Combining Complex Event Models and Timing Constraints

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

Abstract

Sophisticated models of event streams including jitter and bursts as well as the possibility to specify a variety of system-level timing constraints are prerequisites for modern analysis and synthesis techniques in the area of embedded real-time systems. Currently, there is no commonly used specification that models events and timing constraints in a sufficiently general way. In this paper, we first identify a duality between event models and timing constraints and as a result present a specification that can be used for both. Our specification covers most current analysis and synthesis techniques and is easily extensible. We then show how the duality between event models and timing constraints can be applied at different points in a design flow. A real-time video transmission is used as an example.

1 Introduction

Embedded real-time systems operate in an environment with which they interact, e.g. by sending or receiving signals or data. Furthermore, the environment imposes timing constraints such as execution rates or input-output latencies on the system. Each interaction of system and environment can be viewed as an event. Depending on the characteristics of events and timing constraints, different design decisions have to be taken. Both the modeling of events and the specification of timing constraints are thus integral parts of the specification of embedded real-time systems.

Different system-level analysis techniques, in particular schedulability tests, and synthesis techniques such as scheduling or partitioning strategies, make different assumptions about event models, timing constraints as well as the target architecture. There is no single analysis or synthesis technique that can deal with the full complexity of timing constraints, event models and target architectures. The combination of multiple techniques is thus becoming increasingly important due to the growing embedded system complexity and heterogeneity. To facilitate this combination, a specification for event models and timing constraints is desirable, which is expressive enough to cover the different assumptions made by different techniques. However, currently there is no commonly used specification for this purpose.

In this paper we first show a duality between system-level timing constraints and event models. This allows us to use a common specification for both, which, depending on the designer's viewpoint, is interpreted either as timing constraints or event models. Our specification satisfies the following 3 requirements.

(1) Specification should happen at a natural level of abstraction. The designer should be able to describe what he wants, not how to achieve it. This requires a clean separation between timing constraints and event models on one hand, and timing mechanisms such as clocks on the other. (2) The specification should be expressive enough to cover a variety of event and timing constraint models employed by current system-level schedulability tests, scheduling techniques for Real-Time Operating Systems (RTOS) and hardware/software partitioning techniques. Combination of different techniques should be supported to overcome the limited applicability of each, in order to allow modeling, analysis and implementation of complex, heterogeneous systems. (3) The specification of event models and timing constraints should not depend on languages used for system design.

The remainder of the paper is structured as follows. After an overview of related work, we compare event models and timing constraints in Sec. 3. Based on this comparison, a duality between event models and timing constraints is identified in Sec. 4, and a common specification is presented. In Sec. 5, three classes of timing constraints respectively event models are introduced, which together are sufficiently expressive for most modern analysis and synthesis techniques. System-level design flow issues are discussed in Sec. 6. This is followed by a video transmission example in Sec. 7 and a conclusion.

2 Related Work

A variety of schedulability tests, scheduling techniques for RTOSes and (co)synthesis methods exist that target real-time systems. Their combined purpose is to allow conservative, but as accurate as possible analysis of the system behavior, and to base synthesis decisions on this analysis. Each of these techniques has (among others) restrictions in terms of timing constraint classes and event model classes that can be considered, and thus has limited applicability.

Rate monotonic scheduling (RMS) [5] is a well-known real-time scheduling algorithm. It assumes periodic task arrival times, no data-dependencies between tasks, requires task deadline to be at the end of their period and provides worst-case response times for each task. Lehoczky [4] provides a more general analysis of single-CPU static-priority systems with arbitrary deadlines. In the analysis by Yen and Wolf [9] periodic tasks running on a distributed system are assumed, where a task has a fixed period and deadline and consists of processes with data-dependencies. This work considers best-case values for process execution times in addition to worst-case values to capture scheduling anomalies.

Other work, e.g. by Tindell [8] acknowledges the fact that realistic systems exhibit communication jitter and bursts. Tindell considers periodic tasks with arrival-time jitter, as well as sporadically periodic tasks that are executed as a result of bursts of arriving events. However, no data-dependencies are allowed, and like in [4] the analysis is limited to single-CPU static-priority systems. In [6], arrival time jitter and upper and lower bounds on response times are considered. Formulas for event propagation throughout a system are presented.

The co-synthesis systems COSYMA [7] and VULCAN [2] consider periodic input events and periodic output constraints as well as worst-case input-output delay constraints.

