Event Model Interfaces for Heterogeneous System Analysis

Kai Richter, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig / Germany
{richter|ernst}@ida.ing.tu-bs.de

Abstract

Complex embedded systems consist of hardware and software components from different domains, such as control and signal processing, many of them supplied by different IP vendors. The embedded system designer faces the challenge to integrate, optimize and verify the resulting heterogeneous systems. While formal verification is available for some subproblems, the analysis of the whole system is currently limited to simulation or emulation. In this paper, we tackle the analysis of global resource sharing, scheduling, and buffer sizing in heterogeneous embedded systems. For many practically used preemptive and non-preemptive hardware and software scheduling algorithms of processors and busses, semi-formal analysis techniques are known. However, they cannot be used in system level analysis due to incompatibilities of their underlying event models. This paper presents a technique to couple the analysis of local scheduling strategies via an event interface model. We derive transformation rules between the most important event models and provide proofs where necessary. We use expressive examples to illustrate their application.

1. Introduction

With increasing embedded system complexity, there is a trend towards heterogeneous architectures. Automotive systems are composed of different processors running distributed functions with reactive or transformative behavior under the OSEK [16] operating system. Busses and processors run different resource sharing protocols, such as time sharing (TTP [10]) or packet communication (CAN [17]) and static or dynamic priority processor scheduling. Multimedia devices, such as set-top boxes, run telecommunication protocols, coding and signal processing functions on heterogeneous multiprocessors with hardwired or weakly programmable coprocessors and complex memory architectures. These are just two of many similar examples. In all cases, system constraints, system optimization, and reuse of hardware and software IP are the main sources of heterogeneity.

Global analysis of such heterogeneous systems is a challenge and is currently limited to simulation approaches, such as in VCC [3], or Seamless [15]. The known limitations of simulation such as incomplete coverage and corner case identification are aggravated since many of the design errors only result from system integration requiring detailed knowledge which is often not available to the integrator.

The analysis problem can be divided in the analysis of the system functionality and the analysis of the timing and resource sharing of a system implementation on a specific hardware-software platform. In this paper, we focus on the related issues of system timing and resource sharing.

Take an automotive bus which is used by many system functions from many suppliers. Bus load, buffer memory requirement, and response time analysis are among the major problems requiring component knowledge at integration time. By sharing a common resource, previously unrelated system functions influence each others timing. Adding or removing subsystems changes timing and resource requirements. Since timing typically is a central quality and cost factor (dead times in control engineering, jitter in multi-media, buffer sizes, ...), system analysis must iterate in design space exploration and system configuration. To reduce integration problems, people resort to fixed time-slice TDMA protocols (e. g. TTP) without priorities which are not resource efficient and hardly scale to larger systems. With increasing levels of integration and extension of the IP business, we will see this type of problems arriving at all heterogeneous systems including SoCs.

Many of the critical integration problems, such as buffer memory analysis and response time analysis could potentially be resolved by a formal global analysis if the known approaches to local subsystem analysis could be combined without running into the problem of state space explosion. State space explosion would almost inevitably emerge when trying to analyze detailed system functionality, e. g. of a car, if this information were available at all.

Target system timing analysis can be divided in two parts: analysis of a single process executed on a target system component and analysis of resource sharing effects given process execution times. For the first task, there are many recent contributions combining implicit or explicit program path analysis and cycle-true processor modeling, such as [13, 5, 22, 9]. In addition, process communication can be determined to analyze communication channel load and timing [22]. For the second task, there is a huge amount of work, mainly in the domain of real-time operating systems. In almost all approaches, however, homogeneous resource sharing strategies are assumed in literature. There are very few exceptions which consider special cases, such as [18] analyzing response times for static priority process scheduling combined with a TDMA bus protocol.

One reason for this lack of global models are incompatibilities of the input event models. An input event model describes the frequency and type of input events that lead to process or communication execution. In effect, the input events determine the system workload. Resource sharing analysis techniques, therefore, typically assume certain input event models. The system designer can deliberately enforce an event model for a certain component, e. g. by time triggering, to enable a certain resource sharing strategy or to obtain predictable system timing. This increases buffer cost and response times. Our automotive application is a good example.

The importance of transitions between different event models has been widely neglected in literature. In general, global heterogeneous system analysis is currently not possible since the respective local analysis techniques use incompatible event models. In this paper, we introduce a technique to globally analyze coupled local resource sharing strategies using an event stream interface model.

The paper is organized as follows: In Section 2, we analyze and then classify the event models typically used for analysis of widely used resource sharing strategies. An event stream interface model and interface functions which allow to represent timing and event propagation in the context of very different resource sharing strategies are introduced in Section 3. In Section 4, we demonstrate the coupling technique and the intended use of event model interfaces in system-level analysis using a complex example. Finally, we draw some conclusions.

2. Resource Sharing: Strategies & Analysis

Resource sharing strategies include process and communication scheduling and memory allocation. In this section, we briefly review the most popular event models in literature with respect to scheduling analysis.

A simple and efficient assumption is a stream of periodic input events. Here, the arrival of events can be captured by a single parameter, the period T. Periodic events are usually enforced by a timer, such as for input signal sampling in signal processing. Static scheduling strategies with a non-preemptive (i. e. without interrupt) cyclic or cyclo-static process execution order exploit this periodic property for high efficiency [11] when possible phase between different periodic event streams are known. In case of a multi-rate system, the static execution order includes several executions of a process (up to the macro-period).

