The Earlier the Better: A Theory of Timed Actor Interfaces*

Marc Geilen
Eindhoven University of Technology
m.c.w.geilen@tue.nl

Stavros Tripakis
University of California, Berkeley
stavros@eecs.berkeley.edu

Maarten Wiggers
University of Twente
m.h.wiggers@utwente.nl

ABSTRACT

Programming embedded and cyber-physical systems requires attention not only to functional behavior and correctness, but also to non-functional aspects and specifically timing and performance. A structured, compositional, model-based approach based on stepwise refinement and abstraction techniques can support the development process, increase its quality and reduce development time through automation of synthesis, analysis or verification. Toward this, we introduce a theory of timed actors whose notion of refinement is based on the principle of worst-case design that permeates the world of performance-critical systems. This is in contrast with the classical behavioral and functional refinements based on restricting sets of behaviors. Our refinement allows time-deterministic abstractions to be made of time-non-deterministic systems, improving efficiency and reducing complexity of formal analysis. We show how our theory relates to, and can be used to reconcile existing time and performance models and their established theories.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and Interfaces*; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design, Languages, Theory, Verification

Keywords

Actors, Dataflow, Compositionality, Throughput, Latency, Interfaces, Refinement

* This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET) and #0931843 (ActionWebs), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
HSCC'11, April 12–14, 2011, Chicago, Illinois, USA.
Copyright 2011 ACM 978-1-4503-0629-4/11/04 ...$10.00.

1. INTRODUCTION

Advances in sensor, actuator and computer hardware currently enable new classes of applications, often described under the terms *embedded* or *cyber-physical systems* (ECPS). Examples of such systems can be found in the domains of robotics, health care, transportation, and energy. ECPS are different from traditional computing systems, because in ECPS the computer is in tight interaction with a physical environment, which it monitors and possibly controls. The requirements of the closed-loop system (computer + environment) are not purely functional. Instead, they often involve timing or performance properties, such as throughput or latency.

Abstraction and compositionality have been two key principles in developing large and complex systems. Although a large number of methods employing these principles exist to deal with functional properties (e.g., see [5, 9, 24, 25]), less attention has been paid to timing and performance. This paper contributes toward filling this gap.

Our approach can be termed *model based*. High-level models that are suitable for analysis are used as specifications or for design-space exploration. Refinement and abstraction steps are used to move between high-level models, lower-level models and implementations. The process guarantees that the results of the analysis (e.g., bounds on throughput or latency) are preserved during refinement. Our paper defines a general model and a suitable notion of abstraction and refinement that support this process. The model is compositional in the sense that refinement between models consisting of many components can be achieved by refining individual components separately.

Our treatment follows *interface theories* [11], which can be seen as type theories focusing on dynamic and concurrent behavior. Our interfaces, called *actor interfaces*, are inspired by *actor-oriented* models of computation such as process networks [20] and data flow [14].

Actors operate by consuming and producing *tokens* on their input and output ports, respectively. Since our primary goal is timing and performance analysis, we completely abstract away from token values, and keep only the times in which these tokens are produced. Actors are then defined as relations between input and output sequences of discrete events occurring in a given time axis. Examples of such event sequences are shown in Figure 2.

The main novelty of our theory lies in its notion of refinement, which is based on the principle *the earlier the better*. In particular, actor A refines actor B if, for the same input, A produces no fewer events and no later, in the worst case,

than those produced by B. For example, an actor A that non-deterministically delays its input by some time $t \in [1, 2]$ refines an actor B that deterministically delays its input by a constant time of 3. This is in sharp contrast with most standard notions of refinement which rely on the principle that the implementation should have fewer possible behaviors and thus be "more deterministic" than the specification. With the standard notions, actor A does not refine B, although it would refine an actor B' that non-deterministically delays its input by some time $t \in [0, 3]$.

The earlier-is-better refinement principle is interesting because it allows *deterministic abstractions of non-deterministic systems*. System implementations can often be seen as time-non-deterministic because of high variability in execution and communication delays, dynamic scheduling, and other effects that are expensive or impossible to model precisely. Time-deterministic models, on the other hand, suffer less from state explosion problems, and are also more suitable for deriving analytic bounds.

The main contributions of this work are the following:

- We develop an interface theory of timed actors with a refinement relation that follows the earlier-is-better principle and preserves worst-case bounds on performance metrics (throughput, latency). (Sections 4–7).
- Our framework unifies existing models (SDF and variants, automata, service curves, etc.) by treating actors semantically, as relations on event sequences, rather than syntactically, as defined by specific models such as automata or dataflow. (Section 8).

Omitted proofs can be found in the extended version of this paper [1]. The latter also contains additional material omitted from this version due to space limitations.

2. MOTIVATING EXAMPLE

To illustrate the use of our framework, we present an example of an MP3 play-back application. The application is based on a fragment of the car radio case study presented in [31]. Our goal is to show how such an application can be handled within our framework, using stepwise refinement from specification to implementation, such that performance guarantees are preserved during the process.

The layers of the refinement process are shown in Figure 1. The top layer captures the specification. It consists of a single actor SPEC, with a single output port, and a single event sequence τ at this port, defined by $\tau(n) = 50 + {}^{n}/_{44.1}$ ms, for $n \in \mathbb{N}$. That is, the n-th event in the sequence occurs at time $\tau(n)$. SPEC specifies the required behavior of an MP3 player where audio samples are produced at a rate of 44.1 kHz, starting with an initial 50 ms delay to allow for processing.

For simplicity, we do not model input tokens, assuming they are always available for consumption. Also note that the system typically includes a component such as a digital-to-analog converter (DAC) which consumes the audio samples produced at port p, buffers them and reproduces them periodically. We omit DAC since it does not take part in the refinement process.

The next layer is an application model consisting of actors DEC (decoder), SRC (sample-rate converter), and actor D1 explained below. DEC and SRC are timed *synchronous data flow* (SDF) [22] actors. SDF actors communicate by conceptually unbounded FIFO queues. They "fire" as soon

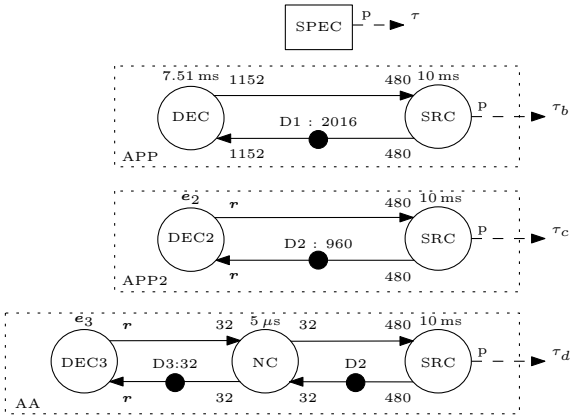


Figure 1: Successive refinements of an MP3 playback application.

as a fixed number of tokens become available at their input queues and, after a fixed duration, produce a fixed number of tokens at their output queues. For instance, DEC consumes and produces 1152 tokens per firing on the queues from and to the SRC actor. Each firing of DEC takes 7.51 ms. For a formal definition of SDF actors see Section 8.1. D1 is an actor modeling 2016 initial tokens on the queue from SRC to DEC. Formally, it is an instance of parameterized actor I_k defined in Example 8. All dataflow actors in Figure 1 (DEC, SRC, DEC2, DEC3 and NC) implicitly have a self-edge with a single initial token so that firings of the same actor do not overlap (i.e., each firing completes before the next one starts).

The global application model is a single composite actor APP obtained by composing the three actors above, first in series and then in feedback, and then hiding all ports except the output port p of SRC. Section 5 precisely defines the compositions and hiding. Because APP is an SDF model and hence deterministic, APP produces a single event sequence τ_b at p. We have captured APP in the dataflow analysis tool SDF3 (http://www.es.ele.tue.nl/sdf3/) and have used the tool to check that τ_b refines τ, i.e., that each event in τ_b occurs no later than the corresponding event in τ. As a result, APP refines SPEC.