Recently, we presented a small, yet powerful set of parameterizable system-level timing constraints [3]. We use them as base for the specification of event models and timing constraints in the work presented in this paper. This work is conducted as part of the **SPI** (**S**ystem **P**roperty **I**ntervals) project [1], which has the goal to enable global system analysis, in order to allow reliable and optimized implementation of heterogeneously specified embedded real-time systems on heterogeneous architectures.

3 Event Models versus Timing Constraints

An embedded system is designed to interact with a certain environment while obeying certain timing constraints (and other types of constraints). Throughout this paper, we use the general term 'event' to refer to any form of communication. In most cases, event timing cannot be specified exactly. Likewise, the specification of exact timing constraints typically over-constrains the design space and in many cases is not implementable at all. The approach we follow is therefore to specify timing constraints and event models using intervals.

A system may be a subsystem of a larger function, in which case the environment includes other subsystems which have already, or are yet to be designed. The following distinctions and relationships between system, event models and timing constraints hold independent of the level of granularity of the design.

Event models constrain the range of *possible* timing of the environment of a system.

A correctly implemented system must be able to deal with the full range of possible event timing. Since such an implementation is conservative, it will typically be able to deal with a larger range of event timing than specified in the event model, without violating its constraints.

Event models are primarily of concern for the design of timing mechanisms at the input of the system. They can also influence output timing, if the event is modeled, that the environment reads data from the system.

Timing constraints constrain the range of *acceptable* timing of the system itself.

A correctly implemented system must not exhibit timing behavior outside the range of its timing constraints. Since such an implementation is conservative, the range of its timing behavior will typically be smaller than the range of its timing constraints.

Timing constraints stem from requirements about the externally observable timing behavior of a system. They are used both to constrain timing of individual system parts (e.g. execution rates, execution time jitter), and of groups of system parts (e.g. latency along a path, synchronized execution times). They are primarily relevant for output timing, but can be useful for input timing, e.g. to sample a continuous value within certain time intervals.

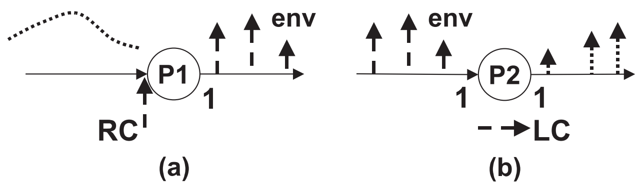

Figure 1. Input event model and output timing constraint (a); Output event model and input timing constraint (b)

As two simple examples, consider Fig. 1. In **(a)**, process **P1** is sampling a continuous signal. The sampling rate, which may have a maximum allowable jitter, is enforced using an execution rate constraint **RC**[1] at its input. The output event stream indicates periodic reading of the system output by the environment **env**. This event model determines the valid output timing of the process.

On the other hand in **(b)**, we have modeled a periodic event stream **env** at the input of process **P2**. **P2** performs some transformation and writes one output for each input. We have also specified an input-output latency constraint **LC** which constrains the timing of each output event relative to the corresponding input event. If **LC** is an interval, the output event stream does not have to be equally spaced, as indicated in the diagram.

Note that in **(a)**, **RC** could be re-interpreted as a timer event with a maximum allowable jitter at the input of **P1**, while **env** leads to an input-output latency constraint in analogy to **(b)**. Such re-interpretation is always possible as will be shown in Sec. 4.

4 Common Specification

In this section we show how to use a common specification for event models and timing constraints. We do not make any assumptions about the language used for the system specification and merely require that some way to annotate timing constraints to system elements exists. In our line of thought, we model the environment of the system as abstract event sources and event sinks.

We now apply timing constraints to those event sources or event sinks and interpret this in the following way. We assume that *an abstract event source or event sink always satisfies its timing constraints*. However, since it is not part of the system we are implementing, we have no additional knowledge about its timing behavior. Consequently, also the event timing between the event sources or event sink on one hand, and the system on the other is neither known nor can be assumed to be time-invariant. Therefore, the full range of event timing within the constraint boundaries of the event sources or event sink has to be assumed during system design. This is re-interpreted as the desired event model.

It is thus a matter of interpretation, whether a particular construct is a timing constraint or an event model. The interpretation is not tied to the notion of system process and abstract event source or event sink that we used in our line of thought. As long as some means to determine the appropriate interpretation is provided, a common specification can be used.