The output of a system with static process execution order is typically periodic with jitter, i. e. with a defined maximum deviation from the execution period. This jitter is caused by data dependent process execution times and, in case of multi-rate systems, by different times between the executions of a process in a macro-period.

If the input events to a statically ordered set of processes arrive with a jitter, then there is less time for execution which either reduces the computation efficiency or requires event buffering to keep efficient periodic execution. The same applies when the phase between periodic event streams is not known exactly. So, even for a single resource sharing strategy, input and output event models can be incompatible.

Periodic input events are also the main model for static priority scheduling strategies. In Rate Monotonic Scheduling (RMS) priorities are statically assigned according to the period of a process, the shorter the period the higher the priority. RMS is optimal for periodically executed processes with deadlines at the end of the respective period [14]. In contrast to the static order process execution, the phase between the different periods has no influence on the scheduling efficiency, and need not be known. The deadline requirement corresponds to a model of periodic output events with jitter. Deadlines can be earlier than the period with Deadline Monotonic Scheduling (DMS) as optimal scheduling strategy or later than the period [12]. RMS/DMS can even be extended to include process graphs with process dependencies without changing the input model [23].

The remaining event models can be subsumed as sporadic events and event streams with possible bursts, in the following called “sporadic event model” and “burst event model”, resp.. The burst model uses two time bases, the minimum event inter-arrival time t and the maximum number of events b which may occur in a time interval T. They are also called “inner” (t) and “outer” (T) periods of the event stream. This burst model is used by Tindell [21] and Audsley [1] to extend the analysis of static priority scheduling to such event streams. Gresser [8, 7] uses a somewhat similar model for analysis of dynamic priority scheduling. It introduces a vector of sequential time intervals rather than a single interval. This gives structure to an event sequence which can be exploited in analyzing Earliest Deadline First (EDF) scheduling. Sporadic events [20] only constrain the minimum inter-arrival time t. This model is e. g. useful to analyze response times.

This survey just selected few landmark and representative approaches out of the host of work on scheduling analysis with the purpose to find an event model classification as a basis for heterogeneous system design. Gresser’s event model is closest to a general event model trying to capture rather complex event stream patterns. Balarin’s event “signatures” [2] used for simulation and abstract load analysis have a similar purpose. Both models target general event stream modeling at the cost of model complexity that excludes direct application of most of the analysis techniques mentioned above. Instead, the next section will introduce transformation functions between simpler models allowing to keep the implementation efficiency of these models. Another problem of the current general models is the lack of formal operations on event streams that allow to merge or split event streams, e. g. on a bus.

The survey suggests to use four event classes: periodic events, periodic event streams with jitter, burst event model and sporadic event model. Some of the models are special cases of others, but, as we will see later, should be differentiated from the more general case for efficiency and analysis reasons. Each of the models represent an event stream by a set of abstract parameters, e. g. the period T. These parameters allow to calculate the worst-case number of event occurrences within a given interval of time, which is required by most of the mentioned analysis approaches. This calculation can be done by a function $n^{+}_{\mathrm{ev}}(\Delta t)$. Table 1 gives a formal definition of these four event stream models.

model	params	$n^{+}_{\mathrm{ev}}(\Delta t)$
periodic	$< T >$	$\left\lceil \frac{\Delta t}{T} \right\rceil$
sporadic	$< t >$	$\left\lceil \frac{\Delta t}{t} \right\rceil$
jitter	$< T, J >$	$\left\lceil \frac{\Delta t + J}{T} \right\rceil$
burst	$< T, t, b >$	$\left\lfloor \frac{\Delta t}{T} \right\rfloor b + \min\left(b, \left\lceil \frac{\Delta t - \lfloor \frac{\Delta t}{T} \rfloor T}{t} \right\rceil\right)$

Table 1. The four most popular event models

3. Event Model Coupling

In this section, we present the actual coupling of event models. We first discuss compatibility issues between the four event models introduced in the preceeding section, and derive event model interfaces (*EMIF*s) to transform the parameters of one event model into those of another model. In those cases, where no simple interface can be derived, we introduce the interposition of event adaptation functions (*EAF*s) like buffers and timers in order to couple initially incompatible event models. This step incorporates buffer sizing and event delay determination due to the additional system functions. Finally, we also investigate event stream superposition and coincidence.

We introduce the basic idea of event model coupling using a simple example. Consider the (sub)system depicted in Figure 1. Two processes $\mathcal{P}_1$ and $\mathcal{P}_2$ exchange data via event stream *ES*.

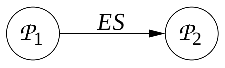

Figure 1. Event stream example

Let us assume the two processes are mapped to two different resources. Other processes are implemented on both resources, too, but they are not shown in the figure. Each resource implements a different resource sharing strategy (*RSST*). Let us furthermore assume, that the known analysis techniques for the two scheduling strategies have the following properties:

- from the timing analysis of $\mathcal{P}_1$ we know that the output events are generally periodic with period T_1 but may experience a maximum jitter J_1 (e. g. resulting from preemptions by other processes)
- for the timing analysis of $\mathcal{P}_2$ there exist only approaches requiring a sporadic input event model with a minimum temporal separation t_2

Clearly, the two local analysis approaches for $\mathcal{P}_1$ and $\mathcal{P}_2$ can not be coupled directly since the two event models (jitter and sporadic) are basically incompatible. However, we are able to *derive the required* parameter values of the sporadic event model *from the known* values of the jitter event model: The minimum temporal separation of two successive events with jitter is the period minus the maximum jitter: $t_2 = T_1 - J_1$. In the following, we generalize this idea.

