HAL open science

A Timed Component Algebra for Services

Benoît Delahaye, José Luiz Fiadeiro, Axel Legay, Antónia Lopes

To cite this version:

Benoît Delahaye, José Luiz Fiadeiro, Axel Legay, Antónia Lopes. A Timed Component Algebra for Services. 15th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOOODS) / 33th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), Jun 2013, Florence, Italy. pp.242-257, 10.1007/978-3-642-38592-6_17 . hal-01515250

HAL Id: hal-01515250

https://inria.hal.science/hal-01515250

Submitted on 27 Apr 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

A Timed Component Algebra for Services

Benoît Delahaye[1], José L. Fiadeiro[2], Axel Legay[1], and Antónia Lopes[3]

[1]INRIA/IRISA, Rennes, France
benoit.delahaye@irisa.fr, axel.legay@irisa.fr
[2]Dep. of Computer Science, Royal Holloway, University of London, UK
jose.fiadeiro@rhul.ac.uk
[3]Dep. of Informatics, Faculty of Sciences, University of Lisbon, Portugal
mal@di.fc.ul.pt

Abstract. We present a component algebra for services that can guarantee time-related properties. The components of this algebra are networks of processes that execute according to time constraints and communicate asynchronously through channels that can delay messages. We characterise a sub-class of consistent networks give sufficient conditions for that class to be closed under composition. Finally, we show how those conditions can be checked, at design time, over timed I/O automata as orchestrations of services, thus ensuring that, when binding a client with a supplier service at run time, the orchestrations of the two services can work together as interconnected without further checks.

1 Introduction

In [9,10], we revisited the notions of interface and component algebra proposed in [7] for component-based design and put forward elements of a corresponding interface theory for service-oriented design in which service orchestrations are networks of asynchronously communicating processes. That algebra is based on an implicit model of time: the behaviour of processes and channels is captured by infinite sequences of sets of actions, each action consisting of either the publication or the delivery of a message. However, such an implicit model of time is not realistic for modelling numerous examples of timed behaviour, from session timeouts to logical deadlines, and is not very effective for the analysis of properties. In this paper, we investigate an alternative model based on *timed traces* [3]. Even if this model assumes a minimal granularity of time, time is no longer implicit and, therefore, more realistic: we record the behaviour that is observed only at those instants of time when networks are active, not at every instant.

In this setting, we study the problem of ensuring consistency of run-time composition of orchestrations based on properties of processes and channels that can be checked at design time. This is important because run-time binding is an intrinsic feature of the service-oriented paradigm – one that distinguishes it from distributed systems in general – and that checking for consistency by actually calculating, at run time, the product of the automata that implement the services being bound to each other and checking for the non-emptiness of the resulting language is simply not realistic.

Not surprisingly, the results obtained in [10] for run-time composition under the implicit-time model do not extend directly to timed traces because the two spaces have different topological structures. Hence, one of our main contributions is the identification of refinement and closure operators that can support the composition of services that do not operate over the same time sequences.

The other main contribution results from adopting, as models of implementations of processes and channels, a variant of timed I/O automata (TIOA) that permits clock invariants on locations and in which all locations are Büchi accepting (as in [12]). Although results on the consistency of the composition of TIOA have been addressed in the literature they are based on a weaker notion of consistency according to which a TIOA that does not accept any non-Zeno timed sequence can still be consistent. In Sec. 5 we give an example of a situation in which the composition of two TIOA can only produce Zeno sequences, which is not acceptable as this would mean that joint behaviour would only be possible by forcing actions to be executed over successively shorter delays to converge on a deadline. The sub-algebra of TIOA that we characterise in Sec. 4 addresses this problem, i.e., we identify properties that can be checked, at design time, over networks of TIOA that ensure that, when binding a client with a supplier service, their orchestrations can operate together without further run-time checks.

2 The component algebra

We start by recalling a few concepts related to traces and their Cantor topology. Given a set A, a *trace* λ over A is an element of A^ω, i.e., an infinite sequence of elements of A. We denote by $\lambda(i)$ the $(i+1)$-th element of λ and by λ_i the prefix of λ that ends at $\lambda(i-1)$ if $i > 0$, with λ_0 being the empty sequence. A *segment* π is an element of A^*, i.e., a finite sequence of elements of A, the length of which we denote by $|\pi|$. We use $\pi{<}\lambda$ to mean that the segment π is a prefix of λ. Given $a{\in}A$, we denote by $(\pi{\cdot}a)$ the segment obtained by extending π with a. A *property* Λ over A is a set of traces. For every property Λ, we define $\Lambda^f = \{\pi : \exists\lambda{\in}\Lambda(\pi{<}\lambda)\}$ — the segments that are prefixes of traces in Λ, also called the *downward closure* of Λ — and $\bar{\Lambda} = \{\lambda : \forall\pi{<}\lambda(\pi{\in}\Lambda^f)\}$ — the traces whose prefixes are in Λ^f, also called the *closure* of Λ. A property Λ is said to be *closed* iff $\Lambda \supseteq \bar{\Lambda}$ (and, hence, $\Lambda = \bar{\Lambda}$).

In this timed model, every trace consists of an infinite sequence of pairs of an instant of time and a set of actions – the actions that are observed at that instant of time. In order to be able to model networks of systems, we allow that set of actions to be empty: on the one hand, this allows us to model finite behaviours, i.e., systems that stop executing actions after a certain point in time while still part of a network; on the other hand, it allows us to model observations that are triggered by actions performed by components outside the system.

This time model falls under what is often known as a 'point-based semantics', as opposed to an 'interval-based semantics' in which observations are made at every instant of time – our systems of systems are discrete and, therefore, a continuous observation model is not required. The advantages of the proposed

model are that, on the one hand, it offers a natural extension of the trace-based model adopted in [10] and, on the other hand, it has been recently studied from the point of view of a number of decidability results [17].

Definition 1 (Timed traces) *Let A be a set (of actions) and $\delta \in \mathbb{R}_{>0}$.*

- *A* time sequence *τ is a trace over $\mathbb{R}_{\geq 0}$ for which there exists a sequence $(d_i \in \mathbb{N}_+)_{i \in \mathbb{N}}$ such that $\tau(0) = 0$ and $\tau(i+1) = \tau(i) + d_i \times \delta$ for every i.*
- *An* action sequence *σ is a trace over 2^A such that $\sigma(0) = \emptyset$.*
- *A* timed trace *over A is a pair $\lambda = \langle \sigma, \tau \rangle$ of an action and a time sequence. We denote by $\Lambda(A)$ the set of timed traces over A.*
- *Given a timed property $\Lambda \subseteq \Lambda(A)$, we define:*
  - *For every time sequence τ, $\Lambda_\tau = \{\sigma \in (2^A)^\omega : \langle \sigma, \tau \rangle \in \Lambda\}$ — the action property defined by Λ and τ.*
  - *$\Lambda_{time} = \{\tau : \exists \sigma \in (2^A)^\omega (\langle \sigma, \tau \rangle \in \Lambda)\}$ — the time sequences involved in Λ.*

The constant δ (fixed for the remainder of the paper) represents the minimal interval between two time observations — the sequence $(d_i)_{i \in \mathbb{N}}$ provides the duration associated with each step i. This implies in particular that time progresses, i.e., the set $\{\tau(i) : i \in \mathbb{N}\}$ is unbounded. Working with such a constant is realistic and endows the space of time sequences with topological properties that are stronger than those of the more general space of non-Zeno sequences.

