On the Specification of Real-time Properties of Streaming Networks *

Raimund Kirner, Simon Maurer

University of Hertfordshire
r.kirner, s.maurer@herts.ac.uk

Abstract. Cyper-physical systems (CPS) are networked embedded systems, having often real-time requirements for individual control tasks. The complexity of CPS due to concurrency can be reduced by modelling it as a streaming network, providing an implicit local synchronisation mechanism.
In this paper we show that specifying real-time requirements for such streaming networks is not straight forward. Especially specifying latency is challenging due to their global context from which they arise. Analysing models at requirements and system design level, we provide practical solutions to the specification of the timing behaviour of such models of streaming networks.

1 Introduction

Streaming networks consists of processing nodes connected via communication channels. Since this communication channels have a single reader and a single writer, streaming networks are a well-recognised for their benefit of coping with the complexity of concurrent systems.

This strength of streaming networks makes them also an interesting paradigm to apply to cyber-physical systems (CPS). CPS are networked embedded systems, thus exposing naturally a high level of concurrency [9]. At the same time CPS have often real-time requirements, often for local subsystems, but sometimes also at a global level.

Lee has proposed coordination languages as a means to cope with the complexity of CPS [9]. We support this observation and also propose to use the streaming network paradigm such the underlying concept for such a coordination language.

In this paper we do not focus on the overall design of a coordination language well suited for CPS. But rather, we focus on the underlying streaming network paradigm and discuss how it suits the specification of real-time requirements. As we show in this paper, it is not so easy to specify real-time properties for streaming networks in a simple and resource-efficient way.

In Section 2 we study the specification of real-time properties for real-time systems of simple structure, exposing the difference between modelling timing behaviour at requirements level and at system design level. In Section 3 we show the challenges that

* This work was partially supported by COST Action IC1202: "Timing Analysis On Code-Level" (TACLe). The research leading to these results has received funding from the FP7 ARTEMIS-JU research project "ConstRaint and Application driven Framework for Tailoring Embedded Real-time Systems" (CRAFTERS).

arise when aiming to annotate real-time properties in streaming networks and propose concrete solutions. Section 4 shows some examples of streaming networks applied to CPS. A selection of related work is discussed in Section 5. Section 6 concludes the paper.

2 Development of Real-time Systems

In order to expose the challenge of specifying real-time properties of streaming networks we first review some fundamentals of developing real-time programs. There are basically two extreme contexts for modelling extra-functional behaviour like real-time requirements:

1. a high-level model that describes how the system is expected to fit into the environment. We call such a model a requirements specification or *requirements model*.
2. a low-level model that describes how the system is designed, with various levels of detail. We call such a model a *design model*.

Depending on the concrete software development processes of specific application domains, there are more fine-grained distinctions of modelling levels. However, for the sake of simplicity, we focus only on these two fundamental ones. Furthermore, in the following we discuss the timing-related aspects of such system models.

To understand modelling of real-time systems from its fundamentals, we assume a simple system structure where a particular services is expected to work on input, derived from sensors, to produce an output for an actuator. This fundamental structure is shown in the top of Figure 1.

Focusing on the system requirements we derive a requirements model. In the context of real-time systems such a requirements model has to include the specification of extra-functional properties, especially timing requirements. The modelling aspects of timing requirements are shown in the bottom half of Figure 1. Based on the simple system structure we can express timing requirements as a tuple $\langle I_x, S_y, O_z, Treq_{x,y,z} \rangle$, meaning that each timing requirement $Treq_{x,y,z}$ spans from a particular input I_x to an output O_z, characterising a service S_y. In the timing domain these requirements typically include the processing rate and the latency of a service. The processing rate is sometimes also called throughput. The variation in rate or latency is called jitter, and can also be included in the timing requirements. While often jitter is exclusively associated with latency, we consider it also applicable to processing rate, especially for such real-time systems where the processing rate is more important than the latency, e.g., in video streaming applications without control loops.

