlexRay, and of the FTT-CAN protocol [16].

2 LTTA

The LTTA protocol [1] is meant to preserve the flow of data between a reader and a writer, with no loss of information. Its main components are: a writer and its node, the bus, and the reader on a remote node. These devices operate according to local clocks that are not synchronized. All definitions are associated with the devices as indicated by the superscripts w, b, and r, respectively. The sequences of writes, transfers and reads are indexed by the set $\mathbb{N}^+$ and the index 0 is reserved for the definition of initial conditions.

The writer At the time $t^w(n)$ of the n-th tick of its clock, the writer generates a new value $x^w(n)$ and a new alternating flag $b^w(n)$ with:

$$b^w(n) = \begin{cases} false & \text{if } n = 0 \\ \text{not } b^w(n-1) & \text{otherwise} \end{cases} \tag{1}$$

and stores both in a private output buffer. At any time t, the buffer content y^w is the "latest" value pair:

$$y^w(t) = (x^w(n), b^w(n)) \text{ where } n = \sup\{n' | t^w(n') < t\}$$

The bus. At the time $t^b(n)$ of its n-th clock tick, the bus fetches the value in the writer's output buffer and stores it, with zero delay, in the reader's input buffer.

At any time t, the reader's input buffer content y^b, is the latest value that was written into it:

$$y^b(t) = y^w(t^b(n)) \text{ where } n = \sup\{n' | t^b(n') < t\} \tag{2}$$

The reader At the time $t^r(n)$ of its n-th clock tick, the reader copies the value y^b into the auxiliary variables

$$(x(n), b(n)) = y^b(t^r(n)) \tag{3}$$

Then, the reader extracts from the x sequence the values x^r corresponding to alternations of b. Variable $m(n)$ counts the number of alternations of b up to cycle n. The values of the extracted sequence $x^r(k)$ are given by:

$$\begin{array}{l} m(0) = 0; m(n) = \text{ Card}\{k \mid k \leq n \wedge b(k) \neq b(k-1)\} \\ x^r(k) = x(l), \text{ where } l = \min\{n' | m(n') \geq k\} \end{array} \tag{4}$$

As defined in [1] the protocol is correct (i.e. preserves the sequence of data produced by the writer) if

$$\forall n : x^r(n) = x^w(n) \tag{5}$$

Under the assumption of perfectly periodic clocks, the above protocol is correct if and only if:

$$T^w \geq T^b \text{ and } \left\lfloor \frac{T^w}{T^b} \right\rfloor \geq \frac{T^r}{T^b} \tag{6}$$

where T^w, T^b, T^r are the periods of the writer, bus and reader [1].

Clock drifts If T is the reference period, and δ_m and δ_M are positive relative bounds, with $0 \leq \delta_m < 1$, clock drifts can be modeled as

$$T(1-\delta_m) \leq \ t(n+1) - t(n) \ \leq T(1+\delta_M) \tag{7}$$

Theorem 1 *[Extension of Th. 1 in [1]] Let the writing, bus transmission, and reading times satisfy equations:*

$$\begin{array}{rrcccc} \forall n: & T_m^w & \leq & t^w(n+1) - t^w(n) & & \\ \forall p: & T_m^b & \leq & t^b(p+1) - t^b(p) & \leq & T_M^b \\ \forall m: & & & t^r(m+1) - t^r(m) & \leq & T_M^r \end{array} \tag{8}$$

Then, the protocol is correct with respect to (5) if

$$T_m^w \geq T_M^b \text{ and } \left\lfloor \frac{T_m^w}{T_M^b} \right\rfloor \geq \frac{T_M^r}{T_m^b} \tag{9}$$

Activation Jitters We can account for clock jitter by:

$$\begin{array}{l} t(n) \ \in \ [\tau(n), \tau(n)+J], \text{ where} \\ T_m = T(1-\delta_m) \leq \tau(n+1) - \tau(n) \leq T(1+\delta_M) = T_M \end{array}$$

In this model, $t(n)$ is the sequence of clock ticks of interest, and $\tau(n)$ is an auxiliary sequence satisfying the second equation. The second equation captures the drift effect, whereas the first equation captures the jitter effect.

