Synthesis of task and message activation models in real-time distributed automotive systems *

Abstract

Modern automotive architectures support the execution of distributed safety- and time-critical functions on a complex networked system with several buses and tens of ECUs. Schedulability theory allows the analysis of the worst case end-to-end latencies and the evaluation of the possible architecture configurations options with respect to timing constraints. We present an optimization framework, based on an ILP formulation of the problem, to select the communication and synchronization model that exploits the trade-offs between the purely periodic and the precedence constrained data-driven activation models to meet the latency and jitter requirements of the application. We demonstrated its effectiveness by optimizing a complex automotive architecture.

1. Introduction

Past work in electronics/controls/software-based (ECS) vehicle architectures and function development has been component-focused, each function being usually deployed to a single control module. Recent architectures feature networking of control modules within application domains (e.g. power train) as well as across domains (e.g. power train and chassis). The implications are an increased number of distributed time-critical functions and multiple tasks in execution on each ECU. Distributed architectures supporting the execution of (hard) real-time applications are also common in avionics, factory and plant control systems. For building these systems with a design-time guarantee that the timing constraints are met, different design and scheduling methodologies are used. Avionics controls, for example, are often built based on static, time-driven schedules. Because of resource efficiency, many automotive controls are designed based on run-time priority-based scheduling of tasks and messages. Examples of standards supporting this scheduling model are the OSEK operating system standard [7]and the CAN [2] bus arbitration model. At the interface between any two resource domains, and very often also at the interface between two abstraction layers (such as, for example, the application and the middleware layers), different interaction models may be implemented. The simplest interaction model consists of the purely *periodic activation* model, where all interacting tasks are activated periodically and communicate by means of asynchronous buffers based on a freshest value (non time-deterministic) semantics. Similarly, message transmission is triggered periodically and each message contains the latest values of the signals that are mapped into it. Another possible activation model is the *data-driven activation*, where task executions and message transmissions are triggered, respectively, by the arrival of the input data and by the availability of the signal data. The periodic activation model has the advantage of higher possible schedulability on all resources, but suffers from possibly very high worst-case latencies on the end-to-end computations. The data-driven activation model, on the other hand, provides for much shorter end-to-end delays and time determinism in the communication, but it may result in time intervals with bursty activations of tasks and messages, hence high instantaneous load on some resources and possibly very high latencies or even impossibility of scheduling for low priority end-to-end computations.

The two competing models of periodic activation with asynchronous communication and data-driven activation are reconciled by a new conceptual framework for the analysis of distributed chains of computations, based on network calculus [3] and its application for evaluating the propagation of event models [4]. In [5] this model is used for distributed schedulability analysis, where the system can be described as an arbitrary mix of data-driven and periodic asynchronous interaction models. Other papers, such as [9], focused on providing lock-free and wait-free communication mechanisms that ensure deterministic delays in the implementation of models integrating both event and time triggered subsystems. Later, the mechanism has not only been extended to EDF scheduled systems, but the authors also provided optimal (tight) bounds for buffer allocation in the implementation of Rate Transition blocks for many-to-one communication channels [10]. Optimization of buffer implementation is also the objective of [1]. Finally, the trade offs between a purely periodic activation model and an event-driven activation semantics are explored in [6] with respect to the composability of subsystems scheduled according to the two models. However, even if these papers

*This work has been supported by General Motors and by CHESS.

provide analysis procedures with increasing speed and precision, the synthesis problem is scantly analyzed: the only approach is provided by [8], where the use of genetic algorithms is proposed for optimizing priority and period assignments with respect to a number of constraints, including end-to-end deadlines and jitter.

Our work is performed in the context of the design of distributed software architectures for next-generation automotive controls, where the application performance requirements impose constraints on end-to-end latencies in the execution of the control functions. We propose a novel synthesis procedure, based on approximate timing analysis to optimize the definition of the activation model in the functional network with respect to the latency constraints. We demonstrate its effectiveness in the tuning of a complex real-world automotive architecture.

2. Definitions, Notation and Assumptions

Our model is a *dataflow* of tasks, represented with a *Directed Acyclic Graph*. The *model* is a tuple $\{\mathcal{V}, \mathcal{E}, \mathcal{R}\}$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ the set of edges, and $\mathcal{R} = \{R_1, \ldots, R_z\}$ is the set of shared resources supporting the execution of the tasks (CPUs) and the transmission of the messages (bus).