The motivation for the third layer is buffer considerations. In this layer, the SDF actor DEC is replaced by the *cyclo-static data flow* (CSDF) [6] actor DEC2. This substitution results in smaller buffers on the queue from SRC to DEC2 [31]. CSDF actors generalize SDF actors by allowing the token consumption/production rates to vary periodically, as an SDF actor that cycles between a finite set of firing *phases*. In our example, DEC2 has 39 phases, captured by the notation $\boldsymbol{r} = [0, 0, [32]^{18}, 0, [32]^{18}]$. In the first two phases DEC decodes frame headers without consuming nor producing any tokens. The subsequent 18 phases each consume and produce 32 tokens, and are followed by a header decoding phase with no tokens consumed or produced. Finally there are 18 more phases that each consume and produce 32 tokens. This sequence of phases is repeated for each MP3 frame. The durations of these phases are given by $\boldsymbol{e}_2 = [670, 2700, [40]^{18}, 2700, [40]^{18}]\mu$s. That is, phase 1 takes 670 μs, phase 2 takes 2700 μs, and so on.

Using arguments similar to those presented later in Example 3, we can show that DEC2 refines DEC. The composite

actor APP2 is produced by first refining DEC to DEC2, and then reducing the number of initial tokens from D1 to D2, while maintaining that APP2 refines APP. The latter is ensured by using SDF3 to compute τ_c for a given D2, and checking that τ_c refines τ_b.

The bottom layer is an *architecture aware* model (AA) that is close to a distributed implementation on a multiprocessor architecture with network-on-chip (NoC) communication. In this layer DEC2 is replaced by the composition of DEC3, D3 and NC. DEC3 is identical to DEC2 except for its firing durations which are reduced to $\boldsymbol{e}_3 = [670, 2700, [30]^{18}, 2700, [30]^{18}]\mu$s, because the communication is modeled separately. NC is an SDF actor that models the NoC behavior. It can be shown that the composition of DEC3, NC and D3 refines DEC2. This and our compositionality Propositions 2 and 3 imply that AA refines APP2.

The final implementation (not shown in the figure) can be compositionally shown to refine the AA model. For instance, the NC actor conservatively abstracts the NoC implementation [16]. It is important to mention that although implementations are time-non-deterministic for multiple reasons, e.g., software execution times or run-time scheduling, the models in Figure 1 are time-deterministic.

3. RELATED WORK

Abstraction and compositionality have been extensively studied from an untimed perspective, focusing on functional correctness (e.g., see [5, 9, 24, 25]). Timing has also been considered, implicitly or explicitly, in a number of frameworks. Our treatment has been inspired in particular by interface theories such as interface automata [11] which use game-theoretic interpretations of composition and refinement, that are more appropriate for open systems. Although interface automata have no explicit notion of time, discrete time can be implicitly modeled by adding a special "tick" output. [12] follows [11] but uses timed automata [2] instead of discrete automata. However, a notion of refinement is not defined in [12]. [10] extends [12] with a notion of refinement in the spirit of alternating simulation [3], adapted for timed systems.

The refinement notions used in all works above differ from ours in a fundamental way: in our case, earlier is better, whereas in the above works, if the implementation can produce an output a at some time t, then the (refined) specification must also be able to produce a at the *same* time t. Thus, an implementation that can produce a only at times $t \leq 1$ does not refine a specification that can produce a only at times $t \geq 2$. Another major difference is that performance metrics such as throughput and latency are not considered in any of the above works.

Our work is about non-deterministic models and worst-case performance bounds and as such differs from probabilistic frameworks such as Markov decision processes, or stochastic process algebras or games (e.g., see [26, 19, 18, 13]). Worst-case performance bounds can be derived using techniques from the *network calculus* (NC) [7] or *real-time calculus* (RTC) [27]. Refinement relations have been considered recently in these frameworks [17, 28]. Semantically, these relations correspond to trace containment at the outputs and as such do not follow the earlier-is-better principle. An important feature of NC and RTC is that they can model resources, e.g., available computation power, and therefore be used in applications such as schedulability analysis. We do not explicitly distinguish resources in our framework. In NC and RTC, behaviors are typically captured by arrival or service curves. Service curves can be seen as a special class of actors [1]. Service curves have limited expressiveness: they cannot generally capture, for instance, languages produced by automata actors. The same can be said of real-time scheduling theory (e.g., see [8]). Automata-based models have been used for scheduling and resource modeling, e.g., as in [30], where tasks are described as ω-regular languages representing sets of admissible schedules. Refinement is not considered in this work, and although it could be defined as language containment, this would not follow the earlier-is-better principle.

$(\max, +)$ algebra and its relatives (e.g., see [4]) are used as an underlying system theory for different discrete event system frameworks, including NC, RTC and SDF. $(\max, +)$ algebra is mostly limited to deterministic, $(\max, +)$-linear systems. Our framework is more general: it can capture non-determinism in time, an essential property in order to be able to relate time-deterministic specification models such as SDF to implementations that have variable timing.

Our work has also been inspired by the work in [32], where task graph implementations are conservatively abstracted to timed dataflow specifications.

4. ACTORS

We consider actor interfaces (in short *actors*) as relations between finite or infinite sequences of input tokens and sequences of output tokens. We abstract from token content, and instead focus on arrival or production times represented as timestamps from some totally ordered, continuous time domain $(\mathcal{T}, \leq)$. $\mathcal{T}$ contains a minimal element denoted 0. We also add to $\mathcal{T}$ a maximal element denoted ∞, so that $t < \infty$ for all $t \in \mathcal{T}$. $\mathcal{T}^\infty$ denotes $\mathcal{T} \cup \{\infty\}$. $\mathbb{N}$ denotes the set of natural numbers and we assume $0 \in \mathbb{N}$. $\mathbb{R}$ denotes the set of real numbers and $\mathbb{R}^{\geq 0}$ the set of non-negative reals.

Definition 1 (Event sequence). *An* event sequence *is a total mapping $\tau : \mathbb{N} \to \mathcal{T}^\infty$, such that τ is weakly monotone, that is, for every $k, m \in \mathbb{N}$ and $k \leq m$ we have $\tau(k) \leq \tau(m)$.*

Thus $\tau(n)$ captures the arrival time of the n-th token, with $\tau(n) = \infty$ interpreted as event n being *absent*. Then, all events $n' > n$ must also be absent. Because of this property, an event sequence τ can also be viewed as a finite or infinite sequence of timestamps in $\mathcal{T}$. The *length* of τ, denoted $|\tau|$, is the smallest $n \in \mathbb{N}$ such that $\tau(n) = \infty$, and with somewhat abusive notation $|\tau| = \infty$ if $\tau(n) < \infty$ for all $n \in \mathbb{N}$. We use ϵ to denote the *empty* event sequence, $\epsilon(n) = \infty$ for all n. Given event sequence τ and timestamp $t \in \mathcal{T}$ such that $t \leq \tau(0)$, $t \cdot \tau$ denotes the event sequence consisting of t followed by τ. The set of all event sequences is denoted Tr.

Event sequences are communicated over *ports*. For a set P of ports, $Tr(P)$ denotes $P \to Tr$, the set of total functions that map each port of P to an event sequence. Elements of $Tr(P)$ are called *event traces over* P. We sometimes use the notation $(p, n, t) \in x$ instead of $x(p)(n) = t$.

Definition 2 (Earlier-than and prefix orders). *For $\tau, \tau' \in Tr$, τ is said to be* earlier than τ'*, denoted $\tau \leq \tau'$, iff $|\tau| = |\tau'|$ and for all $n < |\tau|$, $\tau(n) \leq \tau'(n)$. $\leq$ is called the* earlier-than *relation. In addition we consider the* prefix *relation: $\tau \preceq \tau'$ iff $|\tau| \leq |\tau'|$ and for every $n < |\tau|$,*

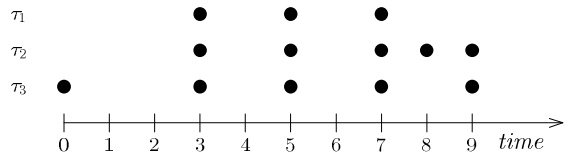


Figure 2: Three event sequences.

$\tau(n) = \tau'(n)$. *We lift* $\leq$ *and* $\preceq$ *to event traces* $x, x' \in Tr(P)$ *in the usual way:* $x \leq x'$ *iff for all* $p \in P$, $x(p) \leq x'(p)$; $x \preceq x'$ *iff for all* $p \in P$, $x(p) \preceq x'(p)$.