To give an example of a timing requirement, we consider the maximum latency for a service, also called a *deadline*. Deadlines are called firm if the utility of a service abruptly drops after exceeding the deadline, otherwise it is called a soft deadline [8]. Figure 2 shows the specification of a deadline for a service S_y from input I_x to output O_z. It is important to include the data-flow path attached to the timing requirement, since the same service, for example, might also write to another output with a different deadline attached to it. The deadline shown in Figure 2 is relative, i.e., the maximum

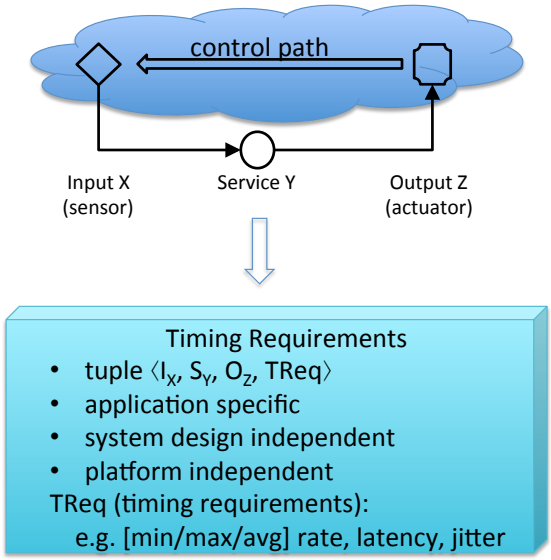

Fig. 1. Derivation of real-time requirements from application context

latency is measured against the trigger instant at input I_x. Each trigger instant of input I_x results into a different absolute deadline:

$$t_{deadline} = t_{input} + t_{deadline,rel}$$

Figure 3 shows the transition from the requirements model to the design model. While the requirements model is solely focused on the demands imposed by the application environment of the real-time control system, the design model shifts its focus to the details of how to build the system, resulting in a design-specific model. As shown in Figure 3, the design model can be expressed at different abstraction stages, ranging from an implementation-independent model to an implementation-dependent model. The implementation-dependent design model includes imposed properties like the choice of platform. But the implementation-dependent design model might also be enriched by annotations, derived from behaviour analysis, making behavioural properties, resulting from the implementation choices, explicit. Figure 4 shows a timing requirement of latency (response time) attached to the design model.

The fundamental difference between Figure 4 and Figure 2 is that the abstract specification of a service in Figure 2 has been replaced by a concrete processing node realising that service. The relation between services and processing nodes realising them is, in general, $n : m$ and not necessarily $1 : m$ or $n : 1$.

Here we want to stress that ideally a design model already reflects the real-time behaviour of the system at a platform-independent stage, if possible even at an implementation-independent stage. The benefit of a platform-independent design model is that the system behaviour can be reasoned about independently of the platform choice, making a

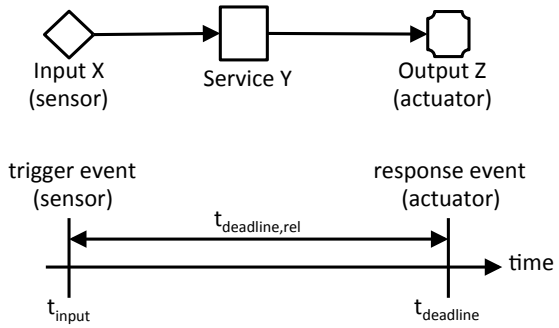

Fig. 2. Example of timing requirements: deadline (maximum latency)

correctness verification more robust against changes of the platform at a later development stage.

However, including the specification of the timing behaviour into the platform-independent design model also comes with a cost. We have to ensure, after the platform choice, that the specified timing behaviour is actually implementable with the chosen platform. While this is nothing surprising by itself, there is a fundamental difference of whether we have to fulfil the timing specification of the requirements model or whether we have to fulfil the timing requirements of an platform-independent design model. The latter may impose additional design-specific constraints that can rise resource constraints which are not imposed by the application context itself. In Section 3 this aspect is discussed in more detail within the context of timed streaming networks.