$\mathcal{V} = \{o_1, \ldots, o_n\}$ is the set of objects implementing the computation and communication functions of the system. o_i can be a task or a message and is characterized by a maximum time requirement C_i and a resource R_{o_i} that it needs to execute or for its transmission. All objects are scheduled according to their priority; π_i is the priority of o_i and indexes are assigned by decreasing priority levels; r_i is the worst case response time of o_i, from the activation of the object to its completion in case it is a task, or its arrival at the destination node in case it is a message. w_i is defined as the worst case time spent from the instant the job is released with maximum jitter J_i to its completion or arrival. An object o_i has conceptually one or more *input ports* and one or more *output ports* that are used to exchange data and optionally *activation signals* or events. Each object runs at a base period T_i (and optionally, jitter J_i). It reads its inputs at the time it starts executing, if it is a task, or it samples the incoming signal values and it is enqueued at the activation time in case it is a message. At the end of its execution or transmission, it delivers its results (task) or its data content (message) and, where required, activation signals on its output ports.

$\mathcal{E} = \{l_1, \ldots, l_m\}$ is the set of *links*. A link $l_i = (o_h, o_k)$ connects the output port of object o_h (the source) to the input port of object o_k (the sink). Alternatively, a link may be labeled with the indexes of the source and destination task as in $l_{h,k} = (o_h, o_k)$. A link l_i may carry the activation signal produced when the source object completes its execution or transmission and instantaneously received on the input port of the sink. However, a different communication and synchronization model is possible, where the sink is activated by a periodic timer and, when it executes, reads the latest value that was transmitted over the link (and stored into a buffer). The source and the sink of link l_i are also denoted by $src(l_i)$ and $snk(l_i)$, respectively.

When an object is activated by the completion of a predecessor we define an *event-driven* activation model. If an object is activated by a single completion event, then the only condition is that its period must be an integer multiple of the predecessor object period. In this case, the activation semantics is of one every k signals. We define a less restrictive activation semantics by allowing an object to be activated by multiple completion events. In this case, the activation is of type AND. The only allowed case for multiple activation events from multiple incoming links is when the links are connected to predecessor objects having periods that are integer dividers of the target object period, have a unique common predecessor, and are scheduled on the same resource. In this case, we define a set of link groups $\mathcal{G} = \{lg_1, \ldots, lg_k\}$ where each link group $lg_i = \{l_{i_0}, \ldots l_{i_{k_i}}\}$ has the following properties, $snk(l_{i_j}) = snk(l_{i_l})$ and $R(src(l_{i_j})) = R(src(l_{i_l}))$ for any link pair $l_{i_j}, l_{i_l} \in lg_i$. If $\tau_{j1} = src(l_{i_j})$ and $\tau_{j2} = snk(l_{i_j})$ then $kT_{j1} = T_{j2}$ for some integer k. Finally, $\forall lg_i, \exists! o_p$ such that $\forall l_{j,k} \in lg_i$ there exists a link $l_{p,j} \in \mathcal{E}$. and there is no other incoming link to o_j. If all the links in a group carry an activation signal, then the source objects must be activated at the same time or they must all be activated by a completion event. These last conditions do not apply to singleton groups. $G(o_k)$ is the set of link groups that are incoming to o_k. For example, in Figure 1, l_1, l_2, l_3 belong to group lg_1, l_4, l_5 to lg_2 and l_6 to lg_3 consisting of only one link. Hence, an object can be activated by a periodic trigger, by a signal coming from a single predecessor object or by the AND composition of signals coming from a single link group. In this last case, the object is actually activated by the completion of the lowest priority object o_r in the group lg_i, which is called *group representative* $o_r = rep(lg_i)$.

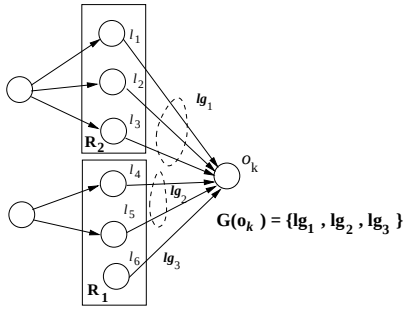

Figure 1. Example of link groups.