3.1. Event Model Interfaces

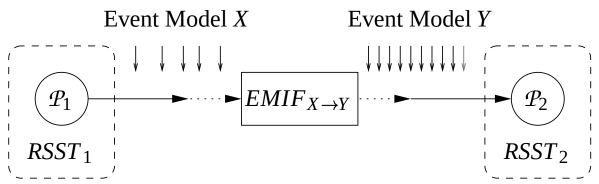

Figure 2. An event model interface (*EMIF*)

The intended use of event model interfaces (*EMIF*s) is shown in Figure 2. $\mathcal{P}_1$'s output event model X results from the selected technique to analyze the $\mathcal{P}_1$'s execution behavior with respect to the resource sharing strategy $RSST_1$. Similarly, the required input event model Y of $\mathcal{P}_2$ is determined by the selected analysis technique for $RSST_2$. Now, it is the $EMIF_{X \to Y}$'s task to translate the event stream's properties from event model X into an instance of event model Y. Such interfaces are generally uni-directional $(X \to Y)$. They can only transform instances of X into instances of Y, not vice versa.

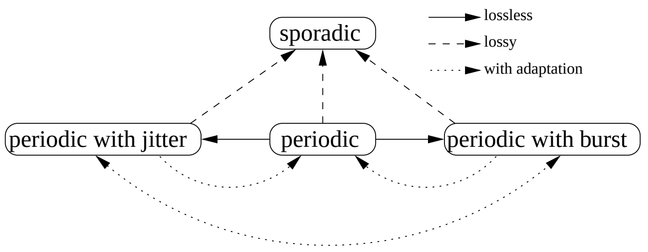

Figure 3. Possible event model interfaces

Figure 3 depicts the possible transformations $(X \to Y)$ for which such an *EMIF* can be found. The solid lines indicate that the interface losslessly (i. e. with the same modeling accuracy) transforms the information of model X into model Y, while the dashed lines indicate a loss of information during the transformation process. The dotted lines indicate that the transformation requires additional adaptation of the input event stream.

In the beginning of this section, we already solved this interfacing for the transformation (jitter$\to$sporadic). The *EMIF*s of the other feasible transformations are given in Table 2. In case when $X = Y$, no event model interface is actually needed, since the parameter values are identical. As can be seen in the table, for only 5 out of 12 (not considering $X = Y$) possible event model transformations an *EMIF* can be given. For the other seven transformations like (jitter$\to$periodic) no *EMIF* can be found. Which model combinations are interfacable can be determined by formally analyzing the corresponding event models: we need to show that the behavior of every instance of event model X is contained in the set of all possible instances of model Y. In other words, for every

$EMIF_{X\to Y}$	Y=periodic	Y=jitter	Y=burst	Y=sporadic
X=periodic	$T_Y = T_X$ (identity)	$T_Y = T_X$, $J_Y = 0$	$T_Y = T_X$, $b_Y = 1$, $t_Y = T_X$	$t_Y = T_X$ (lossy)
X=jitter	—	$T_Y = T_X$, $J_Y = J_X$ (identity)	—	$t_Y = T_X - J_X$ (lossy)
X=burst	—	—	$T_Y = T_X$, $b_Y = b_X$, $t_Y = t_X$ (identity)	$t_Y = t_X$ (lossy)
X=sporadic	—	—	—	$t_Y = t_X$ (identity)

Table 2. *EMIF*s for simple model transformations

instance of an event model X, the task is to find an instance of an event model Y with a n_{ev}^{+}-function that covers the values of the n_{ev}^{+}-function of X:

$$n_{\text{ev},X}^{+}(\Delta t) \overset{!}{\leq} n_{\text{ev},Y}^{+}(\Delta t). \tag{1}$$

Analogously, we have to proof

$$n_{\text{ev},X}^{-}(\Delta t) \overset{!}{\geq} n_{\text{ev},Y}^{-}(\Delta t). \tag{2}$$

This $n_{\text{ev}}^{-}(\Delta t)$-function represents the minimum number of events within a given time interval Δt. Although not always recognized in the existing analysis approaches, this number is of particular importance for the analysis of distributed systems to resolve the so called scheduling anomalies [6]. Table 3 provides the n_{ev}^{-}-functions of our four event models. If both conditions in equations 1 and 2 are satisfied, an interface can be found. Iff we can also show equality for both equations, the transformation is lossless, i. e. the transformation does not reduce the accuracy of the model X. Lossy transformation applies e. g. when mapping from the periodic model to the sporadic one. Equation 1 is satisfied but Equation 2 is not.