EXAMPLE 1. *Figure 2 shows three event sequences* τ_1, τ_2 *and* τ_3 *visualized as black dots on a horizontal time line:* $\tau_1 = 3 \cdot 5 \cdot 7 \cdot \epsilon$, $\tau_2 = 3 \cdot 5 \cdot 7 \cdot 8 \cdot 9 \cdot \epsilon$ *and* $\tau_3 = 0 \cdot 3 \cdot 5 \cdot 7 \cdot 9 \cdot \epsilon$. τ_1 *is a prefix of* τ_2*:* $\tau_1 \preceq \tau_2$*; and* τ_3 *is earlier than* τ_2*:* $\tau_3 \leq \tau_2$. *But* $\tau_1 \not\preceq \tau_2$.

$(Tr, \preceq)$ and $(Tr(P), \preceq)$ are complete partial orders (CPOs). $(Tr, \leq)$ and $(Tr(P), \leq)$ are pre-CPOs (they have no unique minimal element). We use $\bigsqcup_{\trianglelefteq} C$ to denote the least upper bound of a chain C in a CPO with partial order $\trianglelefteq$.

If x_1 is an event trace over ports P_1 and x_2 is an event trace over ports P_2, and P_1 and P_2 are disjoint, then $x_1 \cup x_2$ denotes the event trace over $P_1 \cup P_2$ such that $(x_1 \cup x_2)(p) = x_1(p)$ if $p \in P_1$ and $(x_1 \cup x_2)(p) = x_2(p)$ if $p \in P_2$. $x \uparrow Q$ is identical to x, but with all ports in Q removed from the domain.

DEFINITION 3 (ACTOR). *An* actor *is a tuple* $A = (P, Q, R_A)$ *with a set* P *of input ports, a set* Q *of output ports and an event trace relation* $R_A \subseteq Tr(P) \times Tr(Q)$. *We use* xAy *to denote* $(x, y) \in R_A$ *when we leave the three-tuple of* A *implicit.* A *is called* deterministic *if* R_A *is a partial function. The set of all* legal *input traces of* A *is:*

$$\text{in}_A = \{x \in Tr(P) \mid \exists y \in Tr(Q) : xAy\}.$$

Note that an input trace x models the times that input tokens are *produced* by the environment of the actor, and not the times that these tokens are *consumed* by the actor. Token consumption times can be modeled by adding special output ports to the actor as explained in [1].

In order to study composition and refinement later on, we need to investigate actors with respect to different kinds of monotone changes to their input and output. We therefore introduce the following family of definitions.

DEFINITION 4. (INPUT-CLOSURES, MONOTONICITIES AND CONTINUITIES) *Let* A *be an actor with input ports* P *and output ports* Q. A *is called* input-complete *iff* $\text{in}_A = Tr(P)$. *Given a partial order* $\trianglelefteq$ *on* $Tr(P)$ *and* $Tr(Q)$, A *is called (inverse)* $\trianglelefteq$-input-closed *iff for every* $x \in \text{in}_A$ *and* $x' \in Tr(P)$, $x' \trianglelefteq x$ *(*$x \trianglelefteq x'$*) implies* $x' \in \text{in}_A$. A *is called (inverse)* $\trianglelefteq$-*monotone iff for every* x, y *and* x' *such that* xAy, $x' \in \text{in}_A$ *and* $x \trianglelefteq x'$ *(*$x' \trianglelefteq x$*), there exists* y' *such that* $y \trianglelefteq y'$ *(*$y' \trianglelefteq y$*) and* $x'Ay'$. *Assuming* $\trianglelefteq$ *yields pre-CPOs,* A *is called* $\trianglelefteq$-continuous *iff for every pair* $\{x_k\}$ *and* $\{y_k\}$ *of chains of event traces w.r.t.* $(Tr(P), \trianglelefteq)$ *and* $(Tr(Q), \trianglelefteq)$ *respectively, if* $x_k A y_k$ *for all* k, *then* $(\bigsqcup_{\trianglelefteq}\{x_k\})A(\bigsqcup_{\trianglelefteq}\{y_k\})$.

EXAMPLE 2 (DELAY ACTORS). A variable delay *actor* $\Delta_{[d_1,d_2]}$ *with minimum and maximum delay* $d_1, d_2 \in \mathbb{R}^{\geq 0}$, *where* $d_1 \leq d_2$, *is an actor with one input port* p, *one output port* q, *time domain* $\mathcal{T} = \mathbb{R}^{\geq 0}$, *and such that*

$$x\Delta_{[d_1,d_2]}y \text{ iff } |x(p)| = |y(q)| \wedge \forall n < |x(p)| : \\ x(p)(n) + d_1 \leq y(q)(n) \leq x(p)(n) + d_2 \\ \wedge \big(n > 0 \implies y(q)(n) \geq y(q)(n-1)\big).$$

$\Delta_{[d_1,d_2]}$ *is input-complete,* $\preceq$- *and* $\leq$-*monotone in both directions, and* $\preceq$- *and* $\leq$-*continuous, but not deterministic in general. The* constant delay *actor* Δ_d *is the deterministic variable delay actor* $\Delta_{[d,d]}$.

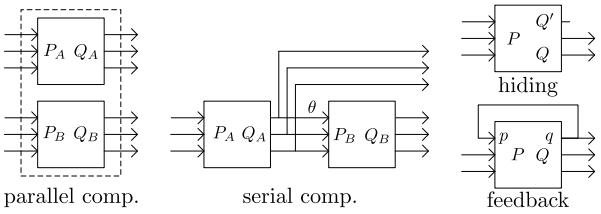


Figure 3: Actor compositions.

5. COMPOSITIONS

Actor interfaces can be composed to yield new actor interfaces. The composition operators defined in this paper are illustrated in Figure 3. Parallel composition composes two interfaces side-by-side without interaction:

DEFINITION 5 (PARALLEL COMPOSITION). *Let* A *and* B *be two actors with disjoint input ports* P_A *and* P_B *and disjoint output ports* Q_A *and* Q_B *respectively. Then the* parallel composition $A||B$ *is an actor with input ports* $P_A \cup P_B$, *output ports* $Q_A \cup Q_B$, *and relation* $A||B = \{(x_1 \cup x_2, y_1 \cup y_2) \mid x_1 A y_1 \wedge x_2 B y_2\}$.

Parallel composition is clearly associative and commutative. It is also easy to see that it preserves all monotonicity, continuity and closure properties if both actors have them.

DEFINITION 6 (SERIAL COMPOSITION). *Let* A *and* B *be two actors with disjoint input ports* P_A *and* P_B *and disjoint output ports* Q_A *and* Q_B *respectively. Let* θ *be a bijective function from* Q_A *to* P_B. *Then the* serial composition $A\theta B$ *is an actor with input ports* $P_\theta = P_A$, *output ports* $Q_\theta = Q_A \cup Q_B$, *and whose relation is defined as follows. First, we lift the mapping of ports to event traces:* $\theta(y) = \{(\theta(p), n, t) \mid (p, n, t) \in y\}$. *The input-output relation of the composite actor* $A\theta B$ *is then defined as:*

$$A\theta B = \{(x_1, y_1 \cup y_2) \in \text{in}_\theta \times Tr(Q_\theta) \mid x_1 A y_1 \wedge \theta(y_1) B y_2\}$$

where: $\text{in}_\theta = \{x \in \text{in}_A \mid \forall y_1 : xAy_1 \implies \theta(y_1) \in \text{in}_B\}$.

in_θ captures the set of legal inputs of the composite actor $A\theta B$. In the spirit of [5, 11], we adopt a "demonic" interpretation of non-determinism, where an input x is legal in $A\theta B$ only if *any* intermediate output that the first actor A may produce for x is a legal input (after relabeling) of the second actor B. In that case, we say that actor B is *receptive* to actor A w.r.t. θ. An input-complete actor is receptive to any other actor. If B is receptive to A or A is deterministic, then $A\theta B$ reduces to standard composition of relations.

Moreover, if both A and B are deterministic (respectively, input-complete) then so is $A\theta B$. Serial composition is associative [1].

The requirement that θ is total and onto is not restrictive. For example, suppose A has two output ports q_1, q_2 and B has two input ports p_1, p_2, but we only want to connect q_1 to p_1. To do this, we can extend A with additional input and output ports p_{p_2} and q_{p_2}, respectively, corresponding to p_2. A acts as the identity function on p_{p_2} and q_{p_2}, that is, for all x, y such that xAy, $y(q_{p_2}) = x(p_{p_2})$. Then we can connect q_{p_2} to p_2. Similarly, we can extend B with additional input and output ports p_{q_2} and q_{q_2}, and connect q_2 to p_{q_2}.

A hiding operator can be used to make internal event sequences unobservable.