An *external event* results from the execution of a virtual object o_i with no input links, representing the environment. External events can be *periodic* with period T_i and jitter J_i, or *sporadic* with a minimum interarrival time, equally denoted by T_i.

An *output* object o_j represents data consumption by the

environment, e.g. when the system updates an actuator.

A *functional chain* or *Path* from o_i to o_j, or $P_{i,j}$, is an ordered sequence $P = [l_1, \ldots, l_n]$ of links that, starting from $o_i = src(l_1)$, reach $o_j = snk(l_n)$ crossing a unique sequence of $n+1$ objects such that $snk(l_k) = src(l_{k+1})$. o_i is the chain's source and o_j its sink.

When task and messages are activated periodically and communicate on a freshest value semantics, several definitions of end-to-end latency (and the associated deadline) are possible. In our work, the end-to-end latency $L_{i,j}$ associated to a path $P_{i,j}$ is defined as the largest possible time interval that is required for the change of the input at one end of the chain to be propagated to the last task at the other end of the chain, whatever is the state of the tasks in the path and regardless of the fact that some intermediate result may be overwritten before it is read.

We assume in this paper that *the application can tolerate the semantic variation when changing from one synchronization model to the other.* In many control applications, the nondeterminism in time introduced by the periodic activation model and the jitter introduced by the event-driven activation can both be tolerated within acceptable ranges.

2.1. Periodic activation model

In the periodic activation model (Figure 2), the release jitter is zero and the worst case end-to-end latency is computed for each path by adding the worst case response times and the periods of all the objects in the path ($r_k = w_k$).

$$L_{(i,j)} = \sum_{k:o_k \in P(i,j)} (T_k + r_k)$$

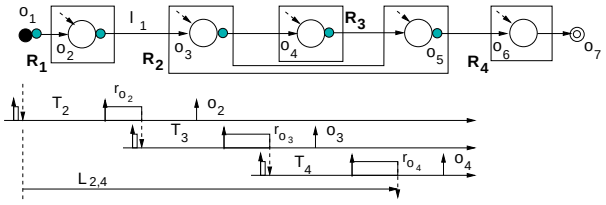

Figure 2. Periodic activation model.

Due to unsynchronized timers, in the worst case (Figure 2) the external event arrives right after the completion of the first instance of task o_2 with minimum (negligible) response time. The event data will be read by the task on its next instance and the result will be produced after its worst case response time, that is, $T_2 + r_2$ time units after the arrival of the external event. The same reasoning applies to the execution of the following objects.

2.2. Data driven activation model

In the data driven activation model (an example in Figure 3), if we assume the same activation period for all the nodes that are activated in a computation chain, then for all the intermediate neighboring nodes $o_i \rightarrow o_j$ it is clearly $r_i = J_j$. The worst case end-to-end latency can be computed for each path by adding the worst case queuing and execution/transmission times of all the objects in the path.

$$L_{(i,j)} = \sum_{k:o_k \in P(i,j)} w_k$$

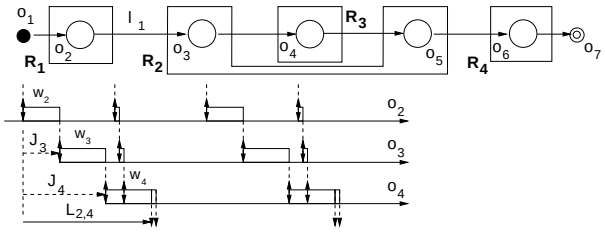

Figure 3. Data driven activation model.

In this case, the worst case jitter of the activation events grows larger as the computations propagate along the chain. The latency is typically lower if compared with the previous case, but the large jitter in the activation of the intermediate tasks and messages means that they may be activated according to bursty patterns of events. These bursts of high priority tasks and messages increase the response time of the lower priority objects that share resources with them.