3 Specification of Complex Systems

In Section 2 we have discussed the specification of timing requirements of real-time systems with a very simple structure. In the context of networked cyber-physical systems we have to deal with much more complex application structures. In the following we discuss challenges of modelling extra-functional properties for both, requirements and design models.

3.1 Timed Requirements Models

One of the challenges of requirements modelling is that requirements have to be developed in a modular way in order to cope with complexity. Like the example shown in Figure 5, services of a system tend to be described in a cascaded way. We cannot use the simple concept of Section 2 where a service is used to characterise the information processing between the inputs and outputs of a system.

Instead of linking services directly to sensor inputs and actuator outputs, we have to use some form of system interfaces. Regardless of the concrete specification methods being used in practice, we abstract from them by using the generic concept of ports. Sensors, actuators, and individual services can now be characterised locally within the

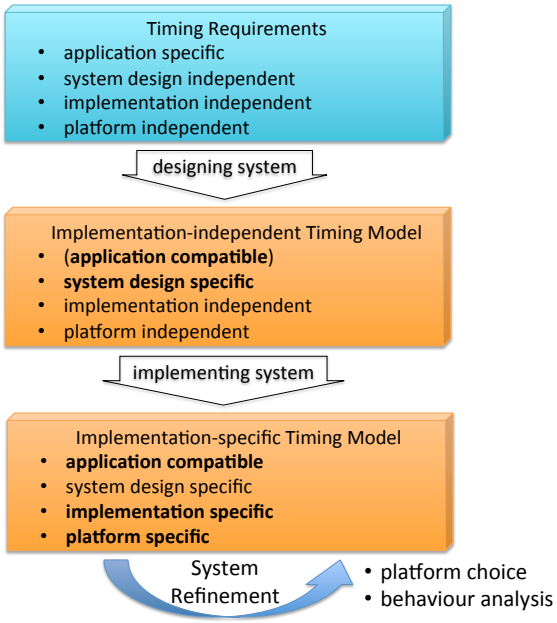

Fig. 3. Derivation of real-time model from application context and refined by analysis

perimeter of their interface ports p_i. Figure 5 shows the use of ports. Not only can the information input by a sensor now be discussed independently of its use, the figure also shows the characterisation of two services SV_1 and SV_2, which are cascaded, i.e., one input of SV_2 is not connected to a sensor but rather to the output of another service. SV_2 might rely on SV_1 on a rather weak basis to provide a refined quality of service, but it can be also the case that SV_2 strictly relies on SV_1 to provide any useful service at all.

The challenge of modelling timing requirements of cascaded services is that timing requirements in their purest form are imposed by the environment and are independent of any internal structuring of the computer system. For example, in Figure 5 there might be a certain relative deadline $d_{2,1}$ from sensor S_2 to actuator A_1. At the same time, there might be a relative different deadline $d_{1,1}$ from sensor S_1 to actuator A_1. The problem with the cascaded services S_1 and S_2 is that one cannot naturally assign them fractions of $d_{1,1}$ and $d_{2,1}$ without creating artificial constraints on the flexibility of the use of resources.

To characterise the latency requirements in their purest form one would have to use a kind of path-based characterisation of latency requirements. Instead of the approach in Section 2 where a timing requirement $Treq_{x,y,z}$ was associated with a context tuple $\langle I_x, S_y, O_z \rangle$, we would now have to match a timing requirement $Treq_p$ with a path specifier. A path specifier pth is a sequence $pth = (a_1, a_2, a_3, \ldots a_n)$ where each element a_i of the sequence is either a service, a port, or an IO node: $a_i \in SERVICES \cup PORTS \cup IO$. A path specifier may also only incompletely describe a path, by which