We illustrate our line of thought in Figs. 2 and 3. Fig. 2 shows the system from Fig. 1 **(b)** extended by an additional

[1]Constraint classes will be introduced in detail in Sec. 5.

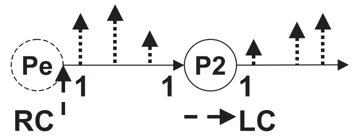

Figure 2. Event model from Fig. 1 (b) re-interpreted as timing constraint

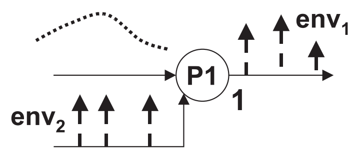

Figure 3. Timing constraint from Fig. 1 (a) re-interpreted as event model

process **Pe** that models the abstract event stream source (indicated by the dashed line). The event stream is enforced by the execution rate constraint **RC** at its output, which replaces the event model **env** in Fig. 1 **(b)**.

The re-interpretation also works in the opposite direction. In Fig. 3, the **RC** from Fig. 1 **(a)** has been re-interpreted as a periodic event **env**$_2$, which triggers the sampling of the continuous signal by **P1**. The maximum allowable jitter of **RC** becomes a maximum possible jitter of **env**$_2$ in this re-interpretation.

5 Constraint- and Event-Classes

A survey of modern system-level analysis and synthesis techniques (Sec. 2) suggests to use three classes of event models respectively timing constraints. Theses three classes have been presented in [3] and used as a specification for system-level timing constraints. The specification is at a natural level of abstraction, independent of the languages used for system specification, and can easily be extended if needed. We give a brief summary here. The re-interpretation as event models is immediate following the line of thought from Sec. 4.

- *Rate Constraints* are used to constrain the starting and completion time of processes, or the read or write time at a process input or output. An interval can be specified both for the period and the period jitter. A rate constraint can be used e.g. to specify relaxed periodic activation by setting the period to a fixed value and allowing for a jitter interval, or sporadic activation with minimum distance by setting the upper period limit to infinity.

- *Burst Constraints* can be used where rate constraints are not powerful enough. A typical example is sending data over a network which generates a burst of packets. An interval can be specified for the outer rate, the inner rate and the burst length. This modeling follows the definition by Tindell [8].

- *Latency Path Constraints* (also known as input-output delay constraints) are used to limit the time for causally-dependent process executions along a certain path. A latency interval can be specified.

No single analysis or synthesis technique currently supports the whole range of expressiveness. This has implications on the design flow as will be shown in Sec. 6.

6 Design Flow Issues

Divide-and-Conquer Design Flow The common specification of timing constraints and event models identified in Sec. 4 seamlessly integrates into a divide-and-conquer design flow, which is typical for larger designs. Assume an embedded system consisting of severals subsystems which are designed separately. The following relationships between timing constraints and event models at the boundary between subsystems can then be exploited.

- *Forward between subsystems* Timing constraints which influence possible event timing at the output of the writing subsystem can be treated as event models at the input of the reading subsystem. This is analogous to the re-interpretation used in Figs. 2 and 3.

- *Backward between subsystems* Timing constraints at the input of the reading subsystem influence the valid event model at the output of the writing subsystem. In the next step this model can be re-interpreted as timing constraints for the writing subsystem as described in Sec. 3.

Design Iterations In an iterative design flow, design decisions lead to narrowing of event model parameters. This in turn leads to relaxation of related timing constraints.

The designer's perspective on what constitutes the system and what constitutes the environment can switch frequently during design iterations. With every switch, timing constraints become event models and vice versa, simply by re-interpreting the specification as described in Sec. 4, thus facilitating iterative design.

Both aspects are shown in the following example. The system in Fig. 4 consists of an event source $\mathbf{Pe}_1$ with a relaxed periodic rate constraint **RC**. On average, events arrive at process **P1** periodically with period **p**, but each event can be delayed for some time within the jitter interval **j**. The output events of process **P1** are buffered in buffer **B**, from where they are read periodically with period **p** by the event sink $\mathbf{Pe}_2$. A latency path constraint **LC** constrains the minimum and maximum delay between event source and event sink.