Implementing LTTA The formulation of the LTTA protocol is based on periodic event streams. However, when readers and writers are implemented as software tasks, the actual time instants at which the reads and writes take place depend on the computation time of the tasks, on scheduling delays and other interferences from the operating system or the I/O handling. In Figure 2 the top row represents the ideal stream of periodic events for a writer. In reality, the writer task is activated by a periodic signal from a HW clock. Because of oscillator inaccuracies, this periodic signal may be affected by drift. The HW clock interrupt signal is managed by the operating system (OS), which activates the corresponding task. The activation signal a^w is delayed by the response time of the OS (a.k.a. *interrupt latency*). Furthermore, the actual write action takes place during the execution of the task. The task may be executed immediately after its activation, as for the instance following $a^w(n)$ in the figure, or it may be delayed by higher priority tasks executing on the same CPU, as for the one activated by $a^w(n+1)$. In the worst case, we should assume that the read or write action is executed as the last action of the task, right before its completion. In this case, the maximum jitter of the write action corresponds to the worst case response time w^i of the task, computed according to the following formula [4]:

$$J^i = w^i = C^i + \sum_{j \in hp(i)} \left\lceil \frac{w^i}{T_m^j} \right\rceil C^j \tag{10}$$

where C^i is the worst case execution time of the task, and the index j spans all the tasks with a higher priority, executed on the same processor. The term $\left\lceil \frac{w^i}{T_m^j} \right\rceil$ (denoted number of *interferences* of τ_j on τ_i) represents the integer number of times the task τ_j preempts τ_i before it completes.

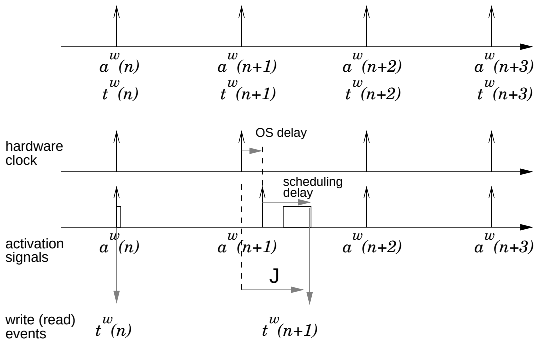

Figure 2. Software implementation of readers and writers. Worst case jitter.

3. Implementing LTTA on CAN

We are interested in mapping the LTTA protocol between a writer and a reader to a CAN-based system consisting of the following stack of components.

- The *writer task* is executed periodically, under the control of a timer interrupt with min-max bounded drift. The writer stores its signal datum in a shared variable. The output write time is subject to jitter caused by scheduling and kernel delays, see (10).

- A *middleware-level task* is responsible for the transmission of all the messages from the same node. This task is activated at the GCD of all message periods. When a message period expires, the task reads the content of the signal variables and packs them into the data field of a new message that is enqueued for transmission. This task determines the period and jitter associated with bus sampling events.

- After a variable delay, caused by the priority-based scheduling of the *CAN bus*, the message is delivered to the remote peripheral hardware. On delivery, an interrupt is generated at the destination and the *driver software* stores the data content of the message in shared signal variables. The message latency can be upper-bounded [6] (see below)

- A *reader task*, periodically activated by a timer, reads the data from the shared variables and uses them. The read time is subject to jitter (10).

This structure is quite general, with the possible only exception of the middleware-level structure. The synchronization in the transmission of messages imposed by the middleware level task is however typical of many implementations and, to the authors' knowledge, in practical use in major automotive electronics manufacturers.

The writer and reader tasks in [1] correspond to the application-level writer and reader tasks. The bus entity, however, is implemented by the union of the middleware-level task, the driver software and the medium access protocol of the bus. To apply (and improve upon) the condition stated in Theorem 1, we need to model CAN latencies.