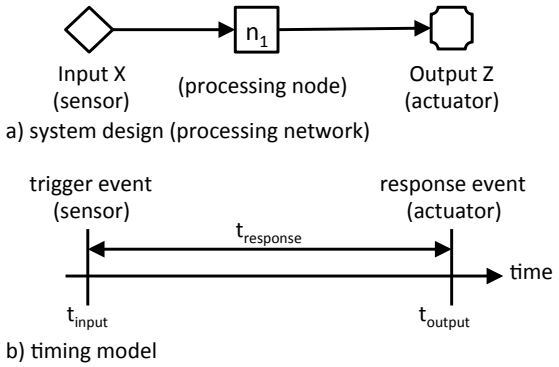

Fig. 4. Timing model derived from application context: Response time

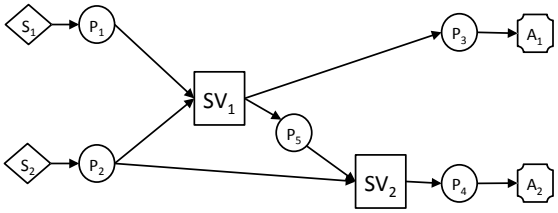

Fig. 5. Specification of multiple Services (Requirements Model)

multiple paths would match the specification. This can be used to assign a requirement to a group of paths. For example, the path specifier (S_2, P_2, P_4, A_2) matches with two concrete paths of Figure 5: $(S_2, P_2, SV_1, P5, SV_2, P_4, A_2)$ and $(S_2, P_2, SV_2, P_4, A_2)$.

The good news is that modelling requirements of processing rates can be done locally and propagated over the network. This approach, for example, is used in Simulink from Mathworks [10]. Nevertheless, clear semantic rules are needed to specify what happens at the interface between different processing rates [11].

Summarising, the challenge of specifying timing requirements for streaming networks is to express them in a pure form, i.e., only implied by the application environment and not by any internal system structuring decisions.

3.2 Timed Design Models

In this section we discuss the issues of specifying timing requirements for design models. As discussed before, the design model focuses on the behaviour of the realised system, where the realised system consists of inter-linked processing nodes with in the general case an $n : m$ mapping between services and processing nodes.

The challenge of how to describe latency in case of cascaded processing nodes is related with the challenge of specifying latency for cascaded services mentioned in Section 3.1. The additional challenge for the design model is that it tends to be more complex than the requirements model in case of individual services being realised by

multiple processing nodes. Figure 6 shows a streaming network with processing nodes fed by multiple sensors and contributing to the control of multiple actuators.

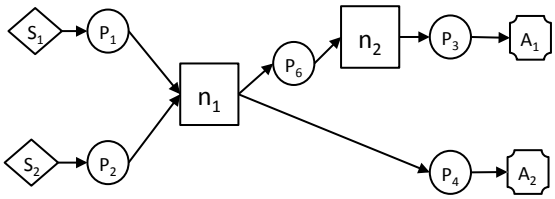

Fig. 6. Processing network with multiple sensors and actuators (Design Model)

What we ultimately want to obtain is a design model which has a clear timing semantics, i.e., it is known what the timing behaviour of different components will be, at least at a certain course granularity level. This is an aim somehow similar to the idea of the *Precision Timed Machine* (PRET) whose machine code has a timed semantics [7, 6]. However, there is a slight difference. With PRET the focus is on a well-specified processor platform with built-in timing semantics. The machine code for PRET would be a platform-specific design model of the computer system.

However, with our ambition for design models of streaming networks we would like to have a platform-independent design model with well-specified temporal behaviour. This means, we would like to know how the system is going to behave, regardless of the implementation details and even more, regardless of the chosen hardware platform. With streaming networks we have the challenge that at a particular position of the network messages originating from different sources can pass through, thus we generally cannot assign the processing latency of a path to any particular place in the network. So if we want to specify the timing behaviour directly at the design model rather than having a separate list of timing constraints, we would have to split the overall latency into multiple local latency values, assigned to individual sections of the streaming network.

For the split latency values there are two different semantics possible:

Summative latencies: the absolute latency along a path from the input to the output is the sum of all the local latency values along that path. Summative latencies do not require a platform-specific design model, so they can be also specified for a platform-independent design model.

Local absolute latencies: each local latency value describes the absolute local latency along a certain subsection of a processing path from the system input to the output. Absolute local latencies require a platform-specific design model, so they cannot be specified for a platform-independent design model.