Functions between sets of actions ('alphabet maps') are useful for defining relationships between processes and the networks in which they operate.

Definition 2 (Maps) *Let $f : A \to B$ be a function (alphabet map).*

- *For every $\sigma \in (2^B)^\omega$, we define $\sigma|_f \in (2^A)^\omega$ pointwise as $\sigma|_f(i) = f^{-1}(\sigma(i))$ — the* projection *of σ over A. If f is an inclusion, i.e., $A \subseteq B$, then we tend to write $_|_A$ instead of $_|_f$; this is a function that, when applied to a trace, forgets the actions of B that are not in A.*
- *For every timed trace $\lambda = \langle \sigma, \tau \rangle$ over B, we define its projection over A to be $\lambda|_f = \langle \sigma|_f, \tau \rangle$, and for every timed property Λ over B, $\Lambda|_f = \{\lambda|_f : \lambda \in \Lambda\}$ — the* projection *of Λ to A.*
- *For every timed property Λ over A, we define $f(\Lambda) = \{\langle \sigma, \tau \rangle : \langle \sigma|_f, \tau \rangle \in \Lambda\}$ — the* translation *of Λ to B.*

We are particularly interested in translations defined by prefixing every element of a set with a given symbol. Such translations are useful for identifying in a network the process to which an action belongs — we do not assume that processes have mutually disjoint alphabets. More precisely, given a set A and a symbol p, we denote by $(p._)$ the function that prefixes the elements of A with 'p.'. Note that prefixing defines a bijection between A and its image $p.A$.

In our asynchronous model, interactions are based on the exchange of messages that are transmitted through channels. We organise messages in sets that we call ports: a *port* is a finite set (of messages). Ports are communication abstractions that are convenient for organising networks of processes.

Every message belonging to a port has an associated *polarity*: $^-$ if it is an outgoing message (published at the port) and $^+$ if it is incoming (delivered at the port). Therefore, every port M has a partition $M^- \cup M^+$. The actions of sending (publishing) or receiving (being delivered) a message m are denoted by $m!$ and $m¡$, respectively. More specifically, if M is a port, we define $A_{M^-}=\{m!{:}m{\in}M^-\}$, $A_{M^+}=\{m¡{:}m{\in}M^+\}$, and $A_M{=}A_{M^-}{\cup}A_{M^+}$ — the set of actions associated with M. Even if a process does not refuse the delivery of messages, it can discard them, e.g., if they arrive outside the protocol expected by the process, and a channel can accept the publication of every message but only deliver some published messages to their destination (this is for instance the case of unreliable channels).

A *process* consists of a finite set γ of mutually disjoint ports — i.e., each message that a process can exchange belongs to exactly one of its ports — and a non-empty timed property Λ over $A_\gamma = \bigcup_{M\in\gamma} A_M$ defining its behaviour.

Interactions are established through channels. A *channel* consists of a set M of messages and a non-empty timed property Λ over $A_M{=}\{m!, m¡ :m{\in}M\}$. Channels connect processes through their ports. Given ports M_1 and M_2 and a channel $\langle M, \Lambda\rangle$, a *connection* between M_1 and M_2 via $\langle M, \Lambda\rangle$ consists of a pair of injective maps $\mu_i{:}M{\to}M_i$ such that $\mu_i^{-1}(M_i^+) = \mu_j^{-1}(M_j^-)$, $\{i,j\}{=}\{1,2\}$ — i.e., a connection establishes a correspondence between the two ports such that any two messages that are connected have opposite polarities. Each μ_i is called the *attachment* of M to M_i. We denote the connection by the triple $\langle M_1 \overset{\mu_1}{\leftarrow} M \overset{\mu_2}{\rightarrow} M_2, \Lambda\rangle$. Notice that every connection defines an injection $\langle \mu_1, \mu_2\rangle$ from A_M to $A_{M_1}{\cup}A_{M_2}$ as follows: for every $m{\in}M$ and $\{i,j\}{=}\{1,2\}$, if $\mu_i(m){\in}M_i^-$ then $\langle \mu_1, \mu_2\rangle(m!) = \mu_i(m)!$ and $\langle \mu_1, \mu_2\rangle(m¡) = \mu_j(m)¡$.

Definition 3 (t-ARN) *A timed asynchronous relational net consists of:*

- *A simple finite graph $\langle P, C\rangle$ where P is a set of nodes and C is a set of edges. Note that each edge is an unordered pair $\{p, q\}$ of nodes.*
- *A labelling function that assigns a process $\langle \gamma_p, \Lambda_p\rangle$ to every node p and a connection $\langle \gamma_c, \Lambda_c\rangle$ to every edge c such that:*
  - *If $c{=}\{p,q\}$ then γ_c is a pair of attachments $\langle M_p \overset{\mu_p}{\leftarrow} M_c \overset{\mu_q}{\rightarrow} M_q\rangle$ for some $M_p{\in}\gamma_p$ and $M_q{\in}\gamma_q$.*
  - *If $\gamma_{\{p,q\}}{=}\langle M_p \overset{\mu_p}{\leftarrow} M_{\{p,q\}} \overset{\mu_q}{\rightarrow} M_q\rangle$ and $\gamma_{\{p,q'\}}{=}\langle M'_p \overset{\mu'_p}{\leftarrow} M_{\{p,q'\}} \overset{\mu'_{q'}}{\rightarrow} M'_{q'}\rangle$ with $q \neq q'$, then $M_p \neq M'_p$.*

For every (T-ARN) α, we define the following sets and mappings:

- *$A_\alpha = \bigcup_{p\in P} p.A_{\gamma_p}$ is the language associated with α.*
- *For every $p{\in}P$, ι_p is the function that maps A_{γ_p} to A_α, which prefixes the actions of A_{γ_p} with p.*
- *For every $c{\in}C$, ι_c is the function that maps A_{M_c} to A_α, which, assuming that $c = \{p, q\}$, translates the actions of A_{M_c} through $\langle p._ \circ \mu_p, q._ \circ \mu_q\rangle$.*
- *$\Lambda_\alpha = \{\lambda{\in}\Lambda(A_\alpha) : \forall p{\in}P(\lambda|_{\iota_p}{\in}\Lambda_p) \wedge \forall c{\in}C(\lambda|_{\iota_c}{\in}\Lambda_c)\}$.*

Note that, for every $p{\in}P$, $(_|_{\iota_p})$ first removes the actions that are not in the language $p.A_p$ and then removes the prefix p. Similarly, for every $c{=}\{p,q\}{\in}C$,

$(_|_{\iota_c})$ first removes the actions that are not in the language $\langle p._\circ\mu_p, q._\circ\mu_q\rangle(A_{M_c})$, then removes the prefixes p and q, and then projects onto the language of M_c.