2.3. Processor scheduling

The worst case response time for a periodic task τ_i, activated with maximum jitter J_i in a generic preemptive and priority based scheduled system is given by:

$$\begin{aligned} &w_i(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i(q) + J_j}{T_j} \right\rceil C_j \\ &w_i = \max_q \{w_i(q) - qT_i\} \\ &r_i = J_i + w_i \\ &\text{for all } q = 0 \ldots q^* \text{ until } r_i(q^*) \leq T_i \end{aligned} \tag{1}$$

where $j \in hp(i)$ means all the object indexes such that $\pi_j \geq \pi_i$ and $R_{o_i} = R_{o_j}$. The need of evaluating the first q instances inside the busy period is caused by the uncertainty about the instance which causes the worst case response time. However, a lower bound on the worst case response time can be obtained by restricting the computation to the first instance. This bound is tight in case $r_i \leq T_i$.

$$\begin{aligned} &w_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \\ &r_i = J_i + w_i \end{aligned} \tag{2}$$

Linear upper and lower bounds for the solution to the previous fixed point equation can be obtained from

$$w_i^{\uparrow} = C_i + \sum_{j \in hp(i)} \left(\frac{w_i^{\uparrow} + J_j}{T_j} + 1\right) C_j \tag{3}$$

$$w_i^{\downarrow} = C_i + \sum_{j \in hp(i)} \left(\frac{w_i^{\downarrow} + J_j}{T_j}\right) C_j \tag{4}$$

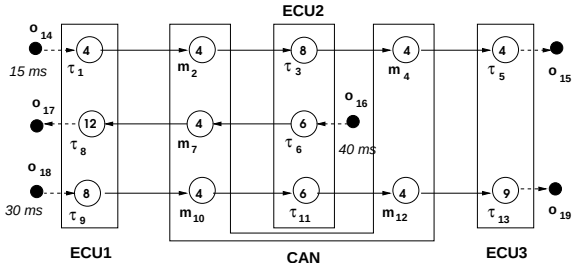

Figure 4. Example graph.

2.4. Bus scheduling

In this paper we assume that *message objects are transmitted over CAN buses*. The evaluation of the worst-case latencies for the messages follows the same rules for the worst-case response time of the tasks, with the exception that an additional blocking term B_i must be included in the formula in order to account for the non preemptability of CAN frames and the transmission time of the message cannot be preempted. The blocking term B_i for a generic message o_i can be computed as the largest worst-case transmission time of any frame having a priority lower than π_i and sharing the same bus resource ($wq_i > 0$ is the queuing delay part of w_i, without the transmission time).

$$\begin{aligned} & wq_i(q) = B_i + qC_i + \sum_{j \in hp(i)} \left\lceil \frac{wq_i(q) + J_j}{T_j} \right\rceil C_j \\ & w_i = \max_q \{C_i + wq_i(q) - qT_i\} \\ & r_i = w_i + J_i \\ & \text{for all } q = 0 \ldots q^* \text{ until } r_i(q^*) \leq T_i. \end{aligned} \tag{5}$$

A lower bound on w_i and r_i can be computed by only considering the first instance ($q = 0$) and, similar to processor scheduling, the response times of messages can be approximated by linear functions of the jitter variables.

3. An example

Figure 4 represents a sample system consisting of 3 ECUs, 1 CAN bus, 8 tasks and 5 messages (priorities, periods and worst-case execution times as in the following table.) Three computation paths are defined, ending respectively in objects o_{15}, o_{17} and o_{19}.

				$periodic$		$event-driven$		
Object	π_i	T_i	C_i	r_i	L_i	J_i	w_i	r_i
τ_1	13	15	4	4	4	0	4	4
m_2	12	15	4	8	27	4	8	12
τ_3	11	15	8	8	50	12	8	20
m_4	10	15	4	12	77	20	12	32
τ_5	9	15	4	8	**100**	32	8	**40**
τ_6	8	40	6	14	14	0	30	30
m_7	7	40	4	16	66	30	28	58
τ_8	6	40	12	20	**130**	58	30	**88**
τ_9	5	30	8	28	28	0	60	60
m_{10}	4	30	4	28	82	60	44	104
τ_{11}	3	30	6	28	140	104	60	164
m_{12}	2	30	4	28	198	164	88	252
τ_{13}	1	30	9	28	**260**	252	60	**312**

The last four columns explain the tradeoffs in the analysis. In the case all objects are activated periodically and communicate by means of asynchronous buffers, the latencies for the three paths, assuming no sampling delay on the first task are shown in the fourth to last column. If, however, the activation of the objects is always driven by the completion of their predecessor, then the latencies are much better for the highest priority paths, but are significantly larger for the lowest priority path ending in o_{19}. Although the jitter analysis is characterized by pessimism (relative offset information and best case response times are not considered in the analysis), the results show the tradeoff between the two models and the opportunity for design optimization.