To be most descriptive, the local latencies have be both, summative latencies as well as local absolute latencies.

To give an example, we assume that the streaming network shown in Figure 6 has as requirement the following absolute latencies from sensor input to actuator output:

	A_1	A_2
S_1	10 ms	4 ms
S_2	10 ms	4 ms

Having only a platform-independent design model, we can still decompose these end-to-end latencies into summative latencies and map them to the streaming network of Figure 6. Figure 7 shows summative latencies for the given example mapped to the streaming network. In this small example we have been able to derive the summative latencies manually. For larger graphs a systematic mapping method would be necessary to use.

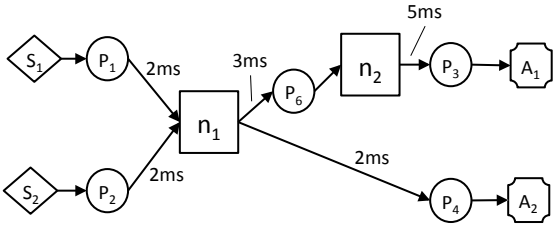

Fig. 7. Processing network with timed semantics (Design Model)

Using such summative latencies for a platform-independent design model are adequate to specify the end-to-end latencies by means of local annotation. However, it would be problematic to interpret them as absolute latencies for local sections of the streaming network. By doing so we would impose additional synthetic resource constraints for implementation and platform choice, not justified by the requirements. Thus our proposal is that for the platform-independent design model we interpret the local latency specifications in general only as summative latency specifications.

From Summative Latencies to Local Absolute Latencies As soon as we have derived a platform-specific design model, we are able to use performance or worst-case execution time (WCET) analysis to refine the model and replace the original summative latencies by another set of latencies that at the summative level are equivalent to the summative latencies of the platform-independent design model, but now also have a local absolute latency semantics.

Using such a refinement step towards the platform-specific design model avoids the introduction of synthetic resource constraints while still providing a fundamental timing semantics at the platform-independent design model. This approach is summarised in Figure 8.

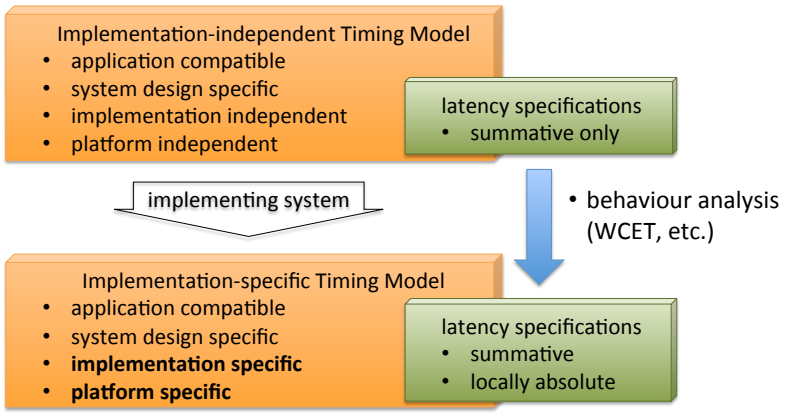

Fig. 8. Derivation of platform-specific semantics for latency

4 Examples of Real-time Streaming Networks

In the following, we show some applications of stream processing networks for real-time computing and also discuss specific issues implied by them which are relevant for modelling.

4.1 Fuel Injection

Figure 9 shows a grossly simplified model of a fuel injection system for internal combustion engines. Fuel injection is a real-time application with very strict timing requirements. Injecting the fuel to late or too early not only reduces the efficiency of the engine, but also increases the mechanical stress of the engine components, resulting in an acceptable outwear rate.