As an example, consider a bank portal that mediates the interactions between clients and the bank in the context of different business operations such as a credit card request. Fig. 1 depicts a T-ARN with two interconnected processes that implement that business operation. Process *Clerk* is responsible for the interaction with the environment and for making decisions on credit card requests, for which it relies on the process *CreditValidator* that validates whether the requesters do not have bad credit (e.g., unpaid collections or recent offences). The behavior of these processes and the channel used for communication are subject to time-related constraints ensuring that the decision on a credit card request is always issued within twenty time units since the reception of the request.

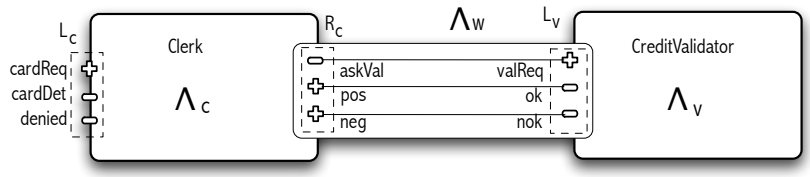

Fig. 1. An example of a T-ARN with two processes connected through a channel.

The graph of the T-ARN consists of two nodes c:*Clerk* and v:*CreditValidator* and an edge $\{c, v\}$:w_{cv}.

- *Clerk* is a process with two ports. In port L_c, the process receives messages *cardReq* and sends *cardDet* (a message carrying the card details) and *denied*. Port R_c has outgoing message *askVal* and incoming messages *pos* and *neg*. The behaviour of *Clerk* is as follows. After the delivery of the first *cardReq* on port L_c, *Clerk* may either simply deny the card request by publishing *denied* or ask an external validation of the requester by publishing *askVal* on R_c. In both cases, the outgoing message is published within five time units since the reception of *cardReq*. Then, *Clerk* waits ten time units for the delivery of *pos* or *neg*, upon which it publishes within three time units, respectively, *cardDet* or *denied*. If none of these messages arrives by the deadline or both arrive together, *Clerk* publishes *denied* on L_c.
- *CreditValidator* is a process with a single port (L_v) with incoming message *valReq* and outgoing messages *ok* and *nok*. When the first *valReq* is delivered, it takes no more than seven time units to publish either *ok* or *nok*.
- The port R_c of *Clerk* is connected with the port L_e of *CreditValidator* through w_{cv}:$\langle R_c \overset{\mu_c}{\leftarrow} \{m, n, k\} \overset{\mu_v}{\rightarrow} L_v, \Lambda_w\rangle$, with μ_c=$\{m \mapsto askVal, n \mapsto pos, k \mapsto neg\}$, μ_e= $\{m \mapsto valReq, n \mapsto ok, k \mapsto nok\}$. The corresponding channel is reliable and introduces at most a delay of five time units in the transmission of messages: $msg¡$ follows within five time units the first $msg!$, for $msg\in\{m, n, k\}$.

We often refer to the T-ARN through the quadruple $\langle P, C, \gamma, \Lambda \rangle$ where γ returns the set of ports of the processes that label the nodes and the pair of attachments of the connections that label the edges, and Λ returns the corresponding properties. The fact that the graph is simple – undirected, without self-loops or multiple edges – means that all interactions between two given processes are supported by a single channel and that no process can interact with itself. The graph is undirected because, as already mentioned, channels are bidirectional. Furthermore, because of the second restriction on the labelling function, different channels cannot share ports.

The alphabet of A_α is the union of the alphabets of the processes involved translated by prefixing all actions with the node from which they originate (see the definition of this translation after Def. 2). We take the set Λ_α to define the set of possible traces observed on α – those traces over the alphabet of the T-ARN that are projected to traces of all its processes and channels. Notice that

$$\Lambda_\alpha = \bigcap_{p \in P} \iota_p(\Lambda_p) \cap \bigcap_{c \in C} \iota_c(\Lambda_c)$$

That is, the behaviour of the T-ARN is given by the intersection of the behaviour of the processes and channels translated to the language of the T-ARN — this corresponds to what one normally understands as a parallel composition. Notice that the translations applied to set of traces effectively open the behaviour of the processes and channels to actions in which they are not involved.

As in [9,10], two T-ARNs can be composed through the ports that are still available for establishing further interconnections, i.e., not connected to any other port, which we call interaction-points.

Definition 4 (Composition) *Let $\alpha_1 = \langle P_1, C_1, \gamma_1, \Lambda_1 \rangle$ and $\alpha_2 = \langle P_2, C_2, \gamma_2, \Lambda_2 \rangle$ be T-ARNs such that P_1 and P_2 are disjoint, and a family $w^i = \langle M_1^i \overset{\mu_1^i}{\leftarrow} M^i \overset{\mu_2^i}{\rightarrow} M_2^i, \Lambda^i \rangle$ $(i = 1 \ldots n)$ of connections for interaction-points $\langle p_1^i, M_1^i \rangle$ of α_1 and $\langle p_2^i, M_2^i \rangle$ of α_2 such that, for every $i \neq j$: (1) $p_1^i \neq p_1^j$ or $p_2^i \neq p_2^j$; (2) if $p_1^i = p_1^j$ then $M_1^i \neq M_1^j$; (3) if $p_2^i = p_2^j$ then $M_2^i \neq M_2^j$. The composition*

$$\alpha_1 \left|\right|_{\langle p_1^i, M_1^i \rangle, w^i, \langle p_2^i, M_2^i \rangle}^{i=1\ldots n} \alpha_2$$

is the T-ARN defined as follows:

- *Its graph is $\langle P_1 \cup P_2, C_1 \cup C_2 \cup \bigcup_{i=1\ldots n} \{p_1^i, p_2^i\} \rangle$*
- *Its labelling function coincides with that of α_1 and α_2 on the corresponding subgraphs, and assigns to the new edges $\{p_1^i, p_2^i\}$ the label w^i.*

Fig. 1 also illustrates the composition of T-ARNs: the depicted T-ARN is the composition of the two atomic T-ARNs defined by *Clerk* and *CreditValidator*.

3 Consistency

In this section, we investigate conditions under which we can prove that a given T-ARN is consistent. Consistency is an important property of any component

algebra [7]: in our setting, it establishes that the processes can work together as interconnected via the channels. We also aim for conditions that are closed under composition so that the consistency of a T-ARN can be derived from that of its parts. Our conditions rely on closure properties and a generalisation of the property of being 'progress-enabled' proposed in [10] for un-timed behaviour.

Definition 5 (Consistent t-ARN) *A* T-*ARN* α *is* consistent *if* $\Lambda_\alpha \neq \emptyset$.

In [10], we defined a sub-algebra of (un-timed) ARNs that are consistent and closed under composition. The characterisation of this sub-algebra relied on the closure operator induced by the Cantor topology over action sequences. The same closure operator can be defined over timed traces but, for the purpose of separating the properties required of the action sequences from those of the time sequences and the way they can be checked over automata (which we do in Sec. 4), it is useful to consider other notions of closure.