Rather than providing formal derivations for all of the model transformations, we only demonstrate how *EMIF*s can be found for the transformation (X=periodic$\to$$Y$=burst). From Table 1 in Section 2, we know:

$$\begin{aligned} n_{\text{ev},X}^{+}(\Delta t) &= \left\lceil \frac{\Delta t}{T_X} \right\rceil \\ n_{\text{ev},Y}^{+}(\Delta t) &= \left\lfloor \frac{\Delta t}{T_Y} \right\rfloor b_Y + \min\left(b_Y, \left\lceil \frac{\Delta t - \left\lfloor \frac{\Delta t}{T_Y} \right\rfloor T_Y}{t_Y} \right\rceil \right) \end{aligned}$$

model	params	$n_{\text{ev}}^{-}(\Delta t)$
periodic	$< T >$	$\left\lfloor \frac{\Delta t}{T} \right\rfloor$
sporadic	$< t >$	0
jitter	$< T, J >$	$\left\lfloor \frac{\Delta t - J}{T} \right\rfloor$
burst	$< T, t, b >$	$\left\lfloor \frac{\Delta t}{T} \right\rfloor b + \max\left(0, \left\lfloor \frac{\Delta t - (\lfloor \frac{\Delta t}{T} \rfloor + 1) T}{t} + b \right\rfloor \right)$

Table 3. The n_{ev}^{-}-functions of the four models

We have an instance of a periodic event model with period T_X. We assume that an instance of a burst event model Y with the parameters $T_Y = T_X$, $t_Y = T_X$, and $b_Y = 1$ has the same n_{ev}^{+}-function. Proof:

$$\begin{aligned} n_{\text{ev},Y}^{+}(\Delta t) &= \left\lfloor \frac{\Delta t}{T_Y} \right\rfloor b_Y + \min\left(b_Y, \left\lceil \frac{\Delta t - \left\lfloor \frac{\Delta t}{T_Y} \right\rfloor T_Y}{t_Y} \right\rceil \right) \\ &= \left\lfloor \frac{\Delta t}{T_X} \right\rfloor + \min\left(1, \left\lceil \frac{\Delta t - \left\lfloor \frac{\Delta t}{T_X} \right\rfloor T_X}{T_X} \right\rceil \right) \\ &= \left\lfloor \frac{\Delta t}{T_X} \right\rfloor + \min\left(1, \left\lceil \frac{\Delta t}{T_X} - \left\lfloor \frac{\Delta t}{T_X} \right\rfloor \right\rceil \right) \end{aligned}$$

Clearly, the term $\left\lceil \frac{\Delta t}{T_X} - \left\lfloor \frac{\Delta t}{T_X} \right\rfloor \right\rceil$ is either zero or one, depending on whether Δt is an integer multiple of T_X or not. Thus, $n_{\text{ev},Y}^{+}(\Delta t)$ turns into:

$$\begin{aligned} n_{\text{ev},Y}^{+}(\Delta t) &= \left\lfloor \frac{\Delta t}{T_X} \right\rfloor + \begin{cases} 0 & \text{if} \frac{\Delta t}{T_X} \in \mathbb{N} \\ 1 & \text{otherwise} \end{cases} \\ &= \left\lceil \frac{\Delta t}{T_X} \right\rceil = n_{\text{ev},X}^{+}(\Delta t) \quad \text{q.e.d.} \end{aligned}$$

Hence, Equation 1 is satisfied. Similarly, we can proof that Equation 2 is satisfied, too, and our assumed model parameter transformations ($T_Y = T_X$, $t_Y = T_X$, $b_Y = 1$) in fact represent the desired event model interface. The other *EMIF*s of Table 2 can be found similarly. As a counter example, consider the already mentioned transformation (jitter$\to$periodic). It is trivial to proof that no general interface can be found: The event model Y=periodic assumes the events to occur with equal temporal separation, what they generally do not since they experience a jitter. Only in the special case with $J_X = 0$, we can find an *EMIF*. Similarly, we can find an *EMIF* for an instance of a burst event model with $b_X = 1$ and $t_X = T_X$. Clearly, both mentioned special cases are periodic without any jitter nor burst, they are only the representation of a periodic event stream using the parameters of jitter or burst models, respectively. However, with slight modifications in the implementation, we can also capture such transformations in more general cases. This is explained in the next section.

$X \rightarrow Y$	$EMIF_{X\rightarrow Y}$	$EAF_{X\rightarrow Y}$	d^+_{EAF}
jitter $\rightarrow$ periodic	$T_Y = T_X$	timed buffer ($T_{EAF} = T_X$, $n_{EAF} = 1$)	T_X
jitter $\rightarrow$ burst	$T_Y = T_X$, $b_Y = 1$, $t_Y = T_Y$	timed buffer ($T_{EAF} = T_X$, $n_{EAF} = 1$)	T_X
burst $\rightarrow$ periodic	$T_Y = \frac{T_X}{b_X}$	timed buffer ($T_{EAF} = \frac{T_X}{b_X}$, $n_{EAF} = b_X - \left\lceil b_X(b_X-1)\frac{t_X}{T_X} \right\rceil$)	$T_X - (b_X-1)t_X$
burst $\rightarrow$ jitter	$T_Y = \frac{T_X}{b_X}$, $J_Y = 0$	timed buffer ($T_{EAF} = \frac{T_X}{b_X}$, $n_{EAF} = b_X - \left\lceil b_X(b_X-1)\frac{t_X}{T_X} \right\rceil$)	$T_X - (b_X-1)t_X$

Table 4. Synchronization *EMIF*s and *EAF*s

3.2. Event Adaptation Functions

As already mentioned, periodic event streams with jitter can not be captured by purely periodic event models. However, in digital signal processing systems, internal events are often assumed periodic since the system's overall input is purely periodic. But due to resource sharing and/or data-dependent process execution times, internal events most likely experience a jitter. To keep up with periodic models, buffers and timers are widely used to re-synchronize such events according to the initial period. This shows that it is generally possible to couple initially incompatible event models for analysis at the cost of additional system functions. We refer these functions to as event adaptation functions (*EAF*s). An example for the use of *EAF*s is given in Figure 4.