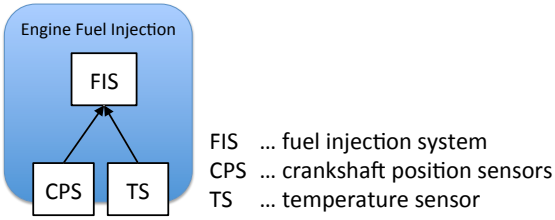

Fig. 9. Example of multi-rate system: Engine Fuel Injection

While any real fuel injection system is much more complex, our simplified model is sufficient to outline a relevant property when modelling it as a streaming network. The

fuel injection system FIS collects inputs from two sources: the current motor temperature from sensor TS and the current crankshaft position from sensor CPS.

The important aspect of this simple model is that we have different rates involved here. The CPS sensor has to deliver its trigger signal in fixed coupling with the crankshaft position, every revolution of the crankshaft. Without the CPS trigger signal available, the fuel injection cannot operate. Also the latencies involved along processing the crankshaft position have to be precisely taken into account. In contrast, the motor temperature from sensor TS has much weaker requirements. Neither the rate nor the latency of that sensor are very significant, since the temperature of the engine changes relatively slowly during correct operation.

4.2 Car Platooning

A car platooning (CP) system is a technology to line up vehicles on a highway into virtual trains, automatically controlling distance and speed [1]. In this section we purely focus on the influence of CPS to the brake control of a car. We want to highlight the different levels of criticality when it comes to brake control in a modern car.

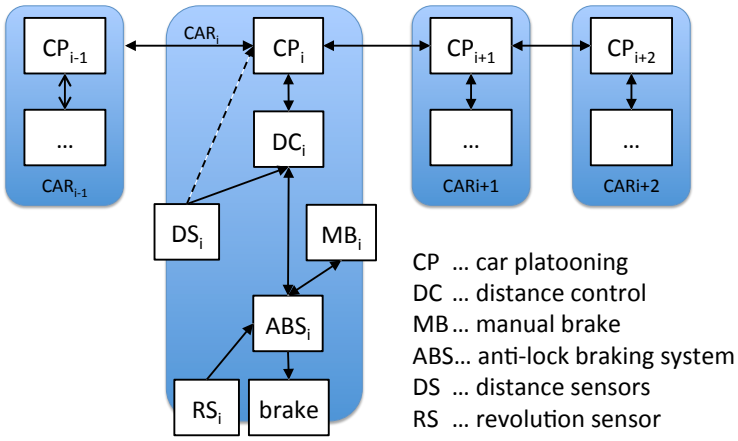

Fig. 10. Example of mixed criticalities: Car Platooning

Figure 10 shows the control chain of components in a modern car that can influence the activation of the brake. Quite standard nowadays is the anti-lock braking system (ABS) which has the highest control over the brake. The driver may activate the braking of the car with the manual brake (MB), but it is the ABS which finally decides when and how long the brake should be actually activated, giving priority to preserve steer-ability over short braking distance. To do so, ABS receives information about the current wheel revolution speed from sensor RS, and lowers the brake activation whenever the speed of a wheel drops. On top of MB acts the distance control (DC) system, which uses distance sensors (DS) in order to keep a minimum distance with other vehicles driving in front

of the car. On top of DC may be a car platooning (CP) system which, to some extent, behaves similar to DC by taking DS into account in order to control the distance to the font car. CP and DC can actually share the input from the same distance sensors of the car. However, CP senses the environment beyond that, being in active communication with the neighbouring vehicles on the road, allowing smoother operation by starting distance adjustment measures before even a change was noticed via the DS sensors.

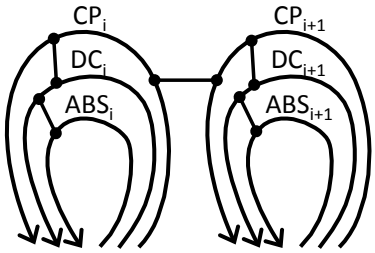

Fig. 11. Example of coupled control loops: Car Platooning

What this use case shows is that besides timing requirements there are also dependability requirements, resulting in different criticalities of the above services. The closer a component is to the brake, the higher is its criticality and the more control it has over the brake. Figure 11 visualises the different automatic control loops involved in that use case. Highest priority is given to ABS, as it is able to pause braking whenever it needs to do so in order to give priority to steer-ability. The manual break MB is not shown in that diagram, as it acts besides ABS since manual brake has to work even if ABS fails. DC has higher priority to CP regarding braking, since DC might detect a road obstacle while CP wants to accelerate in order to keep the distance with the vehicle driving in front of it.