We can use the Cantor topology over $(2^A)^\omega$ to define a notion of closure relative to a fixed time sequence:

Definition 6 (Closure relative to time) *We say that a timed property* Λ *is* closed relative to time *or, simply, t-closed, iff, for every* $\tau \in \Lambda_{time}$, Λ_τ *is closed. A t-closed process/channel is one whose property is t-closed. A t-closed* T-*ARN is one in which all processes and channels are t-closed.*

Processes and channels that are closed relative to time define safety properties in the usual un-timed sense: over a fixed time sequence, which cannot be controlled by the processes or channels, the violation of the property can be checked over a finite trace. We consider now operations on time sequences.

Definition 7 (Time refinement) *Let* $\rho{:}\mathbb{N}\rightarrow\mathbb{N}$ *be a monotonically increasing function that satisfies* $\rho(0)=0$.

- *Let* τ, τ' *be two time sequences. We say that* τ' refines τ *through* ρ, *which we denote by* $\tau' \preceq_\rho \tau$, *iff, for every* $i \in \mathbb{N}$, $\tau(i) = \tau'(\rho(i))$. *We say that* τ' refines τ, *which we denote by* $\tau' \preceq \tau$, *iff* $\tau' \preceq_\rho \tau$ *for some* ρ.
- *Let* $\lambda = \langle \sigma, \tau \rangle$, $\lambda' = \langle \sigma', \tau' \rangle$ *be two timed traces. We say that* λ' *refines* λ *through* ρ *— which we denote by* $\lambda' \preceq_\rho \lambda$ *— iff* $\tau' \preceq_\rho \tau$ *and, for every* $i \in \mathbb{N}$, $\sigma(i) = \sigma'(\rho(i))$ *and, for every* $\rho(i) < j < \rho(i+1)$, $\sigma'(j) = \emptyset$. *We also say that* λ' *refines* λ *— which we denote by* $\lambda' \preceq \lambda$ *— iff* $\lambda' \preceq_\rho \lambda$ *for some* ρ.
- *The r-closure of a set* Λ *of timed traces is* $\Lambda^r = \{\lambda' : \exists \lambda \in \Lambda (\lambda' \preceq \lambda)\}$
- *We say that* Λ *is* closed under time refinement *or, simply, r-closed, iff* $\Lambda^r \subseteq \Lambda$.
- *An r-closed process/channel is one whose property is r-closed. An r-closed* T-*ARN is one in which all processes and channels are r-closed.*

That is, a time sequence refines another if the former interleaves time observations between any two time observations of the latter. Refinement extends to traces by requiring that no actions be observed in the finer trace between two consecutive times of the coarser trace. Therefore, the r-closure of a process

adds all possible interleavings of empty observations to its traces, capturing its behaviour in any possible environment. This is related to mechanisms such as stuttering [1], which ensure that components do not constrain their environment.

It is not difficult to prove that the refinement relation is a complete meet semi-lattice, the meet of two time sequences τ_1 and τ_2 being given by the recursion

$$\tau(i+1) = min(\{\tau_1(j) > \tau(i), j \in \mathbb{N}\} \cup \{\tau_2(j) > \tau(i), j \in \mathbb{N}\})$$

together with the base $\tau(0) = 0$. It is also easy to prove that:

Proposition 8 *If a* T*-ARN α is t-closed (resp., r-closed), then Λ_α is also t-closed (resp., r-closed).*

A property that was found to be relevant in [10] for characterising consistent (un-timed) asynchronous relational nets concerns the ability to make joint progress. In the timed version, we analyse progress in relation to given time sequences.

Definition 9 (Progress-enabled) *For any* T*-ARN α and time sequence τ, let*

$$\Pi_{\alpha_\tau} = \{\pi \in (2^{A_\alpha})^* \colon \forall p \in P(\pi|_{\iota_p} \in \Lambda^f_{p_\tau}) \wedge \forall c \in C(\pi|_{\iota_c} \in \Lambda^f_{c_\tau})\}$$

We say that α is progress-enabled in relation to τ *iff*

$$\epsilon \in \Pi_{\alpha_\tau} \quad \textit{and} \quad \forall \pi \in \Pi_{\alpha_\tau} (\exists \tau' \preceq \tau \exists A \subseteq A_\alpha ((\pi \cdot A) \in \Pi_{\alpha_{\tau'}} \wedge \tau'_{|\pi|} = \tau_{|\pi|}))$$

We say that α is progress-enabled *iff there is a time sequence τ such that α is progress-enabled in relation to every $\tau' \preceq \tau$.*

The set Π_{α_τ} consists of all the segments that the processes and channels can jointly engage in across the time sequence τ. Being progress-enabled relative to τ means that, after any initial joint segment, the processes and channels can make joint progress along a refinement of that time sequence. The reason for using a refinement of τ is that progress may depend on the activities performed at the interaction points of α. Note that, because the intersection of A with the alphabet of any process or channel can be empty, being progress-enabled does not require all parties to actually perform an action.

By itself, being progress-enabled does not guarantee that a T-ARN is consistent: moving from finite to infinite behaviours requires the analysis of what happens 'at the limit'. However, if we work with t-closed and r-closed properties, the limit behaviour will remain within the T-ARN:

Theorem 10 *A* T*-ARN is consistent if it is t-closed, r-closed and progress-enabled.*

We now show how T-ARNs can be guaranteed to be progress-enabled by construction. Every T-ARN that consists of a single node labelled with a process P is progress-enabled in relation to at least a time sequence. This is because processes are consistent. If we take the r-closure of P, then the T-ARN is progress-enabled. In [10], we gave criteria for the composition of two (un-timed) progress-enabled ARNs to be progress-enabled based on the ability of processes to buffer incoming messages – being 'delivery-enabled' – and of channels to buffer published messages – being 'publication-enabled'. In a timed domain, it becomes necessary to identify time sequences across which all parties can work together.

Definition 11 (Delivery-enabled) *Let* $\alpha{=}\langle P, C, \gamma, \Lambda\rangle$, $\langle p, M\rangle{\in}I_\alpha$ *one of its interaction-points, and* $D_{\langle p,M\rangle}{=}\{p.m¡\colon m{\in}M^+\}$. *We say that* α *is delivery-enabled in relation to* $\langle p, M\rangle$ *if, for every* $\tau{\in}\Lambda_{time}$, $(\pi{\cdot}A){\in}\Pi_{\alpha_\tau}$ *and* $B{\subseteq}D_{\langle p,M\rangle}$, *there exists* $\tau'{\preceq}\tau$ *such that* $(\pi{\cdot}B \cup (A{\setminus}D_{\langle p,M\rangle})){\in}\Pi_{\alpha_{\tau'}}$ *and* $\tau'_{|\pi|+1}{=}\tau_{|\pi|+1}$.

That is, being delivery-enabled at an interaction point requires that, for every time sequence, any joint segment of the T-ARN over that sequence can be extended by any set of messages delivered at that interaction-point, after which it will behave according to a refinement of the original trace. Note that this does not interfere with the decision of the process to publish messages: $B{\cup}(A{\setminus}D_{\langle p,M\rangle})$ retains all the publications in A.