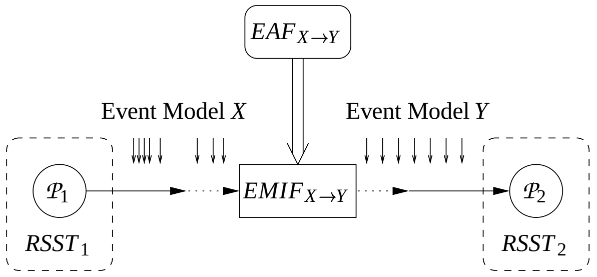

Figure 4. An *EMIF* with event adaptation function (*EAF*)

The actual functionality of these *EAF*s has to be derived from the two event models. For the above mentioned very simple case (X=jitter$\rightarrow$$Y$=periodic), the *EAF* can be found straightforward. A buffer of size 1 and an output issue period of T_Y is needed. Now, we can derive an *EMIF* for this situation by simply setting T_Y to the value of T_X. In general, the parameters of the timed buffers need to be formally derived from the model transformations. Again, we only show this derivation process using one representative, that is the model transformation (X=burst$\rightarrow$$Y$=periodic). We start with finding the period T_Y. From the burst event stream from $\mathcal{P}_1$ we know that there are b_X events every T_X time units. As a result, we obtain an average inter-arrival time of $\varnothing t_X = \frac{T_X}{b_X}$. Clearly, this average inter-arrival time must be the period of the periodic event Y, otherwise the event buffer under- or overflows. The determination of the required buffer size is more complex. On the one hand, the buffer needs to be large enough to temporarily store events during a burst in all possible situations. On the other hand, we are looking for a tight bound on the buffer size in order to optimize buffer memory. Hence, we need to identify worst-case situations. The worst-case buffering situation occurs when the burst events arrive as soon as possible and the periodic event occurs with a maximum phase delay of T_Y with respect to the beginning of the burst. An example for $b_X = 5$ is depicted in Figure 5.

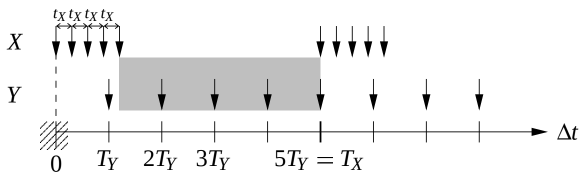

Figure 5. Worst case-buffering scenario for $EAF_{\text{burst}\rightarrow\text{periodic}}$

The maximum buffering delay d^+_{EAF} experienced by the last event of the burst is marked by the gray shaded box:

$$d^+_{EAF} = T_X - (b_X - 1)t_X$$

Since the events are removed from the buffer periodically, the maximum number of events in the buffer (n^+_{EAF}) is:

$$\begin{aligned} n^+_{EAF} &= \left\lceil \frac{d^+_{EAF}}{T_Y} \right\rceil \\ &= \left\lceil \frac{T_X - (b_X-1)t_X}{\frac{T_X}{b_X}} \right\rceil \\ &= b_X - \left\lceil b_X(b_X-1)\frac{t_X}{T_X} \right\rceil \qquad (3) \end{aligned}$$

Similarly, the *EAF*s and *EMIF*s for other model transformations can be derived. A list of *EAF*s and *EMIF*s is provided by Table 4. As can be seen, the transformations (jitter$\rightarrow$periodic) and (jitter$\rightarrow$burst) require the same *EAF*. The same holds for (burst$\rightarrow$periodic) and (burst$\rightarrow$jitter). This is because we use periodic *EAF*s is all cases. Thus, two model transformations require the same $EAF_{X\rightarrow Y}$ when they have the same input event model X. Note that event model interfaces with adaptation functions always perform a lossy transformation of model X into model Y since detailed information of the input event stream is lost due to synchronization.

In this section, we only considered synchronization of single input event streams. However, processes can have several inputs with each input stream being represented in a different model. Here, synchronization is more complex. This is briefly surveyed in the next section.

3.3. Model Extensions

For processes with two inputs, there exist two different possibilities. Either the process can execute whenever there is an event on one of its inputs (OR-process), or the process needs to wait for one event on both inputs (AND-process). Ramanathan [19] provides detailed definitions for AND- and OR-processes but does not consider event streams. With respect to the event stream, the OR-case is comparatively simple. The two input event streams are merged into a new event stream. The analysis in [8] provides means to formally capture event stream merging. However, the model is not applicable to most of the analysis approaches. Hence, we need to find a way to merge event streams within our suggested models. The following example will illustrate this.