CAN Latencies The CAN bus is a wired AND broadcast channel. The time axis is divided in slots, each slot is used for the transmission of one bit. The access protocol works by alternating contention and transmission phases. The contention phase consists of the transmission of the message identifier field, which defines the message priority (11 bits for the standard format). When the channel goes idle, all contending nodes start transmitting the message Id one bit for each slot starting from the most significant. Collisions are solved with the wired AND. If a node reads its priority bits on the channel without changes, it realizes it is the winner and gets the rights to transmit. Otherwise, if one of its bits appears changed when read from the bus, it withdraws.

The message worst-case latencies are computed like the worst-case response time for non-preemptable resources scheduled by priority [3]. If C^i is the worst case transmission time (WCTT) of message M^i, its worst case latency Δ^i_M can be computed as :

$$\begin{aligned} w^i &= B^i + \sum_{j \in hp(i)} \left\lceil \frac{w^i + J^j}{T^j_m} \right\rceil C^j \\ \Delta^i_M &= w^i + C^i \end{aligned} \tag{11}$$

where $w^i > 0$ is the queuing delay, $j \in hp(i)$ spans the indexes of all messages having priority higher than M^i, and the blocking term B^i accounts for the non preemptability of CAN frames and it is equal to the largest WCTT of any frame. The jitter J^i associated to a message is the jitter of the corresponding enqueuing events, equal to the worst case response time of the middleware task responsible for its transmission. The minimum message latency Δ^i_m is the message transmission time C^i itself, and the difference between the maximum and the minimum latency is denoted as

$$\Delta^i_{M-m} = \Delta^i_M - \Delta^i_m = w^i.$$

3.1 Preserving Data Flows

The conditions of Theorem 1 can be pessimistic if the jitter is significantly larger than the drift (as is in most implementations). A less pessimistic bound for analyzing the feasibility of an LTTA implementation follows. The following applies to a single writer-middleware-bus-reader communication sequence. Indexes of tasks and messages are dropped for clarity. In the system graph, conditions on end-to-end computations can be obtained by composing the conditions on the writer-to-reader links. Figure 3 shows the timeline of the communication between writer and reader tasks activated periodically with jitter and exchanging data over a CAN bus. Variations in the timing of the writer, middleware, and reader events are bounded by respective maximum jitters J^w, J^m, and J^r. The CAN bus latency is between a maximal Δ^b_M and a minimal value Δ^b_m.

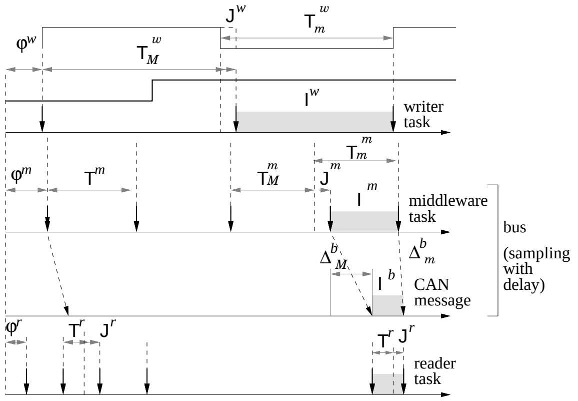

Figure 3. Timeline sequence in the LTTA implementation on CAN.

Using the same formalism used in [1] to describe the system, the writer task produces data at time instants $t^w(n)$ with $n \in \mathbb{N}^+$, with period $T^w \in [T^w_m, T^w_M]$ and writes the result in the signal variable with maximum jitter J^w. Similarly, the middleware task reads data from the signal variable with period $T^m \in [T^m_m, T^m_M]$, and maximum jitter J^m, it packs the signal into a message and enqueues the message for transmission. The message is received at the destination node with variable latency $\Delta^b \in [\Delta^b_m, \Delta^b_M]$. Finally, the reader task reads the data from the bus peripheral at the receiving node with period $T^r \in [T^r_m, T^r_M]$, and worst case jitter J^r.

At each stage, we define the *validity interval* I to be the smallest time interval that a given data value is guaranteed to be maintained before being overwritten. The preservation conditions dictate that at each stage there is at least one sampling event inside the validity interval of any buffer along the chain. Multiple samples of the same value are discarded on reception, based on the alternating bit protocol.