This example demonstrates that in the system design, real-time requirements and criticality properties are orthogonal issues. Having the strongest real-time requirements has nothing to do with having the highest criticality in the system.

To summarise, real-time requirements are an important category of extra-functional properties, but there are other extra-functional requirements as well. So, scheduling resources may not simply be a real-time problem, but also a mixed-criticality problem.

5 Related Work

There are many approaches of modelling distributed systems with specification of extra-functional properties. In the following we present a sample of modelling approaches, including academic research and concrete tools.

An all-round modelling approach is the *Unified Modelling Language* (UML), which combines multiple modelling paradigms in a unified framework [12]. UML allows the modelling of a system at different abstraction levels. For example, with the UML *Use Case Diagram* (UCD) one can model the system application context without focusing

on implementation details and formal interfaces. Regarding the application of UML to real-time systems there are different approaches. For example, *Real-Time UML* describes how to model resources, time, and concurrency [5]. The *UML Profile for Modeling and Analysis of Real-time and Embedded systems* (MARTE) extends Real-Time UML with concepts for timed processing and timed events, introducing also logical and physical clocks as different time sources [13, 3].

The UCD of UML provides an interesting concept of how to model system requirements at a high level, focusing on the different services to be provided. However, when using the mechanisms of UML and its extensions to specify, one also has to face the problems discussed in this paper.

An example of academic approaches of how to model timed systems is *Ptides*, a variant of the *Ptolemy* execution model [4, 2]. Ptides and Ptolemy have been developed by Ed Lee et al. at Berkeley. Ptides is a stream-based event processing model with support to model extra-functional properties and time sources for real-time processing. Ptides allows to model the processing chains on so-called platforms and the communication between multiple platforms, resulting in distributed systems. With Ptides, one can annotate the extra-functional properties of different components, like delays to locally receive or send a message. In addition, one can set the "due date" of messages, i.e., the time at which an output at an actuator should be produced. While this due date is initially set to the message creation time, it can then be incrementally increased by delay blocks along the processing path toward its final destination at an actuator. There are no inherent rules of where to increment the due date by how much, as long as the total delay of the multiple delay blocks along the processing chain add up to the desired time for the output to be produced. By this delay blocks one can obtain a timed system model that has a fixed semantics of the event timings, regardless of the underlying platform. From that point of view, Ptides is well-suited to model the latency of real-time systems in a platform-independent way via summative local latencies as described in Section 3. So far, the published research on Ptides did not address the issue of how to derive platform-specific latencies, which besides summative latencies, would also provide local absolute latencies. As such, Ptides provides a very useful summative timing semantics at platform-independent design model level, but cannot escape the challenge of how to obtain local absolute latencies, as discussed in Section 3, which would require a platform-specific design model.

As an example of a modelling tool with wide-spread industrial use is Matlab/Simulink from Mathworks [10]. In Simulink one can specify processing graphs with multiple update rates of the different components. Simulink is well-prepared for modelling multi-rate systems by a specific "Rate Transition" block [10]. The Rate Transition block can be parameterised in order to trade *data integrity* and *deterministic transfer* for faster response or lower memory requirements. Simulink's focus on update rates works relatively well to address the problem discussed in this paper, but only from the throughput point of view. Simulink does not provide the same flexibility for modelling event latency, as, for example, Ptides is able to offer.

Kirner and Maurer have recently introduced an interface specification for components of stream-based mixed-criticality systems [11]. This model includes the specification of the progress type of components (time-triggered or some form of event-

triggered), but most importantly it offers an explicit way to specify trigger dependencies and trigger-decoupling of subsystems. They also introduced the classification of messages into event messages, state messages and semi-state messages in order to have a semantic justification of trigger coupling/decoupling [11]. Their mixed-criticality interface techniques could applied to other models like Ptides. While these mixed-criticality interfaces are useful to compose the timing behaviour from subsystem, this cannot completely remedy the timing specification problem discussed in this paper.