If the deadlines are defined as $d_{14,15} = 80$, $d_{16,17} = 120$ and $d_{18,19} = 280$, then in neither of the two cases, the deadlines can be guaranteed. However, if the activation model is defined in such a way that messages m_2, m_4 and m_7 are activated periodically, then the worst-case latencies are $L_{14,15} = 70$, $L_{16,17} = 100$ and $L_{18,19} = 208$, with all the deadline constraints satisfied.

Calculating the worst-case response time of tasks and messages means solving a least fixed-point equation. In some cases, the problem may be tentatively approached by using linear upper and lower bounds for the response time of the first object instance in the critical instant hypothesis (which is itself a lower bound of the real value) as in (3), (4).

The question is to determine the amount of pessimism (and optimism) introduced by the linear approximations. The data of the example show that the linear approximations become progressively less accurate when the priority of the objects in the chain is lowered. For example, for the event-driven activation model, the upper and lower bound latencies for the three paths are, respectively, $\{44.36, 130.86, 507.03\}$ and $\{38.91, 79.43, 294.96\}$. However, a linear combination of the linear upper and lower bounds can be sufficiently accurate to be used as an estimator of the actual end-to-end latencies. We will demonstrate the effectiveness of the linear approximation in the following real design case of an automotive system architecture.

4. MILP Solution

A mixed integer linear programming formulation can be used to find a solution with respect to the deadline constraints on the paths. In addition to r_i, J_i, w_i, $L_{s,t}$ we define $y_{h,k}$ as

$$y_{h,k} = \begin{cases} 1, & \text{if the activation of } o_k \text{ is event-driven by } o_h \\ 0, & \text{otherwise} \end{cases}$$

4.1. Feasibility Constraints

The feasibility constraints are modeled according to the rules for computing the jitter, the response times and the latencies at all nodes in the graph.

The jitter inheritance rule is encoded as follows. Consider a scheduled object o_k with multiple incoming link groups. We are only interested in those groups (links) that can possibly carry an activation signal (for all the other links $l_{j,k}$ it is clearly $y_{j,k} = 0$).

We enforce the condition that all the links in one group assume the same activation model. This means that

$$y_{r,k} = y_{s,k} \tag{6}$$

for all the pairs $l_{r,k}, l_{s,k}$ belonging to the same group lg_h. The equivalence must be extended to all the incoming links to the source objects of the group links, in case periodic objects cannot be activated at the same time.

If o_k has more than one incoming link group, only one of the group representatives can provide its activation signal. For each object o_k it must be

$$\sum_{lg_h \in G(o_k)} y_{r,k} \leq 1 \text{ where } o_r = rep(lg_h).$$

If all group links have a periodic activation (all $y_{r,k} = 0$) then o_k is activated periodically and $J_k = 0$. Otherwise, J_k will be equal to the response time of the representative object in the group from which it gets the activation signal. The two alternative ways of computing J_k can be encoded in a pair of constraint sets leveraging a typical formulation in use in integer linear programming.

A very large constant value M is used to nullify one or more constraints by making them always true depending on the value of a set of binary variables ($y_{r,k}$ in our case).

$$J_k \leq \sum_{lg_h \in G(o_k)} y_{r,k} \times M \quad \text{where } o_r = rep(lg_h) \tag{7}$$

$$0 \leq J_k \tag{8}$$

If all $y_{r,k} = 0$, then (7) and (8) constrain the value of J_k to 0. If $y_{r,k} = 1$ for one of the incoming link groups, then the first inequality is redundant and the following two set of constraints (a pair for each $lg_h \in G(o_k)$) make J_k equal to the worst-case response time r_r of the predecessor object o_r that is the representative of the activating group.