To find the worst-case validity intervals, we define at each stage the worst case scenario for the propagation of the values from the writer to the reader.

Writer. The minimum length of the time interval in which a value written by the writer is available is (Figure 3).

$$I^w = T^w_m - J^w \tag{12}$$

shown in Figure 4 as the time from event $w(n)$ at the latest possible time $t^w(n) = \tau^w(n) + J^w$ to $w(n+1)$ at the earliest time $t^w(n+1) = \tau^w(n) + T^w_m$.

Writer to Middleware task. The middleware task samples the signal variable with period T^m and worst case jitter J^m. Hence, the condition for having at least one sample in the worst case validity interval for the writer is (Figure 3)

$$T^m_M + J^m \leq T^w_m - J^w \tag{13}$$

Middleware task to Bus The value read by the middleware task will be available and forwarded in a set of transmitted messages (conceptually it can be imagined as a variable placed before the bus) for a minimum time that can

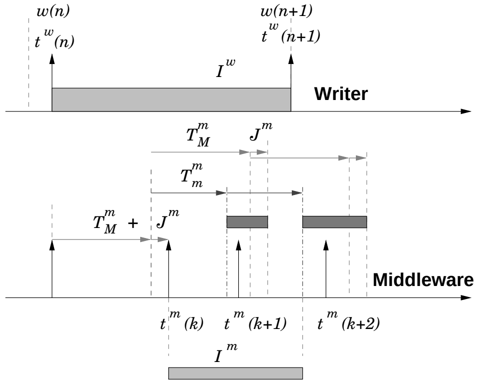

Figure 4. Worst case scenario for the datum validity at the middleware level.

be computed by reasoning on the worst case scenario, depicted in Figures 3 and 4 . Suppose the writer datum has its minimum validity interval from $w(n)$ to $w(n+1)$ and the middleware task samples the writer value at the latest possible time, that is, a middleware read event arrives right before $w(n)$ with zero jitter and the next read event $m(k)$ occurs in $t^m(k)$, $T_M^m + J^m$ time units after $w(n)$. The value sampled in $m(k)$ will start a validity interval at the middleware level I^m that ends when a new value from the writer is read after $t^w(n+1)$, that is, at the first reading event $m(k+l)$ such that $t^m(k+l) \geq t^w(n+1)$. In the worst case, $t^m(k+l) = t^w(n+1)$, and I^m is lower bounded by

$$I_{lb}^m = I^w - T_M^m - J^m. \qquad (14)$$

A less pessimistic bound can be found by considering the bounds on the period and the jitter of the middleware read events (Figure 4). The first middleware read is at

$$t^m(k) = t^w(n) + T_M^m + J^m - \epsilon \text{ (with } \epsilon \to 0)$$

denoted as $t^m(k) = t^{w-}(n) + T_M^m + J^m$.
The next middleware read $t^m(k+1)$ will be in the interval

$$[t^{w-}(n) + T_M^m + T_m^m, \quad t^{w-}(n) + 2T_M^m + J^m]$$

In the general case, the intervals in which all the events $m(k+l)$ with $l = 1, 2, 3, \ldots$ can occur (Figure 4) are

$$[t^{w-}(n) + T_M^m + lT_m^m, \quad t^{w-}(n) + (l+1)T_M^m + J^m]$$

or, alternatively,

$$t^m(k+l) \in [t^m(k) + lT_m^m - J^m, \quad t^m(k) + lT_M^m]$$

The validity interval for the middleware datum ends at the earliest middleware read such that $t^m(k+l) \geq t^w(n+1)$. We therefore need to find the first interval in which a read event may occur that straddles I^m. This defines the minimum validity interval at the next step.