Definition 12 (Publication-enabled) *Let* $h{=}\langle M,\Lambda\rangle$ *be a channel and* $E_h = \{m!{:}m{\in}M\}$. *We say that* h *is publication-enabled iff, for every* $\tau{\in}\Lambda_{time}$, $(\pi{\cdot}A){\in}\Lambda^f_\tau$ *and* $B{\subseteq}E_h$, *there exists* $\tau'{\preceq}\tau$ *such that* $\pi{\cdot}(B{\cup}(A{\setminus}E_h)){\in}\Lambda^f_{\tau'}$ *and* $\tau'_{|\pi|+1}{=}\tau_{|\pi|+1}$.

The requirement here is that, for any time sequence and any segment of a trace over that time sequence, the segment can be extended by the publication of any set of messages, i.e., the channel should not prevent processes from publishing messages when they are in a state in which they could do so. Notice that this does not interfere with the decision of the channel to deliver messages: $(B{\cup}(A{\setminus}E_h))$ retains all the deliveries present in A.

Theorem 13 *Let* α *be a composition of r-closed progress-enabled* T*-ARNs through the connections* $w^i = \langle M^i_1 \overset{\mu^i_1}{\leftarrow} M^i \overset{\mu^i_2}{\rightarrow} M^i_2, \Lambda^i\rangle$, $i = 1\ldots n$, *i.e.,*

$$\alpha = (\alpha_1 \;\|^{i=1\ldots n}_{\langle p^i_1, M^i_1\rangle, w^i, \langle p^i_2, M^i_2\rangle}\; \alpha_2)$$

If, for $i{=}1\ldots n$, α_1 *is delivery-enabled in relation to* $\langle p^i_1{,}M^i_1\rangle$, α_2 *is delivery-enabled in relation to* $\langle p^i_2{,}M^i_2\rangle$ *and* $h^i{=}\langle M^i{,}\Lambda^i\rangle$ *is publication-enabled and r-closed, then* α *is progress-enabled.*

Therefore, the proof that an r-closed T-ARN is progress-enabled can be reduced to checking that individual processes are delivery-enabled in relation to their interaction points and that the channels used for composition are publication-enabled. To guarantee that the T-ARN is consistent, it is sufficient to choose processes and channels that are t-closed (implement safety properties). All the checking can be done at design time, not at composition time (which, in the service-oriented paradigm, is done at run time).

4 The automata-theoretic view

We now show how the properties introduced in the previous section can be checked over orchestrations of services based on automata-based models of processes and channels. We adopt Timed I/O Automata similar to those presented in [6], except that we use discrete time and sets of actions instead of single

actions for transitions. As δ represents the minimal interval between two time observations, all the durations in the automata are in $\mathbb{N}_\delta^+$, the positive multiples of δ. We will use $\mathbb{N}_\delta$ to refer to $\mathbb{N}_\delta^+ \cup \{0\}$.

Let $\mathbb{C}$ be a finite set (of clocks). A *clock valuation* over $\mathbb{C}$ is a mapping v: $\mathbb{C} \to \mathbb{N}_\delta$. Given $d \in \mathbb{N}_\delta$ and a valuation v, we denote by $v+d$ the valuation defined by, for any clock $c \in \mathbb{C}$, $(v+d)(c) = v(c)+d$. Given $R \subseteq \mathbb{C}$ and a clock valuation v, we denote by $v^{\mathbf{R}}$ the valuation where clocks from R are reset, i.e., such that $v^{\mathbf{R}}(c) = 0$ if $c \in R$ and $v^{\mathbf{R}}(c) = v(c)$ otherwise. Let op be the set of relational operators $op = \{\leq, \geq\}$. A *guard* over $\mathbb{C}$ is a finite conjunction of expressions of the form $c \bowtie n$ with $\bowtie \in op$ and $n \in \mathbb{N}$. We denote by $\mathcal{B}(\mathbb{C})$ the set of guards over $\mathbb{C}$.

Definition 14 (DTIOA) *A Discrete Timed I/O Automaton (DTIOA) is a tuple* $\mathcal{A} = \langle Loc, q_0, \mathbb{C}, E, Act, Inv \rangle$ *where:*

- *Loc is a finite set of locations and $q_0 \in Loc$ is the initial location;*
- *$\mathbb{C}$ is a finite set of clocks;*
- *$Act = Act^I \cup Act^O$ is a finite set of actions partitioned into inputs and outputs;*
- *$E \subseteq Loc \times 2^{Act} \times \mathcal{B}(\mathbb{C}) \times 2^{\mathbb{C}} \times Loc$ is a finite set of edges;*
- *Inv: $Loc \to \mathcal{B}(\mathbb{C})$ associates an invariant with every location.*

In addition, we impose that every DTIOA is r-closed*: for all $l \in Loc$, $(l, \emptyset, \phi, \emptyset, l) \in E$ for some valid ϕ in $\mathcal{B}(\mathbb{C})$.*

Being r-closed means that, in every location, it must be possible to make an empty observation without affecting the system. Intuitively, this reflects openness to environments that are involved in the execution of actions not included in Act.

An execution starting in location l_0 and clock valuation v_0 is an alternating sequence

$$(l_0, v_0, d_0) \xrightarrow{S_0, R_0} (l_1, v_1, d_1) \xrightarrow{S_1, R_1} \ldots$$

where: for every i, $l_i \in Loc$, v_i is a clock valuation over $\mathbb{C}$, $S_i \subseteq Act$ and $R_i \subseteq \mathbb{C}$; $d_0 \in \mathbb{N}_\delta$ and, for $i>0$, $d_i \in \mathbb{N}_\delta^+$; and, for every i: (1) $Inv(l_i)(v_i + t)$ holds for all $0 \leq t \leq d_i$, (2) $v_{i+1} = (v_i + d_i)^{\mathbf{R_i}}$ and (3) there is $(l_i, S_i, C, R_i, l_{i+1}) \in E$ such that $C(v_i + d_i)$ holds. The language of $\mathcal{A}$, which we denote by $\Lambda_{\mathcal{A}}$, is the set of executions such that $l_0 = q_0$, $v_0(c) = 0$, for all $c \in \mathbb{C}$, and $d_0 > 0$.

In *deterministic* DTIOAs, for every location l and valuation v such that $Inv(l)(v)$ holds and $S \subseteq Act$, there exists at most one edge $(l, S, C, _, _) \in E$ such that $C(v)$ holds. In this way, in these automata, the current state (l, v), the duration d of the stay in l and the next symbol S determine the next state (l', v') uniquely.