$$J_k \leq r_r + (1 - y_{r,k}) \times M \text{ where } o_r = rep(lg_h) \tag{9}$$

$$r_r - (1 - y_{r,k}) \times M \leq J_k \text{ where } o_r = rep(lg_h) \tag{10}$$

If o_k has only one incoming link from object o_h that can possibly provide an activation signal, then a simpler set of constraints replaces (7), (9), and (10)

$$r_h + (y_{h,k} - 1) \times M \leq J_k \tag{11}$$

$$J_k \leq r_h \tag{12}$$

$$J_k \leq y_{h,k} \times M \tag{13}$$

The worst-case response time r_h for object o_h can be computed as

$$r_h = w_h + J_h$$

Because of the non-linearity and even non convexity of the fixed point formula that provides the exact value of w_h, a linear combination with coefficient $\alpha \in [0, 1]$ of the linear upper (3) and lower bounds (4) is used.

$$w_h = C_h + \sum_{o_k \in hp(h)} (\frac{w_h + J_k}{T_k} + \alpha) C_k \tag{14}$$

where α is chosen as to minimize the following mean square fit function, computed for all $y = 0$ and assuming α does not depend significantly on the value of the y variables.

$$\sum_{P_r \in \mathcal{P}} (\alpha * L^{\uparrow}_{P_r} + (1 - \alpha) * L^{\downarrow}_{P_r} - L_{P_r})^2 \tag{15}$$

where $L^{\uparrow}_{P_r}$ and $L^{\downarrow}_{P_r}$ are the latencies computed on the path P_r using the upper and the lower linear bound respectively.

Finally, for computing the end-to-end latencies, a variable $z_{i,j}$ is defined for each link $l_{i,j}$ to express the link contribution to the end-to-end latencies of all the paths containing it. The variable $z_{i,j}$ is equal to w_j if the link $l_{i,j}$ carries an activation event (first two of the following constraints.) Otherwise, $z_{i,j}$ will be equal to $w_j + J_j + T_j$, considering the fact that o_j may be activated by some other signal with release jitter J_j. Hence, the contribution to the latency depends on the value of $y_{i,j}$ and the usual formulation is used to express the alternative.

$$w_j \leq z_{i,j} \tag{16}$$

$$z_{i,j} \leq w_j + (1 - y_{i,j}) \times M \tag{17}$$

$$z_{i,j} \leq w_j + J_j + T_j \tag{18}$$

$$w_j + J_j + T_j - y_{i,j} \times M \leq z_{i,j} \tag{19}$$

The end-to-end latency $L_{s,t}$ associated with path $P_{s,t}$ is computed as

$$L_{s,t} = \sum_{l_{u,v} \in P_{s,t}} z_{u,v}$$

and should not exceed its deadline.

$$L_{s,t} \leq d_{s,t}.$$

4.2. Objective Functions

Based on the above constraints, in addition to get a feasible solution, which satisfies the deadline constraints, we have the flexibility to get the optimal solution with respect to different cost functions. The minimization of the number of event buffers is expressed by

$$maximize \sum_{lg_h \in \mathcal{G}} y_{j,k}, \text{ where } o_j = rep(lg_h)$$

Other interesting cost functions are the sum of the end-to-end latencies, or the sum of the positive differences between the end-to-end latency of each path and the corresponding deadline over all the paths in the system. In the

second objective function we may assign a penalty for the violation of a specific path deadline through a weight γ_{p_r}.

$$\sum_{p_r \in \mathcal{P}} L_{p_r} \qquad \sum_{p_r \in \mathcal{P}} \gamma_{p_r} * Max(L_{p_r} - d_{p_r}, 0)$$

4.3. Optimization of the example graph

For the example in Figure 4, we used the objective functions defined in the previous section. The results are shown in the following table where $P_1 = o_{14} \rightarrow o_{15}$, $P_2 = o_{16} \rightarrow o_{17}$, $P_3 = o_{18} \rightarrow o_{19}$ and the objective functions are $F_1 =$ minimization of the number of event buffers, $F_2 =$ minimization of the sum of the path latencies, $F_3 =$ minimization of the sum of weighted lateness for all the paths exceeding the deadline and $F_4 =$ minimization of the lowest priority path latency.