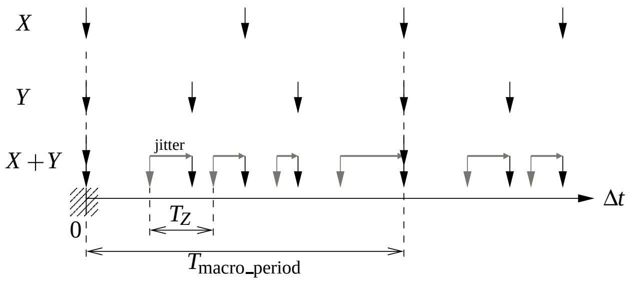

Figure 6. Merging two periodic event streams

Consider two periodic event streams X and Y with different periods $T_X = 1.5\ T_Y$. This is depicted in Figure 6. The result is a periodic event stream with jitter, indicated by the gray arrows. The proof is similar to satisfying equations 1 and 2, only that we have to consider the superposition of event streams. We assume that a periodic model with jitter and the parameters

$$T_Z = \frac{1}{\frac{1}{T_Y} + \frac{1}{T_X}} \quad \text{and} \quad J_Z = T_Z$$

is able to capture the merging of two periodic event streams. Proof:

$$\begin{aligned} n^+_{\text{ev},Z}(\Delta t) &= \left\lceil \frac{\Delta t + J_Z}{T_Z} \right\rceil = \left\lceil \frac{\Delta t + T_Z}{T_Z} \right\rceil = \left\lceil \frac{\Delta t}{T_Z} \right\rceil + 1 \\ &= \left\lceil \frac{\Delta t}{\frac{1}{\frac{1}{T_X}+\frac{1}{T_Y}}} \right\rceil + 1 = \left\lceil \frac{\Delta t}{T_X} + \frac{\Delta t}{T_Y} \right\rceil + 1 \\ &\geq \left\lceil \frac{\Delta t}{T_X} \right\rceil + \left\lceil \frac{\Delta t}{T_Y} \right\rceil = n^+_{\text{ev},X}(\Delta t) + n^+_{\text{ev},Y}(\Delta t) \quad \text{q.e.d.} \end{aligned}$$

Similarly, we can check that the condition of Equation 2 is satisfied, too. Such merging $EMIF_{X+Y \to Z}$s can be found similarly for other event streams, e. g. one periodic and one burst. In general, the merging procedure within OR-processes does not enforce any restrictions to the input stream models. In contrast, the AND-processes require coincidence of the input events. In other words, the number of events ($n_{\text{ev}}(\Delta t)$) in both event streams X and Y must not diverge for large Δt. Otherwise, the required event buffer of one of the streams will overflow. This can be checked by Equation 4.

$$\lim_{\Delta t \to \infty} \frac{n_{\text{ev},X}(\Delta t)}{n_{\text{ev},Y}(\Delta t)} \overset{!}{=} 1 \tag{4}$$

This condition must be true for all instances of event streams within the corresponding models. Thus, we have to guarantee that the upper and the lower bound on the number of events for each input stream itself do not diverge:

$$\lim_{\Delta t \to \infty} \frac{n^-_{\text{ev},X}(\Delta t)}{n^+_{\text{ev},X}(\Delta t)} \overset{!}{=} 1 \tag{5}$$

Only when both coincidence conditions, i. e. equations 4 and 5, are fulfilled, the streams can be AND-merged. To derive the actual resulting event stream is more complex than the superposition in the OR-case.

The example in Section 3.2 already gives an idea about coincidence of events. Here, an event stream with burst was resynchronized into a periodic event stream by means buffers and timers. The timer of the *EAF* was used to periodically output events from the buffer. We can think of this timer as a second event stream, and the synchronization as AND-merging. Trivially, both streams fulfill the above coincidence conditions (since we derived the parameters of stream Y from the parameters of stream X). In other words, coincidence was a result of the event adaptation step. Usually, coincidence is a requirement in AND-merging. However, the underlying ideas with respect to buffer and delay analysis are the same. This is a very simple case of AND-merging. General AND-merging is more complicated and exceeds the scope of this paper.

So far, we have shown how event stream properties can be interfaced between different event models by means of *EMIF*s and *EAF*s. This way, we are now able to couple previously incompatible event models, and thus, analysis techniques. This is a prerequisite for global (system-level) analysis. The next section contains a larger example in order to demonstrate the intended use of event models during system analysis.

4. Application Example

Figure 7 shows a complex heterogeneous platform architecture. It consists of four processors which are connected via a CAN bus [17]. Process $\mathcal{P}_1$ sends data to process $\mathcal{P}_3$ over the communication channel $\mathcal{C}_1$. Similarly, processes $\mathcal{P}_2$ and $\mathcal{P}_4$ exchange data via $\mathcal{C}_2$. The other four processes represent the resource sharing influences on the processors. However, they do not use the bus.

The task is to find event model interfaces and adaptation functions in order to enable coupling of existing local analysis techniques for the individual resources. In this paper, local analysis is only interesting is so far as it determines and constraints the event models. The following list gives a brief overview on the assumed characteristics of the resource sharing strategies (*RSST*) and analysis approaches with respect to the event models in the example:

RSST$_1$ implements static-priority preemptive scheduling. The bursty output behavior (X_1=burst event model) of $\mathcal{P}_1$ results from frequent preemptions by the higher-priority process $\mathcal{P}_5$.

RSST$_2$ is purely time driven and periodically schedules processes, e. g. simple digital signal processing functions with a fixed response time. The output is purely periodic (X_2), too.