DEFINITION 7 (HIDING). *Let $A = (P, Q, R_A)$ be an actor and let $Q' \subseteq Q$. The* hiding of Q' in A *is the actor*

$$A\backslash Q' = (P, Q\backslash Q', \{(x, y \uparrow Q') \mid xAy\}).$$

Note that $\text{in}_{A\backslash Q'} = \text{in}_A$. Hiding preserves all forms of monotonicity and continuity, as well as determinism.

DEFINITION 8 (FEEDBACK). *Let $A(P, Q, R_A)$ be an actor and let $p \in P$ and $q \in Q$. The* feedback composition of A on (p, q) *is the actor*

$$A(p = q) = (P\backslash\{p\}, Q, \{(x \uparrow \{p\}, y) \mid xAy \wedge x(p) = y(q)\}).$$

Feedback is commutative [1].

It is well-known from the study of systems with feedback that the behavior of such a system may not be unique, even if the system is deterministic, or that the behavior may not be constructively computable from the behavior of the actor, depending on the nature of the actor. To effectively apply feedback we will typically require additional constraints on the actor. In the following proposition we describe a case in which a solution can be constructively characterized by a method reminiscent of those used in Kahn Process Networks (KPN) [20]. Our result can also handle non-deterministic actors, however. See also the related Proposition 3.

PROPOSITION 1. *If actor A is input-complete, $\preceq$-monotone and $\preceq$-continuous, then $A(p = q)$ is input-complete, $\preceq$-monotone and $\preceq$-continuous.*

PROOF. Let $A = (P, Q, R_A)$, with $p \in P$ and $q \in Q$. If x is an event trace and p a port of x, then $x[p \to \tau]$ denotes the event trace obtained from x by setting the sequence at p to τ and leaving the sequences at all other ports unchanged. We show here only input completeness because it illustrates the existence of a fixed-point of the feedback. For a detailed proof, see [1]. Let $x \in Tr(P\backslash\{p\})$, then by input-completeness, $x_0 = x[p \to \epsilon] \in \text{in}_A$. Hence there is some y_0 such that $x_0 A y_0$. Now let $x_1 = x[p \to y_0(q)]$. Clearly $x_0 \preceq x_1$. Further by $\preceq$-continuity, there exists y_1 such that $x_1 A y_1$ and $y_0 \preceq y_1$. Repeating the procedure with $x_{k+1} = x[p \to y_k(q)]$ we create two chains $\{x_k\}$ and $\{y_k\}$ in the prefix CPO, such that for all k, $x_k A y_k$. Let $x' = \bigsqcup_{\preceq}\{x_k\}$ and $y' = \bigsqcup_{\preceq}\{y_k\}$, then by construction of the chains and by $\preceq$-continuity, respectively $x'(p) = y'(q)$ and $x' A y'$. Therefore, $x' \uparrow \{p\} = x \in \text{in}_{A(p=q)}$. Thus, $A(p = q)$ is input-complete. □

The assumptions used in the above result may appear strong at first sight. Note, however, that similar assumptions are often used in fixpoint theorems, even for deterministic systems. Although we could have restricted our attention to actors that have such properties by definition, we chose not to do so, since one of our goals is to be as general as possible and to examine the required assumptions on a case-by-case basis. Note that some actor formalisms (e.g., SDF) ensure these properties by definition, however, other formalisms (e.g., automata) don't.

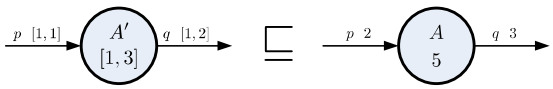


Figure 4: SDF actor A refined by CSDF actor A'.

6. REFINEMENT

Refinement is a relation between two actors A and B, allowing one to replace actor A by actor B in a given context and obtain "same or better" results, in the worst case. If τ_A and τ_B are event sequences produced by A and B, respectively, then "τ_B is same or better than τ_A" means the following: τ_B should have at least as many events as τ_A and for every event they have in common, the event should be produced in τ_B no later than in τ_A. We first capture this relation on event sequences and event traces.

DEFINITION 9. *Event sequence τ* refines *event sequence τ', denoted $\tau \sqsubseteq \tau'$, iff for all $n \in \mathbb{N}$, $\tau(n) \leq \tau'(n)$. $\sqsubseteq$ is lifted to event traces $x, x' \in Tr(P)$ in the standard way: $x \sqsubseteq x'$ iff for all $p \in P$, $x(p) \sqsubseteq x'(p)$.*

For example, for the event sequences shown in Figure 2, we have $\tau_3 \sqsubseteq \tau_2$, $\tau_2 \sqsubseteq \tau_1$, but $\tau_1 \not\sqsubseteq \tau_2$. The refinement relations on event sequences and event traces are partial orders, i.e., reflexive, transitive and antisymmetric. Moreover, the set of traces $(Tr(P), \sqsubseteq)$ equipped with the refinement order is a lattice. The supremum and infimum of traces is the point-wise supremum and infimum respectively. The event sequence $\vec{0}$, defined by $\vec{0}(n) = 0$ for all $n \in \mathbb{N}$, is the least element. The empty sequence ϵ, is the greatest element.

Note that for all $x, x' \in Tr(P)$, both $x' \preceq x$ and $x \leq x'$ imply $x \sqsubseteq x'$ and $x \sqsubseteq x'$ iff there exists $x'' \in Tr(P)$ such that $x'' \preceq x$ and $x'' \leq x'$; also, $x \sqsubseteq x'$ implies that for all $p \in P$, $|x'(p)| \leq |x(p)|$ and if both x and x' are infinite, then $x \sqsubseteq x'$ iff $x \leq x'$.

Knowing what refinement of event traces means, we can now define a refinement relation on actors.

DEFINITION 10 (REFINEMENT). *Let $A = (P, Q, R_A)$ and $B = (P, Q, R_B)$ be actors. B* refines *A, denoted $B \sqsubseteq A$, iff*
(1) $\text{in}_A \subseteq \text{in}_B$; and
(2) $\forall x \in \text{in}_A, \forall y : xBy \implies \exists y' : y \sqsubseteq y' \wedge xAy'$.

Condition (1) states that for actor B to refine actor A, B must accept at least all the inputs that actor A accepts. Condition (2) states that any behavior of actor B is no worse than a worst-case behavior of A on the same input. Note that this is where we deviate from the standard notions of refinement that implement the "more output deterministic" principle, which amounts to using the stronger constraint $y = y'$ instead of $y \sqsubseteq y'$ in Condition (2).

The requirement that both A and B have the same sets of input and output ports is not restrictive. Every output port

of A (resp. input port of B) must also be an output port of B (resp. input port of A): otherwise replacing A by B in certain contexts may result in open inputs. Any output port of B (resp. input port of A) not originally in A (resp. B) can be added to it as a "dummy" port.

EXAMPLE 3 (CSDF REFINING SDF). *Figure 4 shows an SDF actor A refined by a CSDF actor A'. At each firing, which takes 5 time units to complete, A consumes 2 and produces 3 tokens. A' cycles between two firing phases: in the first, which takes 1 time unit, 1 token is consumed and 1 is produced; in the second, which takes 3 time units, 1 token is consumed and 2 are produced. We observe that for the same number of input tokens, A' produces no fewer (and sometimes strictly more) output tokens than A, because A' can fire on a single input token, whereas A requires two. Moreover, because of the earlier activation, as well as the shorter processing time, A' produces outputs no later than A. Therefore, A' refines A.*

It is worth noting that the refinement in the above example would not hold had we used in Definition 10, $y = y'$ as in traditional refinement relations, or even $y \leq y'$ instead of $y \sqsubseteq y'$. This is because, for the input sequence containing a single token, A' produces strictly more tokens than A. As the example of Section 2 shows, it is important to be able to replace SDF actors by CSDF actors in applications, which partly motivated our novel definition of refinement.

Actor refinement is a pre-order: it is reflexive and transitive [1]. However, it is not antisymmetric. Indeed, for the constant and variable delay actors Δ_d and $\Delta_{[d_1,d_2]}$ (see Example 2), and for $d = d_2$, we have both $\Delta_d \sqsubseteq \Delta_{[d_1,d]}$ and $\Delta_{[d_1,d]} \sqsubseteq \Delta_d$. Yet $\Delta_d \neq \Delta_{[d_1,d]}$ when $d_1 < d$.