An execution in $\Lambda_{\mathcal{A}}$ defines a timed trace $\lambda = \langle \sigma, \tau \rangle$ over $Act^I \cup Act^O$ where $\sigma(0) = \emptyset$, $\tau(0) = 0$ and, for $i \geq 0$, $\sigma(i+1) = S_i$ and $\tau(i+1) = \tau(i) + d_i$. We denote by $[\![\mathcal{A}]\!]$ the set of timed traces defined by the set of executions in $\Lambda_{\mathcal{A}}$, which is r-closed in the sense of Def. 7. For example, the timed traces defined by the DTIOA in Fig. 2 are those in which either no input is ever received (in

which case the system is idle forever) or, after the delivery of the first *valReq*, it takes no more than seven time units for the system to publish either *ok* or *nok* (after that, the system is open to inputs but does not publish anything more). Those traces correspond to the property that defines the behaviour of the process *CreditValidator* presented before.

A DTIOA $\mathcal{A}$ is *consistent* if $\Lambda_{\mathcal{A}} \neq \emptyset$ and has *consistent states* if, for every l and v such that $Inv(l)(v)$ holds, there exists an execution of $\mathcal{A}$ starting in (l, v). Notice that a DTIOA that has consistent states is not necessarily consistent. Indeed, although having consistent states implies that there is an infinite execution starting in the initial state, it could be the case that this execution has an initial duration d_0=0. Thus, in the following, we assume that DTIOA are consistent *and* have consistent states.

An important class of DTIOAs are those that are able to receive any set of inputs at all times (input-enabledness) and that, at each step, provide outputs that do not depend on the received inputs (independence). One way to ensure that DTIOA satisfy both requirements is to leverage the notions of delivery-enabledness and publication-enabledness of T-ARNs to the automata setting:

Definition 15 (DP-enabled DTIOA) *A DTIOA* $\mathcal{A} = \langle Loc, q_0, \mathbb{C}, E, Act, Inv \rangle$ *is* DP-enabled *if, for every* $B \subseteq Act^I$*, clock valuation* v*, and edge* $(l, A, C, R, l') \in E$ *such that the following properties hold —* $Inv(l)(v)$*,* $C(v)$ *and, for all* $0 \leq t \leq \delta$*,* $Inv(l')(v^{\mathbf{R}} + t)$ *— there is an edge* $(l, B \cup (A \setminus Act^I), C', R', l'') \in E$ *such that* $C'(v)$ *holds and, for all* $0 \leq t \leq \delta$*,* $Inv(l'')(v^{\mathbf{R}'} + t)$ *also holds.*

For a DTIOA to be input-enabled and independent, all edges need to be adaptable to accept any set of inputs without changing the associated outputs. Although the target locations of edges and clock resets may be modified when changing inputs, they are required to be enabled for execution at least in the same situations as the original ones.

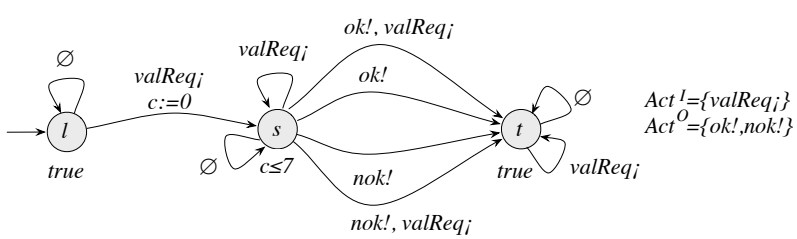

Fig. 2. An example of a DTIOA.

Definition 16 (DTIOP) *A DTIO process consists of a set* $\gamma_{\mathcal{P}} = \{M_1, ..., M_n\}$ *of mutually disjoint ports and a deterministic DP-enabled DTIOA* $\mathcal{A}_{\mathcal{P}}$ *that is consistent, has consistent states and for which* $Act^I = \cup_i A_{M_i^+}$ *and* $Act^O = \cup_i A_{M_i^-}$*.*

The inputs of a DTIOP are deliveries m¡ of incoming messages and outputs are publications m! of outgoing messages at the ports. The language of a DTIOP is

that of its DTIOA, i.e., $[\![\mathcal{P}]\!] = [\![\mathcal{A}_\mathcal{P}]\!]$. For example, the port L_V in Fig. 1 and the DTIOA in Fig. 2 define a DTIOP provided we choose $\delta<1$. The automaton is obviously deterministic, has consistent states and, if $\delta<1$, it is also DP-enabled.

As before, interconnection of DTIO processes is achieved through channel implementations, also defined in terms of DTIOA.

Definition 17 (DTIOC) *A DTIO channel (or DTIOC) consists of a set M of messages and a deterministic DP-enabled DTIOA $\mathcal{A}$ that is consistent, has consistent states and for which $Act^I=\{m!{:}m\in M\}$ and $Act^O=\{m¡{:}m\in M\}$.*

Notice that deliveries are outputs for channels (inputs for processes) and publications are inputs for channels (outputs for processes). This is because, messages published by a process are delivered to another process through a channel: if a process $\mathcal{P}_1$ is connected to a process $\mathcal{P}_2$ via a channel $\mathcal{C}$, the publication of a message m by $\mathcal{P}_1$ is an output for $\mathcal{P}_1$ and an input for $\mathcal{C}$; the delivery of m is an output for $\mathcal{C}$ and an input for $\mathcal{P}_2$.

Every DTIOP $\mathcal{P}$ defines a t-closed and r-closed process $P_\mathcal{P} = \langle \gamma_\mathcal{P}, [\![\mathcal{A}_\mathcal{P}]\!]\rangle$ in the sense of Sec. 2. Similarly, every DTIOC $\mathcal{C}=\langle M, \mathcal{A}\rangle$ defines a t-closed and r-closed channel $C_\mathcal{C}=\langle M, [\![\mathcal{A}_M]\!]\rangle$. Most importantly, $P_\mathcal{P}$ and $C_\mathcal{C}$ meet the conditions required for the application of Theo. 13.

Theorem 18 *If $\mathcal{P}$ is a DTIOP, then $P_\mathcal{P}$ is* DP-enabled *in relation to any of its ports and is* progress-enabled. *If $\mathcal{C}$ is a DTIOC, then $C_\mathcal{C}$ is* publication-enabled.

A DTIO net (or DTION) is defined in the same way as a T-ARN except that DTIOPs and DTIOCs are used instead of processes and channels, respectively. Every DTION $\mathcal{N}$ defines the T-ARN $\alpha_\mathcal{N}$ obtained by replacing the DTIOPs and DTIOCs with the corresponding processes and channels. By construction, $\alpha_\mathcal{N}$ is t-closed and r-closed. The semantics of a DTION can be defined in terms of the classical partially synchronized product of DTIOA, which we recall briefly.