Hence, we need to find the smallest positive integer

$$l = 1, 2, 3 \ldots \text{ such that } t^w(n+1) \leq t^m(k) + lT_M^m$$

(in Figure 4, it is $l = 2$ for the second interval). For the selected l, if $t^w(n+1) \geq t^m(k) + lT_m^m - J^m$, then the validity interval ends in $t^w(n+1)$, otherwise, it ends in $t^m(k) + lT_m^m - J^m$. The validity interval at the middleware level is

$$I^m = \max(t^m(k) + lT_m^m - J^m, t^w(n+1)) - t^m(k). \quad (15)$$

Please note that the worst case is not defined by considering an integer number of periods of length T_m^M, but the smallest interval may result from larger middleware task periods (in the example of the figure, $M(k+2)$ occurs right after the end of I^{wM} only if the middleware events are separated by more than T^M).
Bus to Reader task After transmission on the bus, considering the worst case and best possible message latencies, the bus validity interval at the destination node is

$$I^b = I^m - \Delta_M^b + \Delta_m^b = I^m - \Delta_{M-m}^b \qquad (16)$$

Reader The condition for the reader task to have at least one event inside the validity interval of the bus datum is

$$T_M^r + J^r \leq I^b \qquad (17)$$

Theorem 2 *If conditions (13, 14, 17) are satisfied, then the data flow between a quasi-periodic writer and a quasi-periodic reader through a bus with bounded delay is preserved without any data loss.*

3.2 Latency evaluation

The worst case end-to-end latency between the periodic activation of a writer and the completion of the reader task consuming its datum (in case all communications occur with a single position buffer) is upper-bounded by

$$L_{w,r} \leq T_M^m + J^m + \Delta_M^b + T_M^r + J^r$$

In a communication path, the end-to-end latency can be simply obtained by adding up the contributions at all steps.

4. Assigning periods to tasks and messages

Using the conditions of the previous section, designers may seek an assignment of the periods that allow a guaranteed feasible implementation of LTTA on all the links that require data preservation, or possibly try to find a solution that is not only feasible, but also optimal with respect to some cost function. A mathematical programming formulation allows the use of automatic tools for finding a feasible or optimal solution. The computation times C^i of the tasks, the transmission times C^m of the messages, the communication relation, the task and message deployment, the periods of the independent writer tasks, and the worst case drift δ of all periods are given. The optimization variables include the periods of all the tasks and messages. Additional dependent optimization variables are the latencies of the messages, the response times of the tasks and the jitter in the reading/writing of the shared variables.

For our problem, geometric programming (GP), a special form of convex programming [7], can provide an effective solution. A geometric program in standard form is:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1 \;\; i = 1, \ldots, m \\ & g_i(x) = 1 \;\; i = 1, \ldots, p \end{array}$$

where $x = (x_1, ..., x_n)$ is a vector of positive real-valued decision variables. f is a set of *posynomial* functions, while g is a set of *monomial* functions. A monomial function m has the form shown in Equation 18 while a posynomial function is simply the sum of monomial functions.

$$m(x) = c x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n} \qquad c > 0, a_i \in \Re \tag{18}$$

Posynomials are not the same as *polynomial*. A polynomial's coefficients need not be positive, and, on the other hand, the exponents of a posynomial can be real numbers, while for polynomials they must be non-negative integers. Although the definition of monomial applies to the terms of both a posynomial and a polynomial, the latter is probably the most common. If x contains both integral and real-valued variables, the problem is a mixed-integer geometric program (MIGP). Unlike GPs, MIGPs are non convex and can only be efficiently solved for a limited number of integer variables. The benefits of a mathematical programming (or MP, GP is a special form of MP) optimization approach are particularly relevant to the period synthesis problem. First, in assigning periods, there are a large number of interdependencies between the objects on different paths. Considering one path at a time is not guaranteed to find a feasible, let alone optimal, solution. MP approaches consider all constraints simultaneously. Next, and more importantly, MP approaches can be customized with system-specific issues by simply adding additional constraints. Whereas other solution mechanisms are brittle to changes in the problem assumptions, MP approaches can adapt to different problem assumptions or partial solutions. For example, the existence of legacy tasks and messages whose periods are fixed or otherwise restricted can be handled quite easily with additional constraints.

The main difficulty with using MP approaches is their solution time. The form of the constraints and objective function must be chosen carefully such that it accurately captures the behavior of the system, and yet remains amenable to efficient solving.