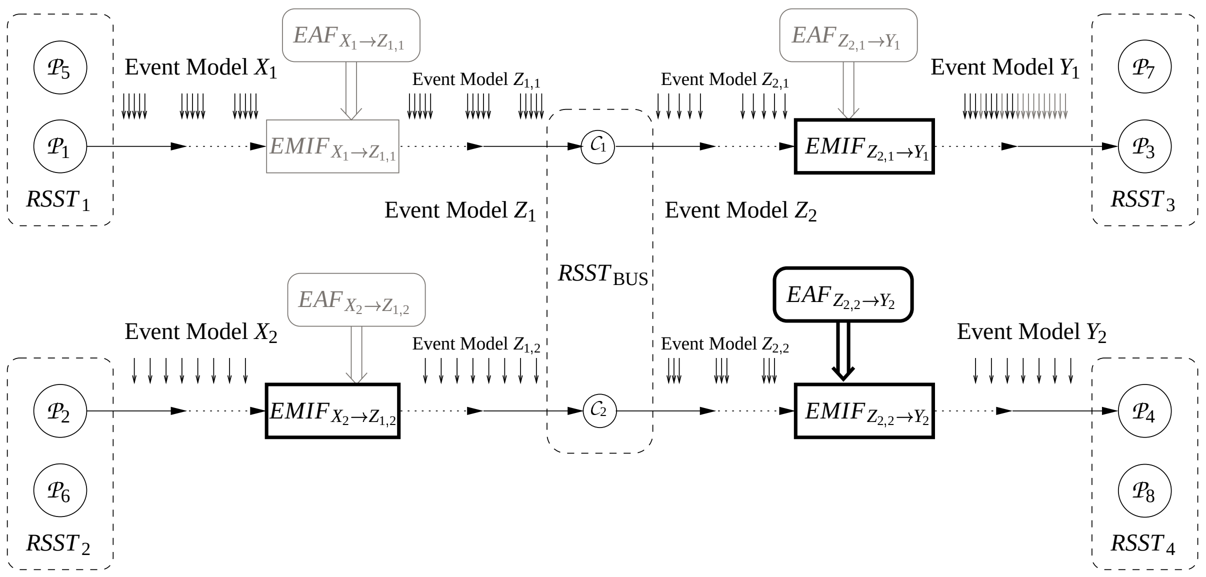

Figure 7. Example: Four processors and one bus

$RSST_3$ requires the input events to be represented using the model of sporadic events (Y_1). This is an example of an input event model constraint enforced by a specific analysis approach. Clearly, the applicable analysis techniques depend on the implemented scheduling strategy. However, for the event model coupling procedure, no detailed information about the scheduling strategy itself is necessary. Only, the required input event model needs to be known.

$RSST_4$ requires periodic (Y_2) input streams. Similarly to $RSST_3$, the actual scheduling strategy is not relevant for Y_2.

$RSST_{\text{Bus}}$ implements the CAN [17] protocol, and assigns static priorities to the logical channels $\mathcal{C}_1$ and $\mathcal{C}_2$. We assume that the input streams need to be captured by the burst event model (Z_1) in order to allow efficient analysis. Furthermore, we assume that the output is provided as bursty event streams (Z_2), too. In order to avoid confusion when deriving the corresponding event model parameters, we will use four event models, $Z_{1,1}\ldots Z_{2,2}$ for the two in- and two outputs, respectively.

With these explanations, we can determine the event model coupling using the *EMIF*s and *EAF*s of tables 2 and 4. We begin with the bus input streams. The output of process $\mathcal{P}_1$ does not need to be transformed, since it is already captured using the event model required by the analysis of $\mathcal{C}_1$. Thus, no event model interfaces nor adaptation functions are needed (unnecessary *EMIF*s and *EAF*s are indicated by the gray color), parameter propagation is sufficient:

$$T_{Z_{1,1}} = T_{X_1}, \quad b_{Z_{1,1}} = b_{X_1}, \quad t_{Z_{1,1}} = t_{X_1}$$

This does not apply to the event models X_2 and $Z_{1,2}$. From Table 2, we see that we can use the burst event model ($Z_{1,2}$) to capture the periodic (X_2) event stream coming from $\mathcal{P}_2$:

$$T_{Z_{1,2}} = T_{X_2}, \quad b_{Z_{1,2}} = 1, \quad t_{Z_{1,2}} = T_{X_2}$$

Now, the input event models for the communication channel are known, and the channel packet response can be calculated using an appropriate analysis technique, such as a properly adapted version of the analysis of Tindell [21] or Lehoczky [12]. The only analysis results required are the output event models (Z_2). In our example, these are known to be bursts. From Table 2, we see that a simple *EMIF* is sufficient to transform the burst event model $Z_{2,1}$ into the sporadic event model Y_1:

$$t_{Y_1} = t_{Z_{2,1}}$$

Again, we do not need an event adaptation function. However, the interface $EMIF_{Z_{2,1}\to Y_1}$ is lossy, i. e. not the complete information of event model $Z_{2,1}$ can be captured by event model Y_1. A more complicated situation is found for the last event interface. Here, a burst event model needs to be transformed into a periodic one. As mentioned in Section 3.2, this can be achieved by adding a buffer and a timer to periodically output the buffered events. This way, the burst event stream is synchronized to a periodic stream. Table 4 provides us with the required *EMIF* and *EAF*:

$$T_{Y_2} = \frac{T_{Z_{2,2}}}{b_{Z_{2,2}}}$$

The output issue rate (T_{EAF}) and the required buffer size (n_{EAF}) to temporarily store events during a burst is given by