Definition 19 (Product) *Two DTIOA $\mathcal{A}_i = \langle Loc^i, q_0^i, \mathbb{C}^i, E^i, Act_i, Inv^i\rangle$ are compatible iff $\mathbb{C}^1\cap\mathbb{C}^2=Act_1^I\cap Act_2^I=Act_1^O\cap Act_2^O=\emptyset$. The composition of two compatible DTIOA is $\mathcal{A}_1\|\mathcal{A}_2=\langle Loc^1\times Loc^2, (q_0^1, q_0^2), \mathbb{C}^1\cup\mathbb{C}^2, E, Act, Inv\rangle$ where:*

- $Act^I = (Act_1^I\backslash Act_2^O)\cup(Act_2^I\backslash Act_1^O)$
- $Act^O = Act_1^O\cup Act_2^O$
- *for all* $(q_1,q_2)\in Loc^1\times Loc^2$, $Inv((q_1,q_2))=Inv^1(q^1)\wedge Inv^2(q_2)$
- $((q_1,q_2),S,C,R,(q_1',q_2'))\in E$ *iff:* $(q_1,S_1,C_1,R_1,q_1')\in E^1$, $(q_2,S_2,C_2,R_2,q_2')\in E^2$, $C = C_1\wedge C_2$, $S_i = S\cap Act_i$ *for* $i = 1,2$, *and* $R = R_1\cup R_2$.

Note that, by construction, when $S\cap Act_1 \neq \emptyset$ and $S\cap Act_2 \neq \emptyset$, all actions on which $\mathcal{A}_1$ and $\mathcal{A}_2$ synchronize (i.e., actions in $S\cap Act_1\cap Act_2$) are necessarily inputs on one side and outputs on the other. After composition these actions become outputs. Furthermore, transitions such that $S\cap Act_i = \emptyset$, which are usually considered as non-synchronizing, are handled as synchronizing transitions with underlying r-closure loops.

Proposition 20 *Given compatible DTIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, $[\![\mathcal{A}_1 \| \mathcal{A}_2]\!] = \iota_1([\![\mathcal{A}_1]\!]) \cap \iota_2([\![\mathcal{A}_2]\!])$, where ι_1 and ι_2 translate the local languages to that of the composition, as in Def. 3.*

In order to show that this notion of product can be used to capture the semantics of DTIONs and that this semantics is compositional, consider the simple case of a DTION $\mathcal{N}$ consisting of two nodes p_1 and p_2 labelled with $\mathcal{P}_1 = \langle \gamma_1, \mathcal{A}_{\mathcal{P}_1} \rangle$ and $\mathcal{P}_2 = \langle \gamma_2, \mathcal{A}_{\mathcal{P}_2} \rangle$, respectively, and an edge c between them labelled with the connection $\mathcal{C} = \langle M_1 \overset{\mu_1}{\leftarrow} M \overset{\mu_2}{\rightarrow} M_2, \mathcal{A}_M \rangle$ where $\langle M, \mathcal{A}_M \rangle$ is a DTIOC. We use prefixing as in Sec. 2, i.e., we denote by $p.\mathcal{A}$ the copy of $\mathcal{A}$ where all actions are prefixed by p.

The connection $\mathcal{C}$ defines a DTIOA $\mathcal{A}_\mathcal{C}$ that is a copy of $\mathcal{A}_M$ except that the alphabet is renamed using the injection $\langle p_1._ \circ \mu_1, p_2._ \circ \mu_2 \rangle$ defined as in Sec. 2 to enforce synchronization of $\mathcal{P}_1$ and $\mathcal{P}_2$ on the ports M_1 and M_2: given a message $m \in M$, the action $m¡$ is renamed $p_i.m¡$ if $\mu_i(m) \in M_i^+$ and the action $m!$ is renamed $p_i.m!$ if $\mu_i(m) \in M_i^-$. More precisely, if $\mathcal{A}_M = \langle Loc^M, q_0^M, \mathbb{C}^M, E^M, Act^M, Inv^M \rangle$, then $\mathcal{A}_\mathcal{C} = \langle Loc^M, q_0^M, \mathbb{C}^M, E^C, Act_C, Inv^M \rangle$ where:

- $Act_C^O = \{p_1.m¡ : m \in M_1^+ \cap \mu_1(M)\} \cup \{p_2.m¡ : m \in M_2^+ \cap \mu_2(M)\}$
- $Act_C^I = \{p_1.m! : m \in M_1^- \cap \mu_1(M)\} \cup \{p_2.m! : m \in M_2^- \cap \mu_2(M)\}$
- $E^C = \{(q_c, \langle p_1._ \circ \mu_1, p_2._ \circ \mu_2 \rangle(A), C, R, q_c') : (q_c, A, C, R, q_c') \in E^M\}$

We can now define the semantics of $\mathcal{N}$ as the product of $p_1.\mathcal{A}_{\mathcal{P}_1}$, $p_2.\mathcal{A}_{\mathcal{P}_2}$ and $\mathcal{A}_\mathcal{C}$. Notice that, because of the renamings, the three DTIOA are compatible and $\mathcal{A}_\mathcal{C}$ ensures that the synchronization only occurs between messages that are related through the maps μ_1 and μ_2. In other words, given a message $m \in M$ such that $\mu_1(m) = m_1 \in M_1^-$ and $\mu_2(m) = m_2 \in M_2^+$: the action $m!$ from $\mathcal{A}_M$ is renamed as $p_1.m_1!$ in $\mathcal{A}_\mathcal{C}$ (an input), and will thus synchronize with $m_1!$ of $\mathcal{P}_1$ (an output), i.e., $p_1.\mu_1(m)!$ in the composition — $\mathcal{P}_1$ synchronizes with $\mathcal{C}$ to publish m_1; the action $m¡$ from $\mathcal{A}_M$ is renamed as $p_2.m_2¡$ in $\mathcal{A}_\mathcal{C}$ (an output), and will thus synchronize with $m_2¡$ of $\mathcal{P}_2$ (an input), i.e., $p_2.\mu_2(m)$ in the composition — $\mathcal{C}$ synchronizes with $\mathcal{P}_2$ to deliver the message. Because of the renaming of the actions of $\mathcal{P}_1$ and $\mathcal{P}_2$ by prefixing them with p_1 and p_2, respectively, no other synchronizations take place.

Theorem 21 (Compositional semantics) *Given a DTION $\mathcal{N}$ as above,*

$$[\![\mathcal{N}]\!] = [\![p_1.\mathcal{A}_{\mathcal{P}_1} \parallel \mathcal{A}_\mathcal{C} \parallel p_2.\mathcal{A}_{\mathcal{P}_2}]\!] = \iota_{p_1}([\![\mathcal{P}_1]\!]) \cap \iota_c([\![\mathcal{C}]\!]) \cap \iota_{p_2}([\![\mathcal{P}_2]\!]) = \Lambda_{\alpha_\mathcal{N}}$$

That is, the semantics of $\mathcal{N}$ is $\alpha_\mathcal{N}$: the product of the DTIOAs that implement the processes and connections of the net generates the set of timed traces obtained through Def. 3 – the semantics of the corresponding T-ARN. The result can be generalized to arbitrary DTIONs by calculating the products corresponding to all interconnections.

5 Related work

Several frameworks have been proposed for component/service-based software systems that exhibit timed properties. Some, such as [15,16], adopt the π-calculus

to address subclasses of timing activities, e.g., timeouts and local urgency in web transactions. Others adopt an algebraic framework: for example, [4] adopts timed data streams for a channel-based coordination model, and [8,11,14,18] address service choreography using timed automata, i.e., they focus on the modelling of the (timed) conversation protocols that characterise the global behaviour of a (fixed) number of peers that exchange services. One of the properties that the latter analyse is compatibility – whether the conversation protocols (modelled as timed automata) followed by the peers lead to deadlocks or time conflicts that prevent them from completing (e.g., reaching final states).