It is easy to show that refinement is always preserved by parallel composition and hiding [1]. Refinement is preserved by serial composition under natural conditions, namely, consuming actor B should not refuse better input and should not produce worse output on better input:

PROPOSITION 2. *(1) If $A' \sqsubseteq A$ and B is $\sqsubseteq$-input-closed and $\sqsubseteq$-monotone, then $A'\theta B \sqsubseteq A\theta B$. (2) If $B' \sqsubseteq B$ then $A\theta B' \sqsubseteq A\theta B$.*

Feedback preserves refinement under the following conditions:

PROPOSITION 3. *Let A be an inverse $\sqsubseteq$-input-closed, $\sqsubseteq$-monotone and $\sqsubseteq$-continuous actor, and let A' be an input-complete, $\preceq$-monotone and $\preceq$-continuous actor such that $A' \sqsubseteq A$. Then $A'(p = q) \sqsubseteq A(p = q)$.*

EXAMPLE 4. *Consider actors $A = (\{p\}, \{q\}, R_A)$ and $A' = (\{p\}, \{q\}, R_{A'})$ with input-output relations*

$$R_A = \{(x, y) \mid (\forall n : y(q)(n) = x(p)(n) + 2) \vee (y(q)(0) = 0 \wedge \forall n : y(q)(n+1) = x(p)(n))\}, \text{ and}$$
$$R_{A'} = \{(x, y) \mid y(q)(0) = 0 \wedge \forall n : y(q)(n+1) = x(p)(n) + 1\}.$$

Both A and A' are input-complete, $\preceq$-monotone in both directions, $\preceq$-continuous and $\leq$-monotone in both directions. A is non-deterministic but A' is deterministic. A' refines A because the unique output sequence of A' can be matched with the (later) output sequence of A produced by the first disjunct. If we connect A in feedback, we get $A(p = q)$ with a single (output) port q, and producing either the empty sequence $y(q) = \epsilon$ or the zero sequence $y(q)(n) = 0$ for all n. A' in feedback produces a single sequence $y'(q)(n) = n$. Any sequence refines ϵ, therefore, $A'(p = q) \sqsubseteq A(p = q)$.

7. PERFORMANCE METRICS

We often care about the performance of our systems in terms of specific metrics such as throughput or latency [7, 27, 23, 15]. In this section we show that our notion of refinement is strong enough to provide guarantees on performance under a refinement process. For simplicity we assume in this section, that $\mathcal{T} = \mathbb{R}^{\geq 0}$.

We begin by defining throughput for an infinite event sequence τ. A first attempt is to define throughput as the limit behavior of the average number of tokens appearing in the sequence per unit of time: $T(\tau) = \lim_{n\to\infty} \frac{n}{\tau(n)}$. By the usual definition of the limit, it exists and is equal to T if

$$\forall \varepsilon > 0 : \exists K > 0 : \forall n > K : T - \varepsilon < \frac{n}{\tau(n)} < T + \varepsilon.$$

However, this limit may not always exist for a given τ. Because among all possible behaviors of an actor, there may be some for which it does not exist, we focus instead on throughput *bounds*, which are more robust against this. We consider lower bounds, which are preserved by refinement.

DEFINITION 11. *Given infinite event sequence τ, its lower bound on throughput is*

$$T^{lb}(\tau) = \sup\{T \in \mathbb{R}^{\geq 0} \mid \exists K > 0 : \forall n > K : n > \tau(n) \cdot T\}$$

where by convention we take $\sup \mathbb{R}^{\geq 0} = \infty$.

$T^{lb}(\tau)$ is the greatest lower bound on the asymptotic average throughput of τ (also known as the *limit inferior* of the sequence $n/\tau(n)$). Multiplying both sides of the inequalities by $\tau(n)$ avoids division by zero problems. For a *zeno* sequence τ, where timestamps do not diverge to ∞, i.e., $\exists t \in \mathbb{R}^{\geq 0} : \forall n \in \mathbb{N} : \tau(n) < t$, we have $T^{lb}(\tau) = \sup \mathbb{R}^{\geq 0} = \infty$. This holds in particular for the zero sequence $\vec{0}$ with $\vec{0}(n) = 0$ for all n.

PROPOSITION 4. *For any two infinite event sequences τ_1 and τ_2, if $\tau_1 \sqsubseteq \tau_2$, then $T^{lb}(\tau_1) \geq T^{lb}(\tau_2)$.*

We next define the throughput bound for an actor A. An actor may have multiple output ports with generally different throughputs. For a given port, the throughput at that port may depend on the input trace as well as on nondeterministic choices of the actor. We therefore consider the worst-case scenario.

DEFINITION 12. *Given actor $A = (P, Q, R_A)$, output port $q \in Q$ and input trace $x \in Tr(P)$, the lower bound on throughput of A w.r.t. q, x is:*

$$T^{lb}(A, x, q) = \inf\{T^{lb}(\tau) \mid \exists y : xAy \wedge \tau = y(q)\}.$$

For example, for the actor SPEC of Section 2, which has no inputs and a unique output port, we have $T^{lb}(\text{SPEC}) = T^{lb}(50 + {}^{n}/_{44.1}) = \sup\{T \in \mathbb{R}^{\geq 0} \mid \exists K > 0 : \forall n > K : n > (50 + {}^{n}/_{44.1})T\} = \sup\{T \in \mathbb{R}^{\geq 0} \mid T < 44.1\} = 44.1$.

For two actors A and B with the same sets of input and output ports P and Q, respectively, we shall write $T^{lb}(A, x) \leq T^{lb}(B, x)$ to mean $T^{lb}(A, x, q) \leq T^{lb}(B, x, q)$ for all $q \in Q$. This notation is used in Proposition 6 below.

We next turn to latency, another prominent performance metric. We define latency as the smallest upper bound on

observed time differences between related input and output events. The pairs of events that we want to relate are explicitly specified as follows:

DEFINITION 13. *An* input-output event specification *(IOES) for a set P of input ports and a set Q of output ports is a relation $\mathcal{E} \subseteq 2^{P\times\mathbb{N}} \times 2^{Q\times\mathbb{N}}$. $\mathcal{E}$ is called* valid for $(x,y) \in Tr(P) \times Tr(Q)$ *iff for every $(E_P, E_Q) \in \mathcal{E}$, if $x(p)(m) \neq \infty$ for every $(p,m) \in E_P$, then $y(q)(n) \neq \infty$ for every $(q,n) \in E_Q$. $\mathcal{E}$ is called* valid for an actor $A = (P, Q, R_A)$ *iff it is valid for every $(x,y) \in R_A$.*

A pair $(E_P, E_Q) \in \mathcal{E}$ says that we want to measure the maximum latency between an input event in E_P and an output event in E_Q, provided all events in E_P have arrived. See Example 5, given below, for an illustration.

DEFINITION 14. *Let $\mathcal{E}$ be a valid IOES for $(x,y) \in Tr(P)\times Tr(Q)$. The upper bound on latency is defined as:*

$$D^{\mathcal{E}}(x,y) = \sup\{y(q)(n) - x(p)(m) \mid (E_P,E_Q)\in\mathcal{E}, \; E_P \subseteq \mathrm{dom}(x), (p,m)\in E_P, (q,n)\in E_Q\}$$

where by convention $\sup\emptyset = 0$ and $\mathrm{dom}(x)$ denotes the set of all pairs (p,n) such that $x(p)(n) \neq \infty$.

$D^{\mathcal{E}}(x,y)$ is the largest among all delays between an input and an output event that occur in x and y and are related by $\mathcal{E}$, provided all other events in the same input group are also in x. Notice that, by the assumption of validity of $\mathcal{E}$ for (x,y), $E_P \subseteq \mathrm{dom}(x)$ implies $E_Q \subseteq \mathrm{dom}(y)$, for every $(E_P, E_Q) \in \mathcal{E}$.

EXAMPLE 5. *Consider a deterministic actor A with two input ports p_1, p_2 and a single output port q. Suppose A consumes one token from each input port, and for every such pair, produces a token at q, after some constant delay, say $d \in \mathbb{R}^{\geq 0}$. Let x_1 and x_2 be two input event traces, with $x_1 = \{(p_1, 2\cdot\epsilon), (p_2, 4\cdot\epsilon)\}$ and $x_2 = \{(p_1, 2\cdot 5\cdot\epsilon), (p_2, 4\cdot\epsilon)\}$. For both x_1 and x_2, A produces the same output event trace $y = \{(q, (4+d)\cdot\epsilon)\}$. This is because, in x_2, A waits for a second input to arrive at p_2 before it can produce a second output. To measure the latency of A, we can define $\mathcal{E}$ to be:*

$$\mathcal{E} = \{(\{(p_1,n),(p_2,n)\},\{(q,n)\}) \mid n \in \mathbb{N}\}.$$

This makes $\mathcal{E}$ a valid IOES for x_1, y, as well as for x_2, y, and gives us $D^{\mathcal{E}}(x_1,y) = D^{\mathcal{E}}(x_2,y) = d$, as is to be expected.