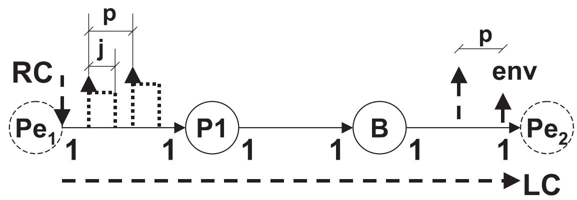

Figure 4. System with jittery event arrival is read periodically by an event sink. The system has to satisfy latency path constraint LC

A conservative implementation of **P1** has to assume worst-case event arrival time at its input, and has to be fast enough to always produce output events in time. This leads to a maximum latency constraint for **P1** (not shown). Output events produced too early are buffered in **B** to avoid violating the minimum delay of **LC**.

The designer now changes his focus and performs a design decision for $\mathbf{Pe}_1$, which narrows the jitter **j** for event arrival at the input of **P1**. This means that **P1** now has more time left to execute its function in the worst case. Therefore, the upper bound of its latency constraint interval can be increased.

Technique Limitations As mentioned in Sec. 2, event models, timing constraints, and analysis and synthesis techniques are strongly related. Therefore, the event models selected by the designer may be incompatible with those needed for analysis.

To give an example, consider a process that is activated by a bursty event stream, but the known analysis techniques for the process assume periodic models. In such cases, the design has to be modified e. g. using additional buffers to smoothen the bursts of events, at the cost of increased latency. This has to be considered in the already mentioned iterative design flow. Not only the parameter values of the models might change during iteration, but also the type of model itself. Design modification in order to enable timing analysis is thus a trade-off to design optimization. Future work includes interfacing between different event models to automatically derive the required design modifications, rather than leaving this problem to the designer.

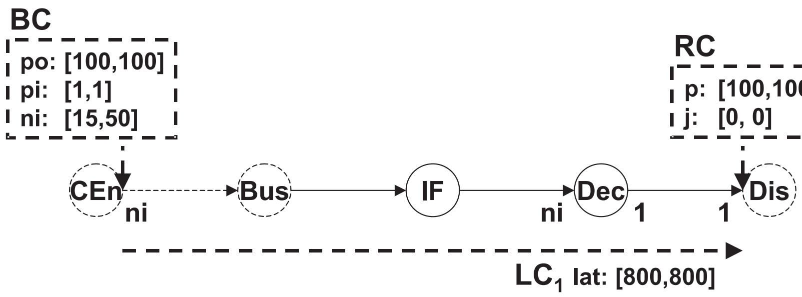

Figure 5. Video transmission example

7 Example

Our application example is a hypothetical model of a real-time video transmission shown in Fig. 5. We are designing the bus interface **IF**, which reads packets from a bus, and the video decoder **Dec**, which generates one video frame from **ni** packets. The camera and video encoder **CEn**, the bus **Bus** and the display **Dis** are considered the environment of the system (indicated by dashed lines). The bus is shared with other traffic (not shown).

Each output events of **CEn** represents a packet. The generation of packets from a video frame is modeled using a burst constraint. The outer period **po** is 100 time units, which might represent $\frac{1}{30}$ of a second, a typical frame rate in video transmission. The inner period **pi** is 1 time unit, and the burst length **ni** is an interval between 15 and 50 packets per frame[2]. The display **Dis** reads the output of the decoder periodically with **p** $= 100$ time units (i.e. equal to the rate, at which frames are generated). A latency path constraint **LC**$_1$ of 800 time units between the encoder and the display is specified. The value is equal to the maximum delay required for the real-time video transmission in our application. The lower bound has to be set to the same value as the upper bound, otherwise it is possible that a frame will be displayed more than once or not at all. Of course, we cannot assume that decoder **Dec** will always produce frames exactly at the right time. Therefore, frames produced too early have to be buffered.

We now assume that the **Bus** delays each packet between 300 and 700 time units, depending on traffic. This information can be modeled as a latency path constraint **LC**$_2$ over the bus as shown in Fig. 6. As explained in Sec. 4, a timing constraint applied to an element modeling the environment is interpreted as an event model. Combined with the information about the bursty output events of **CEn**, event arrival at the input of **IF** can now be modeled.