Although compatibility relates to the notion of consistency that we address in this paper, our emphasis is not on choreography but on orchestration: what we are investigating is in what conditions we can guarantee that the orchestrations of two services can work together when they bind to each other. This has implications on the properties that are required of timed-automata in order to guarantee consistency. Because we aim to support run-time binding and composition, those properties are different from those investigated for choreography (where composition is analysed at design time). An example is the way time is managed: in choreography, this is done globally for the (fixed) set of peers (in the sense that clocks can be set or reset by all peers); in our approach, this needs to be done locally at level of each process because composition is dynamic.

The interaction model is another key aspect of a theory of services. Most service models are synchronous even if message-passing is more adequate for the loosely-coupled operating environment of services [14]. An asynchronous timed model is considered in [11], but only indirectly by simulating buffers in a synchronous setting, which limits the properties that can be analysed.

The problem of guaranteeing the consistency of composition without having to calculate the product of automata (or other models of orchestration) has remained largely ignored in the literature (e.g., [5,6]) as its relevance comes to the fore in service-oriented computing thanks to the crucial distinction that needs to be made between design-time and run-time checks or operations.

Results on the consistency of the composition of Timed I/O Automata (TIOA) are addressed in [6] for the restricted class of TIOA that are input-enabled and allow independent progress, which are directly relevant for our paper. However, their results are based on a weaker notion of consistency according to which a TIOA that does not accept any non-Zeno timed sequence can still be consistent, which is not sufficient to ensure that the composition of TIOA accepting non-empty sets of timed traces is a TIOA that also accepts at least one timed trace. Fig. 3 illustrates this situation: both automata allow independent progress, are input-enabled, and can produce infinite timed traces; however, their composition yields a TIOA that can only produce Zeno sequences.

The same class of TIOA is considered in [13], where their I/O feasibility is investigated. Although this is richer than consistency because in this context TIOA executions are also not necessarily time divergent. Hence, the results that establish sufficient conditions for the composition of I/O feasible TIOA to be

I/O feasible (based on progressive and receptive TIOA) cannot be transposed to the world of sets of timed traces.

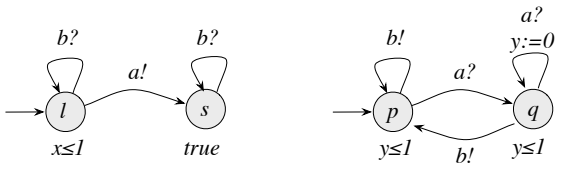

Fig. 3. Input and progress-enabled TIOA that do not generate any joint trace.

6 Concluding remarks

In this paper, we have investigated how a component algebra can be defined over timed traces that addresses run-time composition of services. Services are orchestrated by asynchronous networks of processes and can bind dynamically to required services. Our results include the characterisation of a sub-algebra over which the binding can be proved to be consistent using only design-time properties of the orchestrations, i.e., without having to make further checks at run time (which would undermine real-time operation). We showed how discrete timed I/O automata provide a compositional implementation model for that algebra and identified a class of DTIOA that conform to the properties that ensure consistency of composition – those that are deterministic and DP-enabled. These results extend the literature on TIOA, which so far had not addressed the issues raised by run-time composition.

Our model uses a time unit (in the domain of the reals) for observations but it is not discrete in the sense that, because clock valuations are not restricted to the time unit, the behaviour of TIOA can be constrained by real-time guards. However, this time granularity is shared by all processes. Although this is adequate for service-level agreements in typical business transactions, a non-discrete model would allow us to capture heterogeneity and address a more general class of systems. However, non-Zeno models fail to satisfy the topological properties over which we rely to ensure consistency of networks, namely that refinement defines a complete meet semi-lattice, which could lead to situations in which joint behaviour is only possible by forcing actions to be executed over successively shorter delays to converge on a deadline. We are currently investigating intermediate models over more restricted structures of actions.

We are also investigating t-closure over DTIOA in relation to traditional characterisations of safety properties over time sequences, e.g., nondeterministic Büchi automata [2]. This will allow us to use logics such as Safety MTL [17] to define an interface algebra for T-ARNs similar to [9,10] and investigate the use of model-checking techniques for validating orchestrations in relation to interfaces.

Acknowledgments

This work was partially supported by Fundação para a Ciência e Tecnologia under contract (PTDC/EIA-EIA/103103/2008).

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82(2):253–284, 1991.
2. B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.
3. R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *REX Workshop*, volume 600 of *LNCS*, pages 74–106. Springer, 1991.
4. F. Arbab and J. J. M. M. Rutten. A coinductive calculus of component connectors. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *WADT*, volume 2755 of *Lecture Notes in Computer Science*, pages 34–55. Springer, 2002.
5. C. Chilton, M. Z. Kwiatkowska, and X. Wang. Revisiting timed specification theories: A linear-time perspective. In M. Jurdzinski and D. Nickovic, editors, *FORMATS*, volume 7595 of *LNCS*, pages 75–90. Springer, 2012.
6. 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. ACM, 2010.
7. L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *EMSOFT*, volume 2211 of *LNCS*, pages 148–165. Springer, 2001.
8. G. Díaz, J. J. Pardo, M.-E. Cambronero, V. Valero, and F. Cuartero. Verification of web services with timed automata. *Electr. Notes Theor. Comput. Sci.*, 157(2):19–34, 2006.
9. J. L. Fiadeiro and A. Lopes. An interface theory for service-oriented design. In *FASE*, volume 6603 of *LNCS*, pages 18–33. Springer, 2011.
10. J. L. Fiadeiro and A. Lopes. Consistency of service composition. In *FASE*, volume 7212 of *LNCS*, pages 63–77. Springer, 2012.
11. N. Guermouche and C. Godart. Timed model checking based approach for web services analysis. In *ICWS*, pages 213–221. IEEE, 2009.
12. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.
13. D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Morgan & Claypool Publishers, 2006.
14. R. Kazhamiakin, P. K. Pandya, and M. Pistore. Representation, verification, and computation of timed properties in web. In *ICWS*, pages 497–504. IEEE Computer Society, 2006.
15. C. Laneve and G. Zavattaro. Foundations of web transactions. In *FoSSaCS*, volume 3441 of *LNCS*, pages 282–298. Springer, 2005.
16. A. Lapadula, R. Pugliese, and F. Tiezzi. C-clock-WS: A timed service-oriented calculus. In *ICTAC*, volume 4711 of *LNCS*, pages 275–290. Springer, 2007.
17. J. Ouaknine and J. Worrell. Safety metric temporal logic is fully decidable. In *TACAS*, volume 3920 of *LNCS*, pages 411–425. Springer, 2006.
18. J. Ponge, B. Benatallah, F. Casati, and F. Toumani. Analysis and applications of timed service protocols. *ACM Trans. Softw. Eng. Methodol.*, 19(4), 2010.