6 Summary and Conclusion

The specification of real-time properties in complex systems causes some challenges which we address in this paper. We have put our focus on streaming networks which are a well-suited design paradigm for cyber-physical systems with their omnipresent concurrent behaviour.

We have shown that it is not a straight forward process to specify real-time properties for streaming networks, neither at the requirements level nor at the system design level. More precisely, it is the specification of latency (or deadlines) that is not well suited for streaming networks, mostly because latency requirements are caused by the application environment and do not have a direct imprint at subsystem level. Throughput or processing rate on the other hand can be annotated to streaming networks relatively easy.

We resolve the situation by providing latency specifications with only summative semantics at the level of platform-independent design models. Using temporal behaviour analysis these latency specifications can be refined from a platform-specific design model into latency specifications, having both, a local absolute semantics as well as the original summative semantics at the global level. This result shows to what extent it is possible to specify platform-independent system design models with real-time semantics.

References

1. Carl Bergenhem, Henrik Pettersson, Erik Coelingh, Cristofer Englund, Steven Shladover, and Sadayuki Tsugawa. Overview of platooning systems. In *Proc. 19th ITS World Congress*, Vienna, Austria, Oct. 2012.
2. Janette Cardoso, Patricia Derler, John C. Eidson, Edward A. Lee, Slobodan Matic, Yang Zhou, and Jia Zou. *Systems Design, Modeling, and Simulation using Ptolemy II*, chapter Modeling Timed Systems, pages 355–393. Ptolemy.org, 1st edition, 2014. Available online at `http://ptolemy.org/books/Systems`.
3. Sebastien Demathieu, Frederic Thomas, Charles Andre, Sebastien Gerard, and Francois Terrier. First experiments using the UML profile for MARTE. In *Proc. 11the IEEE Int'l Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 50–57, May 2008.
4. Patricia Derler, Thomas Huining Feng, Edward A. Lee, Slobodan Matic, Hiren D. Patel, Yang Zhao, and Jia Zou. PTIDES: A programming model for distributed real-time embedded systems. Technical Report UCB/EECS-2008-72, EECS Department, University of California, Berkeley, May 2008.

5. Bruce Powel Douglass. *Real Time UML: Advances in the UML for Real-Time Systems (3rd Edition)*. Addison-Wesley Professional, 3rd edition, Feb. 2004.
6. Stephen A. Edwards, Sungjun Kim, Edward A. Lee, Isaac Liu, Hiren D. Patel, and Martin Schoeberl. A disruptive computer design idea: Architectures with repeatable timing. In *Proc. IEEE International Conference on Computer Design*. IEEE, 2009.
7. Stephen A. Edwards and Edward A. Lee. The case for the precision timed (pret) machine. In *Proc. 44th Design Automation Conference (DAC)*, San Diego, California, Jun. 2007.
8. Hermann Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2nd edition, 2011. ISBN: 978-1-4419-8236-0.
9. Edward A. Lee. Cyber physical systems: Design challenges. In *Proc. 11th IEEE International Symposium on Object-oriented Real-time distributed Computing*, pages 363–369, Orlando, Florida, USA, May 2008.
10. The MathWorks Inc., Natick, Massachusetts, USA. *Simulink Reference*, r2015a edition, March 2015. Revised for Simulink 8.5 (Release 2015a).
11. Simon Maurer and Raimund Kirner. Cross-criticality interfaces for cyber-physical systems. In *Proc. 1st IEEE Int'l Conference on Event-based Control, Communication, and Signal Processing*, Krakow, Poland, June 2015.
12. OMG. *Unified Modeling Language: Superstructure (version 2.1.1)*. Object Management Group, Feb. 2007. OMG document number: formal/2007-02-05.
13. OMG. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. Object Management Group, 1.1 edition, June 2011.