Assume that we want to implement **IF** in hardware and **Dec** in software, and that we want to schedule **Dec** together with some other processes (which for simplicity are not shown in the diagram). The timing analysis techniques we have available require a periodic activation of **Dec** to be applicable. It is therefore necessary to buffer the bursty output of **IF** so that **Dec** can be activated periodically. We have to introduce a packet buffer as described in Sec. 6, Technique Limitations. As a benefit, periodic activation of **Dec** will probably also reduce the size of the frame buffer between **Dec** and **Dis**, potentially to one frame if the packet buffer is large enough that all waiting can happen there, and if some synchronization mechanism between **Dec** and **Dis** is provided. This is efficient, since the frame has already been de-compressed by **Dec** and thus consumes more memory than the corresponding packets.

[2] In reality those numbers would be different.

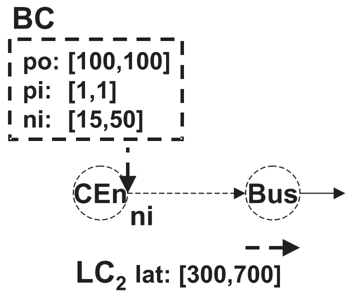

Figure 6. Video transmission example including packet delay on the bus

Let us assume that the implementation of **IF** in hardware leads to a constant input-output delay of $d_{IF} = 1$ time unit for every packet received. The size of this delay, together with **LC**$_1$, **LC**$_2$ and the properties of the encoder output, allow us calculate the available execution time **LC**$_3$ for the decoder. The worst-case arrival time for all packets needed for one frame at the input of the decoder is

$$(ni_{max}-1)*pi_{max}+LC_{2_{max}}+d_{IF} = 49+700+1 = 750$$

time units. The decoder **Dec** thus has only 50 time units in the worst case to execute from the moment the last packet of a frame arrives. Let us now assume that worst-case execution time analysis determines that the decoder cannot be scheduled to satisfy its deadline. In this situation, the analysis can be repeated with later deadlines for **Dec**, until one is found that can be satisfied. For the whole transmission to work, packets simply have to arrive earlier at the input of the decoder by a time equal to the difference between the two deadlines.

One solution to this new timing constraint is to increase the priority of the video packets on the **Bus**, such that their maximum delay is sufficiently reduced. To perform this change, the designer has to switch the system/environment roles.The bus becomes the system, bus interface and decoder become the environment. The scheduling technique identified for **Dec** leads to an event model which specifies the time intervals, during which the decoder may read packets. Providing the packets in time becomes a timing constraint for the output of the bus.

8 Conclusion

In this paper, we highlighted the fact that currently there is no methodology to model events and system-level timing constraints in a sufficiently general way in the area of embedded real-time systems. However, sophisticated models of event streams including jitter and bursts as well as the possibility to specify a variety of system-level timing constraints are prerequisites for modern analysis and synthesis techniques. As a step towards a suitable model, we first identified a duality between event models and timing constraints and as a result presented a specification that can be used for both. We then discussed design flow issues and issues that arise when combining different analysis and synthesis techniques with restricted assumptions about event models and timing constraints. The feasibility and application of our approach was shown using the design of a video transmission as an example.

In the future, we are planing to practically demonstrate our approach in the context of the SPI project. While the design-flow is understood, the biggest challenge will be to apply analysis techniques with restricted assumptions about event models and timing constraints to a system specification that initially does not match those assumptions. Closing the gap between specification and analysis provides many research opportunities to improve on existing techniques, in particular those that consider jitter and bursts. Future work also includes interfacing between different event models to automatically derive the required design modifications, rather than leaving this problem to the designer.

References

[1] 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.

[2] R. K. Gupta and G. De Micheli. Specification and analysis of timing constraints for embedded systems. *IEEE Trans. CAD*, March 1997.

[3] M. Jersak, D. Ziegenbein, and R. Ernst. A general approach to modeling system-level timing constraints. In *Proceedings 4th Forum on Design Languages*, Lyon, France, 2001.

[4] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings Real-Time Systems Symposiom*, pages 201–209, 1989.

[5] C. L. Liu and J. W. Layland. Scheduling algorithm for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20, 1973.

[6] K. Richter, R. Ernst, and W. Wolf. Hierarchical specification methods for platform-based design. In *Proc. of Tenth Workshop on Synthesis And System Integration of MIxed Technologies (SASIMI 2001)*, Nara, Japan, 2001.

[7] A. Österling, Th. Benner, R. Ernst, D. Herrmann, Th. Scholz, and W. Ye. *Hardware/Software Co-Design: Principles and Practice*, chapter The COSYMA System. Kluwer Academic Publishers, 1997.

[8] 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.

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