For our synthesis problem, geometric programming (GP), which is a special form of convex programming [7] balances these tradeoffs nicely. Simpler methods, like MILP, cannot be used given that periods appear as they are in the definition of the bounds required by LTTA for flow preservation and also appear at the denominator of the formulas that are used to compute the worst case response times of tasks and messages.

Problem formulation The design solution must be found subject to the set of constraints that define the correctness of the LTTA implementation and the dependency of the reader and writer jitter from their response time and the periods of the higher priority tasks and messages. To use geometric programming, all the constraints defined in the previous section must be formulated in the form accepted by GP.
Drift constraints The reader, writer and middleware periods, including their upper and lower bounds are optimization variables, constrained by the drift. For each period T^i the drift constraints can be written as

$$\frac{T^i_m}{T^i} + \delta^i_m \leq 1 \quad \text{and} \quad \frac{T^i}{T^i_M} + \delta^i_M \leq 1$$

LTTA constraints The implementation of LTTA requires (13) for all links $l_{i,j}$. The condition can be rewritten as

$$\frac{T^m_M + J_m + J^w}{T^w_m} \leq 1. \tag{19}$$

Furthermore, we need to enforce conditions (12), (14), (16), (17). We add optimization variables for the validity intervals and we define inequalities that constrain the size of the validity intervals according to the previous definitions.

$$\begin{array}{lll} \frac{I^w + J^w}{T^w_m} \leq 1 & \frac{I^m + T^m_M + J^m}{I^w} \leq 1 & \text{from (12, 14)} \\ \frac{I^b + \Delta^b_{M-m}}{I^m} \leq 1 & \frac{T^r_M + J^r}{I^b} \leq 1 & \text{from (16, 17)} \end{array}$$

Task jitter constraints The jitter of the writer, reader and middleware event flows is equal to the worst case response time of the corresponding task (10).

$$J^i = w^i \tag{20}$$

Similarly, the formulation in [3] can be used to compute the message delays from the message transmission times and their periods from (11). The interferences $z^{ij} \in \mathbb{Z}^+$ are used as helper variables to define the number of interferences of higher priority task (message) τ_j on τ_i (M_j on M_i). The worst case response time of the i-th task and message can be bound using, respectively

$$\frac{C^i + \sum_{j \in hp(i)} z^{ij} C^j}{w^i} \leq 1 \qquad \frac{B^i + \sum_{j \in hp(i)} z^{ij} C^j}{\Delta^i_{M-m}} \leq 1$$

The number of interferences z^{ij} (from a higher priority j to a lower priority i) for tasks and messages are specified with (21) and (21), respectively. Since z^{ij} are integers, the problem becomes a mixed integer geometric programming problem.

$$\frac{w^i}{T^j z^{ij}} \leq 1 \qquad \frac{\Delta^i + J^j}{T^j z^{ij}} \leq 1 \tag{21}$$

Minimum and maximum execution periods of tasks and messages may be specified separately, with (4).

$$\frac{T^i_{\min}}{T^i} \leq 1 \qquad \frac{T^i}{T^i_{\max}} \leq 1$$

Deadline constraints In a real-time system, some of the response times and/or some of the end-to-end latencies must meet corresponding given deadlines

$$\frac{w^i}{d^i} \leq 1 \qquad \frac{L^{i,j}}{d^{i,j}} \leq 1$$

Approximation to GP To solve the MIGP problem efficiently, we approximate it with a standard GP problem relaxing the interference variables z^{ij} to real-valued variables with additional parameters $0 \leq \ \alpha^{ij} \leq 1$. The approximated task response-time variables are denoted as s^i. The approximated message latencies as Λ. The approximated response times and latencies of tasks and messages are

$$\frac{C^i+\sum_{j\in hp(i)}(z^{ij}+\alpha^{ij})C^j}{s^i} \leq 1 \qquad (22)$$

$$\frac{B^i+\sum_{j\in hp(i)}(z^{ij}+\alpha^{ij})C^j}{\Lambda^i} \leq 1 \qquad (23)$$

and equations (21) from the MIGP become

$$\frac{s^i}{T^j z^{ij}} \leq 1 \qquad \frac{\Lambda^i+J^j}{T^j z^{ij}} \leq 1 \qquad (24)$$