Keeping the reference input trace fixed, refinement of output traces is guaranteed to not worsen latency:

PROPOSITION 5. *Let x, y_1 and y_2 be event traces such that $y_1 \sqsubseteq y_2$. Suppose $\mathcal{E}$ is valid for x and y_2. Then $\mathcal{E}$ is valid for x and y_1 and $D^{\mathcal{E}}(x,y_1) \leq D^{\mathcal{E}}(x,y_2)$.*

DEFINITION 15. *An IOES $\mathcal{E}$ is valid for an actor A iff $\mathcal{E}$ is valid for every (x,y) such that xAy. For a valid $\mathcal{E}$, the worst-case latency of A on input event trace x is*

$$D^{\mathcal{E}}(A,x) = \sup_{y \text{ s.t. } xAy} \{D^{\mathcal{E}}(x,y)\}.$$

EXAMPLE 6. *A suitable IOES for the variable delay actor $\Delta_{[d_1,d_2]}$ from Example 2 is $\mathcal{E} = \{(\{(p,n)\},\{(q,n)\}) \mid n \in \mathbb{N}\}$. $\mathcal{E}$ is valid for $\Delta_{[d_1,d_2]}$ and $D^{\mathcal{E}}(\Delta_{[d_1,d_2]}, x) = d_2$, for any non-empty input event trace x.*

The following states the main preservation results for throughput and latency performance bounds under refinement:

PROPOSITION 6. *Let $B \sqsubseteq A$ and $\mathcal{E}$ be a valid IOES for A. Then for any $x \in \mathrm{in}_A$, $T^{lb}(B,x) \geq T^{lb}(A,x)$ and $D^{\mathcal{E}}(B,x) \leq D^{\mathcal{E}}(A,x)$.*

Proposition 6 can be used together with Propositions 2 and 3 to guarantee that worst-case performance bounds are preserved during compositional refinement of models, as in the example of Section 2.

8. REPRESENTATIONS & ALGORITHMS

So far, our treatment has been semantical, regarding actors as sets of input-output event traces. In this section, we consider syntactic, finite representations. We show that the semantics commonly associated with these representations can be embedded naturally in our theory. We also provide algorithms to check refinement and compute compositions and performance metrics on such representations. Our intention in this section is not to be complete, but rather to give examples of how our theory can be instantiated and automated.

8.1 Synchronous Data Flow

We have informally used timed SDF actors in previous examples. In this section we formally define them. Typically, in timed SDF models the time domain is the non-negative reals or integers. In the rest of this subsection, we therefore assume that $\mathcal{T} = \mathbb{R}^{\geq 0}$ or $\mathcal{T} = \mathbb{N}$.

DEFINITION 16 (SDF ACTORS). *An actor $A = (P, Q, R_A)$ is a* homogeneous SDF actor with firing duration $d \in \mathcal{T}$, *iff*

$$R_A = \{(x,y) \mid \forall q \in Q : |y(q)| = \min_{p\in P} |x(p)| \wedge \; \forall n < |y(q)| : y(q)(n) = \max_{p\in P} x(p)(n) + d\}.$$

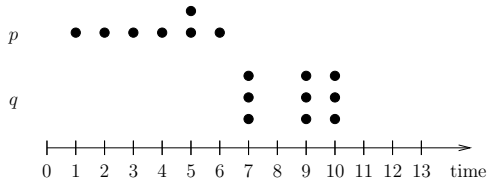
That is, the n-th firing of A starts as soon as the n-th token has arrived on every input. The firing takes d time units, after which a single output token is produced on each output. A is an SDF actor with token transfer quanta $\boldsymbol{r} : P \cup Q \to \mathbb{N}$ and firing duration $d \in \mathcal{T}$ *iff*

$$\begin{aligned} R_A \;=\; & \{(x,y) \mid \forall q \in Q : |y(q)| = \boldsymbol{r}(q) \cdot \min_{p\in P} \big(|x(p)| \div \boldsymbol{r}(p)\big) \\ & \wedge \; \forall n < |y(q)| : y(q)(n) = d + \\ & \max_{p\in P} x(p)\big((n \div \boldsymbol{r}(q) + 1)\cdot \boldsymbol{r}(p) - 1\big)\}. \end{aligned}$$

where $\div$ denotes the quotient of the integer division.

A (non-homogeneous) SDF actor can consume respectively produce more than a single token on its inputs and outputs with every firing, using rates according to $\boldsymbol{r}$. SDF actors are deterministic and have constant delays d. In SDF literature they are often implicitly understood to abstract behaviours with varying (non-deterministic) execution times in a conservative way.

EXAMPLE 7. *Consider the SDF actor A shown in Figure 4. A has an input port p (quantum 2) and an output port q (quantum 3). Its firing duration is 5. An example input-output event trace of A is shown below. The firings of A start at times 2, 4 and 5 and overlap in time. Note that the 7-th input token does not lead to any output.*



CSDF actors like the one on the left of Figure 4 can be formalized similarly, taking into account that they periodically cycle through firings with different quanta and firing durations.

An *SDF graph* represents the composition of multiple SDF actors, as in the examples shown in Figure 1. Edges in SDF graphs are often annotated with initial tokens representing the fact that the initial state of some queues is non-empty. To model this, we introduce an explicit actor:

EXAMPLE 8. *The* initial token actor with $k \in \mathbb{N}$ tokens *is* $I_k = (\{p\}, \{q\}, R_{I_k})$*, where* $(x, y) \in R_{I_k}$ *iff for all* $n \in \mathbb{N}$*:* $y(q)(n) = 0$ *if* $n < k$*, and* $y(q)(n) = x(p)(n-k)$ *otherwise.*

That is, I_k outputs k initial tokens at time 0, and then behaves as the identity function. I_k satisfies all monotonicity and continuity properties.

An SDF graph cannot always be reduced to an equivalent SDF actor. Indeed, in general, the serial or parallel composition of two SDF actors is not an SDF actor [29] (but of course it is an actor in the sense of this paper).

Let A_1 and A_2 be two SDF actors. We want to check whether $A_1 \sqsubseteq A_2$. Clearly, A_1 and A_2 must have the same sets of input and output ports, say P and Q. Suppose A_1 and A_2 have quanta functions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, and firing durations d_1 and d_2, respectively.

PROPOSITION 7. $A_1 \sqsubseteq A_2$ *iff* $d_1 \leq d_2$ *and* $\forall p \in P, q \in Q, n \leq \boldsymbol{r}_1(p) \cdot \boldsymbol{r}_2(p) : \boldsymbol{r}_1(q) \cdot (n \div \boldsymbol{r}_1(p)) \geq \boldsymbol{r}_2(q) \cdot (n \div \boldsymbol{r}_2(p))$.

The above result is generalized in [1] which discusses how refinement can be checked not only on SDF actors but also on SDF graphs, using $(\max, +)$ algebra. [1] also discusses how throughput can be computed on SDF graphs. The proposition that follows summarizes the latter result. For an SDF graph A with external input and output ports P and Q, $\boldsymbol{r}_A : P \cup Q \to \mathbb{N}$ denotes the *repetition vector* of the graph, which assigns to every port the relative rates at which tokens are consumed and produced [22].

PROPOSITION 8. *Let A be a strongly connected SDF graph with input ports P and output ports Q. Let x be an input trace of A. Then A has a computable internal throughput bound T^A, and*

$$T^{lb}(A, x, q) = \boldsymbol{r}_A(q) \cdot \min(T^A, \min_{p \in P} \frac{T^{lb}(x(p))}{\boldsymbol{r}_A(p)}).$$

Similarly, latency of SDF actors as defined in this work, can be computed using existing analysis techniques from SDF literature [15, 23]. It is natural to specify the related input-output events in patterns which repeat with the periodic behavior of SDF iterations.