Objective	P_1	P_2	P_3	periodic objects	event objects
F_1	55	84	304	m_4	remainings
F_2	70	58	266	τ_3, m_4, τ_{10}	remainings
F_3	55	112	236	m_2, m_7	remainings
F_4	70	98	168	τ_3, m_4, τ_8	remainings

5. Case study and Conclusions

The paper presented a novel synthesis framework procedure, based on approximate timing analysis to optimize the definition of the activation model in the functional network with respect to the latency constraints. The proposed approach has been applied to the architecture configuration of an experimental vehicle. The architecture consists of 38 nodes connected by 6 CAN buses. A total number of 100 tasks are executed on the ECU nodes, supporting from 1 to 22 tasks each, and 322 messages are exchanged over the six buses, with a minimum and maximum number of messages of, respectively, 32 and 145 for each group. The number of links in the dataflow graph is 507. Bus utilizations are between 30% and 50% and CPU utilizations are estimated between 5% and 60%. Ten pairs of endpoints have been identified in the graph as sources and destinations of computation paths with deadlines. An analysis of the graph found 184 paths between these 10 pairs of nodes and deadlines ranging from 100 ms to 300 ms have been defined for them.

If all tasks and messages are activated periodically, the end-to-end latencies largely exceed (in the worst case) the desired deadlines. For example, a worst-case latency of 627ms was found for paths with deadline 300 and 302.83 for paths with deadline 100. Of the 507 links, 313 are subject to optimization, including link groups. The sum of the end-to-end latencies was used as the metric function. The problem encoding results in 1673 variables, 313 of which are binary, and 3989 linear constraints. The time required to solve the problem is always close to 0.25 seconds (1.4 GHz PC).

After the first optimization round, the end-to-end latencies were much closer to the desired deadlines, but still not feasible for 12 of the 148 paths. It was necessary to change the period of one more task (from 12.5 to 10) and one more message (from 100 to 80), making it shorter so that an event driven activation could be defined on the corresponding incoming and outgoing links. After another optimization round, all the latencies became lower than the deadlines, with the largest value of 265 for paths with deadline 300, 190 for paths with deadline 200 and 97 for paths with deadline 100. The final result of the optimization was the definition of 115 links and 3 groups (140 total links) to carry an event-driven activation signal. The value of α changed from 0.239, at the start, when all $y = 0$ to 0.224 for the final solution. When repeating the optimization procedure with the new value of α, the same result was obtained, therefore supporting the validity of our linear approximation assumption.

Besides the assignment of priorities to tasks and messages, or the definition of task and message periods, another possible objective for the synthesis of the software architecture is finding the optimal placement of the tasks on the ECUs. However, these optimization variables are not considered in this paper and will be the subject of future work.

References

[1] M. Baleani, A. Ferrari, L. Mangeruca, and A. S. Vincentelli. Efficient embedded software design with synchronous models. In *Proceedings of the 5th ACM EMSOFT conference*, 2005.

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

[3] J.-Y. L. Boudec and P. Thiran. Network calculus - a theory of deterministic queuing systems for the internet. In *LNCS 2050, Springer*, 2001.

[4] S. Chakraborty and L. Thiele. A new task model for streaming applications and its schedulability analysis. In *IEEE DATE*, Munich, Germany, March 2005.

[5] A. Hamann, R. Henia, M. Jerzak, R. Racu, K. Richter, and R. Ernst. SymTA/S symbolic timing analysis for systems. available at http://www.symta.org, 2004.

[6] S. Matic and T. Henzinger. Trading end-to-end latency for composability. In *Proceedings of the 26th IEEE RTSS*, 2005.

[7] OSEK. Osek os version 2.2.3 specification. available at http://www.osek-vdx.org, 2006.

[8] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proceedings of the 11th IEEE RTAS*, pages 160–169, San Francisco (CA), U.S.A., Mar. 2005.

[9] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *6th Euromicro ECRTS*, July 2004.

[10] S. Tripakis, C. Sofronis, N. Scaife, and P. Caspi. Semantics-preserving and memory-efficient implementation of inter-task communication on static-priority or edf schedulers. *Proceedings of the 5th ACM EMSOFT conference*, 2005.