If at the end of the optimization run, the values of all α^{ij} are 1, then the approximation is always conservative, i.e. $s^i \geq w^i$ and $\Lambda^i \geq \Delta^i_{M-n}$. If some $\alpha^{ij} < 1$, no such guarantees can be made. Clearly, the accuracy of the approximation depends upon the α parameters that are used. In [3] the approximation for the computation of the worst case response time is discussed in details (in the context of a different optimization problem) and an iterative algorithm for approximating the α^{ij} values with arbitrary precision is presented.

Objective function A possible objective function is the sum of the end-to-end latencies on selected paths.

$$\text{Minimize} \sum_{i,j} L_{i,j}$$

Each $L_{i,j}$ is the sum of writer-to-reader latency terms. Therefore, the objective function is a sum of jitters, periods and delays and is posynomial.

5. Conclusions

We solved the problem of assigning tasks and message periods in a distributed system implemented on a CAN bus system optimally under the condition that some links preserve data flow constraints. We used the LTTA protocol to ensure the preservation problem and we gave conditions under which the LTTA protocol can be implemented on the CAN system satisfying implementation constraints. We offered an optimization approach based on geometric programming to find the optima periods automatically.

In this paper, we assumed no buffer is available. It is possible to extend our approach to deal with the case in which a $k-$position buffer is used at the interface between write and bus and bus and reader.

References

[1] A. Benveniste and P. Caspi and P. Guernic and H. Marchand and J. Talpin and S. Tripakis. A protocol for loosely time-triggered architectures. In *Proceedings of EMSOFT 2002*, LNCS vol 2491, 252–265, Springer Verlag, 2002.

[2] R. Bosch. Can specification, version 2.0. Stuttgart, 1991.

[3] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. S. Vincentelli. Period optimization for hard real-time distributed automotive systems. In *ACM Design Automation Conference*, San Diego, CA, June 2007.

[4] M. G. Harbour, M. Klein, and J. Lehoczky, “Timing analysis for fixed-priority scheduling of hard real-time systems,” IEEE Transactions on Software Engineering, vol. 20, no. 1, January 1994.

[5] H. Kopetz Real-Time Systems. Kluwer Academic Publishers, Boston, 1997.

[6] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (can) message response times. Control Eng. Practice, 3(8):1163–1169, 1995.

[7] S. Boyd and L. Vandenberghe. Convex optimization. Available at http://www.stanford.edu/˜boyd/cvxbook/, 2004.

[8] K. Koh, S. Kim, A. Mutapcic, and S. Boyd. gpposy: A matlab solver for geometric programs in posynomial form. Technical report, Stanford University, May 2006.

[9] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincentelli, P. Caspi, M. Di Natale, Implementing Synchronous Models on Loosely Time Triggered Architectures IEEE Transactions on Computers, accepted, to appear.

[10] Navet N, Song Y, Simont-Lion F, Wilwert C Trends in automotive communication systems. Proc IEEE 93(6):1204-1223, 2005

[11] Flexray consortium, Protocol Specification V2.1 Rev. A, available at http://www.flexray.com, 2006

[12] Embedded Computing Freescale Silicon Enables World’s First Vehicle with FlexRay(TM) Technology, available at http://www.embedded-computing.com/news/db/?4667

[13] International Organization for Standardization Road vehicles-Controller Area Network (CAN) - Part 4: Time-triggered communication. ISO/DIS 11898-4, 2002

[14] Kopetz H, Bauer G The time-triggered architecture. Proc IEEE 91(1):112-126, 2003

[15] Pop P, Eles P, Peng Z Schedulability-driven communication synthesis for time-triggered embedded systems. Real-Time Systems Journal 24:297-325, 2004

[16] J. Ferreira, P. Pedreiras, L. Almeida and J. A. Fonseca, The FTT-CAN Protocol for Flexibility in Safety-Critical Systems, IEEE Micro, 22 (4), 46-55, 2002