8.2 Discrete-Time Automata

An other natural representation of actors is automata. Automata, in contrast with SDF actors, do not have $\sqsubseteq$-monotonicity and input-closure built-in, and such properties have to be explicitly verified when necessary. There are

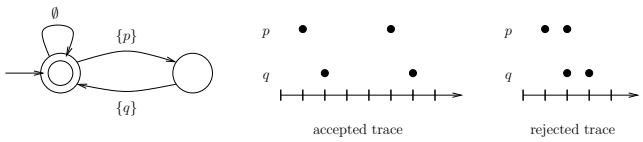


Figure 5: Implicit-tick automaton example.

many automata variants, over finite or infinite words, with various acceptance conditions, finite or infinite-state, and so on. We are not going to propose a single automaton-based model for actors. Instead we will discuss some general ideas as well as some cases for which we have algorithms. We limit our discussion to *discrete-time* automata (DTA) in the sense that time is counted by discrete transitions. DTA generate actors over a discrete time domain, $\mathcal{T} = \mathbb{N}$. The ideas naturally apply also to timed automata [2] and yield actors where $\mathcal{T} = \mathbb{R}^{\geq 0}$.

One possible model is an automaton whose transitions are labeled with subsets of $P \cup Q$, the set of input or output ports. An example is shown in Figure 5. The state drawn with two circles is the *accepting* state. In this *implicit-tick model* each transition corresponds to one time unit. If the transition is labeled by some set of ports $V \subseteq P \cup Q$, then an event at each port in V occurs at the corresponding instant in $\mathbb{N}$. V may be empty, as in the self-loop transition of the automaton shown in the figure. In this case, no events occur at that time instant.

The implicit-tick model cannot capture event traces where more than one event occurs simultaneously at the same port (and therefore an implicit-tick actor cannot be input-complete). An alternative is to dissociate the elapse of time from transitions, by introducing a special label, $\mathtt{t}$, denoting one time unit. Then, we can use automata whose transitions are labeled with single ports or $\mathtt{t}$, that is, whose alphabet is $P \cup Q \cup \{\mathtt{t}\}$. We call this the *explicit-tick model*. We do not further discuss this model here: it is explored in [1].

A DTA M defines an actor $A(M) = (P, Q, R_{A(M)})$ as follows. Every (finite or infinite) accepting run of M generates a (finite or infinite) word w in the language of M, denoted $\mathcal{L}(M)$. Every word w can be mapped to a unique event trace pair $Tr(w) \in Tr(P) \times Tr(Q)$, as illustrated in Figure 5. Then, $R_{A(M)}$ is the set of all event trace pairs generated by words in $\mathcal{L}(M)$, i.e., the set $\{Tr(w) \mid w \in \mathcal{L}(M)\}$, also denoted $Tr(\mathcal{L}(M))$. An *ITA* is an implicit-tick automaton on finite words. An *ITBA* is an implicit-tick Büchi automaton. Finite-word DTA are strictly less expressive than corresponding Büchi versions. Every ITA M can be transformed into an ITBA M' such that $A(M) = A(M')$.

Two distinct words w and w' of an ITA M can result in the same event trace, for instance, if $w' = w \cdot \emptyset^n$, for different $n \geq 1$. To avoid technical complications related to this, we will assume that M is *tick-closed*, that is, for any $w \in \mathcal{L}(M)$, $w \cdot \emptyset^* \subseteq \mathcal{L}(M)$. The ITA of Figure 5 is tick-closed. Any ITA M can be transformed to a tick-closed ITA M' so that $A(M) = A(M')$. In the rest of this section we assume all ITA to be tick-closed. We also assume that automata are finite-state, all their states are reachable from the initial state, and there is no state that cannot reach an accepting state by a non-empty path.

Given actors represented by discrete-time automata, we are interested in answering various questions.

"Given M and M', is $A(M) = A(M')$?" For ITA, there is a bijection between infinite words and event traces, that is, $\forall w, w' \in (2^{P \cup Q})^\omega : w \neq w' \iff Tr(w) \neq Tr(w')$. (Note that $Tr(w)$ may be finite, even though w is infinite, if w ends in $\emptyset^\omega$.) The same bijection does not hold for finite words as explained above. Nevertheless, because ITA are assumed to be tick-closed, we can show:

PROPOSITION 9. *For two ITA (ITBA) M_1 and M_2, we have $A(M_1) = A(M_2)$ iff $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.*

"Given M, is $A(M)$ deterministic?" Determinism of M does not imply determinism of $A(M)$, because of the different role of input and output symbols. $A(M)$ is non-deterministic iff there are two words $w, w' \in \mathcal{L}(M)$, with $(x, y) = Tr(w)$ and $(x', y') = Tr(w')$, such that $x = x'$ but $y \neq y'$.

PROPOSITION 10. *For any ITA or ITBA M it is decidable whether $A(M)$ is deterministic.*

The proof uses a synchronous product of M with itself synchronizing only on input events to check for words with the same input event trace, but a different output event trace [1].

"Given M_1 and M_2, compute M so that $A(M) = A(M_1)||A(M_2)$." M can be computed as a product of M_1 and M_2, so that $\mathcal{L}(M)$ contains exactly those words w such that $Tr(w) = Tr(w_1) \cup Tr(w_2)$, for $w_i \in \mathcal{L}(M_i)$ and $i = 1, 2$. If M_1 and M_2 belong to the implicit-tick model, M is a synchronous product, so that a pair of transitions $\xrightarrow{P_1 \cup Q_1}$ of M_1 and $\xrightarrow{P_2 \cup Q_2}$ of M_2 yields a transition $\xrightarrow{P_1 \cup P_2 \cup Q_1 \cup Q_2}$ in M.

"Given M_1 and M_2, and a bijection θ from the output ports of M_1 to the input ports of M_2, compute M so that $A(M) = A(M_1)\theta A(M_2)$."

Feeding the output of M_1 (after relabeling) into the input of M_2 can be achieved by a product of both automata synchronizing on the corresponding ports. The main challenge in computing the serial composition is to ensure that the constraint in_θ is satisfied (see Definition 6). We assume M_1 and M_2 are ITA. We construct the composite automaton M as the synchronous product of M_1, M_2 and a third automaton M_{in} capturing the constraint in_θ. We can construct M_{in} as an alternating automaton such that $Tr(\mathcal{L}(M_{in})) = \text{in}_\theta$ [1]. M_{in} can then be converted into a non-deterministic automaton using standard techniques.

"Given M, input port p and output port q, compute M' such that $A(M') = A(M)(p = q)$." If M is an ITA or an ITBA, then M' can be easily obtained by removing from M all transitions $\xrightarrow{P' \cup Q'}$ except those that satisfy $p \in P' \iff q \in Q'$.

An important question is to check for actor refinement. *"Given M_1 and M_2, is $A(M_1) \sqsubseteq A(M_2)$?"* We show that checking actor refinement on ITA can be reduced to checking language containment with respect to an appropriate closure. Given automaton M, we construct an (initially infinite state) automaton $M_{\infty\sqsubseteq}$ that recognizes the refinement closure of M, i.e., it accepts all words of M, but also all words that correspond to traces that refine the traces of M. We define $M_{\infty\sqsubseteq}$ for a single output port q, but the construction can be generalized to multiple output ports. Figure 6 shows an example. We add a counter n to count the surplus of q events. Whenever later in a word M requires a q event, we allow this event to be absent and decrease the counter. The following gives a precise definition of this idea, parameterized with a bound k on the counter.

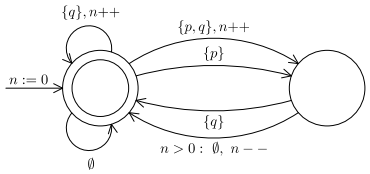


Figure 6: Refinement closure of Figure 5.