$$T_{EAF} = \frac{T_{Z_{2,2}}}{b_{Z_{2,2}}}, \quad n_{EAF} = b_{Z_{2,2}} - \left\lceil b_{Z_{2,2}}(b_{Z_{2,2}}-1)\frac{t_{Z_{2,2}}}{T_{Z_{2,2}}}\right\rceil,$$

and the maximum buffering delay for each event is:

$$d^{+}_{EAF} = T_{Z_{2,2}} - (b_{Z_{2,2}}-1)t_{Z_{2,2}}$$

In this example, we demonstrated the event model coupling technique in different situations. Trivially, event models might already match, and no interfaces are needed, as for the streams X_1 and $Z_{1,1}$. For $X_2\to Z_{1,2}$ a simple *EMIF* was sufficient to transform the event stream without losing modeling accuracy. This was not possible for the coupling $Z_{2,1} \rightarrow Y_1$, but we could only find a lossy $EMIF$. For the transformation $Z_{2,2} \rightarrow Y_2$, no plain $EMIF$ could be found, but a buffer and a timer (EAF) were required for re-synchronization.

5. Conclusion

We presented a technique and rules for event model transformations in order to couple local analysis models for system-level scheduling and buffer memory analysis. We started with a classification of known and practically used resource sharing strategies and their respective analysis techniques. We showed that the analysis techniques heavily depend on the input event model. The more restricted event models, such as periodic events, typically obtain tight analysis bounds allowing higher system optimization. Event model incompatibilities prevent the application of such efficient analysis techniques in global system analysis.

The event stream interface model introduced in this paper provides functions for efficient event model coupling. We showed in which cases exact transformations without loss of information is possible, where accuracy is lost because the subsequent analysis uses a less expressive model, and how we can use an event model adaptation function to derive a more restricted event model from a more general one. Event model adaptation uses buffering and time triggering which reflects what a designer would do in manual design. We have shown how to derive safe buffer sizes for event model adaptations.

In summary, the event model coupling presented in this paper is an enabling technique to system-level scheduling, timing and buffer memory size analysis. It allows to keep the efficiency in subsystem design without compromising global system analysis. A tool for (semi-)automated timing and buffer memory analysis is currently under development as part of the SPI (System Property Intervals) workbench [4], a tool suite for reliable and optimized integration of embedded real-time systems on heterogeneous architectures.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Journal of Real-Time Systems*, 8(5):284–292, 1993.

[2] F. Balarin. STARS of MPEG decoder: a case study in worst-case analysis of discrete-event systems. In *Proc. 9th int. Symposium on Hardware/software codesign CODES'01*, pages 104–108, Longangstraede, Denmark, April 2001.

[3] Cadence. *Cierto VCC Environment*. http://www.cadence.com/products/vcc.html.

[4] R. Ernst, D. Ziegenbein, K. Richter, L. Thiele, and J. Teich. Hardware/software codesign of embedded systems - The SPI Workbench. In *Proceedings IEEE Workshop on VLSI*, Orlando, USA, June 1999.

[5] Christian Ferdinand and Reinhard Wilhelm. On predicting data cache behavior for real-time systems. In *Proceedings of International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 16–30, 1998.

[6] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[7] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123, Oulu, Finland, 1993.

[8] Klaus Gresser. *Echtzeitnachweis ereignisgesteuerter Realzeitsysteme*. PhD thesis, only avaliable in German, Technische Universität München, 1993.

[9] A. Hergenhan and W. Rosenstiel. Static timing analysis of embedded software on advanced processor architectures. In *Proceedings of Design, Automation and Test in Europe (DATE '00)*, pages 552–559, Paris, March 2000.

[10] H. Kopetz and G. Gruensteidl. TTP - a time-triggered protocol for fault-tolerant computing. In *Proceedings 23rd International Symposium on Fault-Tolerant Computing*, pages 524–532, 1993.

[11] E. A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), January 1987.

[12] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings Real-Time Systems Symposiom*, pages 201–209, 1990.

[13] Yau-Tsun Steven Li and Sharad Malik. *Performance Analysis of Real-Time Embedded Software*. Kluwer Academic Publishers, 1999.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[15] Mentor Graphics. *Seamless Co-Verification Environment*. http://www.mentorg.com/seamless/.

[16] OSEK Steering Committee. *OSEK Open systems and the corresponding interfaces for automotive electronics*. http://www.osek-vdx.org/.

[17] Philips Semiconductors. *Controller Area Network CAN*. http://www.semiconductors.philips.com/can/.

[18] P. Pop, P. Eles, and Z. Peng. Bus access optimization for distributed embedded systems based on schedulability analysis. In *Proc. Design, Automation and Test in Europe (DATE 2000)*, Paris, France, 2000.

[19] D. Ramanathan, A. Dasdan, and R. Gupta. High-level modeling of communication in real-time embedded systems. In *In Proceedings IEEE High-Level Design Validation and Test Workshop (HLDVT)*, pages 172–180, November 1998.

[20] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Journal of Real-Time Systems*, 1(1):27–60, 1989.

[21] K. W. Tindell. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, Mar 1994.

[22] F. Wolf and R. Ernst. Intervals in Software Execution Cost Analysis. In *Proceedings of IEEE/ACM International Symposium on System Synthesis*, pages 130–135, Madrid, Spain, September 2000.

[23] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11), November 1998.