DEFINITION 17. *Let $M = (S, s_0, E, F)$ be an ITA with a single output port q, states S, initial state $s_0 \in S$, accepting states $F \subseteq S$, and transitions E. For $k \in \mathbb{N}$, the k-*bounded refinement closure *of M is the automaton $M_{k\sqsubseteq} = (S_{k\sqsubseteq}, s_{k\sqsubseteq,0}, E_{k\sqsubseteq}, F_{k\sqsubseteq})$ such that $S_{k\sqsubseteq} = \{(s, n) \mid s \in S, 0 \leq n \leq k\}$, $s_{k\sqsubseteq,0} = (s_0, 0)$, and $F_{k\sqsubseteq} = \{(s, n) \in S_{k\sqsubseteq} \mid s \in F\}$. For every transition $(s_1, \sigma, s_2) \in E$, we have the following transitions in $E_{k\sqsubseteq}$:*

$$\begin{array}{cc} \big((s_1, n), \sigma, (s_2, n)\big) & \text{if} \quad 0 \leq n \leq k \\ \big((s_1, n), \sigma \cup \{q\}, (s_2, n+1)\big) & \text{if} \quad 0 \leq n < k, q \notin \sigma \\ \big((s_1, n), \sigma \cup \{q\}, (s_2, n)\big) & \text{if} \quad n = k, q \notin \sigma \\ \big((s_1, n), \sigma \backslash \{q\}, (s_2, n-1)\big) & \text{if} \quad 0 < n \leq k, q \in \sigma \end{array}$$

The (unbounded) refinement closure *of M is the automaton $M_{\infty\sqsubseteq}$ defined in a similar way, but where counter n is unbounded.*

LEMMA 1. *Let M_1 and M_2 be ITA with the same input ports P and the same, single output port q. Then $A(M_1) \sqsubseteq A(M_2)$ iff for every $w \in \mathcal{L}(M_1)$ such that $Tr(w) = (x, y)$ and $x \in \text{in}_{A(M_2)}$, $w \in \mathcal{L}(M_{2,\infty\sqsubseteq})$.*

Unfortunately, $M_{2,\infty\sqsubseteq}$ is not a finite-state automaton, in fact, $\mathcal{L}(M_{2,\infty\sqsubseteq})$ is not always regular. Let $\mathcal{L}(M) = (\{p, q\} \cdot \emptyset)^*$, where p is the only input port and q the only output port. Then $\mathcal{L}(M_{\infty\sqsubseteq})$ contains all words of the form $(\{p, q\} \cdot \{q\})^n \cdot \{p\}^n$, for any $n \in \mathbb{N}$. Based on this, we can show that $\mathcal{L}(M_{\infty\sqsubseteq})$ is not regular. Despite this difficulty, we can make use of the finite memory property of M_1 and M_2 to find an upper bound on the size of the refinement closure, which proves that checking refinement for ITA is decidable:

PROPOSITION 11. *Let M_1 and M_2 be deterministic ITA with the same input ports P and the same, single output port q. $A(M_1) \sqsubseteq A(M_2)$ iff for every $w \in \mathcal{L}(M_1)$ such that $Tr(w) = (x, y)$ with $x \in \text{in}_{A(M_2)}$, $w \in \mathcal{L}(M_{2,N\sqsubseteq})$, where $N = |S_1| \times |S_2|$.*

"Given ITBA M with sets of input and output ports P and Q, respectively, and given an output port $q \in Q$ and an input trace $x \in Tr(P)$, what is $T^{lb}(A(M), x, q)$?" To compute this, we need a finite representation for x. A natural choice is to represent x as a deterministic ITBA M_x that only refers to ports in P. We require that M_x generates a single trace x. These assumptions imply that M_x has the form of a "lasso" (a single path eventually returning to an earlier state).

First, we compute a product of M and M_x such that the two automata synchronize on inputs. We remove from the product all strongly connected components (SCCs) that contain no accepting state of M and denote the result by M'.

We assign a weight to each transition $\xrightarrow{P'/Q'}$ of M': weight 1 if $q \in Q'$ and weight 0 otherwise. With these weights M' can be viewed as a weighted directed graph. We run Karp's algorithm [21] to compute the minimum cycle mean of M', denoted MCM. MCM is the minimum over all simple cycles κ in M'' of the ratio $\frac{w(\kappa)}{|\kappa|}$, where $w(\kappa)$ is the sum of weights of all transitions in κ and $|\kappa|$ is the length of κ (i.e., the number of transitions in κ).

PROPOSITION 12. $T^{lb}(A(M), x, q) = MCM$.

(proof sketch) The tricky part is that Karp's algorithm considers all cycles, including non-accepting (in the Büchi sense) cycles. However, all cycles are guaranteed to belong to an accepting SCC (otherwise the SCC is removed by construction of M'). Then, from any cycle it is possible to reach an accepting state and then return to the cycle. This "detour" may increase the throughput by some amount ε, however, ε can be made arbitrarily small by taking the detour very infrequently (but infinitely often, to be accepting).

9. DISCUSSION AND FUTURE WORK

We have proposed an interface theory for timed actors with a refinement relation based on the earlier-is-better principle, suitable for worst-case performance analysis. Our framework is compositional and unifies existing formalisms, allowing different types of models to be used in the same design process, e.g., automata models could refine SDF models.

The earlier-is-better principle may not seem directly applicable in scenarios where outputs should be produced not too late but not too early either. A possible alternative approach is to combine the earlier-is-better refinement with a corresponding *later-is-better* version, obtained by replacing $y \sqsubseteq y'$ by $y' \sqsubseteq y$ in Condition (2) of Definition 10. Separate specifications could then be used, expressing upper and lower bounds on timing behavior, and refined using the earlier- or later-is-better relation, respectively. Examining in detail this hybrid approach is part of future work.

Other directions for future work include examining the algorithmic complexity of the various computational problems and coming up with practically useful algorithms; implementing the algorithms and performing experiments; integrating new representations under our framework; and having a comparative study of the different representations, for instance, in terms of expressiveness and complexity.

10. REFERENCES

[1] Extended version of this paper available at `http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-130.html`.

[2] R. Alur and D. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.

[3] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR*, 1998.

[4] F. Baccelli, G. Olsder, J.-P. Quadrat, and G. Cohen. *Synchronization and linearity. An algebra for discrete event systems.* Wiley, 1992.

[5] R.-J. Back and J. Wright. *Refinement Calculus*. Springer, 1998.

[6] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-static dataflow. *IEEE Tran. on Signal Processing*, 44(2), 1996.

[7] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Springer, 2001.

[8] G. Buttazzo. *Hard Real-time computing systems*. Kluwer, 1997.

[9] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[10] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *HSCC*, pages 91–100, 2010.

[11] L. de Alfaro and T. Henzinger. Interface automata. In *Foundations of Software Engineering (FSE)*. ACM Press, 2001.

[12] L. de Alfaro, T. A. Henzinger, and M. I. A. Stoelinga. Timed interfaces. In *EMSOFT*, pages 108–122, 2002.

[13] L. de Alfaro, R. Majumdar, V. Raman, and M. Stoelinga. Game refinement relations and metrics. *L. Meth. in Comp. Sc.*, 4(3), 2008.

[14] J. B. Dennis. First version of a data flow procedure language. In *Programming Symposium*, pages 362–376. Springer-Verlag, 1974.

[15] A. Ghamarian, S. Stuijk, T. Basten, M. Geilen, and B. Theelen. Latency minimization for synchronous data flow graphs. *Digital Systems Design, Euromicro Symposium on*, pages 189–196, 2007.

[16] A. Hansson, M. Wiggers, A. Moonen, K. Goossens, and M. Bekooij. Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis. *IET CDT*, 3(5), 2009.

[17] T. Henzinger and S. Matic. An interface algebra for real-time components. In *RTAS*, pages 253 – 266, 2006.

[18] H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.

[19] B. Jonsson and W. Yi. Testing preorders for probabilistic processes can be characterized by simulations. *Theor. Comput. Sci.*, 282(1):33–51, 2002.

[20] G. Kahn. The semantics of a simple language for parallel programming. In *Inf. Proc. 74*, 1974.

[21] R. Karp. A characterization of the minimum cycle mean in a digraph. *Discr.Math.*, 23(3):309–311, 1978.

[22] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[23] O. M. Moreira and M. J. G. Bekooij. Self-timed scheduling analysis for real-time applications. *EURASIP J. on Adv. in Signal Proc.*, 2007.

[24] F. Nielson, H. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.

[25] B. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[26] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. of Computing*, 2(2):250–273, 1995.

[27] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS*, pages 101–104, 2000.

[28] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *EMSOFT*, pages 34–43, 2006.

[29] S. Tripakis, D. Bui, M. Geilen, B. Rodiers, and E. A. Lee. Compositionality in Synchronous Data Flow: Modular Code Generation from Hierarchical SDF Graphs. ACM TECS (to appear).

[30] G. Weiss and R. Alur. Automata based interfaces for control and scheduling. In *HSCC*, pages 601–613. Springer, 2007.

[31] M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient computation of buffer capacities for cyclo-static real-time systems with back-pressure. In *RTAS*, pages 281–292, 2007.

[32] M. Wiggers, M. Bekooij, and G. Smit. Monotonicity and run-time scheduling. In *EMSOFT*, pages 177–186, 2009.